Specifying Process-Oriented Hypertext for Organizational Computing *

John Noll
University of Colorado, Denver
and
Walt Scacchi
University of California, Irvine

December 9, 1999

Abstract

Organizations deploy intranets to provide access to documents for those who use them. But the web of computing comprises more than just documents: people, tools, and processes are critical to organizational function. In particular, people may need guidance on how to perform tasks, as well as access to information necessary to carry out those tasks.

In this paper, we present a language for describing process-oriented hypertexts. A process-oriented hypertext links information, tools, and activities into a seamless organizational web. Using such a hypertext, and process performer can enact a process by browsing, and receive guidance on how to perform the process activities, where to find relevant information, and what tools to use.

The PML process modeling language provides a way for process engineers to specify process models in terms of activities, and the sequence in which they should be performed. The specification can be elaborated with descriptions of resources and tools required and provided by activities, and the skills necessary to carry out an activity. The resulting models are then translated into one or more process-oriented hypertexts that represent instances of the process currently being performed.

PML includes features that allow the modeler to specify how the process activities should be dynamically linked to information and resource nodes at the time the process is performed. This enables processes to be described as abstract models that can be instantiated as process-oriented hypertexts in different organizational settings.

We used PML to model processes in a case study of the grants management process at the US Office of Naval Research. We describe some of our experiences applying PML to this study, and conclude with lessons learned and directions for future study.

*Submitted to *Journal of Networking and Computer Applications,* special issue on hypermedia extensibility.

Contents

1 Introduction

Intranets have been widely deployed as a way to make important documents accessible to the diverse range of potential users within an organization. However, the common use of an intranet is mainly as an information delivery mechanism. Users access data warehouses, corporate documents, and ad-hoc department web pages by browsing hypertext links, relying on their own experience to determine which links ultimately lead to the desired documents. Thus, users are responsible for determining which documents are necessary to the task at hand, and for finding and retrieving those documents from the intranet.

By contrast, *workflow* technology uses sequential activity descriptions to automate the flow of documents through the organization. Each step in the workflow modifies the document as it flows through the process. This can be viewed as the delivery of documents to the document consumer, according to some process specification.

Kling and Scacchi describe an array of relationships among organizational processes, workflow tasks, documents, application systems, development tools, users and other resources as the "web of computing" [8]. While conventional notions of intranets and workflows address a particular aspect of the web of computing, neither by itself is sufficient to support the wide spectrum of organizational relationships and activities. A workflow application may deliver a document that is the main focus of a particular task, but the human agent responsible for performing that task may require additional information (procedure manuals, regulatory documentation, historical data) that is part of the work context to complete the task.

For example, a grants officer in a research funding agency may receive a request for funds dispersement from a research investigator via a workflow mechanism. In order to validate and approve the request, the officer requires the original grant text, rules governing allowable expenses, and information describing the current status of the grant funds. Intranets provide accessibility to these supporting documents, but only if the officer knows where to find them.

In order to support the entire information processing and sharing activities of people working in an organization, we must determine and specify what is required in an environment that incorporates widespread information accessibility, guidance in performing individual tasks, and coordination among organizational activities. We call such an environment an "organizational web", to reflect its role in supporting the web of computing [21]. The problem we therefore choose to address is how to describe an organizational web, and how to provide an information environment that supports the definition, specification, analysis and interpretation of organizational webs.

While the concept of the web of computing has been formalized into a knowledge-based process meta-model [13], our interest here is to employ and demonstrate the use of the core of this representation as an ontology for building domain-specific hypertext environments and applications [18].

Hypertext is the appropriate conceptual model for an organizational web. Hypertext linking allows the evolutionary construction of relationships among the wide diversity information artifacts used by an organization. Our own research has shown that the hypertext data organization model is an effective integration approach to diverse information models [14]. In addition, the browsing model of interaction is now familiar to a vast population of computer users. Finally, the notion of *guided tours* through hypertext links can be used to guide users through sequences of tasks.

This latter aspect of hypertext can be exploited to elevate the intranet from a passive data repository to *process-oriented hypertext* that provides active support of organizational processes.

Process-oriented hypertexts are hypertexts of linked documents and tasks across an organizational web. The document web within the organizational web is composed of legacy data sources and new documents; the process web is overlayed (conceptually) onto the document web as a set of linked tasks.

It is possible to hand craft a process-oriented hypertext using integration technologies like Common Gateway Interface (CGI) programs, scripting languages, database application generators, and workflow tools. In this approach, business rules are encoded into scripts or other procedural specifications. Such specifications begin with a model of the process to be implemented, but because the process is implicitly represented as a sequence of application and tool invocations, database transactions, and other operations glued together with procedural code, the process itself is opaque to most users. This makes it difficult to validate the implementation against the actual process, and difficult to analyze the result for improvement. Furthermore, they require a variety of specific programming skills in addition to process analysis and modeling expertise, making it difficult for actual process owners and performers to (re-)design their user or domain specific processes.

In this paper we describe an alternate approach using the PML process modeling language, and related tools, to describe organizational webs in a form that enables modeling, validation, analysis, and execution.

The advantage of a process modeling language like PML is that the process is represented *explicitly*, rather than implicitly in an implementation. A single PML specification can be translated into several process representations (textual or graphic), which are tangible, visual, and explicit. Thus, they can be quickly reviewed, and opportunities for revision can be quickly identified, by process owners and users, without knowledge for how processes might be implemented by an information system

PML specifications can be developed and maintained by process users or process owners, either as program text or through visual modeling tools. Thus, process owners and users can be empowered to design and redesign process flows, resource dependencies, organizational constraints and business rules.

This means process owners and users can readily develop and articulate workflow and information sharing patterns that make their work interesting and more satisfying, since they can see ahead to where their process flows are going, as well as where they came from.

Hypertext environments like the Web are typically used to represent dispersed, linked object spaces. The states of these objects are not usually constituted nor employed as a model of computation, except for the protocols (e.g., HTTP) that support object transfer.

Processes are often represented as directed, attributed graphs, where the attributes of the graph's nodes and links specify or define their data type. Hypertext is a convenient information structure for representing directed, attributed graphs. Hypertext is thus a plausible choice for representing descriptions or models of processes.

Processes are a foundation as an executable model of computation. For example, operating systems are designed to manage computational processes and their associated resources.

Semantic hypertexts, those whose nodes and links are attributed and typed, are well suited as representations for processes that can be modeled, analyzed, and interpreted (or compiled) as an executable model of computation.

Semantic hypertexts or process-oriented hypertexts are useful for modeling, analyzing, interpreting or executing organizational work processes supported by a networked information infrastructure (e.g., a process-directed intranet [21] that serves as an "organizational operating system." Such an organizational operating system thus provides a model of computation which links organizational process users and owners to the information resources–artifacts, documents, application programs, tools, network infrastructure, etc.–that constitute their web of computing [8]. Similarly, it provides the means whereby a virtual enterprise (an internetwork of organizations) can can be modeled, analyzed, visualized, prototyped, and enacted in a way where hypertext-based representations of organizations, processes and associated information can be articulated and evolved [15, 21, 16].

Process-oriented hypertext, together with a hypertext model of the organization, user roles, product model, application model, tool model, and resource model of corresponding objects located in Internet-based repositories, constitutes a domain-specific hypertext environment that can be tailored to different process-oriented domains [18]. This enables the modeling, analysis and interpretive browsing of process flow.

Process-oriented hypertexts are also a central representational component that facilitates the integration of heterogeneous repositories of information artifacts found in domains such as software engineering [14]. Process-oriented hypertexts support versioning and configuration control, especially when multiple repositories are employed, each with different or heterogeneous change-control policies and support mechanisms [15]. Process-oriented hypertext and their associated modeling, compilation and run-time environments, enable *process prototyping*: the rapid prototyping of wide-area, multi-site organizational processes [22, 21] using collaborative process simulators [19].

Process-oriented hypertexts extend traditional hypertext environments by providing mechanisms that support process *guidance*, *control* and *monitoring*. Guidance means providing explicit process control flow constructs that provide *navigational control semantics* that direct a process user to browse designated hypertext-based information objects and to perform user role-specific tasks and tool invocations which evolve the state of the visited objects. Control ensures that process steps are performed in the correct order, and that each step produces the desired result. Monitoring includes process measurement, recording/timestamping of process events, potential replay of process steps, etc. useful for planning and improvement [16].

Process-oriented hypertext supports a multi-level scheme for integrating new or legacy data, applications, and tools that collectively constitute an Internet-based infrastructure for dynamically configuring virtual enterprises [15, 16]. Process-oriented hypertext can also support the collaborative re-engineering of complex organizational processes in ways that facilitate the empowerment of end-users who can access, browse, and manipulate multi-view (textual or graphic) process descriptions [21, 20].

PML constitutes our most recent realization and unification of the preceding concepts and capabilities for developing and using process-oriented hypertext. In the next section, we present the PML language features and representation formats. Then, we discuss enactment of processes specified using PML. Following that, we examine related research. Finally, we conclude with our observations and results.

2 The PML Language

PML is a process specification language that enables process engineers to describe organizational processes in a form that can be translated into a variety of representations. Using PML, one can describe processes in terms of an organizational web of tasks, products, tools, agents, and supporting documents needed to perform the tasks. The resulting process specification can then be compiled into a process-oriented hypertext that represents an instance of the process being performed. When deployed in a networked environment, a process-oriented hypertext serves as the basis for a *process-driven intranet* [21].

PML was developed as part of a research study conducted for the US Office of Naval Research. The study examined ONR's grants management process with the goal of streamlining the process and deploying process support on an intranet [21, 20].

The analysis was conducted using process analysis techniques developed at the University of Southern California's ATRIUM Laboratory [22]. This methodology involved a sequence of process capture and refinement steps:

1. Interview individuals to capture process steps. Interviewers used a tabular format to describe the processes in outline form.

2. Translate interview outlines and notes into a formal, high level process description.

3. Conduct collaborative group validation and analysis of the resulting models, during which process performers critiqued and refined process descriptions by examining graphical depictions of processes generated from the high level specifications.

4. Develop executable prototypes of process enactment systems based on refinements and improvements resulting from the group analysis.

This methodology led to a requirement for a modeling formalism that could be translated into executable prototypes, while at the same time being flexible enough to enable incremental process modeling. We also needed a language that would allow the modeler to generate new models and revise existing models quickly, to enable rapid refinement during "live" validation sessions with actual process performers. Finally, we needed to incorporate existing (and future) tools and documents, that were part of ONR's infrastructure, into the process descriptions, to capture the interaction of processes with the broader organizational context.

2.1 Language Features

A PML process description specifies the tasks or activities that comprise a process, and the sequence in which they should be performed. Thus, a PML specification has two parts: description of the tasks or activities to be performed, and specification of the flow of control defining the order of task performance. We describe each of these in turn.

2.1.1 Actions

The activities of a process are called *actions*. Actions represent primitive steps in a process. An action specification may have several fields:

```
process proposal_submit {
    action submit_proposal {
        requires { proposal_file }
        provides { proposal }
        output { proposal.id }
        agent { PrincipalInvestigator }
        tool { http//gilligan.usc.edu/process/onr/forms/submit-proposal.html }
    }

    branch submit_supporting_docs {
        action submit_certs {
            requires { certification_file }
            provides { certification }
            agent { PrincipalInvestigator }
            tool { http//gilligan.usc.edu/process/onr/forms/submit-certs.html }
        }
        action submit_budget {
            requires { budget_file }
            provides { budget }
            output { budget.id }
            agent { PrincipalInvestigator }
            tool { http//gilligan.usc.edu/process/onr/forms/create-budget.html }
        }
    }
}
```

Figure 1: A proposal submittal process fragment described in PML

Name The name of the action.

Type The action type, either "manual" or "executable." An action typically is a task or activity performed by a human agent. However, many processes also include steps that can be completely automated. PML allows for such steps using the executable type.

Agent A specification of the role an agent performing the action should have.

Script A narrative description of the action. One of the chief advantages of enactable process descriptions is their usefulness for providing guidance to agents performing the process. The description field can be used to provide such guidance, as well as links to supporting documentation.

For executable actions, this field contains a script to be executed by the enactment mechanism to perform the action.

Tool The tool(s) used to perform the action.

Requires, Provides Resources required to perform the action, and provided as a side effect of performing the action. The purpose of enacting a process is to produce or modify some tangible resource, called the *product* of the process. The resource fields of an action specify any objects (file, Web page, string, etc.) input to the action, and the products produced by the action.

Input, Output Fields to specify data passed across the enactment environment boundary, for example input obtained from users via forms, or status messages to users that are not stored in a repository.

Of these, only the name is required; thus, PML specifications can evolve from high level process skeletons, that describe only the names of actions and the sequence in which they are to be performed, to detailed specifications that can be compiled into an executable form to produce dynamic process hypertexts. Figure 1 includes examples of several action declarations.

2.1.2 Control flow

A process specification describes the tasks (actions) that should be performed to carry out the process, and the sequence in which those tasks should be performed. While our experience has shown that most processes are sequential in nature, there are situations when one must specify concurrent or alternate tasks, or sequences of tasks that should be repeated.

PML has four control flow constructs for specifying the order in which actions should be performed: *sequence*, *iteration*, *selection*, and *branch*. Using this small set of control flow primitives, it is possible to capture a wide variety of situations involving decisions and concurrency.

Sequence specifies a set of actions that should be performed one at a time in a specific order. Figure 2 shows an example of a sequence.[1]

[1] The PML examples in this section are fragments of actual models of the grants management process developed during the ONR study.

```
sequence HandleNewContractor {
    action GetContCodeForNewUniversity {
        requires { GrantDocFile }
        agent { PT }
    }
    action AssignNewContractorCode {
        requires { GrantDocInfo }
        agent { PT }
    }
    action RequestContractorProfile {
        requires { GrantDocFile }
        agent { PT }
    }
}
```

Figure 2: An example of a PML sequence

In this example, the sequence represents the set of tasks required to establish a new grant awardee. In general, sequences are useful for modeling the hierarchical structure of processes: sub-processes are represented by sequences, primitive actions by actions. Figure 3 depicts the hierarchical breakdown of a portion of the grants administration process, from which these examples are taken.

Selection *Selections* specify a set of actions, only one of which should be performed. They are used to describe decision points in the process. The example in Figure 4 illustrates how PML models decisions using selections: a request for documents should be sent *either* to the Program Officer (PO) if it is for an internal document, or to the Grant Officer (GO) if it is not.

Branch A *branch* specifies a set of actions that can performed concurrently in any order. The actions enclosed in a branch block must all be completed before the process can proceed past the block, but the order in which they are performed is unimportant.

Branches are useful for describing the situation that occurs in many real processes: there is a nominal sequence in which actions should be performed, but in reality persons performing the actions do them in parallel, switching frequently among tasks.

This example in Figure 5 models compliance review, which has several components: physical property review, purchasing system review, audits, etc. Each of these activities can proceed in parallel with the others; although there is a "normal" sequence in which they are started, this is not a requirement that needs to be enforced.

The parallel nature of branches means that each arm of the branch represents a sub-process that executes concurrently with its neighbors.

```
                    PROCESS GrantAdministration_ToBe

* SEQUENCE AdministrationPrep
     + ACTION HandleNewContractor
          o requires ElecGrantFolder
          o agent PT
* END AdministrationPrep
* SELECT AdministrationFunctions
     + SELECT HandleCompliance
          o BRANCH InstitutionalCompliance
               # ACTION PropertySystemwideReview
                    @ requires ElecGrantFolder
                    @ agent AGO
               # ACTION PurchasingSystemReview
                    @ requires ElecGrantFolder
                    @ agent AGO
               # ACTION AuditComplianceAndResolution
                    @ requires ElecGrantFolder
                    @ agent AGO
               # ACTION FinancialSystemCompliance
                    @ requires ElecGrantFolder
                    @ agent AGO
               # ACTION ReportingSystemCompliance
                    @ requires ElecGrantFolder
                    @ agent AGO
               # ACTION IndirectCostMonitoring
                    @ requires ElecGrantFolder
                    @ agent AGO
               # ACTION FraudWasteAuditInvestigations
                    @ requires ElecGrantFolder
                    @ agent AGO
          o END InstitutionalCompliance
          o BRANCH TransactionCompliance
  ...
* END AdministrationFunctions
```

Figure 3: Task Decomposition Outline for Grants Administration Process

```
select SendRequest {
    action SendToPOifONR {
        requires { DocumentRequest }
        agent { GA }
    }
    action SendToGOifNotONR {
        requires { DocumentRequest }
        agent { GA }
    }
}
```

Figure 4: An example of selection in PML

```
branch InstitutionalCompliance {
    action PropertySystemwideReview {
        requires { ElecGrantFolder }
        agent { AGO }
    }

    action PurchasingSystemReview {
        requires { ElecGrantFolder }
        agent { AGO }
    }

    action AuditComplianceAndResolution {
        requires { ElecGrantFolder }
        agent { AGO }
    }
    ...
}
```

Figure 5: A PML branch

```
iteration FollowUp {
    action FollowUpNCERequest {
        requires { DocumentRequest }
        agent { GA }
    }
}
```

Figure 6: Iteration in PML

Iteration Iteration occurs over sequences of actions; the agent enacting the process decides whether to enter or skip the loop by selecting either the first action in the loop, or the first action following the loop.

In Figure 6, the follow up action is repeated until the grant administrator receives the desired reply (or gives up).

2.2 Resources

Resources required and provided by processes conform to a *product model* of the process target domain. The product model enables the specification of resources according to the Articulator resource taxonomy [13].

The purpose of the PML "provides" and "requires" resource fields is to specify predicates that define pre- and post-conditions of actions in a process.

The resource fields of an action declaration in a PML process model specify what resources (products) the action *requires* for execution, and *provides* after execution. In the process fragments above, a name in a resource field defines a variable that is bound to an instance of a resource at run time; these bindings are resolved within the context of the virtual repository, a logical warehouse or database which contains all of the resource instances to which processes may apply (see section 3).

In addition to resource variable, resource fields may contain expressions that specify constraints on the state of resources required as input for an action (the *requires* declaration), and assert what state those resources have when the action completes (the *provides* declaration). These expressions have the following form:

```
requires { resourceName.attributeName op value }
provides { resourceName.attributeName op value }
```

Op may be any comparison operator: `==`, `!=`, `<`, `>`, etc. The *value* may be any value appropriate for the type of the resource under comparison. In a *requires* field, the resulting expression is a predicate indicating the required state of the input resource. In a *provides* field, the expression is an assertion of the state the resource will be in when the action is performed correctly.

The compiler translates *requires* fields into dynamic link specifications that link action nodes to resources during process enactment.

Some additional fields are used to specify resources used or produced by actions:

Input Specifies input data obtained from outside the product repository.

```
input { name:string, title:string, contents:file }
```

specifies that the action requires a name and title string, and a contents file. These values are typically entered into a form by a human agent.

Output Specifies data that are provided to the world external to the repository.

```
output {  accessCode:string }
```

specifies that the current action produces a string called *accessCode*. This would normally be presented to a human agent, as part of the result display of the action. The implication in this case is that the user should remember the accessCode for future use; the action documentation should include this reminder.

2.3 Compilation

The PML compiler can translate PML process specifications into a variety of different formats for analysis and execution, including process-oriented hypertext. A PML process-oriented hypertext consists of a set of action nodes, static links to incorporate supporting documents, and dynamic links to represent the control flow and implement resource bindings.

Process instances can serve as an integrating component of an organizational web. Users can browse process hypertexts to examine the status of resources, learn what their peers are working on, understand organizational activities, as well as to perform process tasks. Thus the organizational knowledge is linked by the process hypertext to the organizational activity. In addition, the process hypertext presents a particular context for interpreting other on-line resources, one based on the actual work carried out by an organization.

To achieve this integration, PML process specifications must be translated into components of a process-oriented hypertext. There are four such components:

1. Action nodes, that represent the tasks in the process.
2. The Process Graph, that represents the hierarchy and sequence of actions in the process.
3. Document and tool links, that link actions to documents, tools, and other information necessary to complete the task.
4. Resource bindings, that link actions to the resources they require or modify.

The compiler generates an action node for each action specification in a PML model. The action node serves as the primary integration component in a process-oriented hypertext. The action node includes

- A description of the task to be performed. If the action is "executable", this will be an executable script to be passed to the underlying operating system for execution.
- Links to supporting documentation and tools.
- Links to resources that the action will use or modify.

Links between actions represent the control flow of a process. Process performers can enact an instance of the process by traversing these links to visit action nodes. They can also browse the process graph to learn about the process.

Static links from action nodes to essential documentation are specified by markup tags in the description fields of actions. These allow the process modeler to incorporate portions of the organizational web into a process, and are one component of process integration.

The action sequence links, and links to documents and tools, can be determined at compile time, as these are part of the process specification. However, links between actions and the resources they modify must be determined during process enactment. This is because some resources may not even exist until they are created by actions, or may not be in the correct state until modified by a previous action.

Thus, links from actions to resources are computed dynamically by the process enactment mechanism. The compiler generates queries to the hypertext repository that are executed during enactment to determine whether resources are available to an action.

Dynamic links greatly increase the power of a PML description. Using dynamic linking, multiple instances of a given process can be created by overlaying different control flow and resource links for each process instance. The PML enactment engine updates these overlays as the processes are enacted, allowing the same action nodes to be part of multiple process instances.

Returning to the grant award example, this means that the same PML process description can be used to guide a grant officer through the steps required to manage multiple grants; the progress of each grant through the process is tracked by a separate process instance, with its own set of control flow and resource links.

2.4 Process Prototyping

Ultimately, a PML description is translated into a process-oriented hypertext for enactment in a real organizational setting. Before this can happen, the PML action specifications must be elaborated with presentation information: a narrative description of the action for human consumption, links to related documents, and input forms for capturing user input if necessary.

The PML compiler provides a prototyping facility to support incremental development of these action node descriptions.

This facility constructs a JavaScript program from the PML control flow, and the action script definitions, that enables a process engineer to step through the process sequence and view each action screen or form. This allows rapid evaluation of the user interface component of a PML process specification, without the need for the complete PEOS execution environment. An example is shown in Figure 7.,

3 Enacting PML Processes

The PML enactment mechanism, called the Process Enactment Operating System (PEOS), has four main components:

1. Virtual Repository Interface.
2. PEOS server.
3. PML Virtual Machine.
4. Hypertext Browser, and other user interfaces.

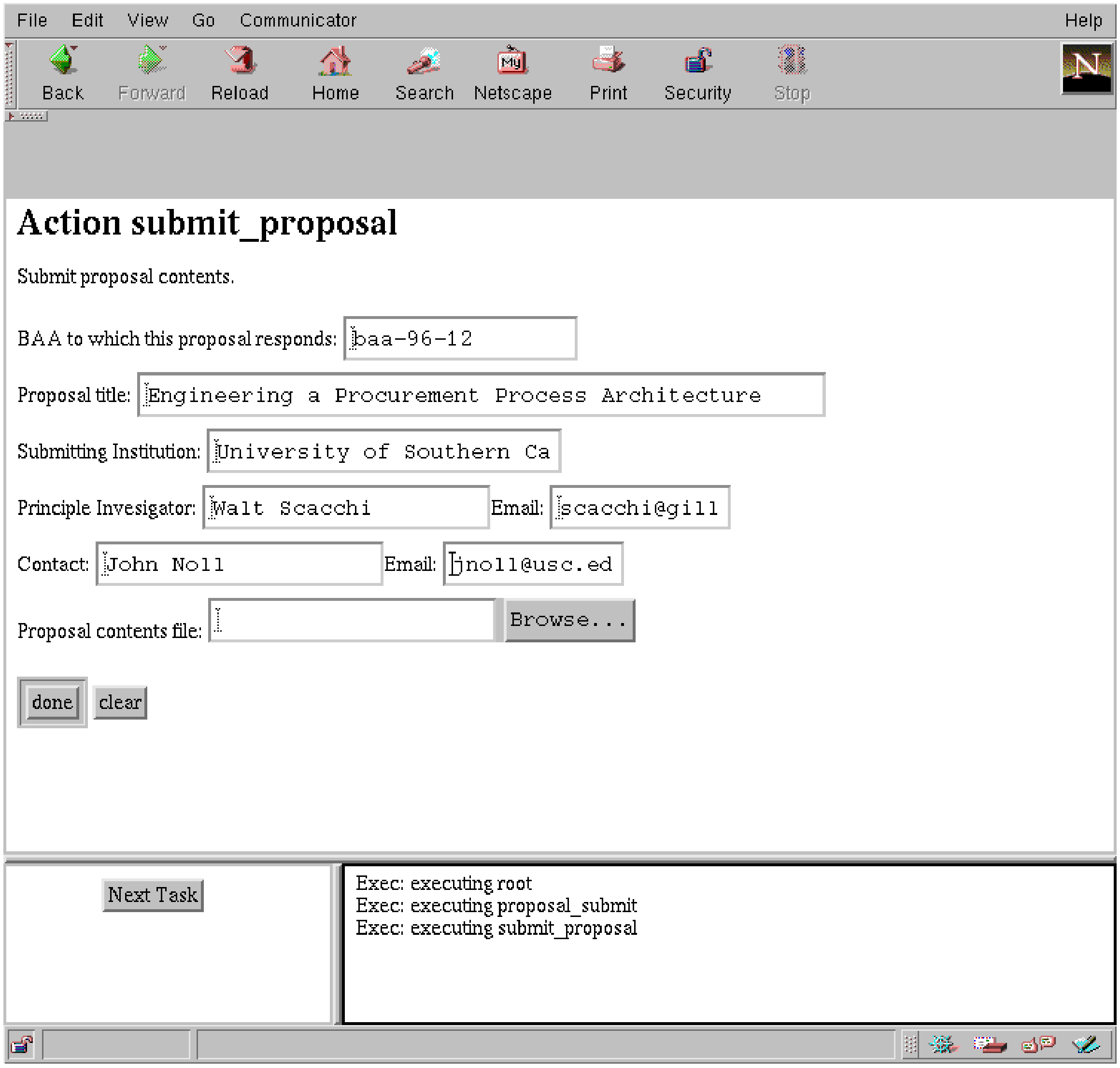

Figure 7: PML Prototyping Facility

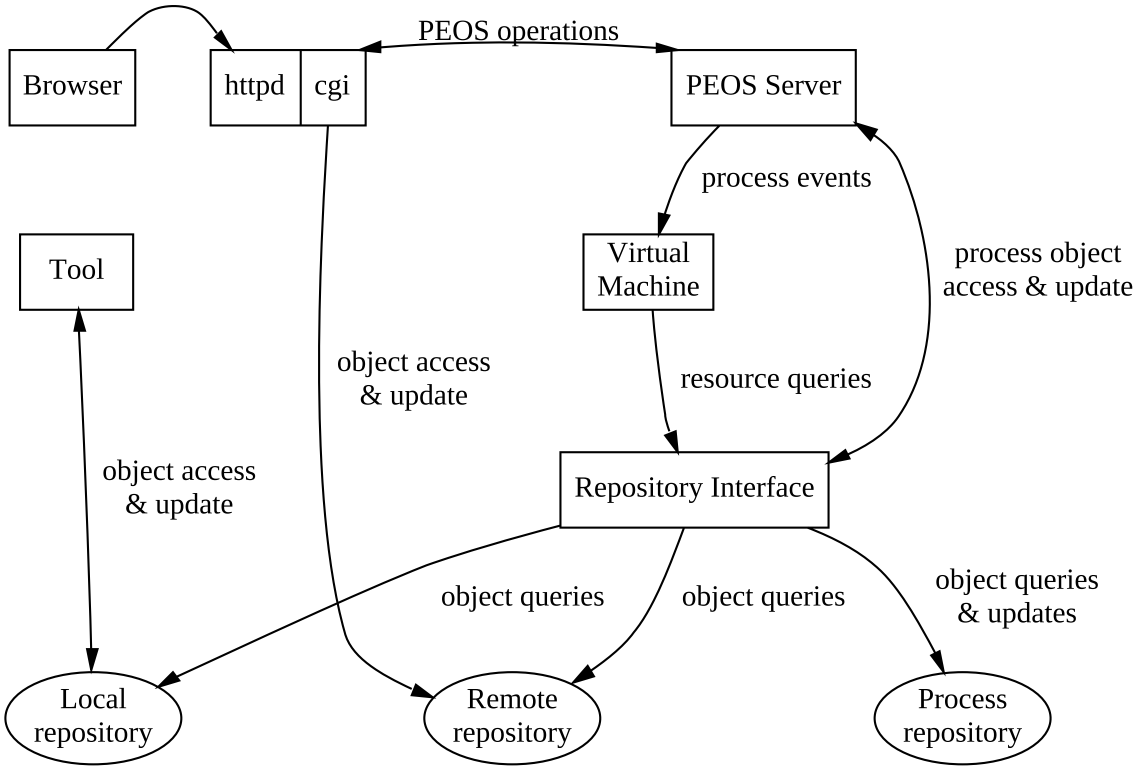

Figure 8: PML Enactment Architecture

These are depicted in Figure 8.

Process performers, and other information users, access both information and process objects primarily by browsing. PEOS also supports additional user interfaces, including a simple command line interface and one based on message exchange using email services.

3.1 Virtual Repository Interface

The Virtual Repository Interface integrates a collection of distributed, autonomous, heterogeneous information sources and storage managers into a logically centralized virtual repository using a mechanism similar to that described in [14].

The objects and relationships in each information store are presented to clients of the virtual repository as objects with unique global identifiers and attributes. Storage management and internal representation remain the responsibility of the respective information store.

In the PEOS environment, both process objects (action nodes) and information objects (documents, etc.) are presented as hypertext nodes; relationships among these objects, including task precedence, document structure, etc. are presented as links among the information or process nodes.

The repository interface exports the operations shown in Figure 1. Note that tools update objects directly through an individual information store's native interface.

Operation	arguments	description
Create	*attributes*	Creates a new object with the specified attributes.
Update	*object attribute=value...*	Updates the specified attributes of *object*. These are used by the dispatcher and virtual machine to maintain process state.
Query	*attribute op value ...*	Queries the virtual repository for the set of objects satisfying the specified comparisons. This operation is used by the virtual machine to determine whether an action's required resources exist in the necessary state.
Assert	*attribute op value ...*	Queries the virtual repository for the set of objects satisfying the specified comparisons, and returns true if objects exist that match the query. This operation is useful for verifying that an action completed with the desired effect.

Table 1: Virtual Repository Interface

Request message	arguments	result or reply
Login	*user id, password*	The specified agent is authenticated, and an session for that agent started; subsequent operations during the session are bound to the agent.
List	*none*	Lists the compiled PML models available for instantiation.
Available	*none*	Returns a list of all active or suspended actions bound to requesting agent.
Select	*action*	The specified action becomes the requesting agent's active action. If there is already an active action bound to the agent, that action is suspended.
Suspend	*action*	Explicitly suspend the specified action.
Done	*action*	The specified action has been completed, and should transition to done.
Abort	*action*	The specified action should be aborted.
Logout	*none*	The session is ended.

Table 2: PEOS Protocol

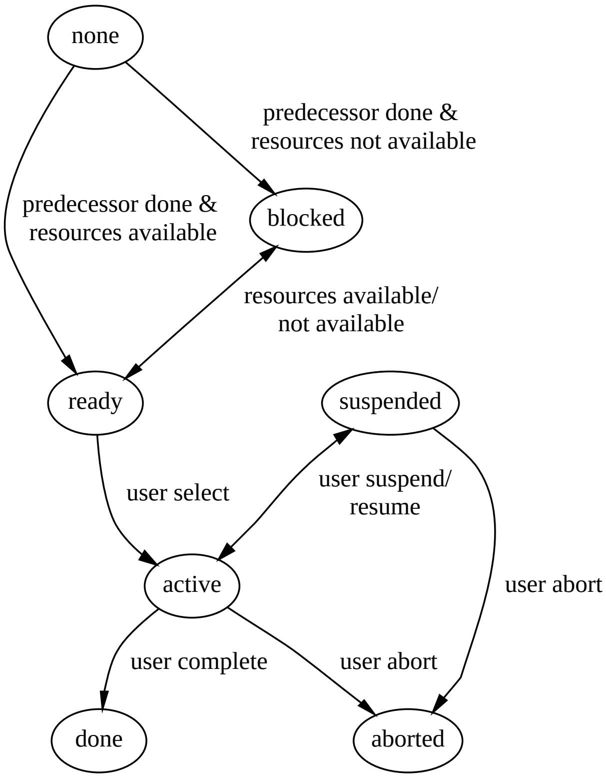

Figure 9: Action State Transitions

3.2 PEOS Server

The PEOS server handles process requests and events received from clients via the PEOS protocol, shown in Table 2.

The **Done** request is particularly important, as it initiates the chain of operations that update process state. When the PEOS server receives a **Done** request, it does the following:

1. Invokes the Virtual Machine (see below) with the event type and identifier of the completed action as arguments.

2. Queries the process repository for any actions in the "blocked" state.

   For each such action found, the server executes the action's required resource query to see if the action may now be unblocked. Since the completion of an action implies that resources have been created or modified, it is possible that some blocked actions' required resources may now be available.

3.3 The PML Virtual Machine

The state of each action in a given process instance changes over time, as the process progresses and resources are created or modified. The state of each action is recorded in the process repository; a process's state is the aggregate of its action states.

Changes in action state are caused by three events (see Figure 9):

1. Completion of a predecessor task. This event causes the next action (or actions, in the case of a branch) in the process sequence to transition to the "active" state, if the successor's required resources are available. If the successor's resources are not available, it transitions to the blocked state.

2. Explicit notification from process performers about action state. Process performers (users) do the actual work in a process and, by selecting actions to perform, performing the actions, and then notifying the system that the action is complete.

3. Modifications to resources, as a side effect of performing process actions.

Selecting an action causes it to become "active"; if the action is part of a selection, other actions in the selection are no longer available, and thus transition to the "none" state. The "active" state indicates that a user is currently performing the action; some actions may require significant time to complete however, spanning days or weeks. Also, users may work on several tasks simultaneously, switching among them for various reasons.

A user can suspend a task explicitly, for example at the end of the work day; or implicitly, by selecting another task to perform (thus switching to another active task). The "suspend" state indicates that an action is part of a user's active tasks, but the user is not currently working on that action. Suspended actions transition to the active state when a user selects them again. Users may also abort actions, which aborts the process as well.

Users perform actions by invoking tools on the action's required resources or input, thus modifying existing resources or creating new resources. When the action is completed, other actions that were blocked because their required resources were not available, may now find that those resources are available, and can then transition to the ready state.

Process performers notify the system that an action is complete by sending "action complete" messages, typically by clicking a button attached to an action node. These events indicate an update to the process state is required. Computing new state in response to action complete events is the responsibility of the PML virtual machine. The PML virtual machine takes the current process state, and the completion event as input, and computes the new process state using a compiled form of the process flow control. This is accomplished in four steps:

1. Verify that the just completed action modified its resources appropriately. The compiler translates the action's *provides* resource specification into an assertion on the repository. The virtual machine executes this assertion when it receives an "action complete" event for an action; if the assertion fails, the action did not accomplish what it was supposed to do, and the process is aborted. If the assertion succeeds, the action transitions to the "done" state.

2. Compute the successor actions. The virtual machine next examines the process control flow to determine the successors of the just completed action. These form the set of possible next actions in the process.

3. Query for required resources. Each successor action may require specific resources before it can proceed. The virtual machine must determine whether each action's required resources are available, before transitioning the action to the "ready" state.

4. Update process state. The virtual machine sets the state attribute of each action to either "ready" or "blocked", then waits for completion events. When a process performer visits an action node whose state is "ready", that action can be performed, ultimately resulting in another completion event. The cycle is then repeated.

The compiler translates each action's *requires* resource specification into a query for the virtual repository interface. The virtual machine executes these queries to determine whether the required resources are available. If the query is successful, the action can transition to the "ready" state; otherwise, it becomes "blocked."

The *requires* field contains a resource name and, optionally, predicates constraining its state. The first reference to a resource name in a PML model may contain a type specifier as well. The compiler generates code for the first reference to each resource, that

1. Queries the repository for objects of the specified type whose state matches any predicates.

2. Binds the resource name to one of the objects matched by the query. Subsequent references to that resource name will refer to the bound object.

3. If no objects match the query, the action is blocked.

This mechanism causes objects to flow through a process in a fashion.

4 Related Work

Two active research communities focus on enactment of complex, dynamic processes: *workflow* research approaches the problem from a business process perspective, while *software process enactment* research is motivated by the goal of automating software engineering activities. While originating in different domains, both communities are converging on a set of common problems [24].

Osterweil's seminal paper [17] established the notion of *process programming* as a way to apply techniques of software development to process modeling and enactment. This notion proposes that processes be described using languages similar to programming languages, enhanced with specific constructs for modeling software processes. Since then considerable effort has yielded a variety of languages for describing software processes, including APPL/A [27], JIL [29], and PML.

Several approaches to executing such descriptions have been implemented.

One approach starts with a conventional programming language (for example, Ada) and enhances it with process modeling constructs. The resulting language is then used to build "process aware" software tools, which are applications compiled from programs written in the enhanced programming language.

Another approach embeds a process language interpreter into a software development environment. The process descriptions may be in the form of rules [2, 3, 12], Petri-Nets [25, 10], or process programming languages [1]. The interpreter uses the process description to direct the development process, by guiding and constraining the invocation of tools integrated into the environment. Garg and Jazayeri present an excellent survey of process oriented software development environments in [7]. Nonetheless, process-oriented software development environments have been criticized for the limited focus on processes that can be carried out using such an environment [30];

The widespread deployment of advanced web browsers has sparked research into providing process support via the World-Wide Web to widely distributed organizations. This is typically accomplished by providing an HTTP gateway to some existing workflow or process engine; a few projects, however, seek to embed process support into the browser itself, for example via Java "agents" [4].

Workflow automation has achieved enormous commercial success automating routine business processes. Commercial workflow software frequently uses a message-based model for task automation, in which human enactors are notified of pending tasks via electronic mail messages, which include the documents to which the tasks apply. Thus the term "workflow" is often used to describe the flow of documents through an organization, where each step in the flow applies some modification to the document [5].

The goal of contemporary workflow research is to expand the scope of workflow automation to "adaptive and dynamic work activity coordination and collaboration" [23], highly distributed "virtual corporations" and processes involving participants outside of a given organization [6]. However, common process representations in workflow software have been limited to directed graphs [5], Petri-nets [28], rules [9], and agents [4], rather than utilizing hypertext structures as suggested here.

Overall, a common thread in workflow and process research is the idea that a process model must move towards an ever more complete description of the organization that performs it; this requires a rich representation formalism that can capture the sequence of activities, the products or resources to which they apply, the human or software agents that perform them [26], and a myriad of relationships that link them. This can lead to powerful modeling notations, but unfortunately also to complicated models that are difficult to construct and understand. It also leads to self-contained execution environments that assume all activities and artifacts are part of the environment.

By contrast, PML separates the organization and its members, artifacts, and processes. Rather than describing the organization as a process, PML describes how processes fit into the larger organizational context; the organization is a framework for performing a collection of processes.

5 Conclusion

We began with the problem of how to integrate process support into the web of computing. Our goal is to provide seamless integration of process guidance, monitoring, and control with information access in the broader context of organizational computing.

We conducted a study at ONR to validate our PML-based solution. We analyzed four phases of the ONR grants administration process: pre-award, grant award, grant administration, and grant close-out. The resulting PML models totaled 128 actions. After translating these into graphical format we presented them to ONR process owners and performers in several all-day group analysis sessions.

The results exceeded our expectations. First, PML was extremely useful during the process capture and analysis phase of our study. The PML specifications, translated into process hierarchy diagrams, were surprisingly accessible to process performers; ONR personnel were able to immediately recognize their process activities, as well as errors in the initial models. PML enabled quick turnaround in correcting these errors and re-generating the diagrams (which, incidentally, spanned an entire wall in the meeting room).

In addition to being accessible, PML process descriptions, when presented as process hierarchy diagrams, revealed obvious opportunities for improvement in the processes. ONR process owners immediately suggested steps that could be streamlined or eliminated; using these suggestions, we were able to redesign ONR's grants administration process, reducing the number of actions by over half, to 54. ONR estimated these improvements could yield savings of up to $15 million per year. In addition, ongoing refinement of these processes at ONR has subsequently led to a reduction of completion time (or "cycle time") for the most common process enactment paths by more than a factor of 20 [20].

A major source of improvement in the ONR grants administration process redesign was the elimination of validation steps in the existing process necessitated by the repeated entry of data into various data system from paper forms. The data on these forms originate from grant awardees; hence, we observed that validation steps could be eliminated if the data were captured from proposal submitters at the beginning of the process (in the pre-award phase).

The PML prototyping facility was used to demonstrate the feasibility of this redesign. We were able to specify an additional up-front process (the proposal submit process depicted in Figure 1) that would capture proposals and related data from submitters, and feed them into the pre-award phase. By translating this process into an executable prototype (Figure 7), we were able to provide a convincing demonstration of how such a process might look to users.

Our experience applying PML in the ONR grants administration process study revealed the difficulty in supporting enactment of complex, concurrent processes. Complex organizations have many concurrent, coordinated processes that operate on data from a variety of information repositories.

The enactment architecture described in Section 3 is a direct outgrowth of the ONR study. In particular, the need to support legacy information systems was a critical requirement for any redesign of ONR processes involving automation. The virtual repository interface is part of our solution to this requirement.

In addition, the ONR study emphasized the need for dynamic binding of processes to resources: it became clear that an instance of the grants administration process would be active for *each* proposal and award. Thus, the enactment mechanism must bind a new process instance to incoming proposals, and to new grants when the process converts a proposal into an award. The result is the virtual machine resource binding mechanism described in Section 3.3.

Finally, ONR's organizational environment confirmed the potential of process-oriented hypertext as an integration mechanism. In performing their grants administration tasks, Grant Administrators and Grant Officers need to consult a variety of information objects, including the grant award document itself, documents provided by the awardee (proposal, budget, certifications), the grant accounts database, procedure manuals, and legal references. These are distribute among several information systems of varying sophistication and maturity, ranging from simple file servers to relational databases. Linking this diversity of information into a process-oriented hypertext provides grants personnel with a seamless information environment that is tailored to the particular domain of grants administration, but not necessarily restricted to that domain.

The widespread deployment of corporate/organizational intranets has emphasized the utility and the problems of instant information access. As the quantity of on-line information grows, users have increasing problems finding the specific data required to perform their tasks.

Process support can solve part of this problem, by bringing information and tasks together.

We conclude that conventional hypertexts make documents and knowledge accessible, but process-oriented hypertexts make them usable.

6 Acknowledgements

Cedric Knight of New Directions Technologies conducted field interviews at ONR that provided the data for constructing the PML models shown in this paper.

This research was supported by ONR through grant N00014-94-1-0889. Numerous ONR personnel contributed to this research through their enthusiastic participation in our process analysis activities.

Additional support for Professor Scacchi was provided by the Defense Acquisition University through grant N487650-27803.

Support of this research should not be construed as an endorsement by ONR, the Defense Acquistion University, or the US Navy, of any statements or findings presented herein.

References

[1] S. Abiteboul, M. Adiba, J. Arlow, P. Armenise, S. Bandinelli, L. Baresi, P. Breche, F. Buddrus, C. Collet, P. Corte, Th. Coupaye, C. Delobel, W. Emmerich, G. Ferran, F. Ferrandina, A. Fuggetta, C. Ghezzi, S.E. Lautemann, L. Lavazza, J. Madec, M. Phoenix, S. Sachweh, W. Schfer, C. Souza dos Santos, G. Tigg, and R. Zicari. The GOODSTEP project: General object-oriented database for software engineering processes. In *Proceedings 1st Asian Pacific Software Engineering Conference*, Tokyo, 1994.

[2] Naser Barghouti and Gail Kaiser. Scaling up rule-based development environments. In *Proceedings of the Third European Software Engineering Conference, Milan*, October 1991.

[3] Israel Ben-Shaul and Gail Kaiser. A paradigm for decentralized process modeling and its realization in the Oz environment. In *Proceedings of the 16th International Conference on Software Engineering, Sorrento, Italy*. IEEE Computer Society, May 1994.

[4] Ting Cai, Peter Gloor, and Saurab Nog. DartFlow: A workflow management system on the web using transportable agents. Technical Report PCS-TR96-283, Department of Computer Science, Dartmouth College, 1996.

[5] Workflow Management Coalition. The workflow reference model. Technical Report TC00-1003, Workflow Managment Coalition, January 1995.

[6] Workflow Management Coalition. Workflow and internet: Catalysts for radical change. Technical report, Workflow Managment Coalition, June 1998.

[7] Pankaj Garg and Mehdi Jazayeri. Process-centered software engineering environments: A grand tour. Technical Report TUV-1841-95-02, Technical University of Vienna, February 1995.

[8] Rob Kling and Walt Scacchi. The web of computing: Computer technology as social organization. *Advances in Computers*, 21:1–90, 1982.

[9] Wenke Lee, Gail Kaiser, Paul Clayton, and Eric Sherman. OzCare: A workflow automation system for care plans. *Journal of the American Medical Informatics Association: 1996 AMIA Annual Fall Symposium, Symposium Supplement*, pages 577–581, October 1996.

[10] F. Maurer, G. Succi, H. Holz, B. Kotting, S. Goldman, and B. Dellen. Software process support over the internet. In *Proceedings 21st International Conference on Software Engineering*, Los Angeles, May 1999.

[11] Pei-wei Mi and Walt Scacchi. Process integration in case environments. *IEEE Software*, 9(2):45–53, March 1992. Reprinted in Computer-Aided Software Engineering, (2nd Edition), E. Chikofski (ed.), IEEE Computer Society, 1993.

[12] Pei-Wei Mi and Walt Scacchi. A knowledge-based meta-model for formulating models of software development processes. *Decision Support Systems*, 17(3), 1996.

[13] Pei-wei Mi and Walt Scacchi. A meta-model for formulating knowledge-based models of software development. *Decision Support Systems*, 17(3):313–330, 1996.

[14] John Noll and Walt Scacchi. Integrating diverse information repositories: A distributed hypertext approach. *IEEE Computer*, 24(12):38–45, December 1991.

[15] John Noll and Walt Scacchi. Supporting distributed configuration management in virtual enterprises. In R. Conradi, editor, *Software Configuration Management*, volume 1235 of *Lecture Notes in Computer Science*, pages 142–160, New York, 1997. Springer-Verlag.

[16] John Noll and Walt Scacchi. Supporting software development in virtual enterprises. *Journal of Digital Information*, 1(4), February 1999.

[17] Leon Osterweil. Software processes are software too. In *Proceedings of the 9th International Conference on Software Engineering, Monterey, CA*, March 1987.

[18] Walt Scacchi. The power of domain-specific hypertext environments. *Journal of the American Society of Information Science*, 40(3):183–191, May 1989.

[19] Walt Scacchi. Experiences in software process simulation and modeling. *Journal of Systems and Software*, 46(2-3):183–192, April 1999.

[20] Walt Scacchi. Reengineering contracted service procurement for internet-based electronic commerce: A case study. *Journal of Information Technology and Management*, (to appear, 1999).

[21] Walt Scacchi and John Noll. Process-driven intranets: Life-cycle support for process engineering. *IEEE Internet Computing*, September/October 1997.

[22] Walt Scacchi and Pei wei Mi. Process life cycle engineering: Approach and support environment. *Intelligent Systems in. Accounting, Finance, and Management*, 6:83–107, 1997.

[23] Amit Sheth. From contemporary workflow process automation to adaptive and dynamic work activity coordination and collaboration. In *Proceedings WFSE '97*, 1997.

[24] Amit Sheth, Dimitrios Georgakopoulos, Stef Joosten, Marek Rusinkiewicz, Walt Scacchi, Jack Wiledon, and Alexander Wolf. Report from the NSF workshop on workflow and process automation in information systems. May 1996.

[25] P. David Stotts. sigmaTrellis: Process models as multi-reader collaborative hyperdocuments. In *Proceedings of the Ninth International Software Process Workshop, Airlie, Virginia*, October 1994.

[26] Stanley Sutton and Leon Osterweil. The design of a next-generation process language. In M. Jazayeri and H. Schaure, editors, *Proceedings of ESEC/FSE '97*, 1997.

[27] Richard Taylor, Frank Belz, Lori Clarke, Leon Osterweil, Richard Selby, Jack Wileden, Alexander Wolf, and Mical Young. Foundations for the arcadia environment architecture. In *SIGSOFT '88: Proceedings of the Third ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, Boston.* SIGSOFT/SIGPLAN, November 1988. Available in SIGSOFT Software Engineering Notes 13(5) and SIGPLAN Notices 24(2).

[28] W.M.P. van der Aalst. Interorganizational workflows: An approach based on message sequence charts and petri nets. *Systems Analysis - Modelling - Simulation*, 34(3):335–367, 1999.

[29] A. Wise. Little-JIL 1.0 language report. Technical report, Laboratory for Advanced Software Engineering Research, University of Massachusetts, Amherst, April 1998.

[30] Alexander Wolf and David Rosenblum. Process-centered environments (only) support environment-centered processes. In *Proceedings 8th International Software Process Workshop*, Wadern, Germany, March 1993.