Journal of Network and Computer Applications (2001) **24**, 39–61
doi:10.1006/jnca.2000.0122, available online at http://www.idealibrary.com on IDEAL®

Specifying process-oriented hypertext for organizational computing

John Noll*[*] and Walt Scacchi

[*]*Dept. of Computer Engineering, Santa Clara University, 500 El Camino Real, Santa Clara, CA U.S.A. 95053 E-mail:* jnoll@cse.scu.edu

[†]*University of California, Irvine, U.S.A. E-mail:* wscacchi@ics.uci.edu

(Received 9 December 1999; accepted 30 June 2000)

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, the 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.

We have developed a process scripting language called *PML* that 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.
© 2001 Academic Press

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 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 [1]. 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 the relationships among organizational processes, tasks, documents, applications, development tools, users and other resources as the 'web of computing' [2]. 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 necessary to complete the task.

For example, a grants officer in a research funding agency may receive a request for funds dispersement 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 [3].

Hypertext is the appropriate conceptual model for an organizational web. Hypertext linking allows the evolutionary construction of relationships among the wide diversity of information artefacts used by an organization. Our own research has shown that the hypertext data organization model is an effective integration approach to diverse information models [4]. In addition, the browsing model of interaction is now familiar to a vast population of computer users. Finally, the notion of guided tours [5] 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 a *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.

Process-oriented hypertexts extend conventional hypertext environments by providing mechanisms that support process *guidance*: explicit process control flow constructs that direct a process user to browse hypertext-based information objects and to perform user role-specific tasks that evolve the state of the visited objects; *control*: ensuring that process steps are performed in the correct order, and that each step produces the desired result; and *monitoring*: process measurement, logging of process events, potential replay of process steps, etc. useful for planning and improvement [6].

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 program 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 to analyse 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 present an alternate approach using an extensible process scripting language called *PML*, and its associated execution mechanism (called *PEOS*), to describe organizational webs in a form that enables the specification, validation, analysis, and execution of process-oriented hypertexts.

Using PML, a process is represented *explicitly* as a script specifying process tasks, the order in which they should be performed, and links to required resources, related documents, and tools. A single PML specification can be translated into several representations, including a process-oriented hypertext, 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 also provides extensibility by allowing executable code fragments or tool invocations to be included in task descriptions; users automatically execute these descriptions or tool invocations by browsing the resulting process-oriented hypertext.

PML specifications can be developed and maintained by process users or process owners, either as program text or through visual modelling 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,

* See, for example, www.seagate.com/support/npf/disc_ata/flowchart_ata.html

since they can see ahead to where their process flows are going, as well as where they came from.

In the next section, we survey related work to place our approach in context. Then, in Section 3 we present the PML language features. In Section 4 we discuss enactment of processes specified using PML. Finally, in Section 5 we conclude with a discussion of our experience using PML to model and prototype processes of the US Office of Naval Research.

2. Related work

2.1 *Process and Hypertext*

While often considered a mechanism for modelling data to be consumed by applications, many researchers have recognized the potential of hypertext as a formalism for representing computation in general, and organizational processes in particular.

Early work in this vein extended the conventional notion of hypertext with executable nodes. For example, KMS provided a built-in scripting language for creating executable nodes, and could also pass nodes to the operating system for compilation and execution [7]. PML supports a generalized version of this concept through its 'script' field.

Proxhy extended this latter mechanism by associating applications with node types, and automatically launching the application when a user visited a node of the associated type [8]. HOSS purports to model 'behaviour' explicitly as a concept in hypertext [9].

Trellis represents a different approach by modelling computation in the structure of the hypertext graph, specifically as a petri-net [10]. An organizational process can be modelled as a petri-net in Trellis, providing guidance through directed browsing based on the propagation of tokens through the petri-net [11]. Trellis also incorporated a language for describing the petri-net hypertext as a program.

CHIPS [12] most closely embodies the notions of organizational webs and process-oriented hypertext. CHIPS attempts to support enactment of both defined and emergent organizational processes; the former are modelled by hypertext graphs that capture process flow, while the latter are supported by 'activity spaces', a hypertext work area supporting group communication [13]. An interesting feature of CHIPS is that it enables process performers to modify the process being enacted by adding or modifying the nodes and links that represent the process.

While the ultimate goal, and realization, of these techniques is similar, our approach differs in two significant ways. First, in each case there is an implicit assumption that the process models and the product structure are developed concurrently, before any products are created. For example, CHIPS instantiates a new activity space containing new information (product) nodes when a new process instance is created. By contrast, an explicit goal of PML is to model and support an organization's current processes, as applied to existing legacy products; and, to support new processes applied to existing products. PML's execution mechanism is designed to co-exist with, but be separate from, an organization's existing intranet or hypertext environment.

Second, the PML approach follows a process engineering model based on Osterweil's concept of *process programming*, an attempt to apply techniques of software development to process modelling and enactment [14]. This notion proposes that processes be described using languages similar to conventional programming languages, enhanced with specific constructs for modelling software processes. As such, PML is similar to Trellis's scripting approach, and contrasts with the model employed by KMS or CHIPS, which is based on hypermedia authoring.

2.2 *Process programming*

The process programming concept has yielded a variety of languages for describing software processes, including APPL/A [15], JIL [16], 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 modelling 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 [17–19], Petri-Nets [11, 20], or process programming languages [21]. 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 [22]. Nonetheless, process-oriented software development environments have been criticized for the limited focus on processes that can be carried out using such an environment [23].

2.3 *Workflow*

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 [1].

The goal of contemporary workflow research is to expand the scope of workflow automation to ‘adaptive and dynamic work activity coordination and collaboration’ [24], highly distributed ‘virtual corporations’, and processes involving participants outside of a given organization [25]. However, common process representations in workflow software have been limited to directed graphs [1], Petri-nets [26], rules [27], and agents [28], rather than utilizing hypertext structures as suggested here.

The widespread deployment of web browsers has motivated many workflow vendors to provide HTTP/HTML support for their workflow products [25]. This is typically accomplished by providing an HTTP gateway to some existing workflow or process engine. The objective is primarily to exploit the ubiquitous browser infrastructure, rather than take advantage of the hypertext-like information space to which browsers provide access. A few research projects, however, seek to embed process support into the browser itself, for example via Java ‘agents’ [28].

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 [29], and a myriad of relationships that link them. This can lead to powerful modelling 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 artefacts are part of the environment.

By contrast, PML separates the organization and its members, artefacts, 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.

3. 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.

* The name ‘PML’ was originally an acronym for ‘Process Markup Language’, a name that reflected our initial attempt to develop an SGML-like notation for specifying processes. Since then, the syntax has evolved considerably to its present from. Thus a more appropriate name might be ‘PSL’ (Process Scripting Language), but we have kept the original name for historical reasons.

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 [3, 30]. The analysis was conducted using process analysis techniques developed at the University of Southern California’s ATRIUM Laboratory [31]. Our methodology comprised 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 the requirement for a modelling formalism that could be translated into executable prototypes, while at the same time be flexible enough to enable incremental process modelling. We also needed a language that would allow the modeller to generate new models and revise existing models quickly, to enable rapid refinement during ‘live’ validation sessions with actual process performers. Also, 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. Finally, we wanted a language that was concise and easy to comprehend.

As can be seen from Fig. 1, the resulting language is straightforward and compact, comprising task descriptions and a minimum set of control flow constructs (sequence, branch, and iteration). In the following sections, we describe the specific features of PML and give examples of their application.

3.1 *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.

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

```
⟨process⟩ →  process ⟨identifier⟩ {⟨primlist ⟩}
⟨primlist⟩ → ⟨primitive⟩ ⟨primlist⟩ | ε
⟨primitive⟩ → ⟨sequence⟩ | ⟨iteration⟩ | ⟨branch⟩ | ⟨action⟩
⟨sequence⟩ →  sequence ⟨identifier⟩ {⟨primlist⟩}
⟨iteration⟩ →  iteration ⟨identifier⟩ {primlist}
⟨branch⟩ →  branch ⟨identifier⟩ {⟨primlist⟩}
⟨action⟩ →  action ⟨identifier⟩ ⟨type⟩ {⟨speclist⟩}
⟨type⟩ →  manual |  executable
⟨speclist⟩ → ⟨spec⟩ ⟨speclist⟩ | ε
⟨spec⟩ → ⟨spectype⟩ {⟨selector⟩}
⟨spectype⟩ →  requires |  provides |  tool |  agent |  script
⟨selector⟩ → ⟨url⟩ | ⟨filename⟩ | ⟨string⟩ | ⟨predicate⟩
⟨predicate⟩ → ⟨identifier⟩ | ⟨query⟩
⟨query⟩ → ⟨term⟩ | ⟨term⟩ ⟨bool⟩ ⟨query⟩ | ε
⟨term⟩ → ⟨identifier⟩ .⟨identifier⟩ ⟨op⟩ ⟨value⟩
⟨bool⟩ →  || |  &&
⟨op⟩ →  == |  != |  < |  > |  <= |  <=
⟨value⟩ → ⟨number⟩ | ⟨string⟩
```

Figure 1. The PML grammar.

- *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* The agent field specifies the role an agent performing the action should have.
- *Script* One of the chief advantages of enactable process descriptions is their usefulness for providing guidance to human agents performing the process. The script field can be used to provide such guidance, as well as links to supporting documentation. It’s contents can be one of the following:
  1. A narrative description of the action, in plain text or HTML markup.
  2. HTML markup specifying a form to be completed as part of the task. The form’s fields are bound to variables in the process instance’s execution context (see Section 4.3), so they can be used in resource predicates as shown.

     Figure 2, which shows a PML description that models a proposal submission process, depicts this use of the script field. The resulting display is shown in Fig. 4; note that the script field has been rendered as an HTML form, as specified.

```
/*
* Process for submitting proposals to ONR.
*/
process proposal_submit {
  action submit_proposal {
    agent { PrincipalInvestigator }
    requires { proposal }
    provides { proposal.contents == file }
    script {
        "<p>Submit proposal contents.
        <p>BAA to which this proposal responds:
        <input name='baa' type='string' size=16>
        <br>Proposal title: <input name='title' type='string' size=50>
        <br>Submitting Institution: <input name='institution'
            type='string' size=25>
        <br>Principle Investigator: <input name='PI' type='string'
            size=20>
        Email: <input name='PIemail' type='string' size=20>
        <br>Contact: <input name='contact' type='string' size=20>
        Email: <input name='contactEmail' type='string' size=12>
        <br>Proposal contents file: <INPUT NAME='file' TYPE='file'>"
    }
  }
  action submit_budget {
    agent { PrincipalInvestigator }
    requires { proposal }
    provides { proposal.budget == file }
    script {
        "<p>Submit budget.
        <br>Proposal title: <input name='title' type='string' size=50>
        <br>Budget file: <INPUT NAME='file' TYPE='file'>
        <br>Email address of contact: <input name='user_id'
            type='string'>"
    }
  }
  action submit_certs {
    agent { PrincipalInvestigator }
    requires { proposal }
    provides { proposal.certs == file && proposal.certifier == user_id
}
    script {
        "<p>Submit electronically signed certifications.
        <br>File containing signed certifications: <INPUT NAME='file'
            TYPE='file'>
        <p>User ID of signature: <input name='user_id' type='string'>"
    }
  }
}
```

Figure 2. A proposal submission process described in PML.

3. For executable actions, the script field contains executable code written in a scripting language (such as TCL, Perl, `sh`, etc.) to be executed by the operating system's shell.

- *Tool* Tools used to perform the action are specified as a string to be executed as a command by hypertext browsers that can run external programs.
- *Requires, Provides* 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, variable, or selection predicate) required to perform the action, and the products produced by the action. Resources are discussed in further detail in Section 3.2

Of these, only the name is required; the type defaults to 'manual', and the other fields are optional. 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 complete process-oriented hypertexts.

3.1.2 *Control flow*. A process specification describes the actions (tasks) that should be performed to carry out the process, and the sequence in which those actions 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 actions, or sequences of actions 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*:

- *Sequence* specifies a set of actions that should be performed one at a time in a specific order. In general, sequences are useful for modelling the hierarchical structure of processes: sub-processes are represented by sequences, primitive actions by actions.
- *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 'SendRequest' example in Fig. 3 illustrates how PML models decisions using selections: a request should be sent *either* to the Program Officer if the grant is an ONR grant, or to the Grant Officer if the grant is from another agency, but administered by ONR.
- *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.

  In Fig. 3, a branch ('PrepareRequest') is used to specify that the request should be sent as both an email message and a written letter. Order is not important, and the actions may be done concurrently.

```
/* No Cost Extension Request. */
sequence HandleNCERequest {
   action ReviewForCompleteness {
       requires { GrantRecord }
       agent { GrantAdministrator }
   }
   branch PrepareRequest {
       action PrepareEmailRequest {
           requires { GrantRecord }
           provides { DocumentRequest }
           agent { GrantAdministrator }
       }
       action PrepareWrittenRequest {
           requires { GrantRecord }
           provides { DocumentRequest }
           agent { GrantAdministrator }
      }
   }
   select SendRequest {
       action SendToPOifONR {
           requires { DocumentRequest }
           agent { GrantAdministrator }
       }
       action SendToGOifNotONR {
           requires { DocumentRequest }
           agent { GrantAdministrator }
      }
   }
   iteration FollowUp {
       action FollowUpNCERequest {
           requires { DocumentRequest }
           agent { GrantAdministrator }
       }
   }
}
```

Figure 3. A PML process fragment modeling ‘No Cost Extension Request’.

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.

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

- *Iteration* Iteration occurs over sequences of actions; the agent enacting the process decides whether to enter the loop by selecting the first action in the loop, or skip/exit the loop by selecting the first action following it.
  In Fig. 3, the 'FollowUp' task is performed until a response is received from the Program or Grant Officer. This allows the process performer to submit as many follow-up requests as necessary; note that the decision to exit the loop is made by the process performer when the desired response is received.

Using this small set of control flow primitives, it is possible to capture a wide variety of situations involving decisions and concurrency.

3.2 *Resources*

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; thus, they define pre- and post-conditions of actions in a process. A name in a resource field defines a variable that is bound to an instance of a resource at run time (either when the process instance is created, or as the result of a query); 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 4).

In addition to resource names, 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 }
requires { resourceName.attributeName op value &&/||
   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.

3.3 *Compilation*

The PML compiler translates PML process specifications into the components of process-oriented hypertext.

The compiler generates an action node template for each action specification in a PML model. When a new process instance is created, these templates are used to create action nodes for the new process instance. 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 an interpreter for execution.
- Static links to supporting documentation and tools.
- Dynamic 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.

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 model.

3.3.1 *Resource binding*. The links between actions, 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 execution mechanism. The compiler generates queries to the hypertext repository that are executed during enactment to determine whether resources are available to an action. The queries are derived from the expressions in an action’s *requires* field; the compiler generates code for these expressions, to:

1. Query the repository for objects of the specified type whose state matches any predicates.
2. If this is the first reference to the resource name, bind the resource name to one of the objects matched by the query. Subsequent references to that resource name will refer to the bound object, which also becomes the destination of dynamic ‘required resource’ links from the action node.
3. If no objects match the query, the action transitions to the ‘blocked’ state; otherwise, it transitions to the ‘ready’ state.

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 execution mechanism updates these overlays as the processes are enacted.

Returning to the ‘No Cost Extension’ example, this means that the same PML process description can be used to guide a grant administrator through the steps required to manage multiple requests; the progress of each request through the process is tracked by a separate process instance, with its own set of control flow and resource links.

4. 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.

These are depicted in Fig. 5.

4.1 *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 [4].

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, however, remain the responsibility of the each 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 objects.

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

Table 1. *Virtual Repository Interface*

Operation	Arguments	Description
Create	*attribute=value* . . .	Creates a new object with the specified attributes.
Update	*object attribute=value* . . .	Updates the specified attributes of *object*.
	These are used by the PEOS Server and PML 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.

4.2 *PEOS server*

The PEOS server handles process requests 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 PML 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.

4.3 *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’s states.

Changes in action state are caused by three events (see Fig. 6):

Table 2. *PEOS protocol*

Request message	Arguments	Result or reply
Login	*user id, password*	Authenticate the agent specified by *user id*, and start a session for that agent; subsequent operations during the session are bound to the agent.
List	none	List the compiled PML models available for instantiation.
Create	*model variable=value…*	Create a new instance of *model*, with the specified variables bound to the specified values.
Available	none	Return 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*	Abort the specified action.
Logout	none	End the current session.

1. Completion of a predecessor task (indicated by a ‘**Done**’ request). This event causes the next action (or actions, in the case of a branch or selection) in the process sequence to transition to the ‘ready’ state, if the next action’s required resources are available. If the next action’s resources are not available, it transitions to the ‘blocked’ state.
2. Explicit notification from process performers about action state (via **‘Select’, ‘Suspend’**, or **‘Abort’** requests). 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.

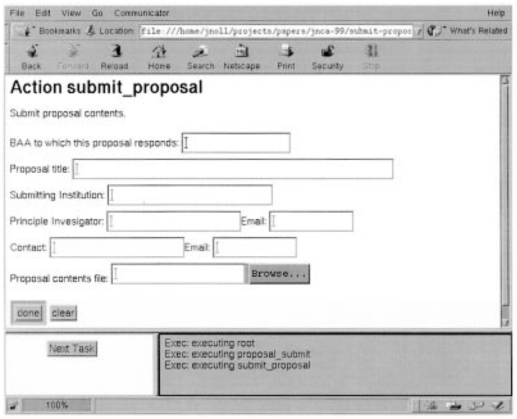
Selecting an action causes it to become ‘active’; if the action is part of a ‘**Select**’ construct, other actions in the selection are no longer available, and thus transition to the ‘none’ state. The ‘active’ state indicates that a user is currently performing the action; some actions may require significant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

Consequently, a user can suspend a task explicitly by sending a ‘**Suspend**’ request, 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, 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 ‘**Done**’ requests, typically by clicking a button attached to an action node (see Fig. 4). These events indicate that an update to the process state is required. Computing new state in response to **Done** requests is the responsibility of the PML Virtual Machine. The PML Virtual Machine takes the current process state and the **Done** request as input, and computes the new process state according to the process flow of 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 (via the Assert operation shown in Table 1) on the repository. The virtual machine executes this assertion when it receives a **Done** event for an action; if the assertion fails, the action did not accomplish what it was


File Edit View Go Communicator Help
Bookmarks Location: file:///home/jnoll/projects/papers/jnca-99/submit-propos What's Related
Back Forward Reload Home Search Netscape Print Security Stop
Browse...
done clear
Next Task
100%

Action submit_proposal

Submit proposal contents.

BAA to which this proposal responds:

Proposal title:

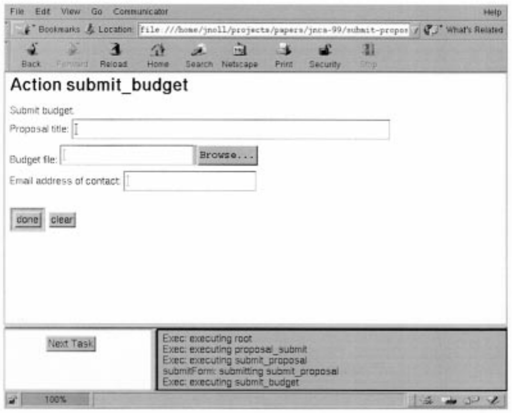
Submitting Institution:

Principle Invesigator: Email:

Contact: Email:

Proposal contents file:

```
Exec: executing root
Exec: executing proposal_submit
Exec: executing submit_proposal
```


File Edit View Go Communicator Help
Bookmarks Location: file:///home/jnoll/projects/papers/jnca-99/submit-propos What's Related
Back Forward Reload Home Search Netscape Print Security Stop
Browse...
done clear
Next Task
100%

Action submit_budget

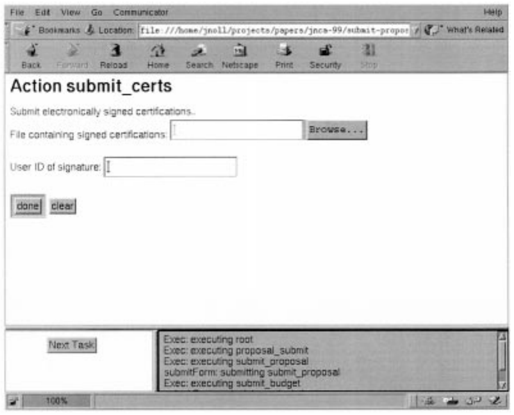
Submit budget.

Proposal title:

Budget file:

Email address of contact:

```
Exec: executing root
Exec: executing proposal_submit
Exec: executing submit_proposal
submitForm: submitting submit_proposal
Exec: executing submit_budget
```


File Edit View Go Communicator Help
Bookmarks Location: file:///home/jnoll/projects/papers/jnca-99/submit-propos What's Related
Back Forward Reload Home Search Netscape Print Security Stop
Browse...
done clear
Next Task
100%

Action submit_certs

Submit electronically signed certifications.

File containing signed certifications:

User ID of signature:

```
Exec: executing root
Exec: executing proposal_submit
Exec: executing submit_proposal
submitForm: submitting submit_proposal
Exec: executing submit_budget
```

Figure 4. Enacting the proposal submission process of Figure 2.

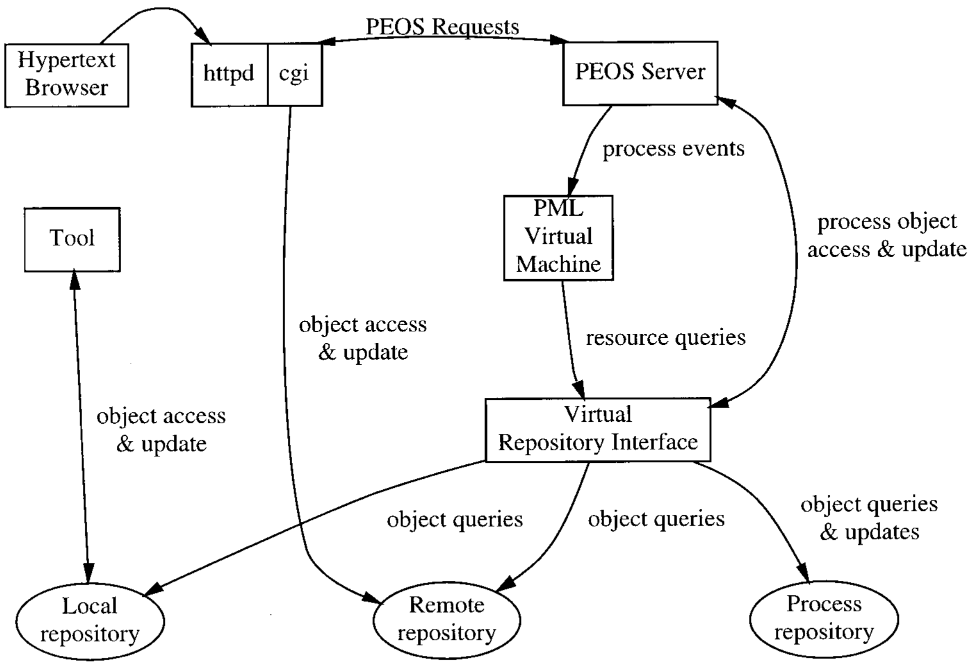


Figure 5. PML enactment architecture.

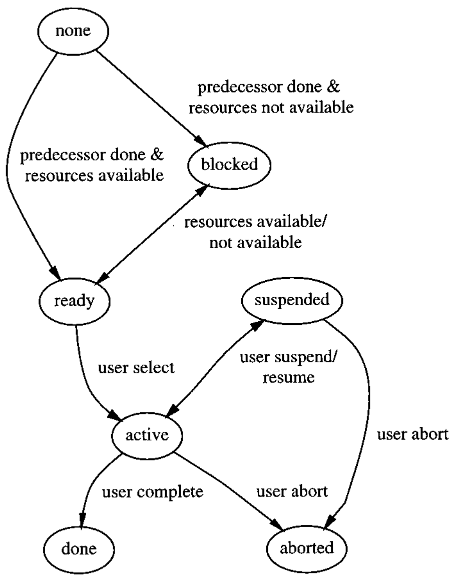

none
predecessor done & resources not available
predecessor done & resources available
blocked
resources available/ not available
ready
suspended
user select
user suspend/ resume
active
user abort
user complete
user abort
done
aborted

Figure 6. Action states.

supposed to do, and the entire 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.

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 the US Navy’s Office of Naval Research (ONR) to validate our PML-based solution. We analysed four phases of the ONR grants administration process: pre-award, grant award, grant administration, and grant close-out. The resulting PML models totalled 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 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

* PEOS’s response to aborted actions is admittedly weak. What to do when an action is aborted, either by the user or system, is a difficult problem and active research area (see, for example, [32] or [33]).

$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 [30].

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.

PEOS 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 Fig. 2) 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 (Fig. 4), we were able to provide a convincing demonstration of how such a process might look to users.

Our experience of 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 4 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: for example, 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 4.3.

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 and 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.

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

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

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.

We would like to thank the reviewers for their excellent comments and suggestions, which have greatly improved 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 Acquisition University, or the US Navy, of any statements or findings presented herein.

References

1. Workflow Management Coalition 1995. The workflow reference model. *Workflow Management Coalition*, Tech. Report No. TC00-1003.
2. R. Kling & W. Scacchi 1982. The web of computing: Computer technology as social organization. *Advances in Computers* **21**, 1–90.
3. W. Scacchi & J. Noll 1997. Process-driven intranets: Life-cycle support for process engineering. *IEEE Internet Computing.* **1**(5), 42–49.
4. J. Noll & W. Scacchi 1991. Integrating diverse information repositories: A distributed hypertext approach. *IEEE Computer* **24**(12), 38–45.
5. P. T. Zellweger 1989. Scripted documents: A hypermedia path mechanis. In *Proceedings of the Second ACM Conference on Hypertext, Pittsburgh, PA USA, 1989.* The Association for Computing Machinery, 1–14.
6. J. Noll & W. Scacchi 1999. Supporting software development in virtual enterprises. *Journal of Digital Information* **1**(4).
7. J. L. Schnase & J. J. Leggett 1989. Computational hypertext in biological modeling. In *Proceeding of the Second ACMM Conference on Hypertext, Pittsburgh, PA USA. 1989.* The Association for Computing Machinery, 181–197.
8. C. J. Kacmar & J. J. Leggett 1991. PROXHY: A process-oriented extensible hypertext architecture. *ACM Transactions on Information Systems* **9**(4), 399–420.
9. P. J. Nurbberg, J. J. Leggett, E. R. Schneider & J. L. Schnase 1996. Hypermedia operating systems: A new paradigm for computing. In *Proceeding of the Seventh ACM Conference on Hypertext, Bethesda, MD USA, 1996.* The Association for Computing Machinery, 194–202.
10. R. Furuta & D. P. Stotts 1989. Programmable browsing semantics in trellis. In *Proceedings of the Second ACM Conference on Hypertext, Pittsburgh, PA USA, 1989.* The Association for Computing Machinery, 27–42.
11. D. P. Stotts 1994. Sigma Trellis: Process models as multi-reader collaborative hyperdocuments. In *Proceedings of the Ninth International Software Process Workshop, Airlie, VA USA, 1994,* 85–89.

12. J. M. Haake & W. Wang 1997. Flexible support for business processes: Extending cooperative hypermedia with process support. In *Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work, Phoenix, AZ USA, 1997*. The Association for Computing Machinery, 341–350.
13. W. Wang & J. Haake 1997. Supporting user-defined activity spaces. In *Proceedings of the Eight ACM Conference of Hypertext, Southampton, United Kingdom, 1997*. The Association for Computing Machinery, 112–123.
14. L. Osterweil 1987. Software processes are software too. In *Proceedings of the 9th International conference on Software Engineering, Monterey, CA USA, 1987*. The Association for Computing Machinery, 2–13.
15. R. Taylor, F. Belz, L. Clarke, L. Osterweil, R. Selby, J. Wileden, A. Wolf & M. Young 1988. 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, 1988*. The Association for Computing Machinery, 1–13.
16. A. Wise 1998. Little-JIL 1.0 language report. Laboratory for Advanced Software Engineering Research, University of massachusetts, Amherst, Tech. Report.
17. N. Barghouti & G. Kaiser 1992. Scaling up rule-based development environments. *Int. J. Softw. Eng. Know.* **2**(1), 59–78.
18. I. ben-Shaul & G. Kaiser 1994. 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, 1994*. IEEE Computer Society, 179–188.
19. P. Mi & W. Scacchi 1996. A meta-model for formulating knowledge-based models of software development. *Decision Support Systems* **17**(3), 313–330.
20. F. Maurer, G. Succi, H. Holz, B. Kotting, S. Goldman & B. Dellen 1999. Software process support over the Internet. In *Proceedings of the 21st International Conference on Software Engineering, Los Angeles, 1999*. The Association for Computing Machinery, 642–645.
21. S. Abiteboul, M. Adiba, J. Arlow, P. Armenise, S. Bandinelli, L. Baresi, P. Breche, F. Buddrus, C. Collet, P. Corte, T. Coupaye, C. Delobel, W. Emmerich, G. Ferran, F. Ferrandina, A. Fuggetta, C. Ghezzi, S. Lautemann, L. Lavazza, J. Madec, M. Phoenix, S. Sachweh, W. Schfer, C. S. dos Santos, G. Tigg & R. Zicari 1994. The GOODSTEP project: General object-oriented database for software engineering processes. In *Proceedings 1st Asian Pacifc Software Engineering Conference, Tokyo, 1994*.
22. P. Garg & M. Jazayeri 1995. Process-centered software engineering environments: A grand tour. *Technical University of Vienna*, Tech. Report No. TUV-1841-95-02.
23. A. Wolf & D. rosenblum 1993. Process-centered environments (only) support environment-centered processes. In *Proceedings 8th International Software Process Workshop, Wadern, Germany, 1993*. International Software Process Association (ISPA), 148–149.
24. A. Sheth 1997. From contemporary workflow process automation to adaptive and dynamic work activity coordination and collaboration. In *Proceedings WFSE '97, 1997*.
25. Workflow Management Coalition 1998. Workflow and internet: Catalysts for radical change. *Workflow Management Coalition*, Tech. Report.
26. W. van der Aalst 1999. Interorganizational workflows: An approach basedon message sequence charts and petri nets. *Systems Analysis—Modelling—Simulation* **34**(3), 335–367.
27. W. Lee, G. Kaiser, P. Clayton & E. Sherman 1996. OzCare: A workflow automation system for care plans. *Jounral of the American Medical Informatics Association: 1996 AMIA Annual Fall Symposium, Sympossium Supplement*, 577–581.
28. T. Cai, P. Gloor & S. Nog 1996. DartFlow: A workflow management system on the web using transportable agent. *Department of Computer Science, Dartmouth College*, Tech. Report No. PCS-TR96-283.
29. S. Sutton & L. Osterweil 1997. The design of a next-generation process language In *Proceedings of ESEC/FSE 197*, M. Jazayeri & H. Schaure, eds., 1997.
30. W. Scacchi (to appear). Reengineering contracted service procurement for Internet-based electronic commerce: A case study. *Journal of Information Technology and Management*.
31. W. Scacchi & P. Mi 1997. Process life cycle engineering: Approach and support environment. *Intelligent Systems in Accounting, Finance, and Management* **6**, 83–107.

32. F. Casati, S. Ceri, S. paraboschi & G. Pozzi 1999. Specification and implementation of exceptions in workflow management systems. *ACM Transactions on Database Systems* **24**(3), 405–451.
33. B. Kiepuszewski, R. Muhlberger & M. E. Orlowsi 1998. FlowBack: Providing backward recovery for workflow management systems. In *Proceedings of SIGMOD '98, Seattle, WA USA, 1998*. Association for Computing Machinery, 555–557.

John Noll recently completed his PhD in Computer Science at the University of Southern California, where he was a Research Associate with the ATRIUM laboratory at USC's Marshall School of Business. He has also taught USC, UCLA, and the University of Colorado (Denver), and held positions at Perceptronics, Inc., Hewlett-Packard Laboratories, and Network Appliance, Inc. Dr. Noll is currently Assistant Professor in the Computer Engineering department at Santa Clara University. His research interest include software process, workflow, and computer supported cooperative work.

Walt Scacchi is Senior Research Scientist and Research Faculty member at the Institute for Software Research at the University of California at Irvine. He received a PhD in Information and Computer Science at University of California, Irvine in 1981. From 1981 to 1998 he was faculty member at the University of Southern California. His interests include knowledge-based systems for modeling and simulating organizational processes, distributed hypertext information systems, open source software production, software acquisition, and organizational analysis of system development projects. Dr. Scacchi is a member of ACM, IEEE, AAAI, and the Software Process Association (SPA). He is an active researcher with more than 90 research publications.