ELSEVIER

The Journal of Systems and Software 59 (2001) 343–354

The Journal of Systems and Software

www.elsevier.com/locate/jss

Modeling and simulating software acquisition process architectures

S. James Choi [a,1], Walt Scacchi [b,*]

[a] *Computer Science Department, California State University, Fullerton, CA 92834-6870, USA*
[b] *Institute for Software Research, University of California, Irvine, CA 92697-3425, USA*

Received 12 July 2000; received in revised form 1 December 2000; accepted 22 March 2001

Abstract

In this paper, we describe our efforts to support the modeling and simulation of processes associated with software system acquisition activities. Software acquisition is generally a multi-organization endeavor concerned with the funding, management, engineering, system integration, deployment and long-term support of large software systems. We first describe our approach supporting the modeling and simulation of software acquisition processes using a software process architecture (SPA). We then introduce how we support the distribution, concurrent execution and interoperation of multiple software process simulations using the high-level architecture (HLA) and run-time infrastructure (RTI) to address the complexity of software acquisition process architectures. To illustrate this, we provide examples from the design and prototyping of a Web-based environment that supports the modeling and simulation of acquisition process architectures. This environment thus serves as a new kind of software process test-bed that can demonstrate and support experiments incorporating multiple software process simulation systems that interoperate in a distributed and concurrent manner across a network. © 2001 Elsevier Science Inc. All rights reserved.

1. Introduction

Software acquisition includes the processes typically associated with the software engineering life cycle. However, acquisition also includes processes that fund, manage, integrate, deploy and support software systems before, during, and after their software engineering life cycle. The need to address processes for systems and software engineering, inter-organization coordination and overall project management together is what establishes our baseline of interest in modeling and simulating software acquisition processes.

Software acquisition processes are often large-scale, involve multiple enterprises and stakeholders, and are expensive, long-lived and frequently plagued with process coordination problems (Boehm and Scacchi, 1996; GAO, 1997; SA-CMM, 2000). Large-scale characterizes the fact that tens-to-hundreds of distinct processes for engineering, project management, and customer/government oversight must be articulated and coordinated. The participation of many enterprises reflects at the topmost level the division of effort between customer, contractor and acquisition program office enterprises. Contractors in turn often organize teams of sub-contractors, sometimes numbering into the thousands, into a virtual enterprise that collectively engineer and deploy the system being acquired. Similarly, the contractor team may involve hundreds to thousands of software developers who will produce and deliver millions of source lines of code. Consequently, program acquisitions for military systems or public infrastructure systems (e.g., air traffic control) cost billions of dollars. Finally, long duration reflects the fact that some programmatic acquisitions for large systems span 10–20 years from initiation through deployment and post-deployment support. Thus, modest improvements in the efficiency or effectiveness of acquisition processes or process configuration, can realize savings in millions of dollars, many person-years (or person-decades) of engineering effort, and improve the quality of the delivered systems (ARO, 1999).

Given the complexity of large acquisition efforts, we choose to examine software acquisition processes from an architectural perspective. In this regard, our position is that modeling and simulating software acquisition processes requires some kind of *factoring* to manage their complexity. Factoring is needed to realize both a *separation of concerns* through a factorable architecture of interconnected and interrelated processes, as well as

* Corresponding author. Tel.: +1-949-824-4130; +1-949-824-1715.
E-mail addresses: sjchoi@ecs.fullerton.edu (S. James Choi), wscacchi@ics.uci.edu (W. Scacchi).
[1] Tel.: +1-714-278-7257; fax: +1-714-278-7168.

facilitating, guiding or managing the *scalable composition of component processes* that together constitute software acquisition. Factoring also enables the *partitioning*, *distribution* and *concurrency* of process activities spread across many participating enterprises. Subsequently, in order to be able to construct and simulate factorable models of software acquisition processes, we require *architectures* that can separate and configure a distributed web of software acquisition processes. An architectural perspective also enables us to explore the potential for formulating families of common software processes into a product line (Bergey et al., 1999), or better said, *process line*. Therefore, we will refer to this structured web as *a process architecture for software acquisition*.

The focus of our research effort here is to describe our approach to modeling and simulating architectures for software acquisition processes (cf. Boehm and Scacchi, 1996; Scacchi and Boehm, 1998; Schooff et al., 1997). We describe our approach supporting the simulation of software acquisition processes within a process architecture. Along the way, we introduce how we employ the high-level architecture (HLA) and run- time infrastructure (RTI) (Kuhl et al., 1999) to support the distribution, concurrent execution and interoperation of multiple software process simulations to address the complexity of software acquisition process architectures. Such an investigation can help us determine whether the HLA can serve as a wide-area or global test-bed that could enable the interoperation of multiple software process simulations that have been independently developed by loosely coupled community of software process researchers or practitioners located around the world. Finally, we introduce the design and prototyping of a Web-based environment that supports the modeling and simulation of acquisition process architectures, as well as a variety of analyses and process prototyping capabilities.

2. Approaches to modeling software process architectures

We describe four concepts in this section. The first is a language we developed for modeling, prototyping and enacting software and business processes, called PML. Second, we describe how we extend and combine PML with software architectural design constructs to model a software process architecture (SPA). Third, we evaluate the use of PML and the HLA as schemes for modeling an SPA. Last, we describe how an SPA can be integrated into a Web-based environment for modeling software acquisition processes that can be simulated across a distributed run-time infrastructure.

2.1. PML: A language for modeling software processes

Noll and Scacchi (2001) have developed and demonstrated the design of PML and its Web-based run-time environment. PML has been used to model a subset of acquisition processes at the US Office of Naval Research in legacy as-is, redesigned to-be, and transitional here-to-there forms (Noll and Scacchi, 2001). The legacy processes span more than 120 problem-solving tasks as process steps that occur in multiple locations within/between ONR's national and international offices.

PML is a declarative language for modeling and specifying complex processes. It acts as an extensible process markup notation that can be compiled into an executable form to support process prototyping and process enactment across the Web, as well as serving as a person-in-the-loop process simulator (Scacchi, 2000). These capabilities enable multi-user process modeling, analysis, walkthrough, redesign, and enactment across a distributed virtual enterprise of cooperating networked enterprises (Noll and Scacchi, 1999; Scacchi and Noll, 1997).

The design of PML was based on compatibility with the knowledge-based software process meta-model that we had previously developed and used in our process modeling and simulation efforts (Mi and Scacchi, 1990, 1996; Scacchi, 1999). According to this process meta-model, agents (people or programs) perform processes using tools that require resources in order to provide intermediate or final products. Process resource requirements and provision are specified using predicate expressions that serve as pre-conditions or post-conditions on process enactment (Noll and Scacchi, 1999, 2000). Process flow is ordered using sequential, conditional, iterative or concurrent control constructs. Processes are also decomposable into a hierarchy of sub-processes or action steps. Finally, processes associate tools for process enactment that are connected through interpretable scripts that explicitly invoke: (a) client-side routines, forms processing, applets or helper applications, or (b) server-side programs or servlets. Subsequently, the run- time environment for PML was designed to operate in a fully distributed manner without a centralized administrative authority (Noll and Scacchi, 1999, 2000). Thus, PML is based on relatively mature software process modeling techniques combined with constructs geared for deployment and use on the Web. Exhibit 1 displays an excerpt of a low-level acquisition process sequence specified in PML.

```
process Proposal_Submit {
  action submit_proposal {
    agent {PrincipalInvestigator}
    requires {proposal}
    provides {proposal.contents = = file}
    script {"⟨p⟩Submitproposal contents.\
    ⟨p⟩BAA to which this proposal responds: \
    ⟨input name = 'baa' type = 'string' size = 16⟩\
    ⟨p⟩CBD source for this BAA: \
    ⟨input name = 'cbd' type = 'string' size = 50⟩\
    ⟨br⟩Proposal title: ⟨input name = 'title'
    type = 'string' size = 50⟩\
    ⟨br⟩Submitting Institution: ⟨input name =
    'institution' type = 'string' size = 25⟩\
    ⟨br⟩Principal 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⟩Submitbudget.\
    ⟨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⟩Submitelectronically signed
    certifications.\
    ⟨br⟩File containing signed certifications:
    ⟨INPUT NAME = 'file'
    TYPE = 'file'⟩\
    ⟨p⟩User ID of signature: ⟨input name = 'user_id'
    type = 'string'⟩"
    }
  }
}
```

Exhibit 1. An excerpt from an acquisition process specified in PML for submitting a software research or development proposal (Noll and Scacchi, 2001).

2.2. Modeling software process architectures

Researchers at CMU, UC Irvine, USC and elsewhere has been investigating new languages, tools and environments that focus attention on software system architectures (e.g., Medvidovic and Taylor, 2000; Shaw and Garlan, 1996). In our work, we chose to adopt architecture design (AD) techniques and constructs from this related research in order to support the modeling of software process architectures. Furthermore, since our focus on SPAs for acquisition is within the purview of government and military enterprises, we chose to explore the viability of the HLA framework in developing distributed simulations of processes within an acquisition SPA.

ADs are used to specify the components, connectors, interfaces and interconnection configuration of composite software systems. *Components* are objects that encapsulate new/legacy application programs or commercial-of-the-shelf software products. *Connectors* are object types that encapsulate application program interfaces (APIs), middleware, protocols, software buses, or other messaging mechanisms that enable the interoperability and exchange of parameter values, data objects or control signals between components. Both components and connectors have *interfaces* that specify application resources. In some AD languages, interfaces may also specify logical pre-conditions of imported resources, and post-conditions on exported resources. Further information about components and connectors can be specified or automatically extracted to include network host address, author/owner, and timestamp attributes (e.g., for time of most recent modification) (Choi and Scacchi, 1990). Finally, the *configuration* of software system architectures specifies which components are connected to which connectors through compatible interfaces. As a result, configurations can be developed and deployed across a network, as well as analyzed to verify its consistency, completeness, traceability and internal correctness (Choi and Scacchi, 1998).

Historically, process architectures were used to provide a conceptual framework for process management tasks, and to provide mechanisms for specifying software processes with entry ("pre") and exit ("post") conditions for each process component (Radice et al., 1985). These early process architectures lacked an explicit process modeling language or execution environment. In contrast, PML provides notational forms for component processes enacted by agents using tools whose resource requirements and product provisions are specified with explicit pre-/post-conditions. PML tool scripts serve as connectors that interconnect application programs to a process component. PML process components are then interconnected through control flow constructs interpreted by the PML run-time infrastructure (Noll and Scacchi, 2001).

In PML, processes, resource interfaces, resources and connectors (tool scripts) are first-class objects. Process models and SPAs specified in PML can therefore be made more specialized or more generic depending on whether instance-level details are included or not. Generic processes specified in PML enable the construction of common families of software processes that can be tailored for reuse across multiple software acquisition or development projects (cf. Bergey et al., 1999). For example, the US Navy has recently begun the acquisition of a new fleet of battleships that will be researched, developed, built and deployed over the next 15–20 years (DD21, 2001). These ships are software-intensive systems involving dozens of mission-critical application programs constituted from millions of source lines of code (Scacchi and Boehm, 1998). As these ships can be acquired in a serial manner, then the opportunity exists to articulate, refine and continuously improve a family of common software acquisition processes, rather than simply using a rigid standard process or developing a custom process for each ship's system acquisition. Thus the potential for a PML-like process modeling language to serve as the basis for a reusable SPA has real, practical and well-motivated applications.

In our view, an SPA should enable the composition, deployment and configuration management of multi-version processes for software development or use in a manner that scales to distributed and networked enterprises (Noll and Scacchi, 1997, 1999, 2001). SPAs should be able to incorporate or reference other process/application software components distributed across an intranet (cf. Scacchi and Noll, 1997) or the Internet (Noll and Scacchi, 1999). This further implies the potential for process components to be mobile and transportable across the Internet, either as part of their deployment or enactment. This means people who seek to collaborate can send/receive or publish/subscribe to software process models, modify or add additional process components, then choose to keep them for local use, or otherwise forward them to someone else. Furthermore, if heterogeneous process modeling notations are to be deployed and made to interoperate, then an SPA must be able to support this compositional capability. Finally, an SPA must also serve as a basis for simulation–that is, simulation of multiple concurrent and distributed software processes, as is found in the domain of software acquisition. To address these needs, we have been investigating the HLA and its associated RTI as a framework for modeling and simulating process architectures supporting software acquisition.

2.3. Modeling SPAs using the high level architecture

The HLA is a proposed IEEE standard for specifying how to structure a distributed and concurrent simulation system that is composed from multiple simulation systems or simulation components. Interested readers unfamiliar with this standard or the commercial technologies that support it should consult its key references (HLA, 1999; Kuhl et al., 1999). However, in simple terms, HLA serves as an architectural framework for integrating and interoperating object-oriented (OO) and non-OO simulation systems, much like CORBA serves as a framework for integrating OO and non-OO applications. In contrast to the PML, HLA uses application program interfaces (or remote method invocation interfaces) to pass data or control signals across its RTI. Thus HLA can be viewed as an implementation level approach to specifying how multiple simulation systems will be integrated in order to interoperate. HLA is also a military standard required for use in the development of distributed simulation systems for military applications.[2] Thus, HLA is more specialized and more domain-specific than CORBA.

Up to this time, there is no record of the use of HLA to support the organization or composition of multiple interacting software process simulation systems or simulation components. Similarly, we could find no evidence of the use of distributed and concurrent software process simulations, though the simulation of other kinds of parallel and networked systems have been addressed (Fujimoto, 1999). So we have chosen to explore the use of HLA as a basis for structuring the organization of multiple process components that can be described using an SPA, then concurrently simulated as a distributed simulation system. Furthermore, the commercial availability of an RTI that supports HLA-based simulations led us to choose to use it to investigate its feasibility in demonstrating distributed and concurrent simulation of a process architecture for software acquisition. Subsequently, the SPAs we have designed were modeled in PML as a process-oriented hypertext (Noll and Scacchi, 2001). At the same time, we sought to implement simulations of these processes using the HLA framework, so that we can evaluate the capability of the RTI to integrate and interoperate distributed simulation components. The following example characterizes one such SPA modeling and simulation effort.

Let us consider a software acquisition process architecture that involves three types of interacting component processes to model the following kinds of entities: software consumer enterprises; software producer (contractor) enterprises; and a program manager to facilitate interactions between them. Additionally, we include a single process connector to interconnect the consumer to produced processes. Such an architecture might be visually depicted as shown in Fig. 1. Fig. 2 then displays a partial view (many object attributes not shown) an HLA object hierarchy (called the HLA Federate Object Model) for the components and connectors shown in Fig. 1.

[2] The use of the HLA framework is mandated by policy of the US Department of Defense when developing distributed simulation systems (HLA, 1999). However as our effort is research oriented, we were not required to use it. Nonetheless, its standardization and widespread use does make it a candidate for evaluation in our research.

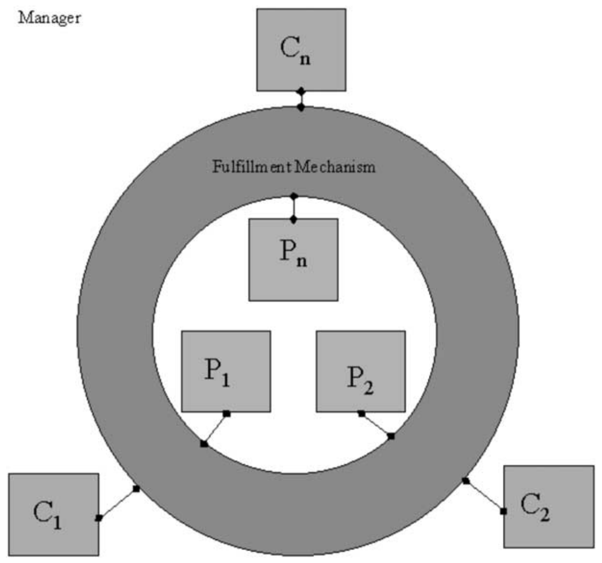


Fig. 1. A software process architecture for acquisition with multiple process components that pass messages using a globally shared process connector (the Fulfillment Mechanism).

When using the HLA to integrate and interoperate multiple simulation systems, developers must satisfy three constraints in order to be compatible with the HLA. These constraints are imposed on any use of HLA as part of conformance to its standards definition and conditions of usage. First, each process simulation component must adhere to a set of ten rules for interoperating with other simulations (Kuhl et al., 1999). Second, each simulation must use an explicit Interface Specification that describes how and in what form it can exchange data and simulation events. Third, each simulation must express data about its public (externally visible) state in form of the HLA Object Model Template (OMT). Each of these requirements bears some further description, though the interested reader should consult the external references for details beyond our exposition here (HLA, 1999; Kuhl et al., 1999).

First, of the 10 rules, five specify constraints on how simulation components interact with one another as a federation. For example, one rule states that when operating as a federation, the representations of all simulation associated object instances shall be in the component simulation, and not in the RTI they use to exchange object instances or values. The other five rules apply to individual simulation components. For example, one such rule states that each simulation component shall be able to update and/or reflect any attributes, as well as send/receive interactions, as specified in their HLA compatible simulation object model.

Second, conforming to the Interface Specification requires use of an HLA RTI that is linked into a simulation to enable interaction with other distributed simulations. The RTI supports six categories of functionality that model and manage how HLA simulations can interact through the global broadcast and synchronization of events that are communicated via shared publish/subscribe registries. The SPA shown in Fig. 1 should be able to conform to this constraint, using its connector as a global mechanism for broadcasting and synchronizing events exchanged across different simulation process components.

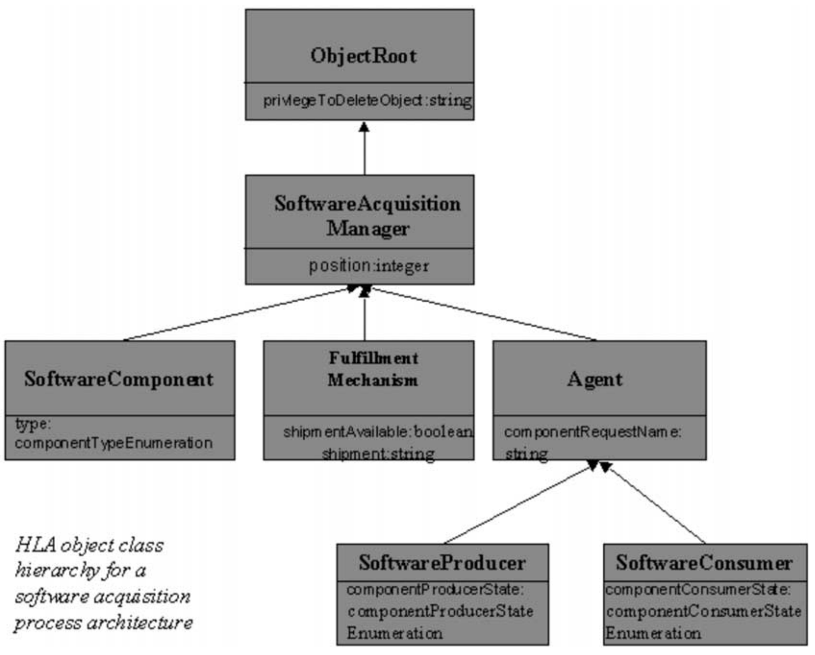

ObjectRoot
privilegeToDeleteObject:string
SoftwareAcquisition Manager
position:integer
SoftwareComponent
type: componentTypeEnumeration
Fulfillment Mechanism
shipmentAvailable:boolean
shipment:string
Agent
componentRequestName: string
SoftwareProducer
componentProducerState: componentProducerState Enumeration
SoftwareConsumer
componentConsumerState: componentConsumerState Enumeration
HLA object class hierarchy for a software acquisition process architecture

Fig. 2. HLA object class hierarchy model for a software acquisition process architecture.

Third, expressing public simulation state data via OMTs suggests a scheme reminiscent of how the extensible markup language, XML, can be used to disseminate the syntax and instances of object data types over the Web. Note that the operational or interpretative semantics of objects is not transmitted, thus the exchange of information requires a prior understanding and agreement as to what the objects and instances mean. This in turn implies that knowledge of objects is distributed among all the simulation components, and thus the potential exists for different simulation components to exchange common objects, but establish their meaning locally. This is in marked contrast to the use of process meta-models, which support process simulation and interoperability through a centralized semantic data model (Mi and Scacchi, 1990, 1996). Maintaining and updating a centralized semantic model is much easier than maintaining distributed simulation object semantics local to each simulation. The effort required to maintain and evolve distributed object semantics does not scale with the incorporation of more simulation components. In fact, it does just the opposite, it generates a combinatorial explosion of possible object meaning inconsistencies and propagated updates.

Thus, we came to the following dilemma in order to use the HLA to model SPAs for distributed simulation: HLA is not a general-purpose architecture for modeling and interoperating application systems or software processes. The three constraints that guide its use impose a specific architectural style that assumes global broadcast and synchronization of events to facilitate interoperability, while sacrificing ease of maintenance and evolution. Nonetheless for prototyping and evaluation purposes, where the semantics of process simulation objects is limited, then the HLA is a plausible candidate to investigate the potential of the distributed and concurrent simulation of interacting software processes, albeit within a pre-determined architectural style.

3. Simulation of software acquisition process architectures

In the previous work, we have demonstrated and comparatively examined different approaches to the simulation of software processes (Scacchi, 1999, 2000). This includes the introduction of person-in-the-loop software process simulators that enable interactive exploration (e.g., browsing, prototyping and walkthrough) of software processes (Scacchi, 2000). Given the approach to modeling and analyzing software process architectures we introduce in our current effort, we need to explain and demonstrate how simulation of software acquisition processes fits into our overall scheme.

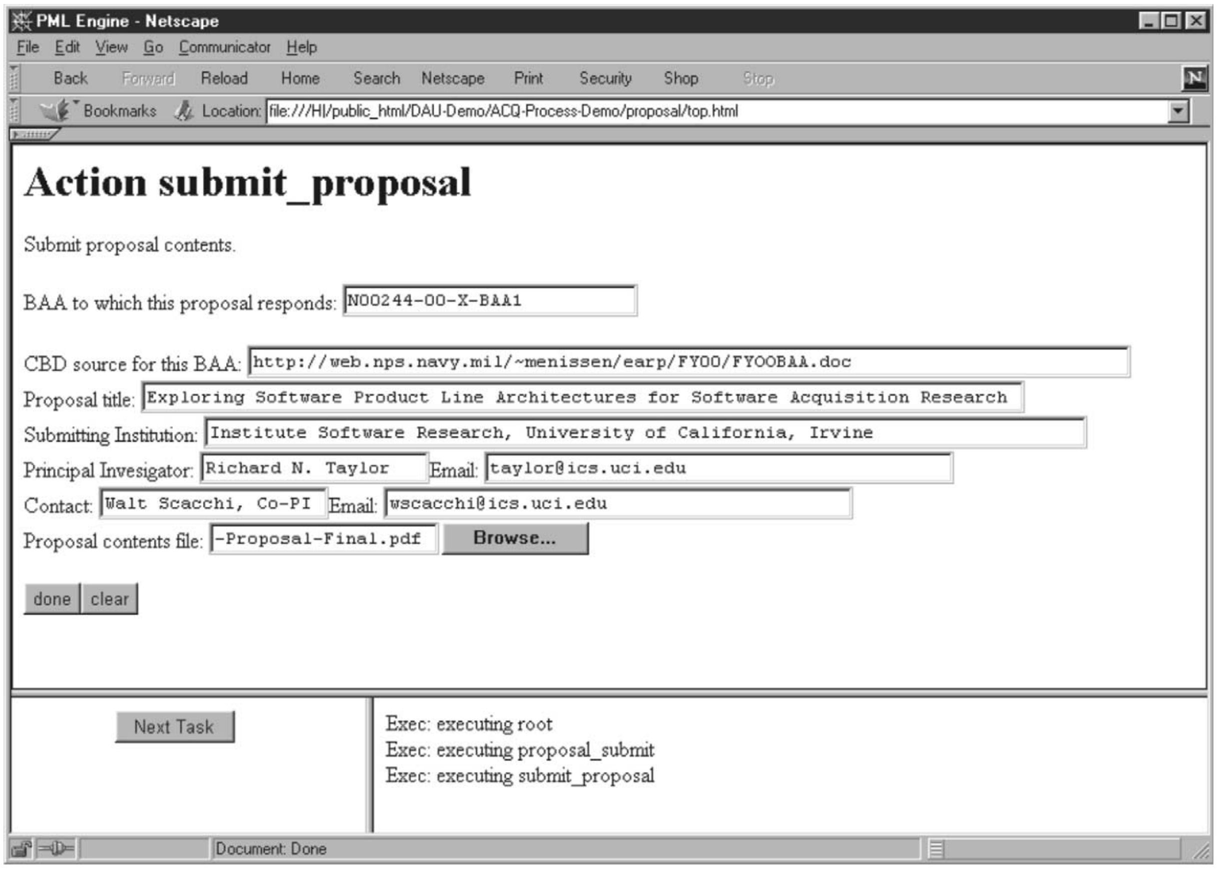
3.1. A software acquisition process simulator

We continue to employ and extend the process simulator techniques noted above, but now we apply them to the domain of software acquisition process architectures. As our software process architectures are configured and interlinked (i.e., "hyperlinked"), then their internal/external representation can be navigated as a process-oriented hypertext (Noll and Scacchi, 1999, 2001). This capability provides a basis for providing Web-based process prototyping, simulator and enactment services. Using PML as the basis for modeling SPAs, we were able to produce a software acquisition process simulator whose operations and capabilities are similar to what we achieved and demonstrated in previous work (Scacchi, 2000). However, now we are able to enrich the experience of people interacting with an acquisition process simulator through the ability to model and link collateral assets for simulating multiple, interacting software processes in a manner that can be distributed and accessed over the Internet/Web, as we will show later. Accordingly, in Fig. 3, we display the view of a software process simulator for one process step modeled in PML (cf. Exhibit 1).

3.2. A test-bed for simulating architectures for software acquisition processes

Beyond providing a process simulator that supports the navigational walkthrough of software acquisition processes one step at a time, we also are investigating the use of architecture-level simulation techniques to assess the dynamic performance of alternative process enactment scenarios associated with different software acquisition processes or process architectures. Here we have been exploring how the RTI for the HLA can be adapted to support the simulation (i.e., simulated enactment of process events or state transitions) of software process architectures. Current implementations of the RTI provide a framework to simulate, monitor, measure and display the performance of a distributed or federated software system architecture (e.g., see http://www.pitch.se/pRTI). However, our challenge is to determine the appropriateness and performance of the RTI as a simulation facility for distributed software processes and software process architectures in general, and for architecture of distributed software acquisition processes in particular. Accordingly, we set out to prototype a distributed and concurrent simulation of an architecture of software processes using the RTI.

According to the HLA, system simulation components are designated as *federates* (Kuhl et al., 1999). So we designed a software acquisition process architecture consisting of four interoperating process federates (i.e., component processes) that could be performed concurrently. These were processes for:


PML Engine - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: file:///H|/public_html/DAU-Demo/ACQ-Process-Demo/proposal/top.html
Browse...
done clear
Next Task
Document: Done

Action submit_proposal

Submit proposal contents.

BAA to which this proposal responds: N00244-00-X-BAA1

CBD source for this BAA: http://web.nps.navy.mil/~menissen/earp/FY00/FY00BAA.doc

Proposal title: Exploring Software Product Line Architectures for Software Acquisition Research

Submitting Institution: Institute Software Research, University of California, Irvine

Principal Invesigator: Richard N. Taylor Email: taylor@ics.uci.edu

Contact: Walt Scacchi, Co-PI Email: wscacchi@ics.uci.edu

Proposal contents file: -Proposal-Final.pdf

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

Fig. 3. Display view of a single process step from a software acquisition process simulator.

- *Software consumer*: Process components of this type simulate a consumer enterprise (e.g., the US Navy) that seeks a new component-based software application system. The enterprise then requests software system components to be developed by a contractor. Each consumer enterprise eventually receives the requested components, then puts the component to use. Then the consumer requests more components until its needs are met. Multiple concurrent instances were allowed to execute in order to simulate multiple consumer enterprises that can independently request software components to be produced and shipped.
- *Software producer*: Process components of this type simulate a contractor enterprise that produces software for a consumer in response to a submitted request for a software component. Once prepared, the requested component is shipped to the consumer as part of its deployment. Multiple concurrent instances were allowed to execute, in order to simulate multiple producer contractors (or a team of contractors) that could service requests for software (product) components, produce and ship them.
- *Fulfillment mechanism* connector: This process component simulates a fulfillment and deployment mechanism used to represent the basic operation of a wide-area workflow infrastructure that transports consumer requests and producer shipments. This process waits for consumer requests, transmits them to the relevant producer whom in turn responds with a product shipment in reply. A single instance of this process was allowed to execute.
- *Manager*: This process component simulates a program manager (or acquisition program office) that facilitates the flow of information from the consumers through the fulfillment and deployment mechanism to the producers, then back to the consumers with the requested and shipped component products. A single instance of this process was allowed to execute.

These processes are relatively simple, yet they represent basic processes involved in the internal operations and external interactions among a group of enterprises that participate in a software acquisition. These processes, as described above, can obviously be modeled and simulated as a single overall process using a conventional single-threaded simulation package. However, our challenge is to model and simulate these as four concurrent process types whose instances can be distributed to run on one or more multi-threaded run-time platforms. We chose to skip the effort to implement our software process architecture simulation test-bed using multiple networked computers, since that seemed to be primarily a task in network programming that would not contribute significant results to our investigation, though the HLA and RTI can support such a capability.

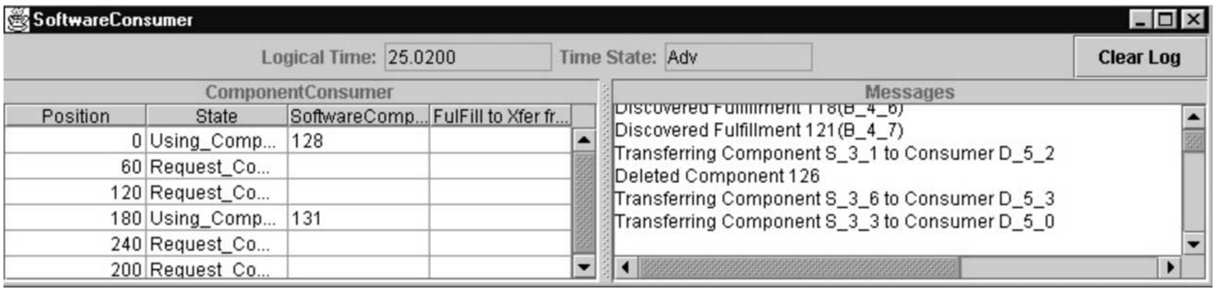
We implemented three process simulation components and one process connector, following the SPA depicted in Fig. 1. Accordingly, we implemented the four software process simulation federates, conforming to the HLA object models (see Fig. 2), in Java. Java was chosen in part for compatibility with our PML and Web-based acquisition process modeling and simulator environment (Noll and Scacchi, 2001; Scacchi and Noll, 1997) and with the Java-based HLA RTI available to us. Each of the four acquisition process simulation component types was implemented as an OO program in approximately one thousand source lines of Java code (about 4000 Java SLOC in total). At least 80% of this code is needed to facilitate use of the HLA RTI interface specification and the simulation messaging OMT required by the HLA standard. The Java code required to create the user interface displays and monitor the execution of the process simulation component is not included in this source code figure, since they employ reusable library packages. The simulation programming task was also simplified through our reuse and modification of a similar multi-federate simulation system example that is supplied by Kuhl et al. (1999) to help document and explain how the HLA and RTI framework is used. Suffice to say that there are many low-level implementation details that we will not describe here involved in the programming of the four types of software acquisition processes in Java to make it conform to the three principal constraints required to use the HLA and RTI. Our results and what we learned from our efforts now follow.

3.3. Results from simulating an SPA when using HLA and RTI

One of the principal results we obtained is an operational prototype of a distributed and concurrent Web-compatible environment for simulating an SPA that entails multiple interacting processes for acquisition. Given that such an accomplishment has not been reported before, it merits consideration for what was achieved and how, as well as what was not realized. In contrast, we did not focus on simulating software processes specific to a particular acquisition program at this time, since this follow-on experimentation requires the modeling and simulation test-bed environment and capabilities that we have developed and describe here. Thus, we will discuss some of the operational capabilities that can be demonstrated and observed at the user interface of this test-bed. More importantly, we can identify six additional results that follow from the creation and evaluation of this approach to simulating software process architectures.

3.3.1. The user interface to a distributed SPA simulation environment

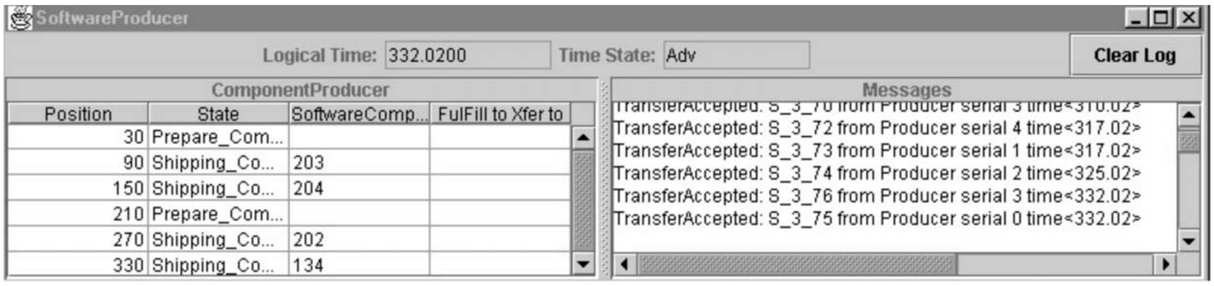
Figs. 4–6 provide a view of the user interface that monitors and displays state transition (on the left sides)



Position	State	SoftwareComp...	FulFill to Xfer fr...
0	Using_Comp...	128	
60	Request_Co...		
120	Request_Co...		
180	Using_Comp...	131	
240	Request_Co...		
200	Request_Co...		

```
Discovered Fulfillment 118(B_4_6)
Discovered Fulfillment 121(B_4_7)
Transferring Component S_3_1 to Consumer D_5_2
Deleted Component 126
Transferring Component S_3_6 to Consumer D_5_3
Transferring Component S_3_3 to Consumer D_5_0
```

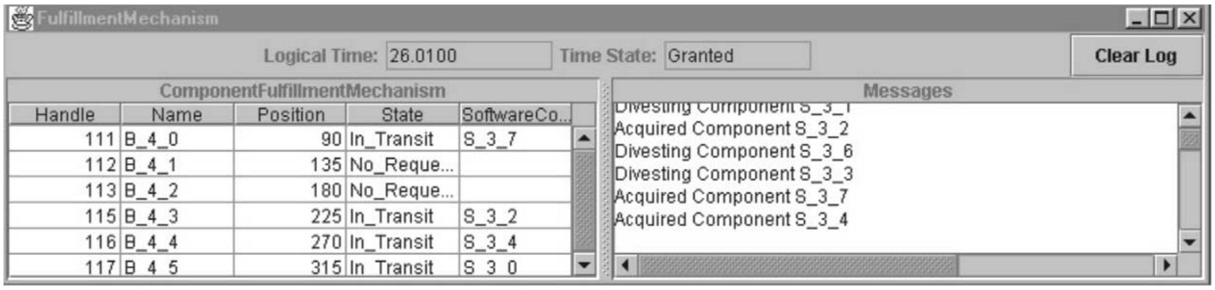
Fig. 4. A UI view of Software Consumer process instance activity.



Position	State	SoftwareComp...	FulFill to Xfer to
30	Prepare_Com...		
90	Shipping_Co...	203	
150	Shipping_Co...	204	
210	Prepare_Com...		
270	Shipping_Co...	202	
330	Shipping_Co...	134	

```
TransferAccepted: S_3_70 from Producer serial 3 time<310.02>
TransferAccepted: S_3_72 from Producer serial 4 time<317.02>
TransferAccepted: S_3_73 from Producer serial 1 time<317.02>
TransferAccepted: S_3_74 from Producer serial 2 time<325.02>
TransferAccepted: S_3_76 from Producer serial 3 time<332.02>
TransferAccepted: S_3_75 from Producer serial 0 time<332.02>
```

Fig. 5. A UI view of Software Producer process instance activity.



Logical Time: 26.0100 Time State: Granted

ComponentFulfillmentMechanism

Handle	Name	Position	State	SoftwareCo...
111	B_4_0	90	In_Transit	S_3_7
112	B_4_1	135	No_Reque...	
113	B_4_2	180	No_Reque...	
115	B_4_3	225	In_Transit	S_3_2
116	B_4_4	270	In_Transit	S_3_4
117	B_4_5	315	In_Transit	S_3_0

Messages

```
Divesting Component S_3_1
Acquired Component S_3_2
Divesting Component S_3_6
Divesting Component S_3_3
Acquired Component S_3_7
Acquired Component S_3_4
```

Fig. 6. A UI view of the Fulfillment Mechanism activity.

and event message histories (on the right sides) associated with instances of three types of software process simulation components. These are, first, the Software Consumer processes that solicit and approve proposals to produce software systems that they acquire. Second, the Software Producer processes that submit proposals in response to solicitation requests to develop software systems then prepare and ship those software systems those whose proposals have been approved by a Software Consumer; and the flow of software artifacts (proposals, software shipments, etc.) through the Fulfillment Mechanism connector. The "Position" field indicates a parameter value that denotes the ordering of messages flowing among the acquisition process participants. The "SoftwareComp" and "Name" are also parameter values that associate an identifier with specific software artifact instances (which are also "components") that are being acquired as they move from Consumers to Producers and back, while traversing the Fulfillment and Manager processes. The Manager view primarily tracks the origination and termination of event notifications and is not shown. Finally, Fig. 7 provides a view of the user interface that graphically depicts an overall global state of a multi-process, multi-instance interaction while simulating a software process architecture for software acquisition.

3.3.2. Interoperation of multiple process simulation components

Using this simulation environment, we are able to demonstrate multiple software process simulations whose interoperation is distributed and concurrent, through the use of independent control threads. The execution and interoperation that is realized through the message passing scheme supported by the HLA and RTI is monitored and displayed through the user interfaces described above. This result represents an advance in the development of new infrastructures that support software process simulation. Furthermore, our expectation is that adding more content and complexity to the process simulation dynamics would have little impact on the simulation code that interfaces to the RTI.

3.3.3. Architectural-level simulation of software processes

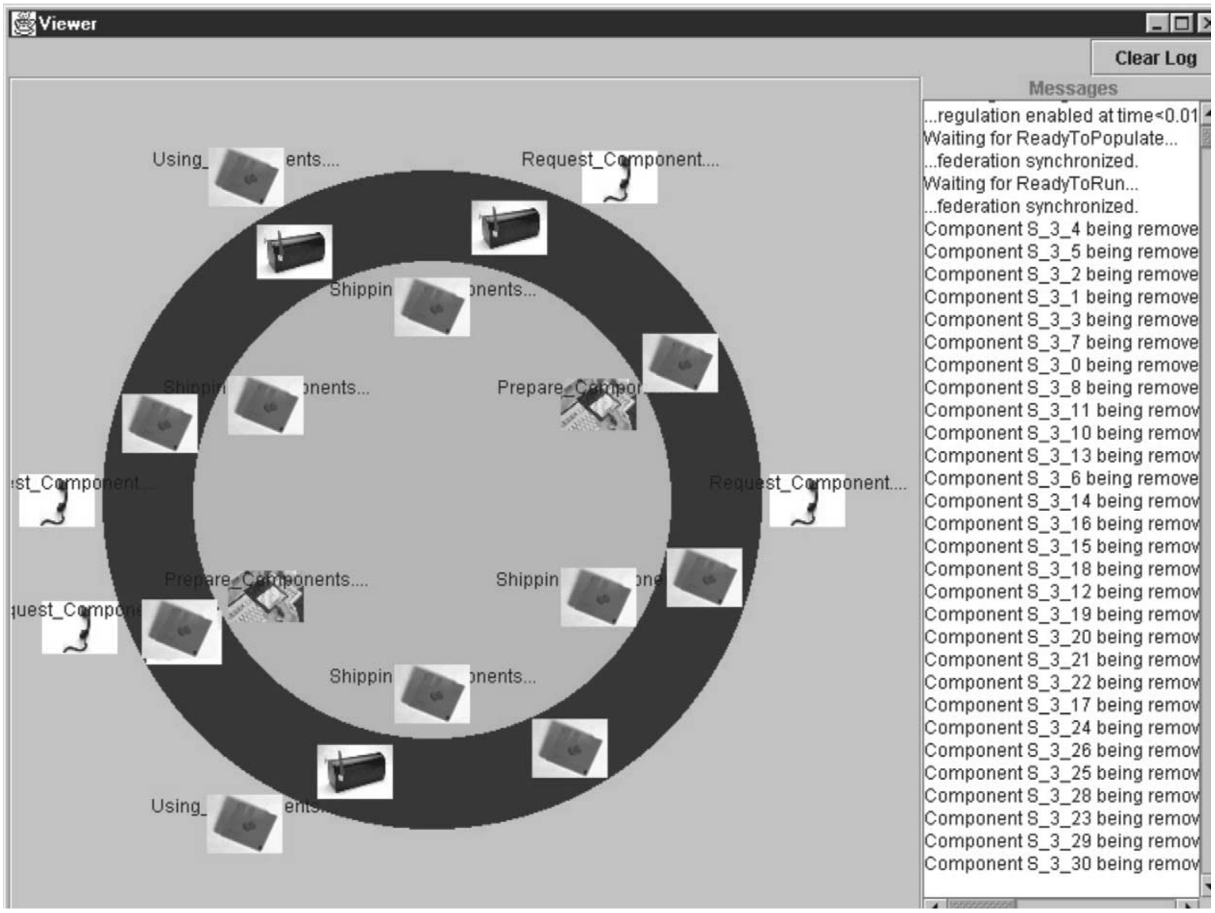
Though our simulations model relatively simple processes and process connectors for software acquisition, they demonstrate the HLA and RTI can be used to implement and simulate software process architectures. The process simulation components that are organized into a SPA may be distributed, interoperate, and execute concurrently. Architecture-level simulation is a technique that enables process simulation at a new, more abstract "system of systems" level of detail, compared to the granularity of conventional software process simulation systems. Such a technique has not previously been employed in simulating software processes, and thus represents a new technique for analyzing complex software processes whose process simulation components may be physically distributed, but logically centralized (cf. Noll and Scacchi, 1999).

3.3.4. Use of Web-compatible technologies

We implemented our distributed software process simulation test-bed using Java to simulate software acquisition processes that were specified and modeled in PML. Such a convenience though not an advance, nonetheless supports the construction, navigation and geographically distributed simulation of software processes and process simulation components. This may enable people working in multiple enterprises that are nationally or internationally distributed to access and refine shared models of software processes, which is an important consideration in a domain like software acquisition (Noll and Scacchi, 1999; Scacchi and Boehm, 1998; Scacchi and Noll, 1997).

3.3.5. Reusable approach and framework for integrating distributed software process simulations

As shown in Kuhl et al. (1999) the impact of adding additional simulation components can be modest, once the cost of interfacing them to the RTI and HLA object model templates is incurred. Thus, part of the attraction to the use of the HLA and RTI for simulating SPAs is the ability to reuse, integrate and interoperate more process component simulations for other software acquisition processes, sub-processes, etc. once they are



```
...regulation enabled at time<0.01
Waiting for ReadyToPopulate...
...federation synchronized.
Waiting for ReadyToRun...
...federation synchronized.
Component S_3_4 being remove
Component S_3_5 being remove
Component S_3_2 being remove
Component S_3_1 being remove
Component S_3_3 being remove
Component S_3_7 being remove
Component S_3_0 being remove
Component S_3_8 being remove
Component S_3_11 being remov
Component S_3_10 being remov
Component S_3_13 being remov
Component S_3_6 being remove
Component S_3_14 being remov
Component S_3_16 being remov
Component S_3_15 being remov
Component S_3_18 being remov
Component S_3_12 being remov
Component S_3_19 being remov
Component S_3_20 being remov
Component S_3_21 being remov
Component S_3_22 being remov
Component S_3_17 being remov
Component S_3_24 being remov
Component S_3_26 being remov
Component S_3_25 being remov
Component S_3_28 being remov
Component S_3_23 being remov
Component S_3_29 being remov
Component S_3_30 being remov
```

Fig. 7. A user interface view of the overall simulated state of a concurrent multi-process SPA for software acquisition shown in Fig. 1.

encapsulated to run with HLA and the RTI. Subsequently, the next step is to add more realism and detail to our software acquisition process simulations in line with what we have already achieved with those modeled in a process-oriented hypertext (Noll and Scacchi, 2001).

Beyond this, the capability we demonstrated with the development of our HLA and RTI test-bed can be independently reproduced with reasonable effort. We have characterized the general terms of our implementation and have indicated reference citations for where others may acquire the HLA and RTI resources from which we started. Thus, our approach to developing a distributed software process simulation environment for experimentation is reusable, as is the framework we employed.

3.3.6. Partial demonstration of scalability

The set of preceding results help demonstrate that there may be a path towards the construction and operation of a scalable approach and infrastructure that can support the integration and interoperation of independently developed software process simulations. Such a "virtual test-bed" for software process simulation does not yet exist, but the preceding results perhaps suggest a way it could (or could not) happen. If researchers and practitioners agree to build and use software process simulations or simulation components that are compatible with the HLA and RTI, then this form of global scalability could be realized. However, our experience with HLA and RTI though suggests that such effort may be undesirable from a technical standpoint, based on the assumption of a single architectural style. Similarly, it may be unrealistic from a pragmatic standpoint, unless participants in the software process simulation community are willing to collectively migrate their efforts onto process simulation servers that are compatible with the HLA and RTI.

3.3.7. Successful demonstration of a novel approach modeling and simulating software processes

Overall, the six preceding results provide evidence as to both the plausibility and viability of modeling and simulating multiple, interacting and distributed software processes through the use of software process architectures. Software architectures and architectural design techniques have emerged elsewhere within the software engineering community. The approach and results we describe indicate that the concepts and techniques associated with software system architectures can be adopted and adapted for use in modeling and simulating complex software processes that span and interlink multiple enterprises. Such an advance may help realize the ability to specify, analyze and understand software processes of a greater organizational and managerial complexity than have heretofore been demonstrated or realized. As such, we have helped move one step closer to the ability to design, redesign, or optimize the web of software processes associated with the acquisition of software-intensive systems.

4. Discussion and conclusions

We now turn to highlight and summarize what is new in this research.

To our knowledge, this work represents the first effort to investigate and provide results on how software process modeling and simulation tools, techniques and concepts can be applied to the domain of software system acquisition. Software acquisition processes are large-scale, involve multiple distributed enterprises and stakeholders, and are expensive, long-lived and frequently plagued with process coordination (interoperation) problems. However, our research challenge is not to simply model and simulate software acquisition processes as just another software process. Instead, we find the domain of software acquisition imposes challenges for modeling and simulating software processes in a way that is factorable into distributed and concurrent components, since acquisition processes in practice are inherently distributed and concurrently in operation in multiple enterprise settings. Thus, we chose to model and simulate software acquisition processes in a manner that reflects and embodies how such processes are physically dispersed, while logically configured to interoperate. To help demonstrate this, we used two Web-compatible approaches to modeling software acquisition processes: one based on a declarative process modeling language PML and its process-oriented hypertext infrastructure; the other based on the implementation of OO programs (in Java) that encapsulate interfaces to the HLA standard and RTI specification.

This in turn serves as motivation for establishing and evaluating software process architectures as a technique to address these challenges. This is the second area in which we have contributed. Up to this time, it appears that software process modeling and simulation efforts have assumed or been targeted to operate with one model at a time in a single thread address space. This is particularly true of efforts that rely in the use of commercially available packages for discrete-event, continuous system (e.g., systems dynamics) or entity-state simulation. Interoperation of multiple, distributed and concurrent software process models or simulations is generally beyond the scope or capability of these packages. In contrast, our interest was to investigate the modeling and simulation of multiple interacting software processes as a system of process simulation systems with interfaces that can be interconnected to enable resource, data or control flow through process connectors. In this regard, we have introduced how software architecture concepts can be used to model and simulate software process architectures that are logically centralized, but physically distributed.

Next, we described how software process architectures could be evaluated with a distributed process simulation environment. We demonstrated a test-bed environment that supports the simulation of the concurrent interoperation of distributed software acquisition processes and multi-threaded process instances. Our test-bed implementation was demonstrated with relatively simple software acquisition processes. Such an environment is best viewed as a test-bed for simulating and evaluating large sets of complex interacting software processes where scalability and networked distribution are required. The acquisition of large military or public infrastructure application systems have such a requirement. Nonetheless, there is a cost to be incurred for the use of such an environment. However, in our view large and multi-enterprise processes for software acquisition may be a well-suited domain for incurring such costs, since the analyses and decision-making insights that are enabled through modeling and simulation are well justified (Brown et al., 2000). In contrast, the test-bed environment is probably too much mechanism to simulate small or simple software processes where distribution and concurrency are not essential aspects of the problem domain.

We also introduced an effort to use and assess the viability of the HLA and RTI as a standards-based platform for simulating the performance of software acquisition processes that are configured as a distributed, concurrent architecture. This effort was posed in contrast to a companion effort based on mature software process modeling language and techniques. Here we came to find that subtle differences in how the semantics of software processes can impact which architectural styles may be most effectively employed when modeling and simulating software process architectures. This was an unexpected result, since to us it represents a barrier for integrating and interoperating multiple independently developed software process models and component process simulations. Thus, the current HLA may not be the best choice for modeling and simulating distributed, interacting software processes.

Finally, we believe software process architectures, together with new architectural frameworks and environments for modeling and simulating distributed, multi-component software process architectures represent promising new areas for further research and development within the software process community.

Capabilities such as these provide new opportunities to conduct experiments and software process performance evaluation studies at a new level of granularity, and with a new kind of test-bed infrastructure. This paper thus describes some initial steps into these areas.

Acknowledgements

This work is supported by contract N487650-27803 to the Institute from Software Research from the Defense Acquisition University as part of their External Acquisition Research Program (EARP). No endorsement of the results or conclusions presented should be ascribed to the research sponsor or the US Government.

References

ARO, 1999. Implementing Acquisition Reform in Software Acquisition. Navy Acquisition Reform Office, http://www.acq-ref.navy.mil/turbo/refs/software.pdf.

Bergey, J.K., Fisher, M.J., Jones, L.G., 1999. The DoD Acquisition Environment and Software Product Lines. Technical Note CMU/SEI-99-TN-004, Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, PA.

Boehm, B.E., Scacchi, W., 1996. Simulation and Modeling for Software Acquisition (SAMSA). Final Report, Center for Software Engineering, University of Southern California, Los Angeles, CA, http://sunset.usc.edu/SAMSA/samcover.html.

Brown, C.D., Grant, G., Kotchman, D., Reyenga, R., Szanto, T., 2000. Building a business case for modeling and simulation. Acquisition Research Quarterly 24 (4).

Choi, S.J., Scacchi, W., 1990. Extracting and restructuring the design of large software systems. IEEE Software 7 (1), 66–73.

Choi, S.J., Scacchi, W., 1998. Formalization and tools supporting the structural correctness of software life cycle descriptions. In: Proceedings of the IASTED Conference on Software Engineering, International Association of Science and Technology for Development (IASTED), Las Vegas, NV, pp. 27–34.

DD21, 2001. DD21 Zumwalt-Class 21st. Century Destroyer, http://sc21.crane.navy.mil/.

Fujimoto, R., 1999. Parallel and Distributed Simulation. Wiley, New York.

GAO, General Accounting Office. Air traffic control–immature software acquisition processes increase FAA system acquisition risks. Report GAO/AIMD-97-47, 1997.

HLA., 1999. High-Level Architecture Web Site. http://hla.dmso.mil.

Kuhl, F., Weatherly, R., Dahmann, J., 1999. Creating Computer Simulation Systems: An Introduction to the High Level Architecture. Prentice-Hall PTR, Upper Saddle River, NJ.

Medvidovic, N., Taylor, R.N., 2000. A Classification and comparison framework for software architecture description languages. IEEE Transactions on Software Engineering 26 (1).

Mi, P., Scacchi, W., 1990. A knowledge-based environment for modeling and simulating software engineering processes. IEEE Transactions on Knowledge and Data Engineering 2 (3), 283–294.

Mi, P., Scacchi, W., 1996. A meta-model for formulating knowledge-based models of software development. Decision Support Systems 17 (3), 313–330.

Noll, J., Scacchi, W., 1997. Supporting distributed configuration management in virtual enterprises. In: Conradi, R. (Ed.), Software Configuration Management. Lecture Notes in Computer Science, vol. 1235, pp. 142–160.

Noll, J., Scacchi, W., 1999. Supporting software development in virtual enterprises. Journal of Digital Information 1 (4).

Noll, J., Scacchi, W., 2001. Process-oriented hypertext for organizational computing. Journal of Network and Computer Applications 24 (1), 39–61.

Radice, R.A., Roth, N.K., O'Hara, A.C., Ciarfella, W.A., 1985. A programming process architecture. IBM Systems Journal 24 (2), 79–90.

Scacchi, W., 1999. Experience with software process simulation and modeling. Journal of Systems and Software 46, 183–192.

Scacchi, W., 2000. Understanding software process redesign using modeling, analysis and simulation. Software Process – Improvement and Practice 5 (2/3), 183–195.

Scacchi, W., Boehm, B.E., 1998. Virtual system acquisition: approach and transition. Acquisition Review Quarterly 5 (2), 185–215.

Scacchi, W., Noll, J., 1997. Process-driven intranets: life-cycle support for process reengineering. IEEE Internet Computing 1 (5), 42–49.

Schooff, R.M., Haimes, Y.Y., Chittister, C.G., 1997. A holistic management framework for software acquisition. Acquisition Review Quarterly.

Shaw, M., Garlan, D., 1996. Software Architecture: A New Perspective on an Emerging Discipline. Prentice-Hall, Englewood Cliffs, NJ.

SA-CMM, Software Acquisition Capability Maturity Model. Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, PA, 2000 http://www.sei.cmu.edu/arm/SA-CMM.html.

James Choi is an Assistant Professor of Computer Science at the California State University at Fullerton (CSUF). He received MS and Ph.D. in Computer Science at University of Southern California, with emphasis in Software Engineering.. His interests are Software Engineering, Software Development Process Modeling, Software Acquisition, Configuration Management and Reverse Software Engineering.

Walt Scacchi is a research computer scientist at the Institute for Software Research at the University of California, Irvine. He joined ISR in 1999 after serving on the faculty at the University of Southern California since 1981. He received his Ph.D. in Information and Computer Science at the University of California, Irvine in 1981. From 1981 to 1991, he directed the USC System Factory Project, and from 1993 to 1998, he directed the USC ATRIUM Laboratory. His interests include organizational studies of Software Development, Software Process Engineering, Software Systems Acquisition, Electronic Commerce, and Collaborative Work Environments. He has published more than 100 research papers, and consults widely to clients in industry and government agencies.