Leveraging Distributed Software Development

As the complexities of distributed collaborative-development environments increase, frameworks specifically designed for such environments will become increasingly essential. The authors describe one such framework—called SoftDock—and the new technologies it exploits.

Junichi Suzuki
Yoshikazu Yamamoto
Keio University

The Internet has been changing the way we collaborate on software development, offering certain advantages but also creating new requirements. Internet-based collaboration does make a wider base of talent available, but the development cycles running at Internet speeds—where your codevelopers might be working as far away as the other side of the planet—require maintaining higher levels of precision.

From a project-management perspective, communication is a key factor in Internet-based development. Internet-based collaboration requires effective team communication so that project managers can point all team members in the right direction. When your development team isn't communicating well, it is nearly impossible to create and validate design solutions and effectively manage your team's deliverables. So while Internet collaboration offers a number of advantages, the friction created by distributed—and therefore *delayed*—communication typically increases the overhead associated with sharing project information.[1]

Furthermore, the technology itself—including system interoperability and the synchronous or asynchronous collaboration tools you use—can create a variety of problems in a distributed development environment. Unfortunately, few development frameworks have attempted to solve some of the problems that have emerged in the context of Internet-based collaboration.

In addition to some of these general issues, several specific issues are raised by the kind of Internet-based collaboration that uses software components. For this kind of distributed collaboration to be truly effective—for it to solve more problems than it creates—the development framework should allow you to

- model software components and the relationships among them,
- describe and share component model information, and
- ensure the integrity of component models.

The SoftDock[2-4] infrastructure does just that. We designed SoftDock to let developers—working in a distributed-collaboration environment—analyze, design, and develop software from component models. In addition to general distributed, component-based development, we are currently using SoftDock as a framework to develop applications as diverse as distance learning,[3] digital libraries, hypermedia model management, and medical records systems.

There are countless additional applications for frameworks like SoftDock. As the complexities of distributed collaborative-development environments increase, frameworks specifically designed for such environments will become increasingly essential.

SOFTDOCK BASICS

SoftDock uses the Object Management Group's Unified Modeling Language (UML) for modeling components. For exchanging information about these UML models, SoftDock uses an application-independent interchange format called the UML eXchange Format (UXF).[4] SoftDock distributes UXF descriptions through W3C's Document Object Model (DOM) interface, which is implemented on top of OMG's Common Object Request Broker Architecture (CORBA).

Figure 1 illustrates SoftDock's architecture. SoftDock stores the XML documents representing UML models—that is, UXF descriptions—in a resource server,

0018-9162/99/$10.00 © 1999 IEEE

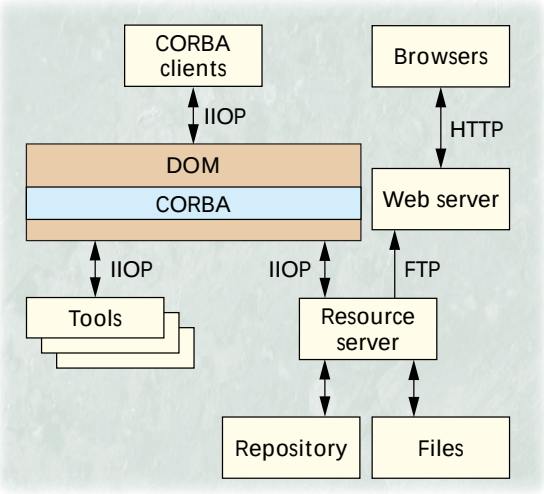

Figure 1. SoftDock's architecture is made up of a number of different elements, all of which are designed to support a high degree of interoperability so that developers in remote locations can collaborate with different development tools without having to sacrifice model information. SoftDock stores these models on its resource server, which facilitates distribution across the Web to XML-compliant browsers or on a network through various interfaces defined by specifications like DOM and CORBA.

which can create, log, and delete documents. The resource server sends notification to client-side applications when a document changes. In theory, this repository could maintain documents either as flat files (for storing model descriptions) or as full-fledged objects in a repository. Currently, though, SoftDock supports only flat file management.

In SoftDock's architecture, the resource server has two ways of distributing or exchanging model information. In the first method, an HTTP route provides one-way broadcasting to client applications. These client applications—including XML-capable Web browsers—access UXF descriptions that the resource server replicates to a Web server through an FTP connection. When a UXF description is updated in the resource server, the server pushes the updated description to the potentially remote Web server.

In the second method, an Internet InterOrb Protocol (IIOP) connection (described more completely later in the article) provides two-way communication between a client and a server. This method is much like the first, except that when users modify remote UXF descriptions, the server immediately reflects any changes.

This architecture has a number of advantages, including

- increasing model reuse by allowing developers to specify software models independently of their implementation technologies, such as language, network, or operating system;
- reducing the information overhead required by the use of different tools in different locations or development phases;
- allowing global scalability and system interoperability by using a standardized interface; and
- supporting state-based development processes[5]—based on systematic design and peer review—to streamline development processes and increase quality control.

We built these characteristics into SoftDock to foster model continuity across development phases and to help improve developer productivity. Specifically, we designed SoftDock to make UML models more universal by providing interoperability on three levels: through UXF, through DOM, and through CORBA.

UXF allows UML models to be interoperable among UML-compliant tools. DOM allows UXF descriptions to be interoperable with XML-compliant tools. And CORBA provides the standard interfaces to allow DOM-compliant tools to interact with each other in a network environment.

CORBA will be familiar to most readers, but UXF (which is our own creation) and DOM (which was only recently finished by W3C) probably won't be. In the remainder of this article, we'll discuss UXF, DOM, and how SoftDock uses these technologies to facilitate distributed collaborative work.

UML EXCHANGE FORMAT

The emergence of UML—created by the joint efforts of object technologists Grady Booch, Ivar Jacobson, and James Rumbaugh—represents one of the most significant developments in object technology. Supported by a broad base of companies, UML defines semantics of object model elements and their notations used by popular analysis and design methodologies to produce a single, universal modeling language that can be used with any method. It is this language that SoftDock uses to create models.

But in a distributed-collaboration development environment—where exchanging models is a crucial activity—it is important that the semantics within a model be described explicitly. Being able to exchange models is particularly important in environments where there are few application-neutral exchange formats that you can use to transfer models between different development tools. To solve this problem, we developed the UML eXchange Format (UXF), which is similar in some respects to OMG's recently completed XML Metamodel Interchange (XMI) format.

Both UXF and XMI are based on XML and serve as a communication vehicle for transferring model information among development tools. We use UXF in the SoftDock system both because OMG hadn't finished XMI when we began developing our project and because we believe UXF is much simpler to use than XMI. However, SoftDock will eventually use XMI by providing a UXF-XMI converter.

Easy communication

The ability to maintain clear technical information during the entire software life cycle is vital to development teams. Every team typically has a large volume of information to manage. Members of the team use this information to manipulate what they're working on and to record the results of their tests. UXF makes it relatively easy to communicate ideas about the UML models by providing a well-structured, standard way of sharing model information.

In addition, because UXF can apply XSL style sheets, you can convert your model information into a variety of media, like RTF, HTML, LaTeX, or PDF. Moreover, you can embed hyperlinks into UXF, which allows you to link UML constructs to each other. We designed all of these features to make it much easier to keep track of model information across all stages of the development process.

Seamless interoperability

UXF consists of a series of XML document type definitions (DTDs), which are schema that specify how a set of tags and their constraints will appear in a certain XML document. Associated with each package in the UML metamodel specification, each DTD maps UML model elements into document tags. Furthermore, every DTD supports the XML namespace, which means you can avoid tagname conflicts between UXF DTDs and other DTDs.

The W3C designed XML as a sophisticated subset of SGML to provide application neutrality, user extensibility, and the ability to represent complex information. Because UXF shares in these qualities, SoftDock can provide the same kinds of features. For example, the software tools used in each phase of development usually employ their own proprietary formats to describe model information. But once encoded with UXF, the model information can be reusable—without modification—in just about any tool.

Because UXF is a natural, transparent extension of the existing Web environment, SoftDock allows you to edit, publish, access, and exchange descriptions as easily as writing and editing HTML. Most existing Web applications—like the XML editor shown in Figure 2—can be used for handling UXF, as can most CASE, documentation, and visual-profiling tools.

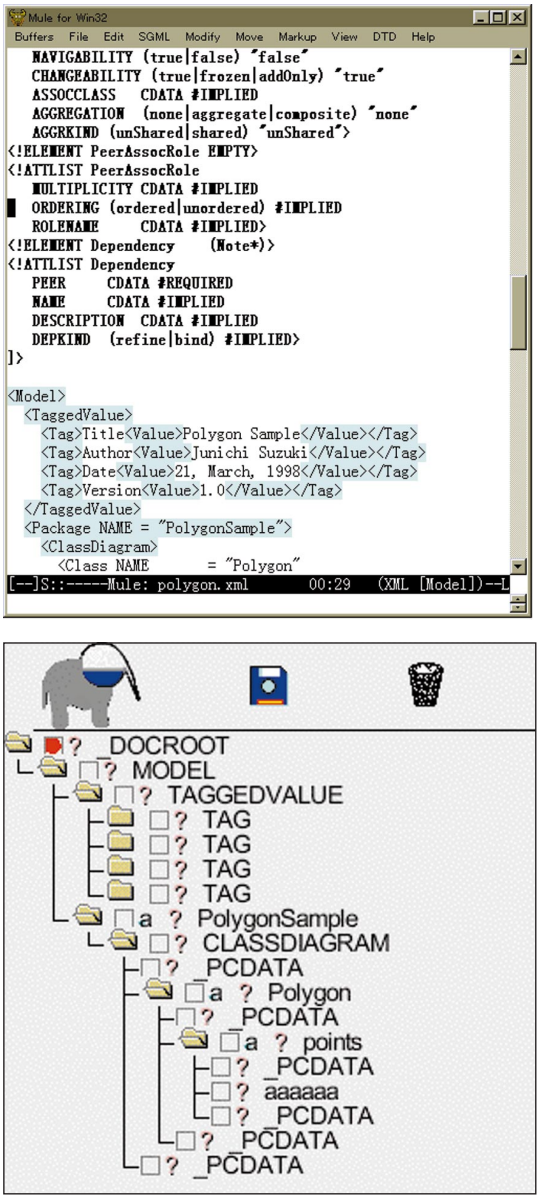

Figure 2. SoftDock makes editing model descriptions relatively easy. The screen shot at the top shows UXF-formatted data within an XML editor, while the one on the bottom shows a hierarchical structure of UXF elements within an XML browser.

DOCUMENT OBJECT MODEL

As we've implied, the primary goal of SoftDock is to provide a framework for sharing and managing UML models in the form of UXF descriptions. In general, the APIs that support XML documents—and, by extension, UXF documents—can be categorized into three groups:

- the source document API, which manages the XML document instances directly,
- the element API, which manages the parsed elements of a document instance, and
- a custom API, which provides application-specific interfaces.

Custom APIs are not appropriate for exchanging generic UXF descriptions because they need to be modified when your UXF DTDs change.

If you were to implement the source document API in SoftDock, a client application would fetch every UXF description from the server and parse it to inspect its model information. This means that the larger a UXF description is, the longer the transfer overhead will be. Since we would like systems as small

```
#pragma prefix "jp.ac.keio.SoftDock"
#include <dom.idl>
#include <CosEventComm.idl>
module SoftDockExtention
{
  exception MetadataNotSupported{};
  exception AlreadyLocked{CorbaDocProxy proxy};
  exception InvalidLockTypeSpecified{};

  interface UXFDescription
    :dom::Document,
    :CosEventComm::TypedPushConsumer,
    :CosEventComm::TypedPushSupplier
  {
    typedef long UID;
    attribute UID nodeId;
    void externalize();
    sequence<string> content();
    sequence<string> metadata()
      raises(Metadata NotSupported);
    sequence<string> query(in string xqlQuery)
    boolean isLocked();
    void requestLock (in CorbaDocProxy proxy
                      in short lockTypeId)
      raises(AlreadyLocked,
        InvalidLockTypeSpecified);
    void releaseLock();
    void update(in short changeTypeId,
                in UXFDescription doc);
  };

  interface CorbaDocProxy
    :UXFDescripion
  {
    attribute dom::UXFDescription remoteDoc;
  };

  interface CorbaDocFactory
  {
    UXFDescription checkIn(in sequence<string>
      doc);
    UXFDescription createDocument(in string
      docName);
    UXFDescription cloneDocument(
      in UXFDescription doc);
    void releaseLock(in UXFDescription target);
    void destroyDocument(in UXFDescription doc);
  };

  interface NodeIterator
    :dom::NodeList
  {
    boolean next(out dom::Node);
    oneway void destroy();
    readonly attribute unsigned long length;
  };
};
```

Figure 3. SoftDock's extension to the DOM interface consists of four IDL interfaces—UXFDescription, CorbaDocFactory, CorbaDocProxy, and NodeIterator—each of which has functions that correspond to its name.

as thin clients to be able to access remote UXF descriptions in a more structured manner, SoftDock uses a generic-element API that is based on an extension of DOM.

DOM's consistent interface

DOM is an API for HTML and XML documents that defines the logical structure of documents and the way a document can be accessed and manipulated. In the DOM specification, the term "document" is used in a fairly broad sense. Increasingly, XML is being used as a way of representing many different kinds of information, much of which would traditionally be seen as data rather than as documents. Nevertheless, XML presents this data as documents that DOM can be used to manage.

W3C designed DOM to be used with any language. In order to provide a precise, language-independent specification of interfaces, W3C defined the specifications in OMG's IDL, a language-neutral interface definition language that is a primary component of CORBA. Implementing DOM interfaces on top of CORBA brings us closer to our goal of making UML model information highly interoperable in a network environment. This strategy allows the SoftDock interfaces to be created independently of UXF DTDs, which means that changes in these DTDs don't have to be reflected in the interfaces.

In short, DOM provides a consistent interface to SoftDock for accessing and manipulating the content and structure of UXF documents. It gives SoftDock a standard set of objects for representing documents, a standard model for how these documents should be combined, and a standard API for accessing the objects. DOM basically serves as a standard parser interface between XML documents and their applications. Having such an API makes it possible for those using SoftDock to develop UXF models using any DOM-compliant XML parsers.

SoftDock's extension to DOM

DOM can be implemented using language-independent systems like COM or CORBA; it can also be implemented using language-specific bindings like Java. In SoftDock, we implement DOM with CORBA. Created by OMG, CORBA is the most popular middleware standard for communications between distributed objects. CORBA provides a way to execute programs written in different languages running on different platforms no matter where they reside in the network. This kind of neutrality and transparency are useful both to DOM and, by extension, to SoftDock.

It is important to realize that the DOM interfaces are abstractions: They are a means of specifying a way to access and manipulate an application's internal repre-

sentation of a document. Interfaces do not imply a particular concrete implementation. Each DOM application can maintain documents in any convenient representation as long as the DOM interfaces are supported.

SoftDock now uses both a publicly available XML parser from IBM and our own DOM-compliant parser to work with DOM interfaces. But we needed to extend the current DOM interfaces for use in a SoftDock distributed environment. Our extension to DOM, shown in Figure 3, provides the essential components for extending the current DOM interfaces to allow SoftDock to handle arbitrary XML descriptions in a CORBA environment.

SoftDock's IIOP connection

One of the most important components in our extension of DOM is the *document factory*, which helps manage the complete life cycle of XML documents. Another important component is the *document proxy*, which fetches remote XML documents and keeps them consistent on the client side. As Figure 4 illustrates, the document factory and the document proxy are crucial elements in the relationship between different APIs that handle XML documents across an IIOP connection.

Generally speaking, a CORBA client at runtime makes requests to remote CORBA objects via an Object Request Broker (ORB). The ORB provides a proxy object in the client's address space, which creates the illusion that the remote object is a local one. (SoftDock also enables local caching of these objects, which will later be resynchronized on the remote server.) The client and server communicate by exchanging messages defined by the General Inter-ORB Protocol (GIOP). GIOP is independent of any specific network transport. However, when GIOP is sent over TCP/IP, it is called IIOP.

In SoftDock, the IIOP connection provides the two-way communication between a client and server, which lets users at different locations create and modify remote XML documents. Client applications can include CASE, documentation, design-metric, and reverse-engineering tools. They can also include source code editors and generators.

To create SoftDock's IIOP connections, you manipulate XML descriptions through either DOM or the simple API for XML (SAX). As a parser interface developed in the XML community (http://www.megginson.com/SAX/index.html), SAX is event-based, while the DOM interface is tree-based. Some DOM-compliant parsers use a SAX-based parser internally, but both APIs are now supported by various parsers implemented in a variety of programming languages. Any parsers supporting either DOM or SAX can be plugged into the SoftDock system.

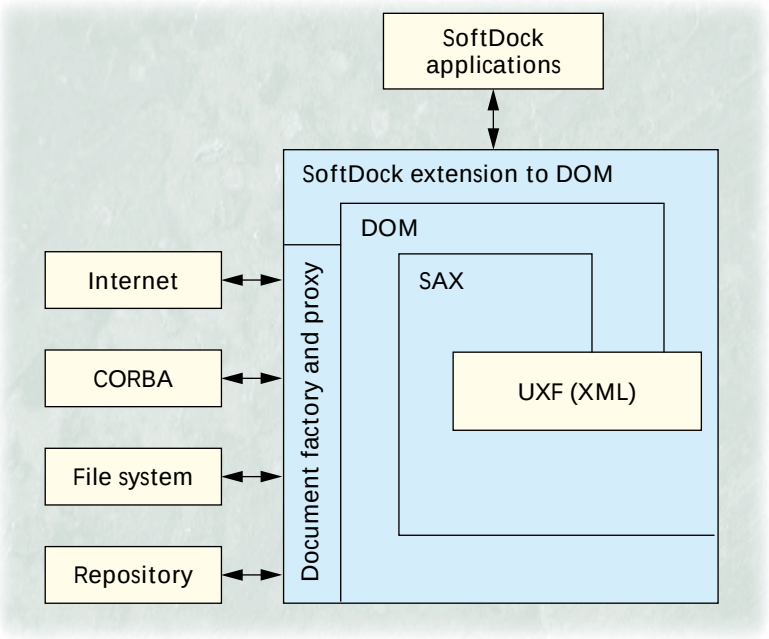

Figure 4. SoftDock employs a number of APIs to handle XML documents—and UXF model descriptions—over an IIOP connection.

Managing UXF descriptions

SoftDock helps you manage UXF descriptions through an IIOP connection by allowing client applications to

- locate and bind to remote UXF descriptions,
- match UXF descriptions with their metadata,
- explore and query internal elements of a UXF document,
- cache remote UXF descriptions at client side,
- manipulate a UXF description synchronously and asynchronously, and
- monitor every change (that's happened) to a UXF description.

Every UXF document is checked into the SoftDock resource server with the document factory. Then it is parsed into an object tree—according to its tag hierarchy—with a DOM-compliant XML parser. The object tree is maintained as a set of CORBA objects, as illustrated in Figure 5. Client applications can find and connect to remote UXF descriptions with a white-page service that uses the CORBA naming service and a yellow-page service that uses the CORBA trading service.

A UXF document has its own metadata, which it exposes to client applications in the form of W3C's Resource Description Framework (described below) or through the CORBA trading service. A client application can then search appropriate UXF descriptions—that meet the client's needs—on the basis of the descriptions' metadata contents.

SoftDock also allows client applications to explore and query the internal elements of a UXF description with an iterator function—see Figure 3—and by using the XML Query Language (http://www.w3.org/TandS/QL/QL98/pp/xql.html), which uses XML as a data model.

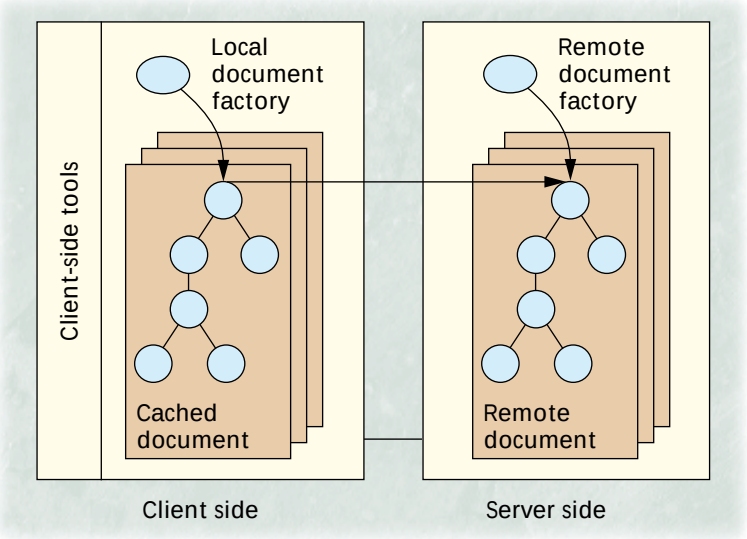

Figure 5. SoftDock caches remote documents locally and maintains them in an object tree as a set of CORBA objects.

Each UXF description can be cached on the client side once it is accessed. SoftDock provides a proxy object in the client address space—called a *document proxy*—which creates the illusion that the remote object is a local one. This reduces the number of messages sent and, by extension, the communication overhead.

SOFTDOCK APPLICATIONS

The SoftDock application we've stressed in this article is distributed, collaborative software development, which will no doubt be one of SoftDock's main applications. So far, we've used SoftDock for a number of such projects, including using it to manage the model information for Java application, applet, and Java Bean development.

Personalizing model information

We deploy Persona,[3] a toolkit we developed for personalizing the content and presentation of XML documents, at the back end of a Web server, where it recognizes every participant's role in the development process by focusing on the client-side information that is transferred with incoming HTTP requests. In response to these requests, Persona delivers customized content and presents information appropriately tailored to each development role, such as project manager, architect, programmer, or customer.

Persona provides two levels of personalization. The first level involves simply generating an HTML document from a requested XML document by applying a particular XSL style sheet. The second level involves rearranging the content or presentation of the generated document. For example, when a project manager accesses a Persona-enabled Web server through the SoftDock HTTP connection, Persona can create a document—based on state-based development processes—that displays a current development status report for all software deliverables. If the same request were made by a programmer, Persona could present detailed model information instead of a status report.

Facilitating distance learning

SoftDock can provide self-paced learning through its HTTP connection and group-paced training through its IIOP connection.[3] In such environments—where roles can include educator, learner, and moderator—Persona can also be useful. And for these environments—where metadata tagging can be particularly useful—SoftDock provides additional services.

For example, SoftDock can use W3C's Resource Description Framework (RDF), which is a specification that defines a way to describe metadata. RDF allows SoftDock to understand metadata for tagging each UXF description. In addition to RDF, SoftDock can use the International Management Systems metadata specification (http://www.imsproject.org/metadata/) to increase metadata interoperability between educational resources and computer-mediated training systems.

Managing hypermedia user interface models

In addition to using SoftDock to facilitate software development and support distributed learning, we've also used SoftDock to manage the user interfaces of Web-based applications. SoftDock can streamline the requisite application design work and automate deployment by sharing Web Interface Definition Language (WIDL)[6] descriptions.

In general, the purpose of WIDL is to enable automation of all interactions with HTML or XML documents. Because WIDL offers a general method of representing request/response interactions over standard Web protocols, SoftDock can define and manage interfaces for data and services that are not under the direct control of programs that require such access.

Before we release SoftDock to the public—which will happen toward the end of 1999—we will investigate more efficient event notification and document updating mechanisms. We are also considering using SSL for document and communications security. And we plan to build in support for new OMG specifications like XMI and the Meta Object Facility (MOF). Our ultimate goal is to create a highly interoperable and semantics-interchangeable infrastructure.

Currently, though, SoftDock supports the iterative and consistent evolution of software model specifications—particularly for distributed collaborative development—by combining some of the best emerging technologies and standards available today. We believe our work on SoftDock is a blueprint of the next logical step in distributed software development. ❖

Acknowledgments

We thank Ken Ichida, Hiroki Kamata, and Kumiko Nakano for their support.

References

1. J. Suzuki and Y. Yamamoto, "Toward the Interoperable Software Design Models: Quartet of UML, XML, DOM, and CORBA," *Proc. 4th Int'l Software Eng. Standards Symp. (ISESS 99)*, IEEE CS Press, Los Alamitos, Calif., 1999, pp. 163-172.
2. J. Suzuki and Y. Yamamoto, "SoftDock: a Distributed Collaborative Platform for Model-Based Software Development," *Proc. 10th Int'l Workshop Database and Expert Systems Applications (DEXA 99)*, Springer-Verlag, Berlin, Aug. 1999, to appear.
3. J. Suzuki and Y. Yamamoto, "Building a Next-Generation Infrastructure for Agent-based Distance Learning," *Int'l J. Continuing Engineering Education and Life-Long Learning*, Nov. 1999, to appear.
4. J. Suzuki and Y. Yamamoto, "Making UML Models Interoperable with UXF," in L. Bezivin and P-A. Muller, eds., *The Unified Modeling Language: Beyond the Notation*, Springer-Verlag, Berlin, 1999.
5. B.D. Tackett and B.V. Doren, "Process Control for Error-Free Software: A Software Success Story," *IEEE Software*, May/June 1999, pp. 24-29.
6. M.G. Wales, "WIDL: Interface Definition for the Web," *IEEE Internet Computing*, Jan./Feb. 1999, pp. 55-59.

Junichi Suzuki is currently completing a PhD in the Department of Computer and Information Science at Keio University. His research interests include object-oriented development methodology, software patterns and frameworks, distributed object computing, reflection, intelligent user interfaces, agent communication, and computational biology. Contact him at suzuki@yy.cs.keio.ac.jp.

Yoshikazu Yamamoto is an associate professor in the Department of Computer and Information Science at Keio University. His research interests include distributed discrete event simulation and modeling, object-oriented programming, agent programming, intelligent interfaces, and documentation. Contact him at yama@ics.keio.ac.jp.