Distributed Management of Component Framework Specifications

Junichi Suzuki and Yoshikazu Yamamoto

Department of Computer Science,
Graduate School of Science and Technology,
Keio University
Yokohama City, 223-8522, Japan
+81-45-563-3925
{suzuki, yama}@yy.cs.keio.ac.jp

Abstract. *The emergence of the Internet has radically changed how to develop and circulate software. It drives rapid software development using plug-compatible components, and also requires distributed software development allowing developers to collaborate in widely disparate places. This paper describes our SoftDock system that leverages distributed component development. It supports the iterative and consistent evolution of component model specifications by combining several emerging standard technologies. We believe our work shows a blue print of a next logical step in the business component research.*

Keywords. *Business components, Business objects, Distributed software development, Model interchange, Model engineering, UML, XML, DOM, CORBA.*

1 Introduction

The emergence of the Internet has radically changed how to develop and circulate software. The development cycle at "Internet speed" requires less time-to-market and more frequent updates, while maintaining high levels of functionality and quality. The notions of component and business object are the key for this issue. A business component is defined in (Herzum et al., 1998) as follows:

A business component is the IS representation, from requirements analysis through deployment and run-time, of an "autonomous" business concept or process. It consists of all the software artifacts necessary to express, implement and deploy the given autonomous business concept as an equally autonomous, reusable element of a larger information system.

It addresses rapid software development and adaptability to survive in changing environment including technological and business changes.

Also, the current software development projects are becoming more network-centric, because effective and economical communication mediums are available, and because the knowledge and skills required for a large project often cannot be found in one location. The team communication among developers critically impacts the quality of their deliverables in the situation where they spread across geometrically distributed places (Dutoit et al., 1998). The friction of distributed, and therefore delayed, communication typically forces the information overhead in teams, thereby works to slow development. It is highly required to develop components independently of physical places and assemble them.

For the above driving forces and requirements, unfortunately, there are few collaborative platforms to manage rapid evolution of business component models. Such an integrated framework requires several challenges for supporting distributed business component development:

– Modeling components and relationships (dependencies, associations, etc.) between them, which involves the modeling language that defines interchangeable semantics of components.
– Describing and interchanging component model information, which involves the model exchange format.
– Providing a uniform means to access and circulate components, which involves a set of published interfaces and communication protocol.
– Ensuring the integrity of component models, which involves the model tracking and storage facility.

This paper describes our SoftDock system which is a platform for distributed software development (Suzuki, 1999a-1999c). SoftDock supports the model-based component development and management. It allows developers to share and manage component model specifications collaboratively, through requirements analysis to deployment, and then produce software from the model.

Our system addresses the above four issues by combining some standard technologies. It provides a solution to the first issue by using the Unified Modeling Language (UML) (OMG, 1999a), to the second one by providing an application-independent interchange format for UML models, called UXF (UML eXchange Format) (Suzuki, 1998a and 1999d), and to the third and fourth issues by distributing UXF descriptions through DOM (Document Object Model) interface (W3C, 1998a) implemented on top of CORBA (Common Object Request Broker Architecture) (OMG, 1998b).

The SoftDock system provides following benefits:

– *It addresses model-based component development.*
  SoftDock allows developers specify component models independently of implementation technologies such as programming language, network facility and operating system.
– *It leverages model reuse by tool interoperability.*
  SoftDock allows component models to be interchangeable and reusable between different development tools with different strengths, throughout the lifecycle of software development. With this capability, development teams do not have to lock-in any specific tools.
– *It addresses cost-effectiveness.*
  The seamless tool interoperability and model reusability reduces the information overhead between tools, project members or development phases. Also, SoftDock has the potential to reduce cost, i.e. time and money, by incorporating promising open standards. Using promising standards helps risk management.
– *It has the potential to speed up model development.*
  The above characteristics increase the model continuity and improve developers' productivity. SoftDock has great potential to streamline distributed business component development.

The remainder of this paper is organized as follows. Section 2 overviews enabling technologies for SoftDock: UML, XML, UXF, RDF, DOM and CORBA. Section 3 describes the SoftDock architecture and its implementation. We conclude with a note on future work, in Section 4 and 5.

2 Foundation Technologies for SoftDock

This section briefly overviews backgrounds, motivations and benefits of enabling technologies used in SoftDock.

2.1 Unified Modeling Language (UML)

The Unified Modeling Language (UML) is the union of the previous leading object modeling methodologies; Booch, OMT and OOSE. It is a standard object-oriented modeling language that defines most of the semantics and their notations required for representing software constructs. UML provides nearly independent 9 diagrams for modeling a given problem domain in terms of various perspectives. UML has been widely accepted by academic and commercial developers, and used for representing various software models including real-time system models, hypermedia models, business models, engineering design models, multi-agent models, etc.

2.2 Component Model Interchange: UXF and XMI

Model interchange is a quite important capability in software development, because there are few application-neutral exchange format between development tools. The most important factor in interchanging model information is that the semantics within the model should be described explicitly and transferred precisely. To address this issue, we developed UXF (UML eXchange Format), which is an interchange format for UML models (Suzuki, 1998a and 1999d). Object Management Group also completed the XMI (XML Metamodel Interchange) format (OMG, 1999). Both UXF and XMI are based on eXtensible Markup Language (XML) (W3C, 1998b), and serves as a communication vehicle for component model information between development tools or developers (Suzuki, 1998a and 1999d).

UXF is carefully designed to be simple and well-structured enough to encode, publish, access and interchange UML models. We are now using UXF in the SoftDock system, because it preceded XMI at the time of beginning of our project, and because it is much simpler than

XMI. However, we are considering to support XMI as well in SoftDock (see Section 4). Note that due to space limitations, the basics and benefits of using XML as an interchange format are not covered here. Please see (Suzuki, 1999c and 1999d) for more depth discussion.

2.3 Distributed Component Model Interchange: DOM and CORBA

SoftDock manages component model information written in UXF through the Document Object Model (DOM) interface implemented on CORBA (Common Object Request Broker Architecture). DOM defines general-purpose interfaces to manipulate a parsed tree structure of a XML document. It provides a set of APIs for the following capabilities:

- *Structure navigation*, which is the navigation of document structures such as accessing and searching elements or attributes.
- *Structure manipulation*, which is the manipulation of document structures such as adding, changing and removing elements or attributes.
- *Content manipulation*, which is the manipulation of document contents such as putting or getting values to elements and attributes.

The DOM interface is defined with the OMG Interface Definition Language (IDL), a primary component in the CORBA specification (see below), because it is designed to be programming language neutral. DOM does not intend to be implemented with CORBA, but implementing DOM interfaces on CORBA is reasonable strategy for our goal to distribute component models on the network environment (Suzuki, 1999a).

Common Object Request Broker Architecture (CORBA) is a standard for object middleware used in the heterogeneous environment. It provides a standard way to interoperate distributed objects. CORBA defines the interfaces and components that organize Object Request Brokers (ORBs). An application accessing the service of a remote object uses an ORB to send a message and receive the results. A series of interfaces in CORBA allows to distribute remote objects on multiple platforms in a transparent way to applications. One of the key components in CORBA is OMG Interface Definition Language (IDL), a language to define the interface of a remote object. It is programming lan-

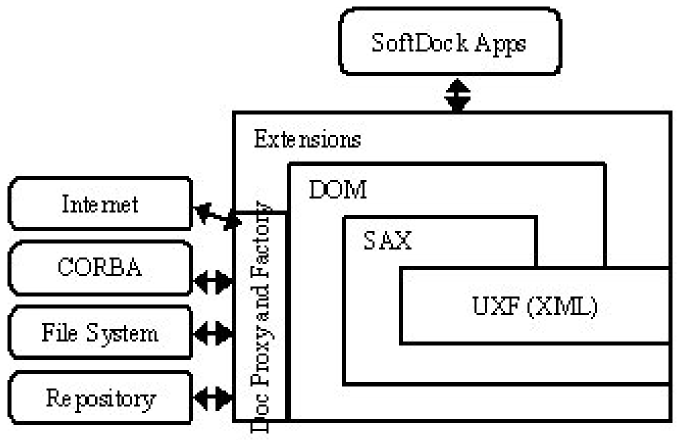

Figure 1: SoftDock Architecture

guage neutral by providing a mapping to various languages. Another important component is Internet Inter-ORB Protocol (IIOP), which is a standard on-the-wire protocol based on TCP. IIOP leverages the interoperability between different ORBs as well as between objects on a single ORB. The benefits from combining DOM and CORBA are described in (Suzuki, 1999c).

2.4 Component Metadata: RDF

Metadata is data about data, or specifically descriptive information about software models, e.g. objects, components, packages, etc., in the context of our system. It is used to index, retrieve, manage, interchange or automate model information. The concept of metadata is becoming important in the web community. The Resource Description Framework (RDF) specification (W3C, 1999) has been completed by the World Wide Web Consortium (W3C). RDF allows us to define metadata of arbitrary web resources including an entire web document, a part of a document, a collection of documents and even an object that is not directly accessible via the web. RDF is an XML-based format.

In the SoftDock system, we use RDF to define metadata of various software models. However, it can integrate any other metadata description languages that are based on XML. For example, we have used the IMS Meta-Data specification (IMS, 1999), which is a metadata description language for educational resources, in order to build a distance learning/training system for teaching software modeling based on SoftDock (Suzuki, 1999c). Section 3.2 shows component metadata is used for describing its non-functional aspects.

3 The SoftDock System

This section describes the foundation architecture, system organization, design strategy, deployment and applications of the SoftDock system.

3.1 Architecture and System Organization

Figure 1 shows the SoftDock architecture, which describes the relationships between different APIs to access component model specifications encoded in UXF. UXF descriptions are manipulated through either DOM or SAX (Simple API for XML) interface (SAX, 1998). SAX is a de-facto standard parser interface, the interface between a XML parser and its applications, which has been developed in the XML community. SAX is an event-based parser interface, while DOM is tree-based (Chang, 1998). DOM compliant parsers often use SAX-based parsers internally. Any parsers supporting either DOM or SAX can be plugged into our architecture without affecting other components in the system. The extension part in the architecture provides a series of utility objects that are useful for building various SoftDock applications (see Figure 1). One of the most important objects in this part is a document proxy. Its responsibilities are:

- Fetching remote model specifications consistently.
- Connecting a parser interface with external environments such as the Internet, CORBA, file systems and repositories.

The design details of document proxy is described in Section 3.3.

Figure 2 shows our current system organization. All the UXF descriptions are stored in the resource server. The SoftDock system provides the connectivity with a web server and CORBA environment. The HTTP connection provides the oneway communication between clients and servers, which aims to allow client applications including web browsers to refer model information within a resource server

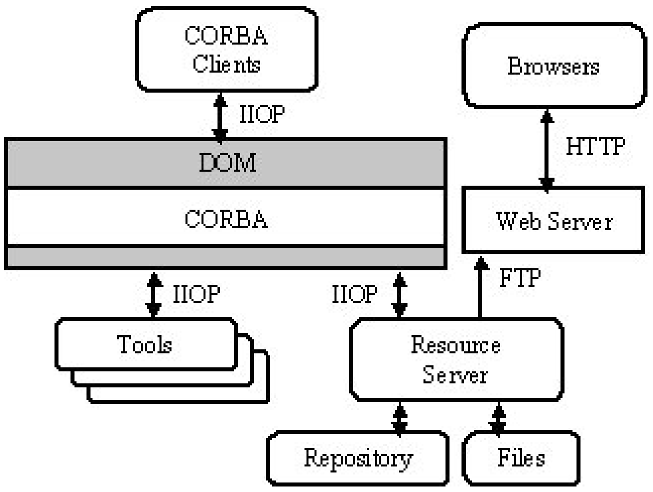

Figure 2: The current SoftDock system organization

through a web server. Whenever a UXF description is updated in resource server, SoftDock pushes the updated description to a web server.

The IIOP connection provides the two-way communication between clients and servers, which aims to allow developers at separated places to refer, create and modify arbitrary model specifications. Client applications include CASE tools, documentation tools, design metrics tools, source code generators, reverse engineering tools.

3.2 Description of Business Components

There is an emerging consensus about business components, apparent in (Herzum et al., 1998 and Herzum, 1998), though there has never been any standard consensus (Sutherland, 1998). It proposes four major dimensions for business components:

– The concept of the business component itself.
– The business component system –the groups of business components that cooperate to deliver the system functionality.
– The internal architecture of the business component.
– The business component system development process.

The SoftDock system supports the categorization and internal structure of business components.

(Herzum et al., 1998) defines the layered categorization of business components:

– *Business process components*, representing business processes and workflows
– *Business entity components,* representing components upon which business processes operate.
– *Utility components*, fine-grained components used by the above components.

A business component at a given level can depend on business components at any of the levels below. SoftDock supports business entity component and utility components by modeling and describing them with UML. Business process components are partially supported by specifying business processes and workflows using UML sequence and activity diagrams. The complete support of business process components is future work, because UML does not have enough model elements to specify business processes and workflows. We are investigating an extension to the UML metamodel for modeling them (see Section 4).

(Herzum et al., 1998) also defines the layered internal structure of each business component: *a presentation layer, a workspace layer* supporting a business transaction, *an enterprise layer* defining business logic, rules and interaction between components, and *a resource layer* supporting transaction, persistence, security and other services. Our system currently supports the description of an enterprise layer.

For the above supports, we extended UXF and introduced the notion of component metadata. UXF is extended to describe the enterprise level of three kinds of components. A sample description is shown in the Appendix. Currently, we suppose all the business components are modeled with UML and UXF, though we are aware the current UML is not sufficient to specify them. What model elements should be prepared for defining business components is beyond the scope of this paper. When further standard consensus or UML extension is emerged, we will plug-in it to the SoftDock system.

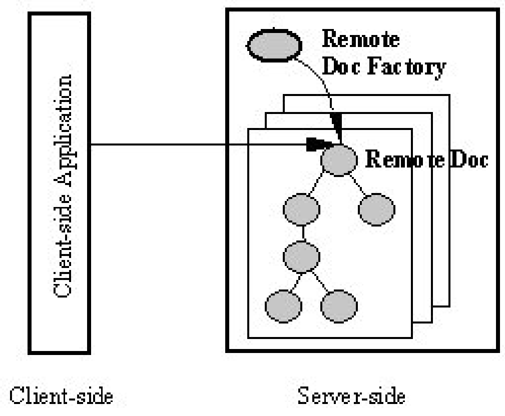

Figure 3: Access to a remote UXF description

Component metadata contains important information to circulate and assemble components. It is described with RDF, and linked to a corresponding component description (see Appendix). Section 3.3 describes how they are transferred and handled in SoftDock.

3.3 Design and Implementation

This section describes design strategies we chose for the IIOP communication subsystem depicted in Figure 2.

3.3.1 Model Management Interfaces

As described earlier, SoftDock allows client applications to access remote UXF-formatted model information through the DOM interface built on CORBA. Every UXF description is parsed into a tree structure according to its tag hierarchy with a DOM-compliant XML parser, and maintained as a set of CORBA objects (Figure 3). The CORBA objects are registered into the Basic Object Adapter (BOA), which is a server-side facility used for managing their lifecycle and dispatching incoming requests to them. Every root node (or object) of tree structures is published to client applications using the CORBA naming service, a white page service for CORBA objects. Client applications locate the root

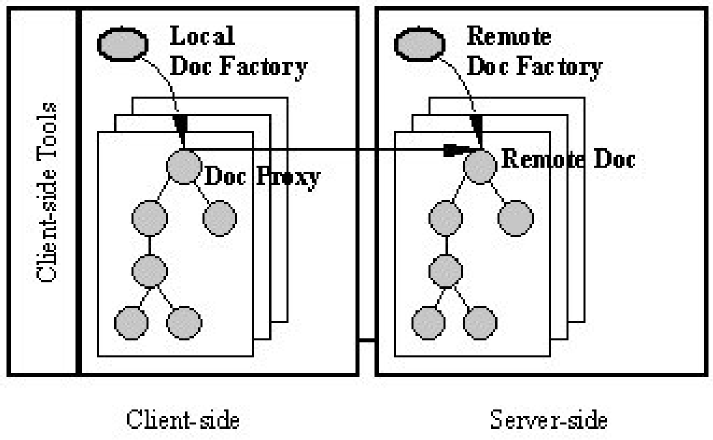

Figure 4: Local cache of a remote UXF description

node of a target UXF description and then navigate its structure and contents.

This design strategy is straightforward, but it does not scale well in the situation where the size of a single UXF document is large or it has many tags. It raises object management problem and increased overhead problem. The object management problem means that an underlying ORB (or BOA, specifically) have to register huge amount of objects potentially. This is because a UXF tag is mapped to a DOM node, which is in turn mapped to a CORBA object. The overhead problem means that a remote method invocation is always required when referring to and/or changing any node in a remote UXF description.

To avoid the overhead problem, SoftDock allows a client application to cache remote UXF documents locally (Figure 4). It provides the IDL interface extending the DOM APIs for fetching and maintaining remote UXF descriptions at client-side. Figure 5 shows three extended interfaces: `UXFDescription`, `CorbaDocAgent` and `CorbaDocProxy`.

```
#pragma prefix "jp.ac.keio.SoftDock"
#include <dom.idl>
#include <CosEventComm.idl>
module SoftDockExtention
{
  typedef long UID;
  interface UXFDescription
    :dom::Document,
    :CosEventComm::TypedPushConsumer,
    :CosEventComm::TypedPushSupplier
  {
    attribute UID nodeId;
    readonly attribute float revision;
    void externalize();
    sequence<string> content();
    boolean isLocked();
    oneway void lockNode(in CorbaDocProxy proxy);
    oneway void releaseNode();
    void update(in string docName,
                in UID    changePoint,
                in short  changeTypeId);
  };
  interface CorbaDocProxy
    :UXFDescripion,
  {
    attribute dom::Document remoteDoc;
    oneway metadata(in sequence<string> mdata);
  };
  interface CorbaDocFactory
  {
    dom::Document createDocument(in string docName);
    dom::Document cloneDocument(in UXFDescriiption doc);
    oneway void releaseNode(in UXFDescriiption node);
    void destroyDocument(in UXFDescriiption doc);
  };
};
```

Figure 5: Extended IDL definition for SoftDock

`UXFDescription` represents a server-side root node of an entire UXF description. It is derived from `dom::Document` defined in the DOM specification, which provides the methods for accessing to the document's data and creating its internal elements. `UXFDescription` is also derived from `CosEventComm::PushSupplier` and `CosEventComm::PushConsumer`, which are the interfaces for push-style event notification in the CORBA event service. These interfaces are used to notify the change events between a remote UXF description and its local cache. The document change notification is described Section 3.3.2. `UXFDescription` has an attribute `revision` to track its revision number and exposes five methods. `externalize()` translates a document's tree structure into a file, and `content()` returns its representation as a sequence of string data. `lockNode()` and `releaseNode()` are used to require and release a lock of the document

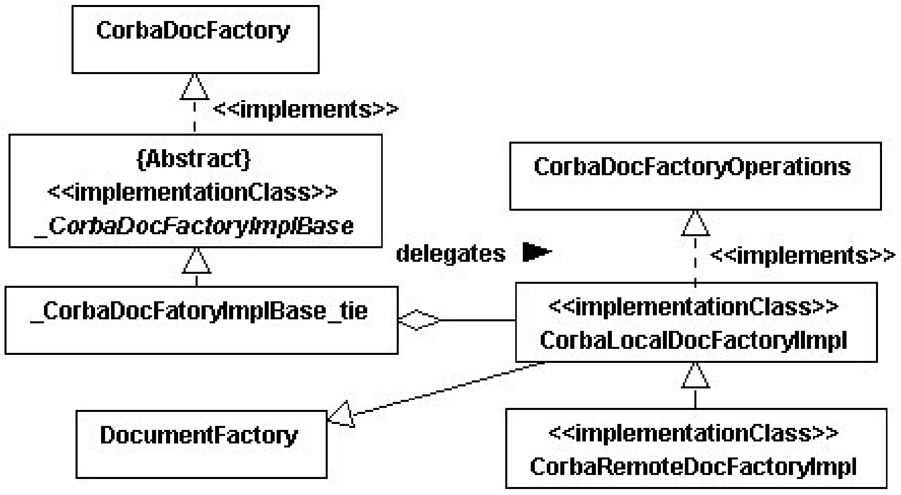

Figure 6: Class structure of the `CorbaDocFactory` interface and its implementations

passed as an argument. `update()` is used to notify any change of a UXF description (see Section 3.3.2).

The `CorbaDocProxy` interface is a client-side proxy for a remote UXF description, shown in Figure 4. This interface is derived from `UXFDescription`. It has references to root nodes of both a remote document and its local cache. The method `metadata()` transfers component metadata (see also Section 3.2).

The `CorbaDocFactory` interface represents the local and remote factory for a UXF description. The DOM specification does not define the way of creating an XML document instance, and therefore this interface provides factory methods for creating, cloning and destroying a document.

Figure 6 shows the server-side class structure of `CorbaDocFactory` and its implementations, which are generated by an IDL-Java compiler. `CorbaDocFactory` in Figure 6 is a Java interface class, which is a Java-side equivalent of the IDL `CorbaDocFactory` interface. `_CorbaDocFactoryImplBase` and `_CorbaDocFactoryImplBase_tie` are skeleton classes for `CorbaDocFactory`, which unmarshal parameters contained in method invocations. We are using the *tie approach* in which a skeleton

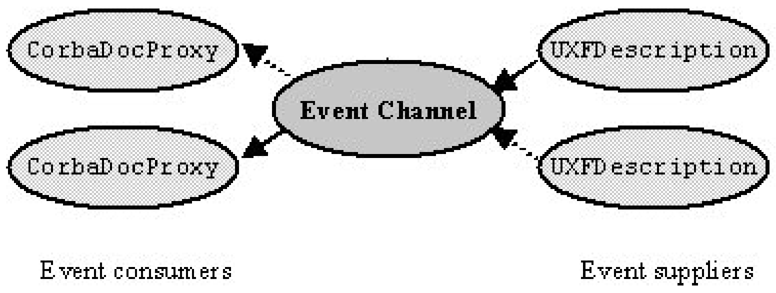

Figure 7: Change notification through an event channel

asks for its delegate class to perform method invocations, instead to perform them by itself. `_CorbaDocFactoryImplBase_tie` delegates method executions to `CorbaLocalDocFactoryImpl` and `CorbaRemoteDocFactoryImpl`. These classes implement the `CorbaDocFactory` interface. They are the client-side and server-side factory classes for a UXF description respectively (see Figure 4).

3.3.2 Change Notification

When a server-side UXF description is updated, the change is notified to the corresponding `CorbaDocProxy` at client-side. This notification is performed by calling the `CorbaDocProxy`'s `update()` method. SoftDock transfers typed events through event channel based on the push model.

Event data is passed by means of the parameters defined in `update()`. The parameter `changeTypeId` shows what kind of change is occurred in an original description at server-side. The kinds of changes include copy, movement, insertion, deletion and modification of a node. An event channel is an intermediate object that allows multiple event suppliers to communicate with multiple event consumers in the decoupled manner. An event channel is both a consumer and supplier of events. It is a standard CORBA object, and the communication with an event channel is accomplished using normal CORBA requests. In SoftDock, `UXFDescription` transfers a push-style change notification to an event channel, and then the event channel pushes the change to `CorbaDocProxy` (Figure 7).

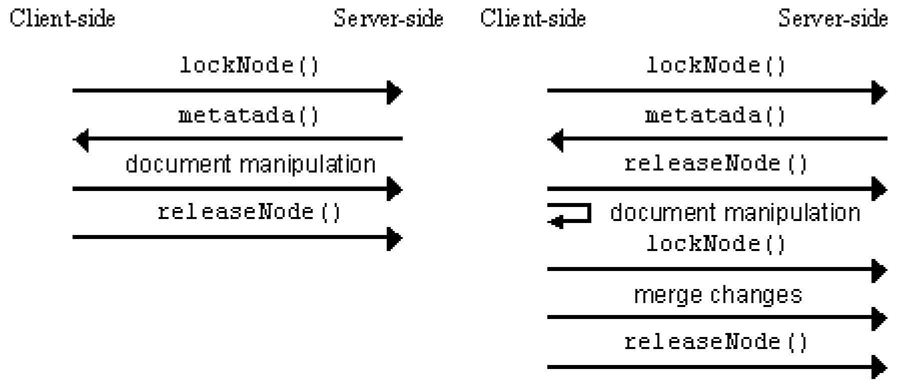

Figure 8: Synchronous (left) and Asynchronous interactions between client and server (right)

3.3.3 Synchronous and Asynchronous Accesses

The caching mechanism described above allows SoftDock to support both synchronous and asynchronous manipulation of a UXF description.

The synchronous editing is performed as shown in the left of Figure 8. A client-side `CorbaDocProxy` locks a remote document using its method `lockNode()` to prevent modifications by other applications. Once the document is locked, its metadata is transferred to its `CorbaDocProxy`. Then, it accesses and manipulates the remote document based on the obtained metadata. After editing the remote documents, `CorbaDocProxy` releases the lock with its method `releaseNode()`. The synchronous editing is a simple model and ensures the document's consistency by preventing overwrites. However, it does not scale well in the situation where many learners are accessing a remote document frequently because every access requires a lock even if the document is not changed.

In contrast to the exclusive lock, our system provides an alternative lock, *shared lock*, which allows a group of applications to share a single lock on a document. This lock enables the asynchronous remote editing. Users can continue to work even when they are offline, e.g. using mobile computers, and merge changes later. Merging a change to the original description, a client application uses the method `update()` of

`UXFDescription`. The asynchronous editing can reduce the number of remote method invocations. The right of Figure 8 shows the interactions between client and server in the asynchronous mode. This mode works best in environments in which participants know each other's activities, or just refer the remote document.

3.4 Deployment

SoftDock has been deployed with:

- DOM compliant XML parser: We have used XML4J (IBM), and are replacing it with our own parser.
- CORBA compliant ORB: We are using two different ORBs, ORBacus (OOC) and Fnorb (DSTC), which are Java-based and Python-based ORBs respectively. The IIOP protocol ensures the interoperability between these ORBs.
- Web server: Our in-house web server, OpenWebServer (Suzuki et al., 1998b, 1999e and 1999f), is used for the HTTP connection (see Section 2).
- Programming languages: CORBA server applications are written with Java, while its client applications are with Java and Python. Our web server is developed with Java.
- Operating systems: CORBA server applications are run on Linux and Windows NT, while its client applications are on Windows 95, Windows NT.

3.5 Applications

We have interchanged some simple component models in SoftDock, e.g. CurrencyBook component, which is designed based on the OMG Finance DTF's Currency specification (OMG, 1998c), Invoice component, Customer component and InvoiceManager component.

SoftDock has a command-line and graphical (Java applet-based and Tk-based) model reference/revision tools. Also, we have developed model documentation tools and connectivity glues for commercial CASE tools. These tools fetch a UXF description from a resource server and translate it into HTML, RTF, PDF, PostScript or proprietary formats used in Rational Rose and MagicDraw.

4 Future Work

We are extending our system from several viewpoints.

As described in Section 3.2, our work has not addressed how business components should be modeled. SoftDock incorporates a minimum extension to UML and UXF. We are investigating some proposals for metamodel extensions and techniques for specifying business processes and workflow with UML (Bock, 1998, Wiegart, 1998 and Hruby, 1998). We plan to build some prototype applications for testing their feasibility.

We are adding some capabilities to SoftDock, which allows more effective distributed model management. The first one is a transparent communication between a remote UXF description and its local cache to whether a target description is cached in client-side, while client applications currently have to know that in advance. The second capability is a yellow page service for each root node of UXF descriptions. It provides more semantics-aware search mechanism than the current white page service. We are now designing it with the CORBA trading service. The third one is secure communication facility. We are considering to use IIOP over SSL. The fourth one is adding a different locking scopes to a UXF description, e.g. a lock for a collection of documents. We are also analyzing the system behavior in the asynchronous editing mode, and investigating the consistent distributed document management.

We also plan to align some other specifications such as the Document Repository Integration by the OMG Business Object DTF (OMG, 1998c), the XMI format (OMG, 1999b), and the Meta Object Facility (MOF) specification (OMG, 1997) for evaluating the possibility and implications of a highly interoperable and semantics-interchangeable infrastructure.

5 Conclusion

This paper describes our SoftDock system that leverages distributed business component development. It supports the iterative and consistent evolution of component model specifications by combining some emerging standard technologies such as UML, XML, DOM and CORBA. We describe our system architecture, design strategies, techniques and implications for managing component models effectively in the distributed environment. We believe our work shows a blue print of a next logical step in the business component research.

Appendix Sample Description of a Component and its Metadata

```
<UXF Version="2.0"
     xmlns:UXF="http://www.yy.cs.keio.ac.jp/~suzuki/project/uxf/">
  <!-- Component description -- >
  <UXF:BusinessComponent>
    <UXF:Name>Invoice</UXF:Name>
    <EnterpriseLevel>
      ...
    </EnterpriseLevel>
    ...
  </UXF:BusinessComponent>
  ...
</UXF>
<!-- Component metadata definition -->
<RDF
  xmlns="http://www.yy.cs.keio.ac.jp/~suzuki/RDF"
  xmlns:UXF="http://www.yy.cs.keio.ac.jp/~suzuki/project/uxf/">
  <Description about="http://www.yy.cs.keio.ac.jp/SoftDock/invoice.uxf">
    <UXF:ComponentCategory>
      Entity
    <UXF:ComponentCategory>
    <UXF:Dependency>
      <UXF:BusinessComponent>
        <Name>CurrencyBook</Name>
      </UXF:BusinessComponent>
      ...
    </UXF:Dependency>
    <UXF:Author>
      <UXF:Name>
      ...
    </UXF:Author>
    <UXF:Date>
      ...
    </UXF:Date>
    <UXF:Version>
      ...
    </UXF:Version>
    ...
  </Description>
</RDF>
```

References

[Bock, 1998] C. Bock. *Suggested Revisions to Activity Models for Business Process Modeling.* OMG document number ad/98-06-13, available at uml.systemhouse.mci.com/artifacts.htm, 1998.

[Chang, 1998] D. Chang and D. Harkey. *Client/Server Data Access with Java and XML*. Wiley, 1998.

[Dutoit et al., 1998] A. H. Dutoit and B. Bruegge. *Communication Metrics for Software Development*. In IEEE Trans. On Software Engineering vol. 24, no. 8, August 1998.

[DSTC] CRC for Distributed Systems Technology. *Fnorb ORB*. available at www.dstc.edu.au/Fnorb/

[Herzum et al., 1998] P. Herzum and O. Sims. *The Business Component Approach*. In the Proceedings of the OOPSLA'98 Workshop on Business Object Design and Implementation, 1998.

[Herzum, 1998] P. Herzum. *The Business Object Component*. OMG document, ormsc/98-09-01.

[Hruby, 1998] P. Hruby. Structuring Specification of Business Systems with UML (with an Emphasis on Workflow Management. In Proceedings of OOPSLA'98 Workshop on Business Object Design and Implementation, 1998.

Systems)

[IBM] IBM. *XML parser for Java*. available at www.alphaworks.ibm.com/tech/xml4j.

[IMS, 1999] IMS Project. *IMS Meta-Data Specification, version 1.02*. February 1999, available at www.imsproject.org/

[OOC] Object-Oriented Concepts, Inc. *ORBacus for Java*. available at www.ooc.com/ob/.

[SAX, 1998] D. Megginson. *SAX 1.0: The Simple API for XML*. available at www.megginson.com/SAX/.

[Sutherland, 1998] J. Sutherland. OOPSLA'98 Workshop Report on Business Object Design and Implementation IV: From Business Objects to Complex Adaptive Systems. 1998, available at www.jeffsutherland.org/oopsla98/.

[Suzuki et al., 1998a] J. Suzuki and Y. Yamamoto. *Managing the Software Design Documents with XML*. In Proceedings of the 16th ACM Annual International Conference of Computer Documentation (SIGDOC '98), pages 127–136, Quebec City Canada, September 1998.

[Suzuki et al., 1998b] J. Suzuki and Y. Yamamoto. *Building an Adaptive Web Server with a Meta-architecture: AISF approach*. In Proceedings of SPA'98, Kusatsu, Japan, March 1998.

[Suzuki et al., 1999a] J. Suzuki and Y. Yamamoto. *Toward the Interoperable Software Models: Quartet of UML, XML, DOM and CORBA*. In the 4th IEEE International Software Engineering Standards Symposium (ISESS '99), pages 163–172, Curitiba, Brazil, May 1999.

[Suzuki et al., 1999b] Junichi Suzuki and Yoshikazu Yamamoto. *SoftDock: a Distributed Collaborative Platform for Model-based Software Development*. In the 10th International Workshop on Database and Expert Systems Applications (DEXA '99), Florence, Italy, September 1999, to appear.

[Suzuki et al., 1999c] J. Suzuki and Y. Yamamoto. *Building A Next-Generation Infrastructure for Agent-based Distance Learning*. In the International Journal of Continuing Engineering Education and Life-Long Learning, November 1999, to appear.

[Suzuki et al., 1999d] J. Suzuki and Y. Yamamoto. *Making UML Models Interoperable with UXF*. In P. Muller and J. Bezivin, editors, Unified Modeling Language, LNCS 1618, 1999, to appear.

[Suzuki et al., 1999e] J. Suzuki and Y. Yamamoto. *OpenWebServer: an Adaptive Web Server Using Software Patterns*. In IEEE Communications Magazine, Vol.37, No.4, pp. 46 - 52, April 1999.

[Suzuki et al., 1999f] J. Suzuki and Y. Yamamoto. *Dynamic Adaptation in the Web Server Design Space using OpenWebServer*. In Proceedings of SPA '99, Atagawa, Japan, March 1999.

[OMG, 1997] Object Management Group. *Meta Object Facility Specification*. OMG document ad/97-08-14, ad/97-08-15 and ad/97-09-04, available at www.omg.org/techprocess/meetings/schedule/Technology_Adoptions.html, 1997.

[OMG, 1998] Object Management Group. *Common Object Request Broker Architecture 2.2*. available at www.omg.org/library/c2indx.html, February 1998.

[OMG, 1998b] Object Management Group. *Currency Specification*. available at www.omg.org/corba/cfinchp.html, 1998.

[OMG, 1998c] Object Management Group. *Document. Repository Integration RFP*. OMG document dtc/98-09-01, 1998.

[OMG, 1999a] Object Management Group. *Unified Modeling Language Specification*. version 1.3R5, available at uml.systemhouse.mci.com, 1999.

[OMG, 1999b] Object Management Group. *XML Metadata Interchange (XMI) Specification*. OMG document, ad/98-10-05 and ad/98-10-06, available at uml.shl.com/xml/xmi.htm, 1999.

[W3C, 1998a] World Wide Web Consortium, M. Champion et. al., editors. *Document Object Model Level 1 Specification.* August 1998, available at www.w3.org/TR/REC-DOM-Level-1/.

[W3C, 1998b] World Wide Web Consortium, Tim Bray et. al., editors. *Extensible Markup Language (XML) 1.0*. February 1998, available at www.w3.org/TR/1998/REC-xml-19980210.

[W3C, 1999] World Wide Web Consortium, O. Lassila and R. R. Swick, editors. *Resource Description Framework (RDF) Model and Syntax Specification*. February 1999, at www.w3.org/Press/1999/RDF-REC.

[Wiegart, 1998] O. Wiegart. *Business Process Modeling & Workflow Definition with UML*. OMG document number ad/98-04-04, available at uml.systemhouse.mci.com/artifacts.htm1998.