A Component-Based Approach to Object-Oriented Distributed Application Software Development

Stephen S. Yau and Bing Xia
Computer Science and Engineering Department
Arizona State University
Tempe, AZ 85287, U.S.A.
{yau, xia}@asu.edu

Abstract

Component-based software development would allow application software to be largely constructed from existing software components. However, it faces many barriers in component integration, including programming languages, operating systems, communication mechanism, interface, etc. In this paper, an approach to cross-platform and cross-language object-oriented distributed application software development through distributed component integration based on Distributed Object Environment is presented. A distributed component model is developed to facilitate easy information retrieval at integration time. By dynamically generating adapters for distributed components using an integration tool, the component connecting process can be transparent to both component developers and application software developers for integration. Group adapters are also developed for replicated component groups to automatically maintain state consistency and active group membership and provide fault-tolerance feature in the resulting application software.

Keywords: Object-oriented component-based software development, integration, distributed system, fault-tolerance, distributed object environment.

1. Introduction

Component-based software is a desirable concept in constructing large-scale application software [1]. By reusing well-designed software parts to construct application software, the productivity of software development can be dramatically improved. On the other hand, the continuous needs to upgrade and reintegrate existing software systems can be achieved by replacing some parts of the system with new components with compliant interfaces. As the object-oriented approach to software development becomes more mature [1][2], it helps the software developers solve some problems in component-based application software development, such as component decomposition and interface definition.

Recently, two new component specifications, Sun's JavaBeans [4] and Microsoft's ActiveX Control [3][5], expand the reuse paradigm to a wider range of software development, such as compound documentation and Internet web services [3]. But, the significant problem that the chosen parts do not fit well together still remains [6] because of various barriers of integration, such as programming languages, operating systems, communication mechanisms, interfaces, etc. In distributed systems, these barriers are more serious due to the heterogeneity of the software running environments and hardware platforms.

In order to overcome these barriers, we need to consider three main aspects of distributed component-based software development:

- **Distributed component model**

A distributed component model describes the common interfaces of distributed components so that the information of these components can be easily retrieved at the time of software development.

- **Distributed component integration**

After software component development, we need to connect these components into an application software system. This process includes retrieving component's information, transparently generating glue code according to the interaction with software developers and tailoring distributed components to be integrated in the application software systems. More aspects need to be considered for additional features of the software specification, such as fault-tolerance or real-time.

- **Running environment for component-based distributed software**

We need a common distributed object environment to encapsulate the features of the distributed application software system, such as different operating systems, networks, and programming languages, from component developers. There are two popular distributed object environments: Microsoft's Distributed Component Object Model (DCOM) [5] and OMG's CORBA [7].

In our approach, we will use our Distributed Component Architecture in [9] which will be summarized in Section 2 for the sake of completeness. In Section 3, we will present our approach to integration, including a distributed component integration tool, component adapter and state consistency maintenance with group component adapters. In Section 4, we will discuss the implementation issues.

2. Distributed component architecture

A distributed software component is a software piece which offers a set of services over a distributed system through its self-describing interfaces that defined in a common representation language and its interface for reconfiguring some of the properties at assembly time to fit into a specific application software system. In our distributed component model, we use Distributed Object Environment as the integration bus because they offer a consistent distributed programming and run-time environment over most commonly used programming languages, operating systems and networks. Their Interface Definition Language (IDL) is suitable for specifying component interfaces without implementation details. A component can be programmed in a variety of languages on different operating systems. If a component implements a common set of IDL interfaces, it can interact with any other components through the common object environment daemon running on its native platform.

Each distributed component can be considered as an individual software part that implements component's common functions. All these common functions are defined in two related interfaces that together form a distributed component architecture as shown in Figure 1. This architecture can be extended for other features by adding more interfaces like Real-time Requirement Interface for real-time component-based software [9].

There are two basic interfaces for a distributed real-time component:

2.1 Distributed component common interfaces

In order to offer a unified method for integration between distributed components with IDL language, a Distributed Component Common Interface is defined using IDL. To support distributed integration, this common interface includes the following four types of services:

- Interface publishing and discovery

In the component common interface, we define public methods that can return service names and their invocation interfaces upon being retrieved by an integration tool at the time of distributed component

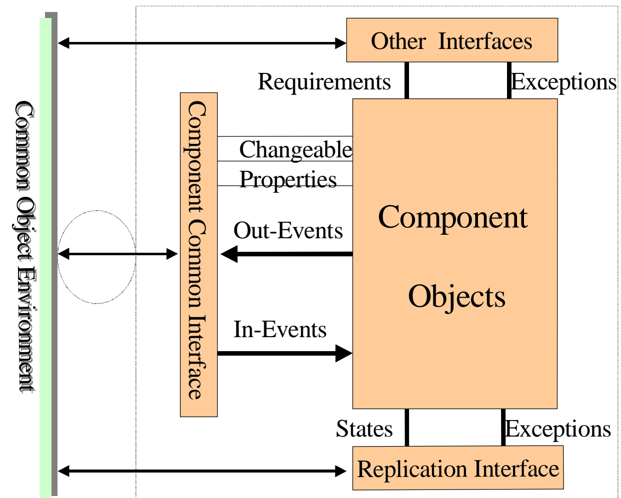

Figure 1: The architecture of a Distributed Component

composition. To find a proper component that offers the needed services, only syntax compatibility is not enough. It happens all the time that two methods or events with similar names perform very different tasks. Hence, additional semantic description about these public services should be part of retrievable information for a distributed component.

- Event handling

This interface allows the related components to react to remote events as well as local events when they are happening on different machines.

- Persistent state maintenance

Persistent state maintenance interface allows distributed components to store their states when the components become inactive and to restore the states before they are reactivated to offer consistent services through unified persistence functions.

- Changeable properties support

This interface allows some attributes of a component to be reassigned by the integration tool at the time of software composition from components so that the component can act according to the new values to fit better into the current software.

2.2 Component replication interface

In order to offer fault-tolerance feature in distributed component-based software, a component can implement additional functions to maintain a replication component group to enhance the availability and provide a certain degree of fault tolerance [9]. A critical service can be offered by several replicated components that reside in different hosts or processes. The replication interface will allow the replicated component group to initiate a global search-and-invoke when a request invokes their services. When some of the replicas that offer

the services are down, the others can still be found and respond to the request.

A key problem that needs be solved in distributed component replication is how to keep the state consistency among the replica [10]. The necessary functions that each replicable component needs to be implemented are defined in IDL format as Distributed Component Replication Interface, which includes the following two basic services:

– **Run-time component state retrieval and update**

These functions will combine component state variables into a predefined format (e.g. string stream) and the receiver of this state information will update its internal variables accordingly.

– **Exception handling**

When one or several members of a replicated component group are down because of process crash or network connection being broken, its exception generating method will invoke other members' exception handling methods for member states update.

3. Our approach to distributed component integration

The goal of defining distributed component interfaces is to support distributed real-time application software development through component integration. The integration process is shown in Figure 2. A distributed component can be developed by wrapping a legacy software unit with the Distributed Component Common Interface. In addition, if the software unit also implements the Component Replication Interface, it has the capability to work as a group member. If it also implements the Real-time Specification Interface, it has the information and capability for real-time method invocations.

The next step is to compose distributed software from independently developed components. The integration is accomplished by an interactive process based on an integration tool that understands these component interfaces and has the capability to 'glue' and 'tailor' these separate components into a system.

3.1 Distributed component integration tool

This component integration tool can interactively put the components together and automatically generate skeleton and repetitive code for the components. Microsoft's Development Studio and Sun's Visual Studio are such environments using visual programming, which allow software developers to visually compose application software using existing software components [10]. Visual

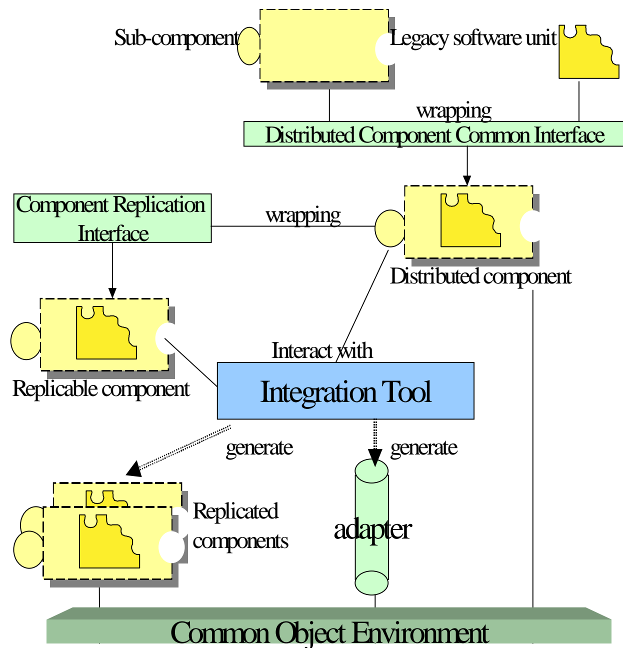

Figure 2: Our integration process of distributed components

Programming enables rapid prototyping and rapid software development.

Traditional integration environments do not support distributed component integration. In this section, we will present our visual integration tool with transparent glue-code generation to facilitate integration and replication of distributed components. Its basic functions are:

- **Visualize the distributed components and their interfaces**

The integration tool allows the software developer to open the components that can be retrieved over the common object environment to construct a whole software structure using Drag-and-Drop.

- **Generate component adapter**

The integration tool generates an adapter for each component to link distributed methods using Distributed Component Common Interface based on user interaction. The link information is not available when a component is built. Only at integration stage, the developer specifies source and target methods of connections. The adapter stores this information for the component and establishes remote method invocations through the common object environment at run-time.

- **Modify properties**

The tool lists the public changeable properties of a component and allows the developer to modify their values so that a versatile component can be tailored to a specific software system.

- **Generate group adapters**

To keep state consistency maintenance among replicated distributed components transparent to developers, we use special group adapters with extra negotiation methods besides regular components. At the time of integration, the integration tool assigns the membership list and the prime membership ID into each group adapter's changeable attributes. Upon receiving remote requests, a group adapter will forward these requests to the prime member who will then broadcast to the others. All members of the group respond in the same way and therefore update their states accordingly while only the prime component will be allowed to send out remote requests through its group adapter if the current request triggers other components. If a member fails to contact the prime host, it will broadcast exception to all the remaining members using Distributed Component Replication Interface, negotiate for a new prime replica, retrieve and re-establish state consistency when a new member joins as a new backup component.

3.2 Component adapter

In traditional distributed application software development, clients are connected to servers using communication API like sockets. This requires the clients to know the exact network addresses of servers. This restricts the dynamic features that a system can offer. In current Distributed Object Environment, like CORBA and DCOM, the situation is improved by introducing Naming Service: a client uses a name string to locate and connect to a server and this is called *location transparency*. But, for distributed components, it is still difficult to require the components to know the names of servers at the time of component implementation. Usually, this information is available when software developer connects components' outgoing methods to server components' incoming interfaces.

In our approach, we attach a component adapter to each component that has outgoing invocation/methods. The adapter will record the connection information and redirect the remote method invocations/events from the component to the outside target components accordingly. In current CORBA, public methods defined in an interface can only be connected to a server component that implements the exactly same interface. By using component adapters, we can overcome this limitation to that every outgoing method of distributed components is connected to different server components with different interfaces, as long as these methods have the same signature of incoming method or event. This will give more flexibility to component designers and allow more existing components to be reused.

There are two basic functions of a component adapter:

- **Redirect invocations/events to developer-defined server components**.

This relay process is described in Figure 3. Each component and its adapter are assigned a unique name, like 'SC', 'TC' and 'A1' by the integration tool. Through Distributed Component Common Interface, the integration tool informs the component of its adapter's ID, so that the component will direct all the remote invocations and events to the adapter. Every time when the software developer connects an outgoing method to other server components, the integration tool will insert a connection record (outgoing_method → target_server × incoming_method) into the connection table in the component's adapter. At run time, when the adapter receives a method invocations/events, it will forward the event to the target server's proper method by looking up its connection table. The adapter is only visible to its serving component. The other part of the distributed system can only see and access the component directly. Hence, only those components that have outgoing methods have adapters.

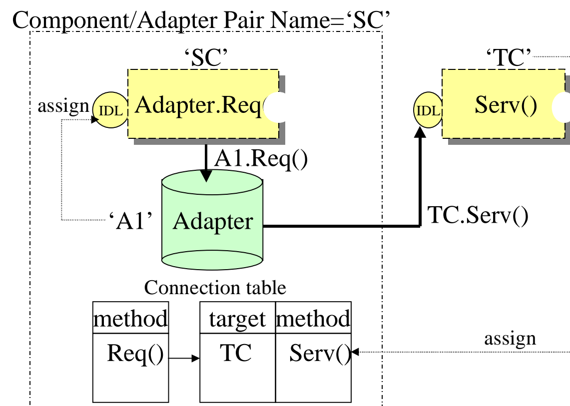

Figure 3: The connection between two distributed components through adapter

- **Re-establish connections with other components in case of temporary communication errors or server relocation, or generate exceptions when connections are lost.**

When the server components are down or a server component is re-allocated to another machine or substituted by a new component during online upgrade, the next invocation to the server will fail. The adapter will try to re-establish the connection by repeating the whole server location process. If it is successful, the component will continue its operations without even noticing this temporary failure. If there is no response, the component's

exception handling function will be called to allow the component to take correspondent actions.

3.3 State consistency of replicated component group

As we mentioned in Section 2, we use replicated component group to provide fault-tolerance to the application software against processes crash and network failure. A very important problem for the component group is to maintain the state consistency between these replicas. One way to do this is to let the component designers add the group maintaining code at the time of design which will increase dramatically the development effort and make the components less flexibility in reuse. In our approach, we offer transparent automatic state consistency maintenance to the replicable components through group adapters.

Group adapter is a superset of general component adapter as illustrated in the previous section. Besides the functions of event redirection and reconnection, it has the knowledge and capability to negotiate with all the other members of the group to keep the whole group in the same states and to reconstruct the group in case some members crash. Figure 4 shows the relationship of group adapters and their replicated component groups. The integration tool assigns the same name to all the group adapters, like ‘GA2’, and only distinguishes them using different sub-names like ‘GA2.1’ and ‘GA2.2’. The group adapters fully represent their components. The components are only visible to their group adapters. The other part of the distributed system can only see and connect to the group adapters using the shared main name, like ‘GA2’.

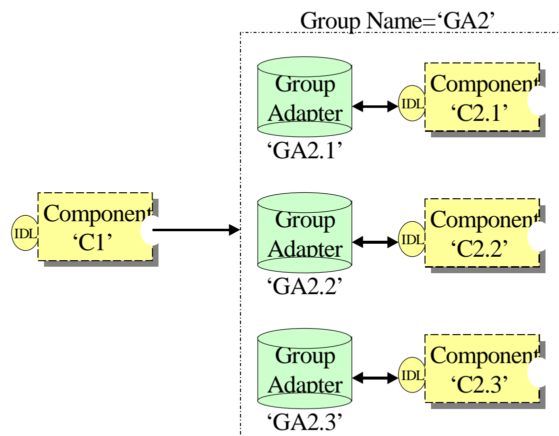

Figure 4: Connections between a group of replicated components and other components through group adapters

Inside the group, every group adapter has a list of all the group members’ IDs (names and sub-names) from the integration tool. The integration tool will also assign a prime group adapter during software composition. When anyone of the group adapters receives a request, the adapter will forward it to the prime adapter who will broadcast this request according to its membership list. Then, every member sends acknowledgement to the prime adapter. The request will be passed to the component only after the prime adapter confirms that all members have received it. Each component starts from the same state and handles the same events so that their states will stay consistent. In order to avoid multiple actions to be invoked by one request, only the prime group adapter is allowed to redirect the outgoing events from the component to the outside of the group. Figure 5 illustrates this automatic state consistency maintenance mechanism.

During the broadcast of a request or confirmation, if some group adapters or their components cannot be reached, the group will enter fault-tolerance process. The group will negotiate to update their active group membership list or elect a new prime adapter if necessary to exclude the crashed components and their adapters. When the crashed or new components join the group, it will bind to the current prime adapter for membership list and retrieve/update the state from the prime component through Replicated Component Interface.

4. Implementation

We have implemented the distributed component framework and the integration tool. We are currently developing associated supporting adapter libraries based on IONA’s Orbix 2.2 CORBA environment. This prototype is used to present and test the capability and efficiency of distributed component-based application software development through cross-language (C, C++, Java) and cross-platform (Windows 95/NT, Solaris) component integration.

The distributed component interfaces are defined in IDL 2.0. They can be translated into C/C++ on either Solaris or Windows 95/NT using Orbix 2.2’s IDL compiler. We also use IONA’s Java version CORBA, OrbixWeb 3.0, to generate Java code from these interfaces defined in IDL [9].

Our integration tool, which is being developed on both Windows NT and Solaris, is a distributed software based on Orbix 2.2. In order to achieve portability, we use Java to develop the GUI interface. The tool retrieves methods and events that are defined in distributed components from Interface Repository, and displays them as icons under a component structure tree. Hence, software developer can drag these icons to a working area and modify

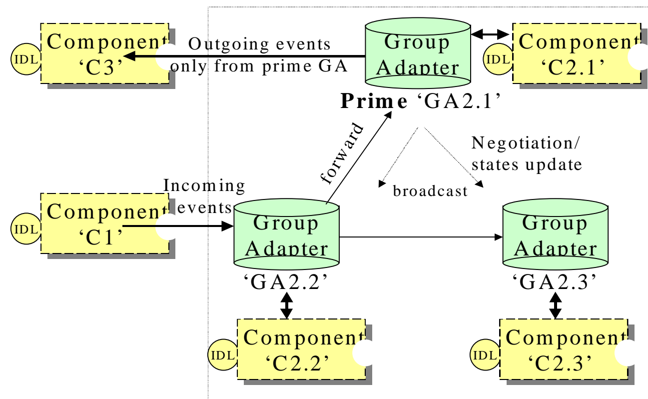

Figure 5: State consistency maintenance through group communication between group adapters

their public attributes and select a proper remote handler for the outgoing events of the components.

During integration, the integration tool needs to dynamically generate the adapters for components with different interfaces. We develop the adapter itself as a distributed component with changeable attributes like Connection_Table and Group_Member_List. When the software developer makes connection between components, the integration tool can adjust these public attributes to configure and tailor the general adapter component into the distributed software system. In order to allow the adapter to connect to different server components across these interfaces, we use CORBA's Dynamic Skeleton Interface (DSI) [7][8] to accept all the requests from its own component and pass them on to the corresponding targets using Dynamic Invocation Interface (DII) [7][8].

In a replicated component group, all the group adapters of the members share the same service name and are distinguished only by different *markers* [9] assigned by the integration tool. As long as one of the member components and its adapter are working, the client still can access the services through name binding service by binding the group name, instead of the name of individual component/adapter pair.

5. Discussions

In this paper, we have presented an approach to object-oriented distributed component software development. Based on the distributed component architecture we have defined, we have developed the integration process of distributed software and the use of component adapter in connecting components' methods/events across different interfaces of the components. In order to have fault-tolerance feature to component-based software, an automatic state consistency maintenance mechanism through replication components' group adapters is introduced. To expedite distributed component-based application software development, we have developed a CORBA-based integration tool with transparent adapter/group adapter generation for regular and replicated components based on this approach. We have developed several distributed system demonstrations using this approach and achieved high productivity. But, same as other component based approaches, our distributed component-based integration introduces some overhead for additional request relay and group communication between replicated components comparing to directly well-designed distributed software. In the future, we plan to introduce system resource management in the component adapter mechanism and extend the distributed component architecture to handle real-time requests over real-time communication service, like Real-time CORBA [11].

References

[1] O. Nierstrasz, S. Gibbs and D. Tsichritzis, "Component-Oriented Software Development", *Comm. ACM*, Vol.35 No. 9, September 1992, pp.160-164.

[2] S. S. Yau, D. H. Bae and K. Yeom, "Object-oriented Development of Architecture Transparent Software for Distributed Parallel Systems", *Jour. Computer Communication*, Vol.16 No.5, May 1993, pp. 317-327.

[3] J. Montgomery, "Distributing Components", *Byte*, April 1997, pp. 93-98.

[4] Sun Microsystem, JavaBeans Specification, Version 1.0, 1997.

[5] Microsoft Corporation, *The Component Object Model Specification*, 1995.

[6] D. Garlan, R. Allen, and J. Ockerbloom, "Why Reuse is So Hard", *IEEE Software*, November 1995, pp.17-26.

[7] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, Revision 2.0, 1995.

[8] IONA Technologies PLC, *Orbix 2 Programming Guide*, March 1997.

[9] S. S. Yau and B. Xia, "An Approach to Distributed Component-based Real-time Application Software Development", to be published in *Proc. 1st IEEE Int'l Symp. on Object-oriented Real-time distributed Computing*, April 20-22, 1998.

[10] S. P. Reiss, Connecting Tools Using Message Passing in the Field Environment, *IEEE Software*, pp. 57-66, Vol. 7 No. 4, July 1990, pp. 57-67.

[11] D. C. Schmidt, A. Gokhale, T. H. Harrison and G. Parulkar, "A High-performance Endsystem Architecture for Real-time CORBA", *IEEE Communications*, Vol.14, No.2, February 1997, pp. 34-41.