Using Adapters to Reduce Interaction Complexity in Reusable Component-Based Software Development

David Rine
George Mason University
Computer Science Department
4400 University Dr.
Fairfax, VA, USA
+703 993-1546
Drine@cs.gmu.edu

Nader Nada
George Mason University
Computer Science Department
4400 University Dr.
Fairfax, VA, USA
+703 993-1568
Nnada@cs.gmu.edu

Khaled Jaber
Case Western Reserve University
Computer Science Department
Cleveland, Ohio. 44106
USA
+614 860-2149
Jaber@Lucent.com

ABSTRACT

Many software development organizations support reuse by moving towards assembling software products using multi-use components that can fit a wide range of products and environments. In order to support the design and integration of multi-use components we are presenting the usage of component interaction adapters. Adapters are used to encapsulate, interconnect and manage multi-use component interactions. Isolating and managing the components' interactions outside the components using adapters decrease components' complexity, increase their reusability, and eases their integration.

Keywords

Complexity, Multi-use component, Requesting component, Reusability, Servicing component, and Syntactic interface.

1. INTRODUCTION TO COMPONENT BASED REUSE

There is a great deal of interest in the idea of reusing the architecture of software applications. The development of new products from the product-line reusable architecture [25, 28] assumes that the following scenario or one similar to it, has taken place. Consider a company where one of their products is doing well in the market. Because of an identified possible expansion of the current customer base, a conservative expansion of the current product is formulated by identifying a few variation points in the requirements based on a variety of new services or functions [9, 13, 14, 15].

Second, the company develops a product-line software architecture as their center piece of this expanding market's new products, advantage of the variation points to leverage the reuse of this architecture.

The product-line reusable architecture is, therefore, initially comprised of abstract components and connectors representing the topology of the architecture and applications (products) to be derived from it [1, 2, 12, 30]. At the end points of same connectors there are specifications of variation points where adapters will be introduced [11]. The variation points were determined as part of the requirements of the software architecture and were selected in cooperation between market and development departments as variations between new product releases. The specification of each variation point in the requirements is represented by an adapter in the design. The representation of each adapter at this point is incomplete, since at this point adapters that are supposed to communicate and interact with each other on behalf of corresponding components' services have not been introduced. And none of the numerous syntactical mismatches between client and server signatures have yet been resolved.

Unfortunately, neither of the two types of reuse, component and architecture is adequately supported with current technologies. The software architecture and its details are usually not explicitly defined; rather they are implicit in the software components' implementations, making it difficult to reuse. Components are difficult to reuse as well because they are tied to runtime environments and to interactions and must often be modified before reuse. The issue of reuse is even more difficult for components where only the interface of the component is available (e.g. COTs products) [7, 8, 9, 10].

In this paper we describe a first step of our research into the development of scalable tools and techniques that increase the reusability of software components and software architectures. Our approach is based on the idea of developing adapters outside (external to) components to interconnect and isolate their interactions in message--based systems supporting the utilization of many components. These adapters also need to be quality of service aware in order to allow interoperation over entirely different network types. The requirements of the component interfaces and of the interactions are expressed separately from the component, meaning that we can generate adapters automatically. Because interconnection and interaction issues are addressed separately and implemented automatically, this method decreases the components' perceived complexity, increases their reusability and simplifies their integration.

We describe issues that make integration in a component-based system difficult and provide a simple telecommunications-based case. We use this case to illustrate some of the integration problems and how adapters can be used to solve these problems.

1.1 Integration of Components with Architecture

Many current technologies, including CORBA, DCOM and Java RMI, can provide a defacto standard for local and distributed object communication [4, 14, 18, 22, 24, 25, 31]. For these tools to be useful, the component implementations must be prepared to conform to the defacto standard imposed by the environment. These technologies provide client/server communications. However, each of these uses different incompatible styles. This simple requirement may make functionally useful commercial products impossible to use in some chosen environment.

Because interaction code for a given environment is part of the component's implementation, moving it to a different environment involves making changes to source code; changing the source code for a commercial product is generally not an option. In addition, as software components move to new environments the underlying network often changes. Different networks offer different types of reliability and Quality-of-Service (QoS), and are managed and controlled in different ways. This heterogeneity is a source of complexity for software architects and designers of distributed applications, especially as the variety of network types and QoS options increases.

Integration can be difficult when even considering a single runtime environment and a defacto standard interaction style such as client/server [5, 13, 33, 34], particularly in the context of reuse and maintenance [23, 27, 28, 22]. Many problems can arise: differences in parameter ordering, differences in data types, differences in the expected interaction mechanism and differences in the meaning or semantics of the interfaces themselves. These differences can sometimes be fixed by adapting the component source code; however this usually leads to multiple different versions of the same component and means that the new version of the component should be re-tested, negating one of the benefits of reuse.

In general solving the interface problems of software components involves the study of three dimensions:

- varying interface syntax;
- varying interface semantics; and
- varying interface pragmatics.

The problems addressed in this paper include the varying mismatches between the interfaces of different software components.

Network performance is characterized by various Quality-of-Service parameters, such as bandwidth, reliability and end-to-end delivery delay. Different networks offer different QoS service levels, such as best-effort, controlled and guaranteed [35]. Best-effort service networks do not provide applications assurance that a particular QoS request will be satisfied, but rather are engineered to deliver the best possible performance. An example of a best effort network is the current Internet, which at times experiences significant levels of congestion and poor performance. Controlled and guaranteed service networks provide assured performance levels through combinations of admission control tests, traffic policing mechanisms and packet scheduling policies. Examples of these types of networks include the Next Generation Internet proposals [6] and some types of ATM networks [3].

1.2 Management of Components' Interactions

During the design phase of multi-use components the management of these components' interactions is a very important issue and should be treated as a first class priority. One of the problems with current techniques for designing reusable component is having the component's functionality and the component's interactions as a part of the component. This makes components dependent on the environment. Replacing or modifying components interacting with the given component will have an impact on the components' reusability.

- Managing component's interactions outside the component using adapters has the following advantages: Reducing component complexity; it is easier to understand a component since its interactions are not bundled in the component.
- Increasing component reusability since component is not tied to other components interacting with it. It takes less time and effort to maintain and move the component to a different environment. Therefore when a component is moved to a different environment, the focus will be on modifying the component adapter not the component itself.

1.3 Present Adapter Technologies

Most generally, adapters have been specified as design patterns [21, 12]. An adapter class implements an interface known to its clients and provides access to an instance of a class not known to its clients. An adapter object provides the functionality promised by an interface without having to assume what class is used to implement that interface. Context that suggests use of adapters is as follows:

- One wants to use a class that calls a method through an interface, but one wants to use it with a class that does not implement that interface; modifying that class to implement the interface is not an option for two reasons. Either one does not have the course code for the class or the class is a general-purpose class, and it is inappropriate for it to implement an interface for a specialized purpose.
- One wants to dynamically determine which methods of another object that an object wants to call., but one

wants to accomplish that without the calling object having knowledge of the other object's class.

CORBA (Common Object Request Broker Architecture, www.omg.org) adapters for object-oriented data base applications, as well as Basic Object Adapters (BOA) whose functions and features require the following:

- An implementation repository that enables programmers to install and register an object implementation.
- Ways to generate and interpret the object references.
- Ways to activate and deactivate object implementations.
- Ways to invoke methods and pas parameters through stubs and skeletons.
- A security method that includes the mechanism for authenticating the client making the call. While CORBA specifies the existence of these two features, implementation details of these features depend on the organization developing a CORBA-compliant product. Moreover, evaluations of the numerous ORB products has been limited. Moreover, the uses of CORBA object-oriented data base and basic adapters is presently limited.

Another to adapters technology is the polylith software bus, which is designed to help programmers for execution in heterogeneous environment [18]. Components that communicate in CORBA, COM/DCOM, or polylith environments must adhere to their given component interfaces or continually add new interfaces. Interconnecting components using adapters for some applications, for example single processor applications, is easier since components themselves do not have to adhere to a complex way of managing interfaces and references to the services of other components. Moving a component that uses an adapter to work in a different environment takes less integration and maintenance effort than a component that is designed to work in CORBA, COM/DCOM, or polylith environments. This is because component interactions are encapsulated in the adapter for that component which is outside the component, i.e., component might not need to be modified.

1.4 Paper Specifics

In this paper, we present a method for using adapters outside the components (externally) to interconnect and isolate their interactions in a message-based system. We also show through a case study how this method can decrease the components' complexity, increase their reusability, and simplify their integration. The case study presented in this paper uses a telephone telecommunications application as an example to show how to use adapters to interconnect and mange components' interactions.

Section 2 describes the approach that is used to interconnect and manage components' interactions in a message-based system. Section 3 is the case study. Section 4 is the conclusion.

2. INTERCONNECTING AND MANAGING COMPONENTS' INTERACTIONS IN THE ARCHITECTURE

Adapters are introduced to interconnect components, contain interaction's protocol, and manage component interactions. For each component in a system there is an adapter associated with it. Component talks to its associated adapter, the adapter talks to its associated component, and adapters communicate with each other to fulfill components' interactions. The terms requesting component [1]and a servicing component [2] are used throughout this paper.

2.1 Components Talk to Adapters

Components request services from each other through their associated adapters. The adapters provide the following interface to be used by its associated component: *A-put(service name, paramater1: parameter1 type, parameter2: parameter2 type, ...).* Service name and parameters refer to a requested service.

This service is used by associated components to send requests for outside services. A requesting component does not need to know the exact signature of the service performing the request. The adapter associated with the servicing component is responsible for solving the syntactic interface mismatch. This is shown in section 2.2.
The service used by a requesting component to send the result of a performed request is called: *A-result (service name, result).*

2.2 Adapters Talk to their Associated Components

Components explicitly define their interface. It defines the syntactic interface, e.g., service name, parameter type, parameters order, for the services it supports. Also components are required to provide services as part of their interface to receive the results of a requested service.

Adapters are responsible for invoking a component's services based on a request from other adapters or deliver the result of a requested service. The adapters presented in our case study use a two-part method applying configuration files to solve the syntactic interface mismatch, service name, parameters type, parameters orders, between a requested service and an exact signature of the service performing the request. Also the two-part method is used by adapters to find the service that needs the result of a performed service.

What does a configuration file do? A configuration file contains:

- A mapping between the requested service signature and the signature of the service that will perform the request.

[1] A requesting component is a component that makes a request for a service provided by other component.
[2] A servicing component is a component that service a request from other components.

- A mapping between a requested service signature and the signature of the service that will receive the result of that requested service.

Adapters can use, for example, macros that take a requested service name and its parameters. Macros will return the exact format of the service signature that will perform the request based on the data found in the configuration file. Macros can also be used to take a requested service name and return the exact signature of the service that needs the result of the requested service.

2.3 Adapters Interactions

Adapters usually communicate using two different software architectures. In the first software architecture adapters communicate with each other through a mediator or a facilitator. In the second architecture adapters communicate with each other directly. One of the disadvantages of having adapters communicate through a mediator or a facilitator is related to decreased components complexity at the expense of decreased interactions performance. Hence, this section describes the second case where adapters communicate directly without using a mediator or a facilitator.

For adapters to communicate, they need to recognize each other, i.e., adapter-a talks to adapter-b. Adapters usually use the system's communication mechanisms, e.g., TCP/IP, shared memory queues, etc. to communicate with each other. In this paper a configuration file that contains the mapping between adapters is used. It might also contain information needed to establish the connection between adapters. For example, if adapters use the inter-process communication mechanism TCP/IP for communication, port numbers between adapters will be included in the configuration file. Adapters read the configuration file at initialization time.

3. CASE STUDY OF ADAPTER-ORIENTED REUSE

3.1 Purpose of Case Study

The purpose of this case study is to demonstrate our method by showing how to interconnect and manage components' interactions using adapters in a message-base system. The case study uses a telephone telecommunications application as an example. In this case study the impact of replacing and modifying systems' components is also presented.

3.2 The Model System

This system is developed as an application for telephone telecommunications. The system uses a signaling protocol called Signaling System 7 (SS7). The system contains two components, one component that monitors the signaling links and is called the monitor component (MT). The other component is used to manage the signaling links and called manage component (MG). The MT component monitors the signaling links and makes requests to bring links in service, inhibit links, and takes links out of service. The MG component puts links in service, inhibit links, and takes links out of service. The following is a high level details for the system:

3.3 System Assumptions

- The MT component makes requests to bring links in service, inhibit links, and takes links out of service. The MT component services are invoked through a display monitor.
- The MG component services requests to put links in service, out of service, and inhibit links.
- The adapter associated with MT component is called MT-Adapter, and Adapter associated with the MG component is called the MG-Adapter.
- The communication mechanism used is shared memory message based queues. A shared memory object captures the system's communications mechanism and uses it to establish the connection between adapters to communicate.
- The method used for introducing adapters in the system is a configuration file called adapters.config.
- The method used for solving the syntactic interface mismatch is a configuration file called interf.config.
- Adapters communicate by sending and receiving messages containing requests.

Note: This case study presents the scenario putting a link in service; inhibiting and taking a link out of service scenarios are similar.

3.4 Component' Interfaces

The following sections present interfaces for the MT and the MG components, the MT adapter and the MG adapter, and the shared memory object.

3.4.1 Monitor Component's Interface

```
MT interface {
Monitor-links( );                          //To monitor a link.
Inservice-result(result: integer );
          //Receives result of in service request
Inhibit-result(result: integer );
          //Receives result of inhibit request
Outservice-result(result: integer);
          //Receives result of out service request
                    }
```

3.4.2 Management Component's Interface

```
MG interface {
In-service (link-number: long integer, time-in-serv: float);
          //Bring a link in service.
Inh-link (link-number: long integer, time-inh: float);
          //Inhibit a link
Out-service (link-number: integer, time-out-serv: float);
          //Take a link out of service.
                    }
```

3.4.3 Communications Mechanism Interface

```
Sh-mcm-object interface {
Result create-queue(queue-id: integer);
          //Create a shared memory queue.
Result attach-queue(queue-id: integer);
        //Attach to shared memory queue.
Result put-message(Msg);
        //Put message on a queue.
Result get-message(Msg);
        //Get message from a queue.
                            }
```

3.4.5 Adapter's Interface

Adapter-interface {
A-put (Service-name, parameter1-type: parameter1, parameter2-type: parameter2, …).
//To send a request for a service provided by other component.
A-result (Service-name, result-type: result);
//To send a result for a performed service. }

The following shows the format of the message used between adapters:
Message-format {
Service-name, Parameter1: Parameter1-type, ..., parameter n: parameter n-type }.

3.5 Configuration Files

3.5.1 Adapters.config File

The adapters.config file is used to introduce the MT-Adapter and the MG-Adapter to each other. The MT-Adapter sends messages to the MG-Adapter using queue id 100. MG-Adapter sends messages to MT-Adapter using queue id 200. Table 1 shows the format for adapters.config file. Other formats can be used as well.

Table 1 Adapters.config File Template

Source Adapter	Destination Adapter	Queue Id
MT-Adapter	MG-Adapter	100
MG-Adapter	MT-Adapter	200

3.5.2 Interf.config File

The interf.config file contains the resolution for the syntactic interface mismatch for the services "in-service", "inh-link", and "out-service". This file also contains the interface for the services that need the result of a performed request. Tables 2 and 3 represent one format for interf.config file. Other formats can be used as well.

Table 2 Resolution for the Service "In-service"

Requested Service	Actual Service
Inservice	In-service
link-id: short	link-number: long integer
time: integer	time-in-serv: float

Table 3 The Result of the Request "In-service"

Requested Service	Service Receives Result
Inservice	Inservice-result
link-id: short	Result :integer
time: integer	

Adapters can use, for example, macros that take a requested service name and its parameters. The macros will return the exact format of the service signature that will perform the request based on the data found in the configuration file. Macros can also be used to take a requested service name and return the exact signature of the service that needs the result of the requested service.

3.6 Adapters and Components Interactions

The section describes the communications between adapters and components to service a request for putting a link in service.

3.6.1 MT Component and MT-Adapter Communications

- The MT component makes a request to bring a link in service by calling the service A-put(inservice, time: integer, link-id:short) provided by the MT-Adapter. Notice, the difference between the passed service name and parameters with the signature of the service. This is shown in the MT interface that will service the request.
- When the result of "inservice" arrives, the MT-Adapter calls the service "inservice-result".

3.6.2. MG Component and MG-Adapter Communication

The MG-Adapter invokes the service "in-service" on the MG component. The MG component sends the result of "in-service" to MG-Adapter by calling the service A-result(in-service, result).

3.6.3. MT-Adapter and MG-Adapter Communication

To establish the connection between the MT-Adapter and the MG-Adapter both adapters at initialization time read the queue ids that will be used for communication. They also call the services create-queue and attach-queue to establish the connection. In this case the MT-Adapter uses queue id 100 and the MG-Adapter uses queue id 200.
To fulfill the "in-service" request the following steps are needed:

- The MT-Adapter builds a message containing "in-service" request and sends it to the MG-Adapter using the shared memory object services. The message will take the following format: *Service name: inservice, Parameter 1: link-id, Parameter 1-type: short, Parameter 2: time, Parameter 2-type: integer.*

- The MG-adapter gets the request from the MT-Adapter. It calls a macro that resolves the syntactic interface mismatch. The MG-Adapter invokes the MG component service "in-service". The MG component sends the result to the MG-Adapter by calling "A-result" service provided by the MG-Adapter. The MG-Adapter builds a message containing the result and sends it to the MT-Adapter using the shared memory object services.
- The MT-Adapter gets the result of the "inservice" request. It calls a macro that returns the service that needs to be invoked on the MT component, i.e., "in-service-result". The MT-Adapter invokes the MT component service "in-service-result".

3.7 Replacing and Modifying Components

This section describes the impact of modifying the MG component interface on the system. It also discusses the results of replacing the MG component by another component that provides the same functionality with a different interface.

3.7.1 Modifying the MG Component Interface

Assume that MG signature for the service "in-service" is changed to in-service (link-id: long integer). The time value is removed and used as a default value.
In this case the MT component would not be impacted because it continues calling the service A-put(inservice, link-id: short, time: integer). The MT-Adapter would not be also impacted because it continues sending the same message. On the other hand the Interf.config file would be impacted. Table 4 presents the new mapping for the "in-service". Macros used to perform the mapping are impacted. The MG-Adapter is not impacted. It calls the same macro.

Table 4 New Mapping for the "In-service"

Requested Service	Actual Service
inservice	in-service
link-id: short	link-number: long integer
time: integer	

3.7.2 Replacing MG component

Assume that a new component called, NEW-MG, provided by a different vendor replacing the MG component.

In this case the MT component and the MT-Adapter would not be impacted. The interf.config file is modified. The MG-Adapter is modified to work with the new interface. Adapter.config file might need to be modified. The only part that might need to be modified is the one that invokes services for the NEW-MG component.

4. CONCLUSION

A case study using a telephone telecommunications application as an example is presented to show how to interconnect and manage components' interactions in the architecture using adapters.

The goal of designing a multi-usable component is to make it fit a wide range of products and environments. In order to design reusable components we should make a distinction between the component's functionality and their interactions. Component's interactions should be managed outside the component. Isolating and managing the components interactions outside the components using adapters decreases the components complexity, increases their reusability, and ease their integration.

Some software developers may consider the development of adapters as an overhead. However, adapters are reused most of the time; they will pay off their initial development cost. Introducing adapters that capture component's interactions make components independent and cleaner, i.e., reusable and less complex.

5. REFERENCES

[1] Abowd, G., Allen, R., Garlan, D., Formalizing Style to Understand Descriptions of Software Architecture, ACM Transactions on Software Engineering and Methodology, 4 (October 1995), 319-364.

[2] Allen, R., Garlan, D., A Formal Basis for Architectural Connection, ACM Transactions on Software Engineering and Methodology, 6(July 1997), 213-249.

[3] ATM Forum Traffic Management Draft Standard aftm-0056.000, April 1996.

[4] Baker, S., CORBA Distributed Objects: Using Orbix, Addison-Wesley, Inc., (and ACM Press), New York, 1998.

[5] Beach, B., Connecting Software Components with Declarative Glue, in Proceedings of the 14th International Conference on Software Engineering, (May 1992).

[6] Bradshaw, J., Software Agents, American Association for Artificial Intelligence. ISBN 0-262-52234-9.

[7] Brown, A., and Wallnau, K., Engineering of Component-Based Systems, Proceedings of the 2nd IEEE International Conference on Engineering of Complex Systems, (1996). IEEE Computer Society Press 1996.

[8] Clements, P. From Subroutines to Subsystems: Component- Based Software Development, American Programmer, (November 1995). IEEE Computer Society Press 1996.

[9] Dellarocas, C., Towards a Design Handbook for Integrating Software Components, Fifth International Symposium on Assessment of Software Tools and Technologies (SAST97), (June 1997). ftp://pound.mit.edu/pub/ Dellarocas-Ph.D.

[10] Dellarocas, C., The Synthesis Environment for Component-Based Software Development, The 8th International Workshop on Software Technology and Engineering Practice (STEP'97), (July 1997). ftp://pound.mit.edu/pub/ Dellarocas-PhD.

[11] Jaber., K.; Nada, N.; Rine, D.; Towards the Design and Integration of Multi-use Components, The Proceedings of International Conference on Information Systems Analysis and Synthesis, (July 1998).

[12] Garlan, D., R. Allen, J. Ockerbloom. Architectural mismatch or why its hard to build systems out of parts. Proceedings of the 17th International Conference on Software Engineering, IEEE, (May 1995).

[13] Meyer, B., Design by Contract, Computer, IEEE Computer Society, October, 1992.
[14] Mowbray, T., and Ruh, W., Inside CORBA: Distributed Object Standards and Application, Addison-Wesley, New York, 1997.
[15] Nada, N.; Rine, D.; Tuwaim, S., Best Software Reuse Practices Require Reusable Software Architecture in Product Line Development, Proceedings of the Second Workshop on Software Architectures in Product Line Acquisitions, (June 1998).
[16] Nada, N.; Rine, D.; Component Management Infrastructure: A Component-Based Software Reuse Reference Model M., Proceedings of the ICSE98 International Workshop on Component-Based Software Engineering, (Japan 1998).
[17] Nada, N.; Rine, D.; Jaber., K.; Towards Components-Based Software Development, , The Proceedings of European Reuse Workshop (ERW'98), (Spain, November 1998).
[18] Object Management Group (OMG), CORBA 2.3 (1998), CORBA 30. (1999).
[19] Orfali, R., Harkey, D., and Edwards, J., The Essential Distributed Objects Survival guide, Addison-Wesley, 1996.
[20] Prieto-diaz and Neighbors, J., Module Interconnection Languages, The Journal of Systems and Software, 6(1986) 307-334.
[18] Purtilo, J., The Polylith Software Bus, Tec. Rep. TR-2469, University of Maryland, 1991.
[21] Rine, D. and Retnadhas, C. Design of a ring-based local area network for microcomputers: improved architecture using interface node adapters, Proceedings of the Fifth Conference on Local Computer Networks, IEEE-CS, ACM, Minneapolis, Minnesota, October, 1980.
[22] Rine, D. and R. Sonnemann, Investments in Reusable Software: A Study of Software Reuse Investment Success Factors, Chapter in Measuring Information Technology Investment Payoff, 1998.
[23] Rine, D. and J. Chen, Testing trainable software components by combining genetic algorithms and backpropagation algorithms, Proceedings of the Conference on Artificial Neural Networks in Engineering, November, The American Society of Mechanical Engineers, Corp. Sponsor, 1996.
[24] Rine, D. and J. Baldo, A framework for software reuse: comprised of technical and non-technical attributes', special issue of the ACM StandardView, (June, 1997), The ACM.
[25] Rine, D. and R. Sonnemann, A software manufacturing framework for the software reuse function with OMG CORBA compliance, special issue on software reuse, International Journal of Applied Software Technology, (March 1997).
[26] Rine, D. and M. Ahmed, A reusable intelligent autopilot: a framework, Special Issue of the International Journal of Applied Software Technology, (October 1997), Corp. Sponsor.
[27] Rine, D., Supporting reuse with object technology, Computer, IEEE Computer Society, (October 1997).
[28] Rine, D., Sonnemann, R., Investments in Reusable Software. A Study of Software Reuse Investment Success Factors , The Journal of Systems and Software, 41(1998), 17-32.
[29] Sametinger, J. Component Interoperation, WISR8 March 23, 1997. [http://www.umcs.maine.edu/~ftp/wisr/wisr.html].
[30] Shaw, M., Garlan, D., Software Architecture, Prentice-Hall, Inc., 1996.
[31] Trevor, J., Rodden, T., and Mariani, J., The use of adapters to support cooperative sharing, Proceedings of CSCW'94, ACM, 219-230.
[32] Wang, Y. COM/DCOM Resource. [http://akpublic.research.att.com/~ymwang/resources/resources.htm]
[33] White, E., Purtilo, J. Integrating the Heterogeneous Control Properties of Software Modules, Proceedings of the Fifth ACM SIGSOFT Symposium on Software Development Environments, (December 1992), 99-108.
[34] White, E. Control Integration in Heterogeneous Distributed Systems Ph.D. Thesis, University of Maryland, College Park, April 1995.
[35] Zhang H., ``Service Disciplines for Guaranteed Performance Service in Packet-Switching Networks, Proceedings of the IEEE, 1996.