

Service Oriented Architecture (SOA) for Integration of Field Bus Systems

Xiaohua Tian*, Yu Cheng*, Rose Q. Hu[†] and Yi Qian[‡]

*Department of Electrical and Computer Engineering, Illinois Institute of Technology
Chicago, IL, USA 60616, email: {xtian3, cheng}@iit.edu

[†]Department of Electrical and Computer Engineering, Mississippi State University
Mississippi State, MS, USA, email: hu@ece.msstate.edu

[‡]Department of Electrical and Computer Engineering, University of Puerto Rico at Mayaguez
Mayaguez, PR, USA, email: yqian@ece.uprm.edu

Abstract—The current trends in service consolidation over Internet Protocol (IP) also stimulates the integration of the industrial automation system with the information technology (IT) infrastructure for more efficient information access and more cost-effective production and management. Field buses have been the de facto communication standard in industrial automation, but mostly based on manufacture-specific protocols. Thus, the interoperability between the manufacturer-specific field bus systems and the external operating environment is the critical factor in enabling the networked industrial automation systems. However, most of the existing field bus integration solutions lack either flexibility or scalability. In this paper, we propose a service-oriented architecture (SOA) based field bus integration architecture (SOAFBIA), where each field bus system is encapsulated with optional interface, manageability interface, and semantic descriptions in a stand format to facilitate interoperability. Moreover, a resource agent is proposed as an enhanced service broker, which implement not only the standard service registry functionality in SOA, but also the resource management functions including admission control, service scheduling, and load balancing.

I. INTRODUCTION

In the recent decade, the Internet has been evolving into a common communication infrastructure for all kinds of services, which significantly propel the proliferation of the service-oriented architecture (SOA) [1], [2] in the manufacturing industry for seamlessly integrating industrial automation systems with the business level information technology (IT) systems. Status information can be retrieved from and control messages can be delivered to manufacturing and executive entities through corporate wide or worldwide networks, to enable more efficient and cost-effective production and management [3], [4]. Within the industrial automation system, field buses have been the de facto means of communications between sensors, actuators and control units, but mostly based on manufacture-specific protocols [5]. Many efforts are being made to improve the interoperability between field bus systems so that system integration can be executed in a finer granularity to further improve the flexibility and adaptivity of the whole manufacturing system to meet the market demand.

One approach for field bus integration is to establish a common information model supported by all field bus systems to facilitate the message exchange between them [6], [7].

However, since the information model deeply depends on the subjectivity of the information user, no much breakthrough has been yielded after years of effort to reach a globally accepted information model. Another approach is proposed in [5], where each field bus system and the external management tool can have different information models; the field bus system will connect to a centralized “translation directory” to retrieve the mapping relationship between different information models to achieve the interoperability. It is not difficult to see that the translation directory based approach tends to suffer from the scalability problem. In a large scale system with frequent message exchange, the communication with the translation directory will result in significant processing overhead, which is not suitable for the industrial automation system where the rapid response is of great importance [3], [8].

In this paper, we propose a service-oriented architecture (SOA) based field bus integration architecture (SOAFBIA), where each field bus system is encapsulated with optional interface, manageability interface, and semantic descriptions to form a manageable Web service component [9], [10] to facilitate interoperability. According to the SOA principle, a service request can be bound to a service component (operating as a service provider) through the optional interface, while a management operation can be delivered to the manageable component through the manageability interface. Particularly, embedding semantics in the interface descriptions enables information model mapping in a distributed manner, which could be implemented either in the message sender side or in the receiver side. Moreover, a resource agent is proposed as an enhanced service broker, which implements not only the standard service registry functionality in SOA, but also the resource management functions including admission control, service scheduling, and load balancing. To the best of our knowledge, this is the first work to combine resource management schemes with the field bus integration schemes.

The remainder of this paper is organized as follows. Section II describes the SOAFBIA design. Section III presents a design of the resource agent and associated resource management techniques. Section IV presents some experiment results to demonstrate the performance of SOAFBIA. Section V gives the conclusion remarks.

II. SOA BASED FIELD BUS INTEGRATION ARCHITECTURE (SOAFBIA)

In SOA, various software programs, intelligent devices, and networking resources are encapsulated via standardized common interfaces as loosely-coupled service components, and the service is provisioned according to the “find, bind and execution” paradigm [1]. In SOA, a service component can be reused/repurposed readily for different functionalities and upgraded conveniently with an advanced implementation, which enables cost reduction in the integration of new services as well as in the maintenance of existing services.

The proposed SOAFBIA design is shown in Fig. 1, where each field bus system is encapsulated with optional interface, manageability interface, and semantic descriptions to form a manageable Web service component [9], [10] to facilitate interoperability; the service components are connected either through cooperate wide Ethernet or worldwide Internet. The optional interface describes the operations that the component performs and the properties of the component, where a specific operation is modeled as a pair of inputs to and outputs from the component, and each input and output is modeled as a pair of name and data type. The manageability interface is defined in the similar format as that of the optional interface, but with syntax and semantics to provide manageability capabilities including identification, configuration, metrics, status, operations, and events generated by manageable entities for management purpose. In SOAFBIA, a client perform either as a customer sending out certain service requests or as a manager sending out control and management messages. According to the SOA principle, a service request can be bound to a service component (operating as a service provider) through the optional interface, while a management operation can be delivered to the manageable component through the manageability interface.

The semantic description aims at adding machine-interpretable information to the Web service interfaces to enable automatic service discovery. In order to model the semantics of a Web service component, the entities of *concepts* can be defined to represent abstract ideas, which are then used to annotate the semantics of the operations, inputs, outputs and properties of the component [11]. A concept can also be used to specify the relationship between two concepts. With the concept annotation, the machine-understandable description of a Web service can be in the format of a *semantic graph* that consists of nodes and labeled links. Nodes in the semantic graph represent operations, inputs, outputs and properties of a component, as well as their data types and concepts. Labeled links in the semantic graph represent the relationship among the nodes [11]. Using the semantic descriptions, the mapping relationship between the information models can be established by the mapping between the corresponding semantic graphs.

In SOAFBIA, each service component will publish its location and service description to a *resource agent* (RA). A service request or management message from a client is

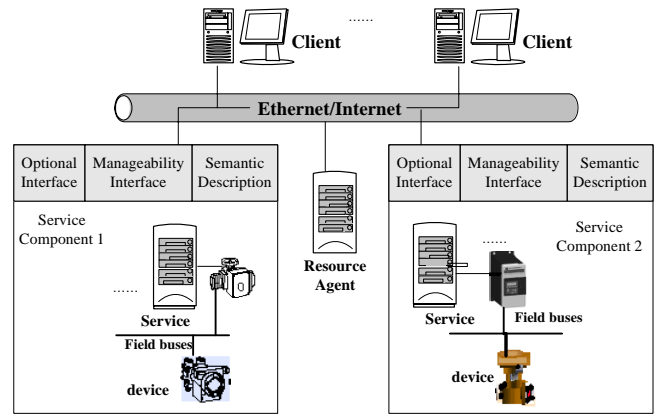


Fig. 1. SOA based field bus integration architecture (SOAFBIA).

processed by the RA first, which will find the matched service component in its database and send the associated location information and service descriptions back to the requester. The client can then deliver the message to the located service component, where the information model mapping can be either on the sender side or the receiver side with the awareness of semantic descriptions. It is noteworthy that a field bus system can also behave as a service requestor to use services provided by other field bus systems. In SOAFBIA, the RA also implements resource management functions, in addition to performing as a standard service registry. The RA can monitor and maintain the status of each service component, regarding quality-of-service (QoS) or resource availability, through the manageability interface. Such status information can be used to determine an optimal service component as a service provider when a new service request comes in.

SOA is an abstract reference model, implementable by different techniques. Currently, the SOA implementation based on Web Services [12] is becoming popular and being standardized in the industry, where the service definition is Extensible Markup Language (XML) based in the form of Web Services Description Language (WSDL) and the messages are usually in XML format communicated over the SOAP protocol [13]. However, SOA can also be bound with other implementation techniques. As XML coding is verbose, not suitable for real-time processing in the industrial automation systems, we consider each service component also provide a CORBA endpoint [8], [14] in addition to the default Web service endpoint to expedite the message processing.

III. RESOURCE MANAGEMENT IN SOAFBIA

In this section, we focus on the CORBA based implementation to illustrate the resource management in SOAFBIA.

A. Resource Agent Architecture

Consider that each entity in the SOAFBIA supports a CORBA interface and the message exchange in the system is supported by the CORBA Object Request Broker (ORB). An implementation with the detailed RA structure is shown in Fig. 2, where multiple groups servers are connected to

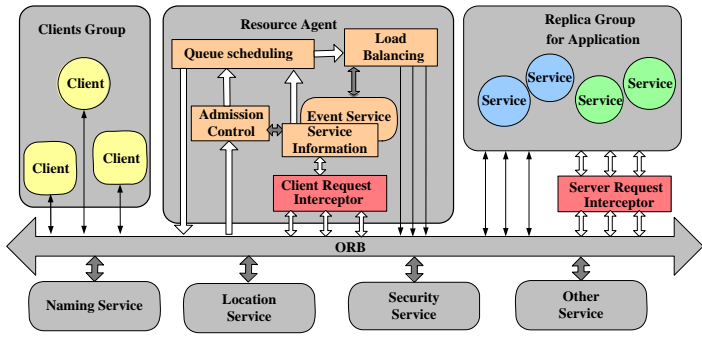


Fig. 2. Resource agent structure.

serve groups of clients according to the SOA principle. In the perspective of resource management, the RA will monitor the delay performance of each server and maintain a status table. Specifically, the Admission Control module is responsible for comparing the client's delay requirement attached in a service request message with the server delay information maintained in the RA to decide whether to accept the request. The admitted requests for different servers are then put into corresponding service queues and managed by the Queue Scheduling module. The Load Balancing module is used to forward each request to the optimal service provider in the service group. The Service Information module monitors and updates the server performance information automatically and regularly. The interceptors attached to the ORB are for delay calculation. The working flow of the architecture is as following:

- 1) Clients send service requests to the system, with the QoS requirement of a delay bound encapsulated in requests.
- 2) A request enters the RA. The admission control module parses the service request message and compares the extracted delay requirement with the server delay information. If a server that can meet the delay requirement exists, the service request will be accepted and forwarded to the corresponding service queue; otherwise, the request is rejected.
- 3) The queue scheduling module uses the priority scheduling to serve the queued service requests. Normally, there are two types of requests in the industrial automation system: requests for device operation, which appear sparsely but requires high reliability, and requests for data retrieve, which appear frequently but requiring relatively low reliability (as those data are retrieved cyclically). The first type of requests are served with higher priority.
- 4) The request selected by the scheduling module is passed to the load balancing module, which selects the server with the best delay performance to serve the request.
- 5) Periodically, the service information module sends probe messages to the service groups by using the ORB event service and the Portable Interceptors. Interceptors enable the RA to calculate the service delay of each request.

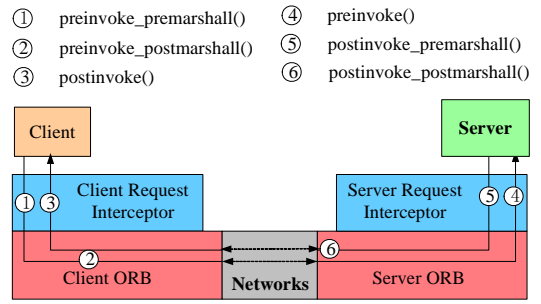


Fig. 3. Portable interceptor operations.

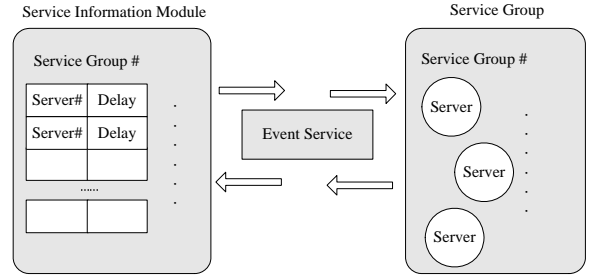


Fig. 4. Service information updating.

B. Resource Management

We implement the CORBA based SOAFBIA with Borland C++ Builder 6.0, which contains Borland's CORBA product - VisiBroker 4.5 [15]. Here we discuss the proposed resource management techniques under the VisiBroker context.

1) *Delay Measurement*: Delay is one of the most important parameters in industrial automation systems. In the RA, portable interceptors are used to measure different types of delays. As shown in Fig. 3, interceptors [15] at both client and server side are independent of application functions; they provide six extensible interfaces for developers to use. The operation *preinvoke_premarshall* and *preinvoke_postmarshall* are invoked at the start and end of the marshalling process at client side. After marshalling, requests are translated into the common format which can be recognized by other objects. The operation *postinvoke* is triggered when the result of the request is returned to the client side; the corresponding operations are also invoked at the server side. We can record the time when each important operation is invoked by programming those extensible interfaces, and get the following delays:

- Round trip time of a request = $\text{time}(3) - \text{time}(1)$,
- Service processing delay = $\text{time}(5) - \text{time}(4)$,
- Propagation delay = $(\text{time}(3) - \text{time}(6)) - (\text{time}(2) - \text{time}(1))$,

where $\text{Time}(1)$ to $\text{Time}(6)$ are the time instants when the six operations as indicated in Fig. 3 are invoked, respectively. The measured delays are then stored in the service information module to indicate the status of associated servers.

2) *Service Information Update*: In the service information module, there resides a performance table for each service group, as shown in Fig. 4. It also contains the interface of event service and location service for updating service information. In the method of *UpdateServiceInfo* in the service information

module, the interfaces of event service and location service are invoked. Event service is specified in CORBA standard, but location service is an extension to CORBA standard made by VisiBroker. It updates the information of servers at a regular interval automatically [15].

3) *Request Redirection*: In CORBA mechanism, request redirection can be implemented by interceptors. There are two exceptions `LOCATION_FORWARD` and `LOCATION_FORWARD_PERMANENT` in CORBA standard. When one of the two exceptions is thrown out, ORB will direct the request to a different service instance specified in an attribute in the exception definition. Interceptor can be loaded at client side to capture each request before it goes into ORB, then the interceptor examines the object ID specified in the request. If the object is not the optimal, interceptor will release the exception `LOCATION_FORWARD` to ORB, after that, the client will reissue the request to the object specified in the exception. This implementation is practical from the functional perspective, nevertheless, each client side will have to know the information of entire service group to determine which server is optimal. Such information is hard to be obtained in a scalable approach.

The RA implements request redirection by coordinating several components of VisiBroker. In VisiBroker, there is a dynamic distributed directory service named Smart Agent. When a client tries to bind a remote server, the smart agent will be checked automatically, then, it will set up a connection between the client and the server specified in parameters of *bind* operation. In our implementation, we use location service to retrieve information of all servers running in the network. Each object is identified by its instance name and object ID. Smart agent updates objects directory every two minutes, at this time, probe messages can be sent to all the latest updated objects according to their object descriptions from smart agent. Interface *Agent* is provided by location service for developers to have access to current system operation information [15] such as which objects are running, where they are, how to identify them. Then, the load balancing module selects the information of the optimal object from service information module and fills the parameters in *bind* operation with object description of the optimal server.

IV. PERFORMANCE EVALUATION

A. Resource Management Efficiency

In this section, we run experiments over a testbed to evaluate the performance of the RA based resource management. The testbed consists of a few computers to perform as clients, servers, and the RA, with configuration details given in Table I. All the entities are interconnected with 100Mbps Ethernet.

We refer to the *modified power formula* defined in [16]

$$Power = \frac{Throughput^\alpha}{\bar{d}(\lambda)} = \frac{(\sum T_i)^\alpha}{\frac{1}{\lambda} \sum_{i=1, N} \lambda_i \bar{d}_i} \quad (1)$$

as a performance measure of the resource utilization efficiency. In (1), $\sum T_i$ is the total throughput of all the queues, λ_i the request arrival rate to the *i*th service group, \bar{d}_i the average

TABLE I
EXPERIMENTS CONFIGURATIONS

Role	ID	CPU	Memory	OS	
Resource Agent Server	RAS	Intel Celeron 2.8GHZ	512M	Win XP (Prof)+SP2	
Client NO.1	C1	AMD Duron 900	256M	Win2000(Prof)	
Client NO.2	C2	AMD Duron 650	256M	Win2000(Prof)	
Service Group 1	Server 1	S1	Intel Celeron 2.8GHZ	512M	Win XP (Prof)+SP2
	Server 2	S2	Intel P4 2.0GHZ	256M	Win XP (Prof)
	Server 3	S3	AMD Duron 650	256M	Win XP (Prof)
	Server 4	S4	Intel Celeron 2.8GHZ	512M	Win XP (Prof)+SP2
Service Group 2	Server 5	S5	Intel P4 2.0GHZ	256M	Win XP (Prof)
	Server 6	S6	AMD Duron 650	256M	Win XP (Prof)

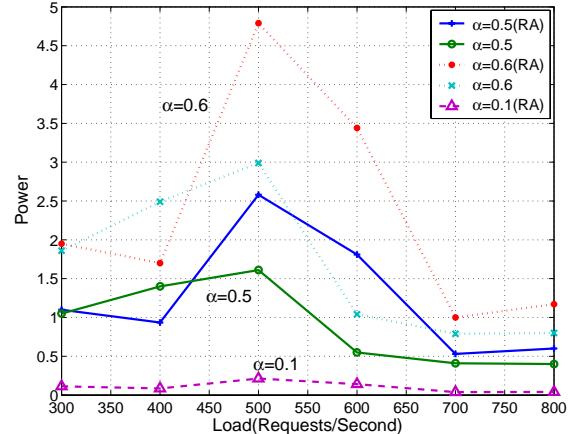


Fig. 5. The Power graph with resource management.

queuing delay of the *i*th service group, and $\bar{d}(\lambda)$ the average queuing delay of the entire system. $\alpha \in [0, 1]$ is a constant that determines the relative weight of throughput in the performance evaluation. When α is close to 0, basically only the delay is of importance in the performance evaluation; when α close to 1, the evaluation also takes the throughput into account.

We consider a scenario where two service groups send service requests with $\lambda_1 = 2\lambda_2$. To measure \bar{d}_i , we take one measurement sample by averaging over 100 requests; the \bar{d}_i is obtained by averaging over the samples with enough requests sent to guarantee a small 95% confidence interval. We experiment under two cases, with and without RA applied. When RA is not applied, the CORBA scheme statically bounds the requests to a fixed server, behaving as a traditional Client/Server (C/S) system.

The power graph according to (1) is given in Fig. 5. It is not difficult to understand that there is a tradeoff relationship between throughput and delay, as higher throughput tends to increase the queue length and result in longer delay. The results in Fig. 5 clearly indicate that there exists an optimal load point leading to the optimal tradeoff between delay and throughput and therefore the maximum power value. Moreover, the RA based resource management can significantly improve the power value under the optimal work load, reflecting a more efficient resource utilization. From Fig. 5, we can also observe that at first RA architecture performs worse than the C/S scheme, but with the increase of the traffic load, the power of RA exceeds the C/S scheme. The reason is that when the traffic load is small, direct binding requests to the

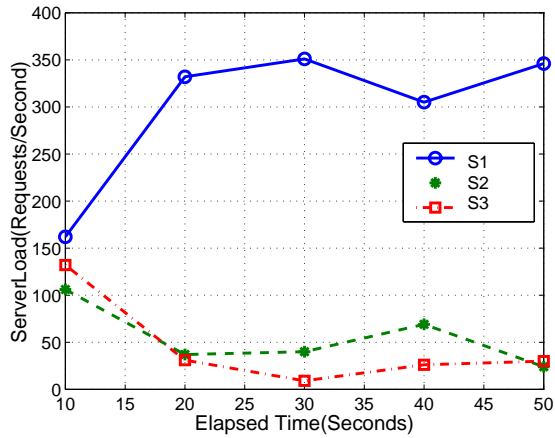


Fig. 6. Load distributions with resource management ($\lambda_1 = 400$).

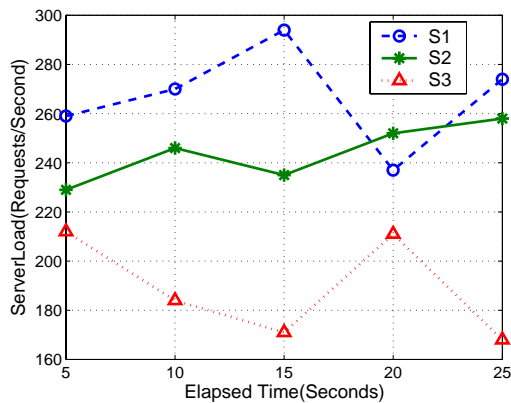


Fig. 7. Load distribution with resource management ($\lambda_1 = 700$).

server is faster than binding to a server via the RA processing. When the traffic load increases, the benefit from resource management will become more significant and surpass the cost due to RA processing overhead.

B. Load Balance

In this experiment, both C1 and C2 send requests to service group1 with $\lambda_1 = \lambda_2$. We examine the load distribution among the servers in the same service group. The “service request arrival rate versus time” curves are shown in Fig. 6 and Fig. 7 to demonstrate the dynamic load balancing achieved by the RA. In the case of $\lambda_1 = 400$, where the load is smaller than the optimal value, Fig. 6 shows that S1 is assigned most of the work, since its hardware configuration is the best. RA always binds requests to S1 in this light load condition. In the case of $\lambda_1 = 700$ as shown in Fig. 7, where the load of the system is larger than the optimal load. The server S1 can not always achieve the smallest delay due to the extra work load allocated; upon such a point, the traffic load will be distributed to other servers, leading to a reduced delay in S1, which in turn becomes a hot point again to attract traffic load. Such dynamic traffic load adjustment implies the load balancing effect: the different serving capacity at each server is balanced by traffic

load distribution so that each server achieves approximately the same delay.

V. CONCLUSION AND FUTURE WORK

In this paper, we present a SOA based field bus integration architecture, where each field bus system is encapsulated into a manageable Web service component for interoperability. In addition, a resource agent is proposed to combine the resource management functions with the service registry functions to enable SOA based service enabling as well as efficient resource utilization. Performance of the proposed SOAFBIA design is demonstrated via testbed experiment results. The results show that the SOA based approach is more effective than the traditional client/service model especially when the work load is heavy. For future work, we plan to investigate Application Oriented Networks (AON) [17] as the network infrastructure to implement resource allocation in automation system. The AON integrates message-level processing capability into network devices, which has the potential to form a more effective and convenient SOA based industrial information system.

REFERENCES

- [1] T. Erl, *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services*, Prentice Hall, 2004
- [2] J. Pasley, “How BPEL and SOA are changing Web services development,” *IEEE Internet Computing*, vol. 9, pp. 60–67, May-Jun. 2005.
- [3] F. Jammes and H. Smit, “Service-oriented paradigms in industrial automation,” *IEEE Trans. Ind. Informat.*, vol. 1, no. 1, pp. 62–70. Feb. 2005.
- [4] A. P. Kalogeras, J. V. Gialelis, C. E. Alexakos, M. J. Georgoudakis, and S. A. Koubias, “Vertical integration of enterprise industrial systems utilizing web services,” *IEEE Trans. Ind. Informat.*, vol. 2, no. 2, pp. 120–128, May 2006.
- [5] S. Eberle, “Adaptive Internet Integration of Field Bus Systems,” *IEEE Trans. Industrial Informatics*, vol.3, no.1, pp. 12–20, Feb. 2007
- [6] ODVA, *CIP Network Specifications*, <http://www.odva.org/>
- [7] OPC Foundation, “OPC – making the fieldbus interface transparent,” white paper, <http://www.opcfoundation.org/SiteMap.aspx?MID=Downloads>
- [8] N. Komoda, “Service Oriented Architecture (SOA) in Industrial Systems,” in *Proc. IEEE International Conference on Industrial Informatics*, Aug. 2006, pp. 1–5.
- [9] M. Potts, I. Sedukhin, and H. Kreger, “Web services manageability – concepts (WS-manageability),” white paper, Sept. 2003. <http://xml.coverpages.org/WS-Manageability-ConceptsV10.pdf>
- [10] OASIS, *Web Services Distributed Management: Management Using Web Services (MUWS 1.1) Part 1*, OASIS Standard, Aug. 2006.
- [11] K. Fujii and T. Suda, “Semantics-based dynamic service composition,” *IEEE J. Sel. Areas Commun.*, vol. 23, pp. 2361–2372, Dec. 2005.
- [12] K. J. Ma, “Web services: what’s real and what’s not?” *IT Pro*, vol. 7, pp. 14–21, Mar.-Apr. 2005.
- [13] J. Bih, “Deploy XML-based network management approach,” *IEEE Potentials*, vol. 24, pp. 26–31, Oct.-Nov. 2005.
- [14] IONA, “CORBA and a service-oriented architecture (SOA): A CORBA integration planning guide,” white paper, <http://www.iona.com/whitepapers/CORBA%20and%20SOA.pdf>
- [15] Borland Corp., *VisiBroker for C++ 4.5 Information Resource, Programmers’ Guide*
- [16] Y. Jiang, C. Lin and J. Wu, “Research on performance evaluation for packet scheduling algorithms,” in *Joint 4th IEEE International Conference on ATM (ICATM 2001) and High Speed Intelligent Internet Symposium*, pp.181–185, 2001
- [17] Cisco Systems, “Cisco application-oriented networking,” white paper, http://www.cisco.com/application/pdf/en/us/guest/products/ps6438/c1650/cdecont_0900aecd802c1f9c.pdf