In compliance with the
Canadian Privacy Legislation
some supporting forms
may have been removed from
this dissertation.

While these forms may be included
in the document page count,
their removal does not represent
any loss of content from the dissertation.

Performance of CORBA-Based Parallel Architectures

by

Ming Huo, B. Eng.

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Applied Science

Ottawa-Carleton Institute for Electrical Engineering

Faculty of Engineering

Department of Systems and Computer Engineering

Carleton University

Ottawa, Ontario, Canada, K1S 5B6

August 18, 2003

© Copyright 2003, Ming Huo

National Library
of Canada

Bibliothèque nationale
du Canada

Acquisitions and
Bibliographic Services

Acquisitions et
services bibliographiques

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-612-88876-2

Our file Notre référence

ISBN: 0-612-88876-2

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

Canada

*To My Parents Jianyu, Xin,
and My Wife Yifu*

Abstract

Distributed Computing Systems have shown advantages over centralized systems with regard to reliability, scalability, extensibility, high performance and cost effectiveness. However, the heterogeneous environment inherent in Distributed Computing Systems is a major challenge for software developers. With wireless data computing, this challenge is even greater, due to the variety of mobile devices, and differing access technologies and communication protocols. Middleware provides a good solution for distributed systems in terms of interoperability, compatibility, and easy application development in wired and wireless networks. However, middleware often adds system overheads that can be detrimental to system performance.

To satisfy the performance requirements of wired and wireless data computing, three new CORBA-based parallel client-server architectures are investigated in this thesis: the Handle-driven Process Planner Architecture (HPP), the Forwarding Process Planner Architecture (FPP), and the Synchronous Process Planner Architecture (SPP). This thesis investigates the relative performances of these three architectures, as well as the traditional sequential architecture in wired and wireless environments. This research thoroughly analyzes the impact of multithreading, network bandwidth, message size, server demands, and the wireless link packet drop rate on the latency and scalability attributes of the three architectures.

Acknowledgments

I would like to express my sincere thanks to my supervisor, Professor Shikharesh Majumdar, for his seasoned guidance, valuable advices, and strong support over the past two years.

I would also like to convey my thanks to the support from my RADS lab colleagues, especially Changshuang, Umar, and system administrators, Naren, John. At the same time, I am also thankful to all those who have helped me in this research. We are grateful to IONA for providing a free evaluation copy of the middleware used in the research.

Last but not least, I would like to give my deep appreciation and thanks to my wife, Yifu Li, for her support and encouragement without which this research would not have been possible.

Table of Contents

List of Figures.....	viii
List of Tables.....	xiii
Glossary of Terms and Symbols	xvii
 Chapter 1 Introduction.....	 1
1.1 Background	1
1.2 Motivation	3
1.3 Goals of this Research.....	4
1.4 Thesis Contributions.....	5
1.5 Thesis Outline.....	6
 Chapter 2 CORBA Overview.....	 7
2.1 Introduction	7
2.2 The Structure of the Object Request Broker	8
2.2.1 Object Request Broker.....	9
2.2.2 Clients and Servers	10
2.2.3 Object References	11
2.2.4 OMG Interface Definition Language and Language Mapping.....	11
2.2.5 IDL Stub and Skeleton.....	13
2.2.6 Dynamic Invocation Interface (DII)	14
2.2.7 Dynamic Skeleton Interface (DSI)	14
2.2.8 Object Adapter (OA).....	15
2.2.9 ORB Interface	16
2.2.10 Interface Repository.....	16
2.2.11 Implementation Repository.....	17
2.2.12 Inter-ORB Protocols	17
2.2.13 CORBA Naming Service	18
2.3 Data Parallel CORBA	19
2.3.1 Data Parallel CORBA Overview	19
2.3.2 Programming Model.....	20
2.4 Wireless Access and Terminal Mobility in CORBA	21
2.5 Related Work	22
2.5.1 Wireless CORBA.....	23
2.5.2 Parallel CORBA	25
 Chapter 3 The Parallel Interaction Architectures	 27
3.1 Introduction	27

3.2 CORBA Application Development in C++	27
3.3 The Parallel Architectures	28
3.4 Handle-Driven Process Planner	30
3.4.1 The HPP Interaction Architecture	30
3.4.1.1 The Client	32
3.4.1.2 The HPP-daemon	33
3.4.1.3 The Server	33
3.5 Forwarding Process Planner	34
3.5.1 The FPP Interaction Architecture	34
3.5.1.1 The Client	36
3.5.1.2 The FPP-daemon	37
3.5.1.3 The Server	37
3.6 Synchronous Process Planner	38
3.6.1 The SPP Interaction Architecture	38
3.6.1.1 The Client	40
3.6.1.2 The SPP-daemon	40
3.6.1.3 The Server	41
Chapter 4 Instrumentation and Performance Measurement.....	43
4.1 The Workload	43
4.2 The Experimental Environment	46
4.2.1 NIST Net	46
4.2.2 Nodes in the Prototypes	47
4.2.3 The Request Paths	49
4.2.3.1 The HPP System Request Paths	49
4.2.3.2 The FPP System Request Paths	52
4.2.3.3 The SPP System Request Paths	54
4.2.3.4 The Traditional Sequential System Request Paths	57
4.3 Results of Experiments	59
4.3.1 Performance Measures	60
4.3.2 CPU and Elapsed Time Measurements	61
4.3.3 Length of Experiments	62
4.3.4 Data Collection and Logging	62
4.3.5 Validity of the Experiment Results	63
Chapter 5 System Performance Comparison	65
5.1 Effect of Multi-Threading	66
5.1.1 Single Threaded Systems	66
5.1.2 Multi-Threaded Systems	74
5.2 Effect of Service Times	77

5.2.1 Wireless Client Only Mode.....	78
5.2.2 All Wireless Mode	83
5.3 Effect of Message Length.....	83
5.3.1 Wireless Client Only Mode.....	85
5.3.2 All Wireless Mode	89
5.4 Effect of Network Bandwidth	94
5.4.1 All Wired Mode and Wireless Client Only Mode.....	95
5.4.2 All Wired Mode and All Wireless Mode.....	98
5.5 Effect of Drop Rate	100
5.5.1 Wireless Client Only Mode.....	101
5.5.2 All Wireless Mode	104
Chapter 6 Conclusions	106
6.1 Introduction	106
6.2 Summary	106
6.2.1 The Impact of Multi-Threading	106
6.2.2 The Impact of Service Demands.....	107
6.2.3 The Impact of Message Length.....	108
6.2.4 The Impact of Network Bandwidth	108
6.2.5 The Impact of Drop Rate	109
6.3 Conclusions	110
6.4 Future Work.....	111
References:	113
Appendix A: Supplementary Results of the Experiments Investigating the Effect of Service Demand	118
Appendix B: Supplementary Results of the Experiments Investigating the Effect of Message Length	122
Appendix C: Supplementary Results of the Experiments Investigating the Effect of Network Bandwidth	125
Appendix D: Supplementary Results of the Experiments Investigating the Effect of Drop Rate.....	128

List of Figures

Figure 2 - 1: The Structure of Object Request Interfaces	9
Figure 2 - 2: Operations on Parallel Objects.....	21
Figure 3 - 1: Composite Request and its Constituent Components	29
Figure 3 - 2: High-level Interaction Graph for the HPP Architecture.....	31
Figure 3 - 3: High-level Interaction Graph for the FPP Architecture	35
Figure 3 - 4: High-level Interaction Graph for the SPP Architecture	39
Figure 4 - 1: The HPP System Request Path	51
Figure 4 - 2: The FPP System Request Path	53
Figure 4 - 3: The SPP System Request Path	56
Figure 4 - 4: The Traditional Sequential System Request Path	58
Figure 4 - 5: Determination of the Run Length for Experiments	62
Figure 5 - 1: The Response Time of the Single Threaded Systems with Service Time 50, 55 ms in the All Wireless Mode.....	69
Figure 5 - 2: The Throughput of the Single Threaded Systems with Service Time 50, 55 ms in the All Wireless Mode.....	70
Figure 5 - 3: CPU and Process Utilization of the Single Threaded SPP System in the All Wireless Mode	72
Figure 5 - 4: CPU and Process Utilization of the Single Threaded HPP System in the	

All Wireless Mode	72
Figure 5 - 5: CPU and Process Utilization of the Single Threaded FPP System in the	
All Wireless Mode	73
Figure 5 - 6: CPU and Process Utilization of the Sequential System in the All Wireless	
Mode	73
Figure 5 - 7: The Response Time of the Multi-Threaded Systems with Service Time 50,	
55 ms in the All Wireless Mode	75
Figure 5 - 8: The Throughput of the Multi-Threaded Systems with Service Time 50, 55	
ms in the All Wireless Mode	75
Figure 5 - 9: CPU and Process Utilization of the Multi-Threaded SPP System with	
Service Time 50, 55 ms in the All Wireless Mode	76
Figure 5 - 10: The Response Time of the Multi-Threaded Systems with Service Time	
10, 11 ms in the Wireless Client Only Mode	79
Figure 5 - 11: The Throughput of the Multi-Threaded Systems with Service Time 10,	
11 ms in the Wireless Client Only Mode	80
Figure 5 - 12: The Response Time of the Multi-Threaded Systems with Service Time	
50, 55 ms in the Wireless Client Only Mode	82
Figure 5 - 13: The Throughput of the Multi-Threaded Systems with Service Time 50,	
55 ms in the Wireless Client Only Mode	82
Figure 5 - 14: The Response Time of the Multi-Threaded Systems with L=10 KB in	

the Wireless Client Only Mode.....	87
Figure 5 - 15: The Throughput of the Multi-Threaded Systems with L=10 KB in the Wireless Client Only Mode.....	87
Figure 5 - 16: The Response Time of the Multi-Threaded Systems with L=10 KB in the All Wireless Mode.....	92
Figure 5 - 17: The Throughput of the Multi-Threaded Systems with L=10 KB in the All Wireless Mode	93
Figure 5 - 18: CPU and Process Utilization of the Multi-Threaded Systems with L=10 KB in the All Wireless Mode.....	93
Figure 5 - 19: The Response Time of the Multi-Threaded Systems with Bw=100 Mbps in the All Wired Mode	97
Figure 5 - 20: The Throughput of the Multi-Threaded Systems with Bw=100 Mbps in the All Wired Mode.....	97
Figure 5 - 21: The Response Time of the Multi-Threaded Systems with D=2% in the Wireless Client Only Mode.....	103
Figure 5 - 22: The Throughput of the Multi-Threaded Systems with D=2% in the Wireless Client Only Mode.....	103
Figure A - 1: The Response Time of the Multi-Threaded Systems with Service Time 100, 110 ms in the Wireless Client Only Mode	119
Figure A - 2: The Throughput of the Multi-Threaded Systems with Service Time 100,	

110 ms in the Wireless Client Only Mode	119
Figure A - 3: The Response Time of the Multi-Threaded Systems with Service Time 10,	
11 ms in the All Wireless Mode	120
Figure A - 4: The Throughput of the Multi-Threaded Systems with Service Time 10,	
11 ms in the All Wireless Mode	120
Figure A - 5: The Response Time of the Multi-Threaded Systems with Service Time	
100, 110 ms in the All Wireless Mode	121
Figure A - 6: The Throughput of the Multi-Threaded Systems with Service Time 100,	
110 ms in the All Wireless Mode	121
Figure B - 1: The Response Time of the Multi-Threaded Systems with L=1500 Bytes	
in the Wireless Client Only Mode.....	123
Figure B - 2: The Throughput of the Multi-Threaded Systems with L=1500 Bytes in	
the Wireless Client Only Mode.....	123
Figure B - 3: The Response Time of the Multi-Threaded Systems with L=1500 Bytes	
in the All Wireless Mode	124
Figure B - 4: The Throughput of the Multi-Threaded Systems with L=1500 Bytes in	
the All Wireless Mode.....	124
Figure C - 1: The Response Time of the Multi-Threaded Systems with Bw=10 Mbps in	
the Wireless Client Only Mode.....	126
Figure C - 2: The Throughput of the Multi-Threaded Systems with Bw=10 Mbps in	

the Wireless Client Only Mode.....	126
Figure C - 3: The Response Time of the Multi-Threaded Systems with Bw=10 Mbps in the All Wireless Mode.....	127
Figure C - 4: The Throughput of the Multi-Threaded Systems with Bw=10 Mbps in the All Wireless Mode.....	127
Figure D - 1: The Response Time of the Multi-Threaded Systems with D=2% in the All Wireless Mode	129
Figure D - 2: The Throughput of the Multi-Threaded Systems with D=2% in the All Wireless Mode	129

List of Tables

Table 4 - 1: Levels for Workload Parameters	45
Table 5 - 1: Parameters of the Experiment Investigating the Effect of Multi-Threading	66
Table 5 - 2: Latency of Single Threaded Systems at N=1 (L=50 bytes, D=0%, S _A =S _C =50 ms, S _B =S _D =55 ms)	67
Table 5 - 3: Number of Message Exchanges per Client Cycle	68
Table 5 - 4: System Throughput of Single Threaded Systems at N=10 (L=50 bytes, D=0%, S _A =S _C =50 ms, S _B =S _D =55 ms)	74
Table 5 - 5: Latency of Multi-Threaded systems at N=1 (L=50 bytes, D=0%, S _A =S _C =50 ms, S _B =S _D =55 ms)	76
Table 5 - 6: System Throughput of Multi-Threaded Systems at N=10 (L=50 bytes, D=0%, S _A =S _C =50 ms, S _B =S _D =55 ms)	77
Table 5 - 7: Parameters of the Experiment Investigating the Effect of Service Demands	78
Table 5 - 8: Effect of Server Demand on Latency at N=1 (L=50 bytes, Wireless Bw=1 Mbps, Wired Bw=100 Mbps, D=0%, Multi-PP-daemon, Wireless Client Only Mode).....	79
Table 5 - 9: Effect of Server Demand on System Throughput at N=10 (L=50 bytes, Wireless Bw=1 Mbps, Wired Bw=100 Mbps, D=0%, Multi-PP-daemon, Wireless Client Only Mode)	81

Table 5 - 10: Effect of Server Demand on Latency at N=1 (L=50 bytes, Wireless Bw=1 Mbps, D=0%, Multi-PP-daemon, All Wireless Mode).....	84
Table 5 - 11: Effect of Server Demand on System Throughput at N=10 (L=50 bytes, Wireless Bw=1 Mbps, D=0%, Multi-PP-daemon, All Wireless Mode)	84
Table 5 - 12: Parameters of the Experiment Investigating the Effect of Message Length	85
Table 5 - 13: Effect of Message Length on Latency at N=1 ($S_A=S_C=50$ ms, $S_B=S_D=55$ ms, Wireless Bw=1 Mbps, Wired Bw=100 Mbps, D=0%, Multi-PP-daemon, Wireless Client Only Mode)	86
Table 5 - 14: Effect of Message Length on System Throughput at N=10 ($S_A=S_C=50$ ms, $S_B=S_D=55$ ms, Wireless Bw=1 Mbps, Wired Bw=100 Mbps, D=0%, Multi-PP-daemon, Wireless Client Only Mode).....	88
Table 5 - 15: Effect of Message Length on Latency at N=1 ($S_A=S_C=50$ ms, $S_B=S_D=55$ ms, Wireless Bw=1 Mbps, D=0%, Multi-PP-daemon, All Wireless Mode).....	90
Table 5 - 16: Effect of Message Length on System Throughput at N=10 ($S_A=S_C=50$ ms, $S_B=S_D=55$ ms, Wireless Bw=1 Mbps, D=0%, Multi-PP-daemon, All Wireless Mode).....	90
Table 5 - 17: Parameters of the Experiment Investigating the Effect of Network Bandwidth.....	94
Table 5 - 18: Effect of Network Bandwidth on Latency at N=1 ($S_A=S_C=50$ ms, $S_B=S_D=55$ ms, L=50 Bytes, D=0%, Multi-PP-daemon, All Wired Mode and	

Wireless Client Only Mode)	95
Table 5 - 19: Effect of Network Bandwidth on System Throughput at N=10 ($S_A=S_C=50$ ms, $S_B=S_D=55$ ms, L=50 Bytes, D=0%, Multi-PP-daemon, All Wired Mode and Wireless Client Only Mode)	96
Table 5 - 20: Effect of Network Bandwidth on Latency at N=1 ($S_A=S_C=50$ ms, $S_B=S_D=55$ ms, L=50 Bytes, D=0%, Multi-PP-daemon, All Wired Mode and All Wireless Mode)	99
Table 5 - 21: Effect of Network Bandwidth on System Throughput at N=10 ($S_A=S_C=50$ ms, $S_B=S_D=55$ ms, L=50 Bytes, D=0%, Multi-PP-daemon, All Wired Mode and All Wireless Mode)	99
Table 5 - 22: Parameters of the Experiment Investigating the Effect of the Drop Rate	101
Table 5 - 23: Effect of Drop Rate on Latency at N=1 ($S_A=S_C=50$ ms, $S_B=S_D=55$ ms, Wireless Bw=1 Mbps, Wired Bw=100 Mbps, L=50 Bytes, Multi-PP-daemon, Wireless Client Only Mode)	102
Table 5 - 24: Effect of Drop Rate on System Throughput at N=10 ($S_A=S_C=50$ ms, $S_B=S_D=55$ ms, Wireless Bw=1 Mbps, Wired Bw=100 Mbps, L=50 Bytes, Multi-PP-daemon, Wireless Client Only Mode)	104
Table 5 - 25: Effect of Drop Rate on Latency at N=1 ($S_A=S_C=50$ ms, $S_B=S_D=55$ ms, Wireless Bw=1 Mbps, L=50 Bytes, Multi-PP-daemon, All Wireless Mode)	105
Table 5 - 26: Effect of Drop Rate on System Throughput at N=10 ($S_A=S_C=50$ ms,	

$S_B=S_D=55$ ms, Wireless Bw=1 Mbps, L=50 Bytes, Multi-PP-daemon, All Wireless

Mode)..... 105

Glossary of Terms and Symbols

API	Application Programming Interface
CDR	Common Data Representation
CORBA	Common Object Request Broker Architecture
DCE	Distributed Computing Environment
DII	Dynamic Invocation Interface
DOC	Distributed Object Computing
DSI	Dynamic Skeleton Interface
FPP	Forwarding Process Planner
GIOP	General Inter-ORB Protocol
GPRS	General Packet Radio Service
HPP	Handle-Driven Process Planner
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
IOR	Interoperable Object Reference
IP	Internet Protocol
IR	Interface Repository
LW-IOP	Light Weight Inter-ORB Protocol
OA	Object Adapter

OMA	Object Management Architecture
OMG	Object Management Group
ORB	Object Request Broker
PDA	Personal Digital Assistant
POA	Portable Object Adapter
POM	Parallel Object Manager
SII	Static Invocation Interface
SPMD	Single Program Multiple Data
SPP	Synchronous Process Planner
TCP	Transmission Control Protocol

Chapter 1 Introduction

1.1 Background

Distributed Computing Systems have shown advantages over centralized systems with regard to reliability, scalability, extensibility, high performance and cost effectiveness.

However, the heterogeneous environment inherent in Distributed Computing Systems is a major challenge for software developers. With wireless data computing, this challenge is even greater, due to the variety of mobile devices, and differing access technologies and communication protocols. Middleware provides a good solution for distributed systems in terms of interoperability, compatibility, and easy application development in wired and wireless networks. For example, with the help of middleware, it is possible to have different applications running on different Operating Systems to access the same computing service provided by a robust server.

Distributed Object Computing (DOC) has been widely adopted in distributed systems due to advantages such as data encapsulation and abstraction, reusability, extensibility, and portability [SCH-95]. CORBA stands for Common Object Request Broker Architecture, proposed by the Object Management Group (OMG). CORBA adopted the Distributed Object Computing concept and is an Object-Oriented Middleware that resides between the application and the operating system. CORBA can provide

interoperability and location transparency services to clients in a heterogeneous environment, through a special entity called the Object Request Broker (ORB). ORB provides a variety of services, such as name-to-address mapping, load balancing, fault tolerance, and so on.

The unprecedented successes of the Internet and wireless communication in the 1990s, along with the emergence of more and more portable devices, such as laptops, palmtops, personal digital assistants (PDAs), have increased the research, development and implementation of the wireless broadband networks technology, focusing on wireless packets data communication with much higher bandwidth than traditional GPRS data communication. This leap makes wireless data computing over high-speed wireless networks a reality for the world of business and industry. As today's Distributed Computing Systems already prevail within wired networks, its feasibility for wireless data computing merits a thorough investigation.

Although middleware provides interoperability, it often adds system overheads that can be detrimental to system performance. This thesis focuses on using parallel processing techniques for achieving high performance CORBA-based systems. Particular attention is paid to systems characterized by wireless links.

1.2 Motivation

As today's wireless broadband networks evolve into a new era, wireless data computing can be expected in the near future. The main concerns of wireless data computing include its feasibility under a restrained wireless environment, such as lower bandwidth and higher packet drop rate over the wireless link; a less powerful CPU; limited memory; limited battery life in mobile devices, etc. To satisfy the performance requirements of wireless data computing, three new CORBA-based parallel client-server architectures are investigated in this thesis.

Based on the CORBA-based client-server architecture, namely the Process Planner ORB (P-ORB), investigated in Abdul-Fatah's thesis [ABD-97], this research suggests three new CORBA-based parallel client-server architectures: the Handle-driven Process Planner Architecture (HPP), the Forwarding Process Planner Architecture (FPP), and the Synchronous Process Planner Architecture (SPP). This thesis investigates the relative performances of these three client-server architectures, as well as the traditional sequential architecture in the wired and wireless environments. This research reports on the impact of different system parameters on the performances of the three architectures. Performance prototypes of the three architectures are implemented using Orbix E2A Application Server Platform 5.1, a commercially available and widely used CORBA 2.3-compliant middleware from IONA. The

prototype is executed on a set of Pentium II and Pentium IV machines running Red Hat Linux 6.0 and 7.2.

1.3 Goals of this Research

The main goal of this research is to investigate the suggested three CORBA-based parallel client-server architectures: the Handle-driven Process Planner Architecture (HPP), the Forwarding Process Planner Architecture (FPP), and the Synchronous Process Planner Architecture (SPP), measure their performances and analyze the results of the experiments. The relative performances of these architectures were thoroughly investigated under different system and workload conditions. This thesis is motivated by the following questions:

- a) Are CORBA-based parallel architectures feasible in the wireless environment?
- b) Which architecture best fits within the limitations of the wireless networks?
In particular, what are the impacts of the wireless link drop rate and wireless link bandwidth on system performance?
- c) What effect does the size of the messages exchanged between the client and the server have on system performance?
- d) Which architecture performs best under different system and workload conditions?

- e) Are the three architectures scalable?
- f) Where are the performance bottlenecks in the different architectures? What are the methods for handling such bottlenecks?
- g) How does multithreading affect system performance?

1.4 Thesis Contributions

The main contributions of this research include the following:

- The design and implementation of the performance prototypes of the three CORBA-based parallel architectures, based on Orbix E2A ASP 5.1.
- A thorough investigation of the performances of the three parallel architectures and the traditional sequential architecture in wired and wireless environment. To the best of our knowledge, this is one of the first works in wireless parallel middleware systems.
- Making clear the observation from the experiments that the multithreaded SPP architecture excels in most wired and wireless environments, and is favored over the traditional Sequential architecture.
- An analysis in detail of the effect of the multithreading technique and of different workload parameters on the performances of the three parallel architectures.
- Demonstrating that the three parallel architectures can be implemented, even

without access to the source code of the middleware product.

1.5 Thesis Outline

This thesis is composed of six chapters. Chapter 2 introduces CORBA and its components, programming techniques in CORBA, and CORBA in wireless environments. The concept and design of the three parallel architectures, and issues related to their implementation, are described in detail in Chapter 3. Chapter 4 discusses the experiments carried out, and the performance measurement techniques used. Chapter 5 provides a thorough comparison and analysis of the experimental results. Conclusions and suggestions for future research are presented in Chapter 6.

Chapter 2 CORBA Overview

2.1 Introduction

Heterogeneity is the nature of distributed computer systems. The variety of computer hardware architectures, operating systems and network communication protocols makes real world application programming extremely difficult. In addition, the use of different software developing tools and libraries make the applications themselves heterogeneous [HEN-99]. The Object Management Group (OMG) was formed in 1989 to help develop portable and reusable applications for heterogeneous distributed systems.

OMG proposed and maintained the CORBA specification, the latest version of which is 3.02 [OMG-02]. CORBA offers a homogeneous application development layer to hide the lower level heterogeneous network details from the application developers, so the developers can focus on their immediate problems. CORBA provides an extensive set of flexible abstractions and important distributed services to address the problems of distributed computing.

This chapter will begin with a high level overview of the CORBA model and the components. This is followed by a brief introduction of Data Parallel CORBA and

Wireless Access and Terminal Mobility in CORBA specifications. A survey of the current CORBA-related literature in parallel computing, and in the wired and wireless environments, is presented in the remainder of the chapter.

2.2 The Structure of the Object Request Broker

The Common Object Request Broker Architecture (CORBA) has the following main components: OMG Interface Definition Language (IDL), language mappings, ORB Core, ORB Interfaces, Object Adapters and an Inter-ORB protocol. They will be described in the following sections.

Figure 2-1 [OMG-02] shows the structure of the Object Request Broker and the relationships between each component. In this figure, the client and the server are located in different nodes, connected by a network. The client can request services from the server either through the Dynamic Invocation Interface (DII) or the IDL-specific stub, or the client can request services from the ORB through the ORB Interface. The server can receive the client's request either through the Static IDL Skeleton or the Dynamic Skeleton Interface (DSI). The server may also request services from the ORB through the Object Adaptor (OA), while performing the request from the client or at any other time.

Figure 2 - 1: The Structure of Object Request Interfaces

2.2.1 Object Request Broker

The ORB is a congregation of interface implementations. ORB implementations have many choices, which are compliant with the CORBA specification, as long as it can offer a set of services of different properties and qualities to the clients and the servers. The client may access the services from different ORB implementations simultaneously. With the help of the ORB, the client can communicate with the server without knowing where the server is located, in which programming language the

server is implemented, or what state the server is in. All the client needs to know is the Object Reference of the targeting server. Thus, location transparency, program language independence and platform independence can be realized in complex distributed computer systems.

The ORB Core is the part of the ORB which provides the communication infrastructures for the other ORB components, the clients, and the servers. The ORB components above the ORB Core encapsulate the differences between the ORB Cores to ease the pressure on application developers. In addition, increased efficiency and reusability of the software can be gained through the use of pluggable standard software modules.

2.2.2 Clients and Servers

A client is an object which invokes operations in a server. In CORBA, the client only knows the logical structure of the server through its interface, and experiences its services through invocations. The client may also act as a server when it provides services to other objects.

A server provides the semantics of the object, usually by defining the object instance data and the object's method code. It is possible to support any style of object

implementation through the use of additional object adapters. Servers may be implemented independent of ORB or the client invocations. The server may select a different Object Adapter to access ORB-dependent services.

2.2.3 Object References

The following description of object reference is obtained from [OMG-02]. “An Object Reference is the information needed to specify an object within an ORB. Both clients and servers have an opaque notion of object references according to the language mapping, and thus are insulated from the actual representation of them. Two ORB implementations may differ in their choice of Object Reference representations”.

2.2.4 OMG Interface Definition Language and Language Mapping

Object interfaces are defined in the OMG Interface Definition Language in CORBA. A set of operations, and the parameters for these operations, comprise an interface. The operations and the objects cannot be implemented in IDL, because IDL is not a programming language. The interfaces written in IDL are independent of any programming language; thus it is possible for the applications implemented in different programming languages to interoperate. CORBA, by means of IDL, supports heterogeneous distributed systems and integrates the separately developed applications.

A sample IDL definition is shown below:

```
Interface GPA_Inquiry {  
  
    double Get_GPA(in long number);  
  
};
```

This example defines an interface named `GPA_Inquiry`, which contains an operation named `Get_GPA`. The `Get_GPA` operation takes a student number as an input argument and returns a double variable as the student's GPA. The parameters of IDL operations must have *in*, *out* or *inout* as their direction declarations, so the ORB knows whether to send their values to the target server, vice versa or both. The keyword *in* indicates that the student number argument should be sent from the client to the target server. While the *out* keyword indicates that the arguments, like the return value, should be sent from the target server back to the client, the *inout* keyword specifies that the client initializes the argument and sends to the server. The server then modifies it and sends it back to the client.

IDL interfaces can inherit from one or more other interfaces. All IDL interfaces implicitly inherit from the `Object` interface, defined in the CORBA module. The `Object` interface provides operations common to all CORBA objects.

Language Mappings define how to translate the IDL into different programming languages. The OMG now provides standard language mappings for C, C++, Java, Smalltalk, COBOL, and Ada. These mappings enable the developers to implement different parts of a distributed system in different programming languages. For example, the server can be written in C++ for efficiency and the client can be written as Java applets for easy access from the Web.

2.2.5 IDL Stub and Skeleton

Usually, the client stubs and the server skeletons are generated by the IDL compiler after the IDL source code has been compiled. They are language-specific and are compiled into the applications. They give the applications static knowledge of the programming language types and the operations mapped from the IDL description of the remote objects.

A stub is a client-side method that allows the remote method invocation to be made like a normal local method call. The stubs make calls on the rest of the ORB using the interfaces of the particular ORB Core. The stubs usually have marshalling /unmarshalling functions for the remote operation parameters and results, as well as exception handling mechanisms [ABD-97].

A skeleton, like a stub, is a server-side method that allows the remote method invocation to be received by a server and dispatched to the appropriate servant. The skeleton is generally an up-call interface, in that the object implementation develops methods that conform to the interface and the ORB calls the methods through the skeleton [OMG-02]. Similarly, the skeleton usually also has marshalling/unmarshalling functions for the remote operation parameters and results, as well as exception-handling mechanisms [ABD-97].

2.2.6 Dynamic Invocation Interface (DII)

Instead of remote method invocation through the stubs, the client can perform a dynamic construction of object invocations through the Dynamic Invocation Interface (DII). The client must provide information on the operation to be performed and the types of parameters being passed (acquired from an interface repository or some other runtime source) [OMG-02]. Although DII gives the clients ultimate power to get the required services at runtime, it also puts heavy overhead on the system performance. Performing method invocations through the stub/skeleton pairs is far more efficient than through the DII [VIN-93].

2.2.7 Dynamic Skeleton Interface (DSI)

The Dynamic Skeleton Interface handles the dynamic object invocations for a server.

That is, instead of having a skeleton for each of the servants in a server, all the servants in the server can be reached through the DSI in a manner similar to the client side's Dynamic Invocation Interface. In DSI, the static knowledge of the methods may be used, or the dynamic knowledge (obtained from an Interface Repository) may also be used. The DSI is typically an up-call interface.

Dynamic skeletons may be invoked through both the client stubs and the dynamic invocation interface. Either type of client request construction interface returns identical results to the client. The introduction of DSI in CORBA 2.0 is mainly to support gateways between ORBs, utilizing different communication protocols [VIN-97].

2.2.8 Object Adapter (OA)

The servers access services provided by the ORB primarily through the object adapters. A few object adapters, with interfaces that are appropriate for specific kinds of objects, are widely available. The services provided by the ORB through the object adapters include the following:

- Object and implementation activation and deactivation
- Registration of implementations
- Object reference generation and interpretation

- Method invocation
- Object reference to implementation mapping
- Interaction security
- Demultiplexing requests

CORBA specified the Basic Object Adapter (BOA) in version 2.1 and the earlier versions. It has specified the Portable Object Adapter (POA) in version 2.2 to replace the BOA. The POA is now the most widely used object adapter, due to its full portability and interoperability.

2.2.9 ORB Interface

The ORB Interface goes directly to the ORB and is the same for all ORBs. The ORB Interface does not depend on the object's interface or object adapter. The ORB Interface provides access to just a few operations, common across all objects, which are useful to both clients and servers. Most of the ORB's services are provided through the object adapter, skeleton, stub, or dynamic invocation.

2.2.10 Interface Repository

The Interface Repository is a service which provides IDL information at run-time. The ORB, the client and the server may request information from the Interface Repository for an object which is unknown at compile time, ascertain information on the valid

operations on the object, and perform invocations on them. The Interface Repository is also a common place to store additional information related to the interfaces of ORB objects, like debugging information, libraries of stubs or skeletons, etc.

2.2.11 Implementation Repository

The Implementation Repository provides information that the ORB needs to locate and activate object implementations. Most information in the Implementation Repository is ORB or operating environment-specific. The Implementation Repository is also a common place for storing additional information related to ORB object implementations, such as debugging information, administrative control, resource allocation, security, and so on.

2.2.12 Inter-ORB Protocols

To allow the ORBs from different vendors to interoperate with each other, CORBA 2.0 introduced and defined the General Inter-ORB Protocol (GIOP) and the Internet Inter-ORB Protocol (IIOP). GIOP defines the transfer syntax and a standard set of message formats for inter-ORB communication over any connection-oriented transport [OMG-02]. IIOP defines how to implement GIOP over Transmission Control Protocol/Internet Protocol (TCP/IP). GIOP specifies the following three elements:

- Common Data Representation (CDR) is the transfer syntax that maps the OMG IDL types from the native end system format to a bi-canonical (supports both little-endian and big-endian) format used to transfer data over networks.
- Seven GIOP Message Formats for locating objects, managing channels, sending requests, and receiving results.
- GIOP transport assumptions name types of transport protocols that may carry GIOP messages.

To enable ORB interoperability, the standard object reference format is defined, and is known as the Interoperable Object Reference (IOR). IORs contain information for ORBs to establish communications between clients and the target objects. They are opaque to applications [HEN-99].

2.2.13 CORBA Naming Service

The CORBA Naming Service is one of the basic standardized CORBA services. It provides a name-to-object reference mapping, so that the service returns an object reference stored under the given name upon inquiry.

The advantages of the Naming Service are as follows [HEN-99]:

- The applications do not need to deal with the object reference changes in their

code. They just use the object name to get the most recent object reference from the Naming Service, regardless of object implementation changes.

- The developers can use meaningful names for the objects instead of having to deal with the stringified object references.
- The application components can obtain access to the object references for an application from the Naming Service without needing to store them as stringified references in files.

2.3 Data Parallel CORBA

The data parallel CORBA specification defines the architecture for data parallel programming in CORBA and addresses data parallelism, as opposed to pipeline parallelism and functional parallelism in distributed systems [OMG-01].

2.3.1 Data Parallel CORBA Overview

Parallel applications are characterized by a set of processes executing in parallel. Data parallel applications in particular process a large data set that is divided between the participating processes. Data is usually redistributed between a set of sending processes and a set of receiving processes.

Data Parallel CORBA enables developers to take advantage of parallel computing

resources to achieve scalability and high performance. Traditional clients may use parallel objects transparently and efficiently without direct support from the Data Parallel CORBA [OMG-01]. A parallel object is a set of partial implementations executing in parallel. Data and request partitioning and distribution are defined in order to make use of parallel objects.

2.3.2 Programming Model

Figure 2-2 [OMG-01] shows the typical parallel ORB interactions. Here, the parallel ORB is an ORB that supports the Data Parallel CORBA specification, as opposed to non-parallel ORBs, which do not. There are three types of objects. Singular objects are traditional CORBA objects which process requests in sequence. Parallel objects can carry out the requests of one or more “parts”, and process them concurrently. The Part Objects are parts of a parallel object and work together to process requests for the parallel object.

There are two types of clients. Parallel clients are virtual CORBA clients that execute associated with some parallel objects. Parallel clients can only run on the parallel ORB. Singular clients are traditional CORBA clients without awareness of parallel ORB or parallel objects, but can receive reference to and invoke operations on parallel objects transparently.

Figure 2 - 2: Operations on Parallel Objects

Data parallelism is exploited typically in scientific applications processing large data sets. This thesis focuses on exploiting functional parallelism for improving system performance that can be used in the context of a broad range of systems.

2.4 Wireless Access and Terminal Mobility in CORBA

The Wireless Access and Terminal Mobility in the CORBA specification [OMG-03] provides a minimal useful functionality for CORBA applications, in which the client,

the server, or both run on mobile hosts. Mobility management is handled effectively by this specification.

2.5 Related Work

Extensive research has been conducted in the field of the performances of CORBA-based client-server systems. For example, four CORBA interaction architectures are suggested and investigated in [ABD-02]. An adaptive ORB implementation and its performance are described in [SHE-00]. Performance enhancement techniques that exploit limited heterogeneity in CORBA-based systems are discussed in [WU-01] and [AHM-01]. Several multi-threaded architectures and their impacts on performances of CORBA-based systems are presented in [SCH-98].

Although conventional CORBA satisfies the best-effort requirements for the traditional request/response applications, a growing class of real-time systems requires more stringent end-to-end quality-of-service (QoS) support including bandwidth, latency, and priority preservation. To meet this challenge, a Real-Time CORBA (RT CORBA) specification is proposed [OMG-02a]. The main characteristics and requirements of Real-Time CORBA specification are summarized in [SCH-00]. The architecture, the real-time scheduling service and optimizations of a real-time CORBA middleware called TAO are presented in [GIL-01, PYR-00, SCH-97, SCH-98a].

2.5.1 Wireless CORBA

As wireless technologies have continued to evolve rapidly in recent years, mobile computing has become a desirable paradigm for research. The success of CORBA in wired distributed computing motivated a great deal of early research in order to apply it to wireless distributed computing. The primary challenges in the wireless domain are as follows [HAA-01]:

- Limited resources in mobile devices: limited processing power, memory, battery life, etc.
- The wide variation of network characteristics of the mobile host in terms of latency, bandwidth, reliability, and protocols.
- Physical host mobility management: rapid change of object reference.

Most early research focused on the latter two issues. In [MAS-02], researchers carried out a comprehensive survey on mobile computing middleware. Several classes of middleware for mobile computing, including reflective middleware, tuplespace-based middleware, context-aware middleware, and data-oriented middleware, were identified, illustrated and discussed comparatively.

A proxy platform Π^2 for Wireless Access and Mobility in CORBA is proposed and

investigated in [RUG-00a, RUG-00b]. The proxy platform Π^2 acts on behalf of mobile users and is based on special equipment at the borders between the wireless and wired domains.

ALICE (Architecture for Location-Independent CORBA Environments) [HAA-99, HAA-01] introduces mobility gateways to realize connectivity management, location management, and disconnected operations. ALICE uses a session layer-type approach in conjunction with application support [HAA-99].

The DOLMEN (Service Machine Development for an Open Long-term Mobile and Fixed Network Environment), which is the basic framework for Wireless Access and Terminal Mobility in the CORBA specification, introduces interoperability bridges using Light-Weight Inter-ORB Protocol (LW-IOP) [TRI-98, LIL-98]. DOLMEN shows wireless support without the change of CORBA specifications.

A working implementation of Mobile CORBA is demonstrated in [ADW-01]. The researcher found that IIOP-based CORBA systems have very predictable performance. In addition, with higher bandwidth, the performance obtained is comparable to wired CORBA systems.

2.5.2 Parallel CORBA

Computation power-critical tasks like scientific computing and engineering can now take advantage of high performance networking technologies and rely on a cluster of computing resources. CORBA plays an important role in distributed parallel computing.

PARDIS (PARallel DIStributed), based on CORBA, is a system designed to address distributed parallel programming problems [KEA-97]. PARDIS provides a distributed environment in which data parallel objects, called SPMD (Single Program Multiple Data) objects, and non-parallel objects can interact with each other across software systems and platforms [KEA-97].

The CORBA-based Object Group Service and the Join Service are presented in [ALE-00a, ALE-00b, ALE-01]. These two services facilitate the development of CORBA-based, distributed parallel applications where a client makes requests to a set of servers and expects one single result in response.

A multi-threaded architecture built around the event nature of a CORBA invocation is presented in [DHO-01]. This architecture uses a library of classes and allows developers to use method invocations similar to the well known *send()* and *receive()*

message passing calls [DHO-01].

Scalable Coherent Interconnect (SCI) cluster programming [PRI-99], Code Coupling [PRI-00], and Portable Parallel CORBA Objects [DEN-01] are similar progressive approaches to distributed parallel computing. [PRI-99] presents a CORBA-based runtime system to provide resource allocation services for the execution of parallel objects. [PRI-00] modifies the IDL and provides a new IDL compiler to generate stubs and skeletons that manage the collection of objects in a fashion transparent to users. [DEN-01], like PARDIS, introduces SPMD programming into CORBA to realize efficient parallel computing. Data distribution mechanisms within the parallel CORBA objects are discussed in [KAM-00].

Chapter 3 The Parallel Interaction Architectures

3.1 Introduction

This chapter starts with a brief description of the general CORBA application development steps using application development in C++ as an example. Three CORBA-based parallel client-server architectures: the Handle-driven Process Planner Architecture (HPP), the Forwarding Process Planner Architecture (FPP), and the Synchronous Process Planner Architecture (SPP), are then described in detail in the remainder of the chapter.

3.2 CORBA Application Development in C++

A C++ CORBA application usually comprises two executables: a client and a server.

The general development steps include [HEN-99]:

1. Design application's objects and define their interfaces in OMG IDL.

According to the required server functionalities, the relationships of the interfaces are also defined. This is often an iterative process.

2. Use the IDL compiler to compile the designed IDL definitions into client stubs and server skeletons. IDL compilers usually generate C++ header files and the initial C++ implementation files. The C++ header files contain declarations for proxy classes, server skeletons, and other supporting types.

The C++ implementation files implement the classes and types declared in the header files. The developer can write clients and servants based on these generated codes.

3. Declare and implement the C++ servant classes to fulfill the designed functionalities. Some vendors, like IONA, provide an IDL toolkit to generate the declaration of the C++ servant classes as well. The developers just need to implement the generated classes.
4. Write a server main program to initialize the ORB and the POA, and create some servants.
5. Create the server executable by compiling and linking the server implementation files with the IDL-generated stubs and skeletons.
6. Write, compile and link client code with the IDL-generated stubs to create the client executable. The client normally first gets the object references of the servants, then invokes the requests on the remote servers.

These development steps may vary depending on the nature of the application.

3.3 The Parallel Architectures

The parallel architectures investigated in this thesis are explained in the following subsections. These architectures exploit functional parallelism that is often present in a set of requests to different objects issued by a client. For example, a client may invoke

multiple servers to complete a specific *job* or a *composite request*. Some of the server invocations in the composite request may proceed concurrently. This research makes use of a simple composite request, as shown in Figure 3-1 [ABD-97]. This composite request comprises two single service requests: service A and service B. They are independent of each other, and can be executed in parallel. Composite requests with more complex precedence relationships among its components may be used by applications. However, a composite request of the type shown in Figure 3-1 gives rise to concurrency in server executions and is a reasonable choice for investigating the research questions outlined in Section 1.3. Analyzing the impact of more complex composite requests on performance forms an interesting direction for future research.

Figure 3 - 1: Composite Request and its Constituent Components

3.4 Handle-Driven Process Planner

This section contains a discussion of the Handle-Driven Process Planner Architecture (HPP). The Handle-Driven approach is based on the Object Management Group-adopted strategy for the name-to-object reference resolution in the Naming Service. However, the Naming Service does not have support for resolving multiple names in one request. The HPP-daemon provides multiple name resolution in one request and reply, thus eliminating communication traffic between the client and the daemon. This section describes the main components of this architecture and its execution scenario.

3.4.1 The HPP Interaction Architecture

In the Handle-Driven Process Planner Architecture, the HPP-daemon returns multiple service handles to the requesting client. The handles are the object references that contain all the information needed to interact with the servers and the servants for the services sought. The client can use these returned handles to invoke multiple servers in parallel. The servers will send the replies back to the client after performing the services. The client processes the returned results and goes on with its execution. Figure 3-2 shows the general HPP high-level interaction architecture. Note that the clients, the HPP-daemon, and the servers can communicate through networks that interconnect the nodes.

Figure 3 - 2: High-level Interaction Graph for the HPP Architecture

The object references realize the location transparency. For example, the client does not need to know whether the sought server is local (in the same process), on the same computing node, or connected over the network. A detailed description of the behavior

and the roles of the main components of the HPP interaction architecture are presented next.

3.4.1.1 The Client

- 1) The client starts and enters its initialization phase, which includes reading the CORBA domain configuration file, initializing the ORB, activating the POA, getting the HPP-daemon service information, registering with the ORB, and preparing the arguments for the requests, and so on.
- 2) Due to the location transparency policy in CORBA, the client does not know the locations of the sought servers and their object references. The client must contact the HPP-daemon to get the handles of the servers. The client sends a get-handle request, including the multiple required service names to the HPP-daemon, and waits for the reply (see Figure 3-2).
- 3) When the client receives the multiple handles from the HPP-daemon, it then sends multiple service requests to the appropriate servers concurrently, using the returned handles (see Figure 3-2). The client performs the concurrent service invocation by lightweight multithreads.
- 4) Upon receiving all the replies from the sought servers, the client processes the

results and starts another request cycle (goes to step 2).

3.4.1.2 The HPP-daemon

- 1) The HPP-daemon rises and enters its initialization phase, which includes reading the CORBA domain configuration file, initializing the ORB, activating the POA, registering with the ORB, and getting the object references of the managed servers in its domain.
- 2) The HPP-daemon enters a listening mode to wait for requests from the clients.
- 3) When a request from the client arrives, the HPP-daemon locates the servers providing the constituent services, activates the servers if needed and sends the handles of the servers back to the client.
- 4) The HPP-daemon returns to its listening mode and waits for new client requests. Note that the HPP-daemon updates its managed server information at predefined regular intervals of time.

3.4.1.3 The Server

- 1) Similar to the client and HPP-daemon, the server starts and enters its initialization phase, which includes reading the CORBA domain configuration file, initializing the ORB, activating the POA, and registering with the ORB.
- 2) The server enters its listening mode and waits for client requests.

- 3) When a client request arrives, the server executes the requested method. In the performance prototype, the method burns a specific amount of CPU time, according to the argument passed in the client request. After completing the processing, the server returns the result to the client.
- 4) The server returns to its listening mode and waits for new client requests.

3.5 Forwarding Process Planner

This section discusses the Forwarding Process Planner Architecture (FPP). The FPP-daemon provides multiple parallel request forwarding for the clients. This section describes the main components of this architecture and its execution scenario.

3.5.1 The FPP Interaction Architecture

In the Forwarding Process Planner Architecture, the FPP-daemon forwards multiple service requests to the sought servers. The servers process the requests in parallel and return the results to the client directly after performing the service. The client processes the returned results and goes on with its execution. Figure 3-3 shows the general high-level FPP interaction architecture. All the requests are asynchronous, without any return value, and only TCP level acknowledgements are returned. The client's object reference is sent along with the requests to the FPP-daemon and the servers. The servers send the results back to the client via the client's object reference.

The client can do other work while waiting for the results. A detailed description of the proposed behavior and the roles of the main components of the FPP client server architecture are presented.

Figure 3 - 3: High-level Interaction Graph for the FPP Architecture

3.5.1.1 The Client

- 1) Similar to the client in the HPP architecture, the client starts and enters its initialization phase, which includes reading the CORBA domain configuration file, initializing the ORB, activating the POA, getting the FPP-daemon service information, registering with the ORB, and preparing the arguments for the requests. In this architecture, the client also registers itself with the Naming Service to accept the results from the servers.
- 2) Due to location transparency policy in CORBA, the client does not know the locations of the sought servers and their object references. The client sends the composite request to the FPP-daemon. Because the client does not expect a reply from the FPP-daemon, the composite request from the client is a “*send-oneway*” call. The asynchronous “*send-oneway*” call returns control to the client immediately without an application level acknowledgement. The client can perform other work while waiting for the results from the servers.
- 3) The client in the performance prototype, that is aware of the number of results to be returned, periodically checks the availability of the results from the servers. The servers invoke special methods in the client for returning the results. The client sleeps for 0.2 ms between successive checks. This ensures a low CPU overhead and a negligible error in the measurement of response times. When the client has received all the results from the servers, it then processes

the results and starts another request cycle (goes to step 2).

3.5.1.2 The FPP-daemon

- 1) Similar to the HPP-daemon, the FPP-daemon rises and enters its initialization phase, which includes reading the CORBA domain configuration file, initializing the ORB, activating the POA, registering with the ORB, and getting the object references of the managed servers in its domain.
- 2) The FPP-daemon enters a listening mode to wait for the requests from the clients.
- 3) When a request from the client arrives, the FPP-daemon decomposes the composite request into its constituent services. Then the FPP-daemon locates the servers that are providing the constituent services, activates the servers if needed and forwards the constituent service requests to the servers in parallel. Asynchronous “*send-oneway*” calls are also used in this case.
- 4) The FPP-daemon returns to its listening mode and waits for new client requests.

Note that the FPP-daemon updates its managed server information at regular predefined intervals of time.

3.5.1.3 The Server

- 1) Similar to the client and FPP-daemon, the server starts and enters its

initialization phase, which includes reading the CORBA domain configuration file, initializing the ORB, activating the POA, and registering with the ORB.

- 2) The server enters its listening mode and waits for client requests.
- 3) When a client request arrives (forwarded by the FPP-daemon), the server executes the requested method. In the performance prototype, the method burns a specific amount of CPU time, according to the argument passed in the client request. Once the processing has been completed, the server returns the result to the client by invoking the special method for receiving results in the client. Asynchronous “*send-oneway*” calls are also used here.
- 4) The server returns to its listening mode and waits for new client requests.

3.6 Synchronous Process Planner

This section discusses the Synchronous Process Planner Architecture (SPP). The SPP-daemon provides multiple parallel requests invocation and results processing for the clients. This section describes the main components of this architecture and its execution scenario.

3.6.1 The SPP Interaction Architecture

In the Synchronous Process Planner Architecture, the SPP-daemon invokes multiple service requests in the sought servers for the client. The servers process the requests in

parallel and return the results to the SPP-daemon after performing the services. The SPP-daemon processes the returned results from the servers and returns the final result to the client. Figure 3-4 shows the general high-level SPP interaction architecture.

Figure 3 - 4: High-level Interaction Graph for the SPP Architecture

All the requests are synchronous in this architecture. The client does not need to have stubs for the server objects in its executable and is therefore very small in size. The behavior and the roles of the main components of the SPP architecture are presented next.

3.6.1.1 The Client

- 1) The client starts and enters its initialization phase, which includes reading the CORBA domain configuration file, initializing the ORB, activating the POA, getting the SPP-daemon service information, registering with the ORB.
- 2) Due to the location transparency policy in CORBA, the client does not know the sought server locations and their object references. The client sends the composite request to the SPP-daemon in its domain and waits for the result from the SPP-daemon.
- 3) Once the client has received the results from the SPP-daemon, it then starts another request cycle (goes to step 2).

3.6.1.2 The SPP-daemon

- 1) Similar to the other two daemons, the SPP-daemon rises and enters its initialization phase, which includes reading the CORBA domain configuration file, initializing the ORB, activating the POA, registering with the ORB, and

getting the object references of the managed servers in its domain.

- 2) The SPP-daemon enters a listening mode to wait for the requests from the clients.
- 3) When a request from the client arrives, the SPP-daemon decomposes the composite request into its constituent services. The SPP-daemon then locates the servers providing the constituent services, activates the servers if needed and sends the constituent service requests to the servers in parallel. The SPP-daemon waits for the results from the servers.
- 4) Upon receiving all the results from the servers, the SPP-daemon processes them and returns a single result to the client.
- 5) The SPP-daemon returns to its listening mode. Note that the SPP-daemon updates its managed server information at predefined regular intervals of time.

3.6.1.3 The Server

- 1) The server starts and enters its initialization phase, which includes reading the CORBA domain configuration file, initializing the ORB, activating the POA, and registering with the ORB.
- 2) The server enters its listening mode and waits for requests.
- 3) When a request (sent by the SPP-daemon) arrives, the server executes the requested method. In the performance prototype, the method burns a specific

amount of CPU time, according to the argument passed in the client request.

After completing the processing, the server returns the result to the

SPP-daemon.

- 4) The server returns to its listening mode.

Chapter 4 Instrumentation and Performance Measurement

This chapter discusses the performance prototypes of the parallel interaction architectures and a sequential architecture that were used to investigate the characteristics of the three CORBA-based architectures: the HPP, the FPP, and the SPP. In order to compare the performance advantages of the parallel architectures over the traditional sequential request processing CORBA system, a prototype of the traditional sequential system was also built and studied. The workload and the testing environment are described in detail in the next two sections. The request processing paths that vary with the underlying CORBA architecture are also presented in detail.

4.1 The Workload

The system modeled consists of a variable number of clients and four servers: Server A, Server B, Server C, and Server D. The workload parameters, which are expected to have significant impacts on the system performance, are controlled in the experiments. The workload parameters used in this research are described.

- 1) *Message Length (L)*: This is the message size in bytes, which the client sends to the sought servers in the request. This is also the size of messages that servers send back to the clients in the replies. The message is a byte stream,

composed of the same character of L bytes, which is included in the request and reply. In each experiment, the message length was held at a fixed given value.

- 2) ***Request Service Times S_A , S_B , S_C , and S_D*** : These are the total CPU demands for Server A, Server B, Server C, and Server D, in milliseconds. They represent the actual CPU processing time for the service only and do not include other overheads, such as marshalling/unmarshalling. In the experiments, S_A is equal to S_C , and S_B is equal to S_D . Unless otherwise stated, S_A , S_B , S_C , and S_D are the mean of service times that are characterized by an exponential distribution.
- 3) ***Number of Clients (N)***: All the experiments are conducted in closed systems. N is the number of active clients. Changing this parameter changes the system load.
- 4) ***Bandwidth (Bw)***: The bandwidth is the network traffic flow rate in bits per second. As wireless and wired network technologies vary, network bandwidths differ. How the systems perform within the constrained wireless environment is an important question.
- 5) ***Wireless Link Packet Drop Rate (D)***: This is another important characteristic of the wireless environment. It is useful for investigating the effectiveness of the proposed architectures in the wireless environment.

Note that we did not model fading in wireless channels. Its impact on performance

warrants further research.

Multithreading: This is a common technique for solving the software bottleneck problem. The multithreading used in this research is based on a thread pool provided by the Orbix E2A ASP 5.1, which has four default threads in the pool and increases the number of threads as needed [ION-02].

The levels for the workload parameters used in the experiments are shown in Table 4-1. N was the primary factor in most experiments and was used to control system load. Different server demands and message lengths were used to achieve various types of systems. We experimented with various types of systems in which some or all of their links can be wired or wireless (See Chapter 5 for details). Two wireless bandwidths, 1 Mbps and 10 Mbps were investigated. The wired bandwidth was fixed at 100 Mbps. Packet drop rates of 0% and 2% were used for investigating wireless links. The number of factors and their levels were chosen in such a way that a reasonable time for experimentation is achieved. Investigating the effect of higher drop rates forms an important direction for future research.

Table 4 - 1: Levels for Workload Parameters

Parameters	Levels
N	1, 2, 4, 6, 8, 10
S_A, S_B (msec)	(10,11); (50,55); (100,110)
L (bytes)	50, 1500, 10K
Wired Bw	100 Mbps
Wireless Bw	1 Mbps, 10 Mbps
D	0%, 2%

Using different combinations of L , S_A , S_B , Bw , and D enables both computation-bound systems and communication-bound systems to be studied. In a communication-bound system, the message length is large and the bandwidth is small, whereas the request service times are small. The computation-bound system features heavily loaded servers, but the message size is small.

4.2 The Experimental Environment

The prototype systems in all the architectures use two Pentium IV 2.4 GHz workstations running Redhat Linux 7.2; one Pentium II 266 MHz running Redhat Linux 8.0; and four Pentium III 500 MHz workstations running Redhat Linux 6.0. All the workstations are connected by a 100 Mbps Ethernet LAN. This section introduces the emulation tools used in the experiment as well as the lower level interactions of the four systems: the HPP, the FPP, the SPP, and the conventional Sequential CORBA system.

4.2.1 NIST Net

NIST Net is a network emulation package developed by the National Institute of Standards and Technology [NIS-02]. The NIST Net network emulator is a general-purpose tool designed to allow controlled, reproducible experiments with network performance sensitive/adaptive applications in a simple laboratory setting.

NIST Net is operated in the IP level running on Linux operating systems and allows a PC-based router to emulate numerous complex performance scenarios, including: tunable packet delay distributions, congestion and packet loss, bandwidth limitations, and packet reordering / duplication [NIS-02]. Using NIST Net, we emulated wireless data communication (IP-based) networks in our experiments. The bandwidth and packet loss characteristics of the wireless networks are handled effectively in NIST Net.

4.2.2 Nodes in the Prototypes

Each of the architectures uses seven nodes. These nodes are distributed and interconnected by a 100 Mbps Ethernet. A description of these nodes is presented.

- 1) *The NIST Net Node:* NIST Net, the wireless link emulator, is running on the Pentium II 266 MHz workstation under Redhat Linux 8.0, acting as a gateway between the client node, the PP-daemon node and the server nodes, and provides the wireless environment in the experiments.

Note that the impact of the NIST Net node on the performance of the interaction architectures is minimal. The NIST Net overhead was observed to be around 10 microseconds that is very small in comparison to the measured response times most of which are of the order of hundreds of milliseconds. The

CPU utilization of the NIST Net node was observed to be always less than 10%. This indicates that this node did not act as a performance limiting bottleneck device.

- 2) ***The Client Node:*** All the clients run on a Pentium IV 2.4 GHz workstation with a Redhat Linux 7.2 operating system. A predetermined number of identical client processes run in a closed system. Because the clients put very little computational demand on the workstation, the underlying CPU utilization of the client node is always under 10% to ensure the accuracy of the system performance measurement. During each experiment, each client executes a predefined number of cycles. A client issues a single composite request in each cycle. Once a client has finished one request, it starts another cycle immediately, without any thinking time. A typical client composite request consists of services from Server A and Server B or from Server C and Server D. Note that in the Sequential CORBA architecture, two requests are generated in a client cycle. One for Server A or Server C, and another for Server B or Server D. A cycle requesting services from Server A and Server B is followed by a cycle requesting services from Server C and Server D. The client node runs a NIST Net emulator to provide a wireless link from the PP-daemon node to the clients.

- 3) ***The PP-daemon Node:*** A Pentium IV 2.4 GHz workstation with Redhat Linux

7.2 operating system runs the PP-daemons (see Chapter 3 for details) as well as the Orbix E2A ASP 5.1 CORBA services such as the Naming Service, Event Service. In the traditional sequential system, there is no PP-daemon involved, so just the CORBA services run on this node. The PP-daemon node also runs a NIST Net emulator to provide a wireless link between the client node and the PP-daemon node. This node provides all the CORBA services that the clients and the servers need to communicate with each other.

- 4) *The Server Nodes:* Server A, Server B, Server C, and Server D each run on a Pentium III 500 MHz workstation under Redhat Linux 6.0. The four server nodes provide services to all the clients.

4.2.3 The Request Paths

This section discusses the request paths for the four architectures in detail. In each experiment, the client requests the services from Server A and Server B in the first cycle and the services from Server C and Server D in the second. The client then repeats the two cycles iteratively.

4.2.3.1 The HPP System Request Paths

The complete client request sequence path per cycle is shown in Figure 4-1. All the remote method invocations are synchronous and the client is blocked while waiting for

a reply from the HPP-daemon or a server.

- **Steps 1-2:** The client obtains the handles of Server C and Server D from the HPP-daemon. The packet sent is 54 bytes in length. When required the NIST Net in the HPP-daemon and client nodes emulates the wireless link between the two nodes.
- **Steps 3-6:** The client sends out the two service requests to Server C and Server D concurrently, using the obtained handles. The message size of the service requests varies in the experiments, so that its impact can be studied. The service requests first go to the NIST Net node, and then NIST Net acts as a gateway in order to emulate the wireless links between the client and the servers, and sends the packets to the appropriate servers.
- **Steps 7-10:** The servers reply to the client after performing the services. The message size of the reply is the same as the service request. The replies go through the NIST Net node as well.

In the next cycle, the client will invoke Server A and Server B instead of Server C and Server D. The client continues to iterate these two cycles until the end of the experiment.

Note that steps 3 and 4 are concurrent and the order in which they occur can vary from one client cycle to another. This is also valid for steps 5, 6; steps 7, 8; and steps 9, 10.

Figure 4 - 1: The HPP System Request Path

4.2.3.2 The FPP System Request Paths

The complete client Request Sequence Path per cycle is shown in Figure 4-2. In this system, some remote method invocations are asynchronous (“*one way*”) calls, so the client and the FPP-daemon are not blocked during the service.

However, the other calls are synchronous, and the servers are blocked, awaiting an acknowledgement from the client. Note that the servers become clients when they invoke the method in the client to return the results.

- **Step 1:** The client sends the composite service request to the FPP-daemon. This call is an asynchronous invocation. The message size of the service requests varies in the experiments, so that its impact can be studied. NIST Net in the FPP-daemon and client nodes emulates the wireless link between the two nodes.
- **Steps 2-5:** The FPP-daemon sends out the two service requests to Server C and Server D concurrently, after analyzing the composite request from the client. These remote method invocations are asynchronous calls. The message size of the service requests is the same as the client request. The service requests first go to the NIST Net node, and then NIST Net acts as a gateway to emulate the wireless links between the client and the servers, and sends the packets to the appropriate servers.

Figure 4 - 2: The FPP System Request Path

- **Steps 6-9:** The servers reply to the client after performing the services by invoking the method for receiving results in the client. The message size of the reply is the same as the client request. The replies go through the NIST Net node as well.

In the next cycle, the client will invoke Server A and Server B instead of Server C and Server D. The client continues to iterate these two cycles until the end of the experiment.

Note that steps 2 and 3 are concurrent and the order in which they occur can vary from one client cycle to another. This is also valid for steps 4, 5; steps 6, 7; and steps 8, 9.

4.2.3.3 The SPP System Request Paths

The complete client Request Sequence Path per cycle is shown in Figure 4-3. All the remote method invocations are synchronous and the client is blocked, waiting for the reply from the SPP-daemon. The SPP-daemon is also blocked, waiting for the reply from the servers.

- **Step 1:** The client sends a composite service request to the SPP-daemon. This call is a synchronous invocation. The message size of the service requests varies in the experiments, so that its impact can be studied. NIST Net in the

SPP-daemon and client nodes emulate the wireless link between the two nodes.

- *Steps 2-5:* The SPP-daemon sends out the two service requests to Server C and Server D concurrently, after analyzing the composite request from the client. These remote method invocations are also synchronous calls. The message size of the service requests is the same as the client request. The service requests first go to the NIST Net node, and then NIST Net acts as a gateway to emulate the wireless links between the client and the servers and sends the packets to the appropriate servers.
- *Steps 6-9:* The servers reply to the SPP-daemon after performing the services. The message size of the reply is the same as the service request. The replies go through the NIST Net node as well.
- *Step 10:* The SPP-daemon processes the returned results and returns the final result to the client. The message size is also the same as the client request.

In the next cycle, the client will invoke Server A and Server B instead of Server C and Server D. The client will continue to iterate these two cycles until the end of the experiment.

Note that steps 2 and 3 are concurrent and the order in which they occur can vary from one client cycle to another. This is also valid for steps 4, 5; steps 6, 7; and steps 8, 9.

Figure 4 - 3: The SPP System Request Path

4.2.3.4 The Traditional Sequential System Request Paths

The complete client Request Sequence Path per cycle is shown in Figure 4-4. All the remote method invocations are synchronous and the client is blocked, waiting for the reply from the CORBA service and the servers.

- *Steps 1-2:* The client obtains the handle of Server C from the CORBA Naming Service. The NIST Net in the CORBA Services node and client node emulates the wireless link between the two nodes.
- *Steps 3-4:* The client obtains the handle of Server D from the CORBA Naming Service.
- *Steps 5-6:* The client sends out the service request to Server C. The message size of the service requests varies in the experiments, so that its impact can be studied. The service request first goes to the NIST Net node. NIST Net acts as a gateway to emulate the wireless links between the client and the servers and sends the packets to the appropriate servers.
- *Steps 7-8:* Server C replies to the client after performing the services. The message size of the reply is the same as the service request. The reply goes through the NIST Net node as well.

Figure 4 - 4: The Traditional Sequential System Request Path

- *Steps 9-10:* The client sends out the service request to Server D. The message size is the same as in the service request to Server C. The service requests first go to the NIST Net node. NIST Net acts as a gateway to emulate the wireless link between the client and the server and sends the packets to the appropriate server.
- *Steps 11-12:* Server D sends a reply to the client after performing the services. The message size of the reply is the same as the service request. The replies go through the NIST Net node as well.

In the next cycle, the client invokes Server A and Server B instead of Server C and Server D. The client iterates these two cycles until the end of the experiment.

Performance prototypes of the four architectures were constructed and tested. During testing, messages describing the system state were logged while different parts of the prototypes were executing. The messages showed that the prototypes demonstrate the expected behaviors.

4.3 Results of Experiments

This section introduces the performance measurements used in this research and the validity of the results. The performance measurements are based on closed systems.

4.3.1 Performance Measures

The following performance metrics were collected and compared to show the impact of different workload factors on system performance.

- 1) *Mean Client Response Time (R)* in milliseconds: This is the mean total time elapsed from when the client starts a cycle to the time all the results have been received and processed in the client. Each client request cycle includes two services: either from Server A and Server B or from Server C and Server D. The mean client response time is a measure of system latency.
- 2) *Overall Mean System Throughput (X)* in client cycles per second: This performance measure shows the average total system throughput obtained by the summation of all the client's mean throughputs. The throughput of a client is the average number of cycles executed per second. This parameter is a measure of system capacity. The mean client response time and the mean system throughput are related by Little's Law ($N=X*R$) [JAI-91].
- 3) *Process and CPU Utilization (U)*: These utilizations provide insight into system performance and are useful in identifying the hardware and software bottlenecks. The two utilizations are calculated as shown below:

Process Utilization = request processing time / total elapsed time, including idle and waiting time.

CPU Utilization = CPU time of a process / total elapsed time, including idle and waiting time.

To investigate the impact of each parameter, a factor-at-a-time approach was adopted for the experiments. With such an approach, one of the parameters was varied while the others were held constant. The number of clients N is increased from 1 to 10 to simulate low to high system load conditions. The final results are presented and analyzed in Chapter 5.

4.3.2 CPU and Elapsed Time Measurements

The following Linux system calls are used in the appropriate position of the source code to measure the elapsed and CPU times during the experiment.

- 1) *gettimeofday()*: This system call obtains the system's notion of the current time, expressed in elapsed seconds and microseconds from 00:00 Universal Coordinated Time, January 1, 1970. The elapsed times in the client, servers, and PP-daemon are all measured using this function, unless otherwise stated [RED-02].
- 2) *Time [process]*: This system call returns the process's CPU utilization, as well as the user level CPU time and the system level CPU time. CPU utilizations are measured using this function in the experiments [RED-02].

4.3.3 Length of Experiments

3,000 client cycles were executed for each client in an experiment. This number was determined from results of some preliminary runs in which the single threaded SPP architecture was used with a message length of 50 bytes, a network bandwidth of 100 Mbps Ethernet, and server service times were fixed at: $S_A=S_C=50\text{ms}$, $S_B=S_D=55\text{ms}$. The number of clients in the system was 10. The number of cycles in a run was varied. Figure 4-5 shows that the mean client response time varies with the number of cycles initially, but converges when the number of client request cycles is over 2000. The number 3000 is chosen because it ensures that the system is in steady state with a reasonable experiment run length.

Figure 4 - 5: Determination of the Run Length for Experiments

4.3.4 Data Collection and Logging

The data collection and logging tasks are incorporated into the clients, servers, and

PP-daemons' source codes to eliminate the overhead engendered by the extra data collection process. The servers and the PP-daemons send the collected data to the clients with a reply message after the service. The client saves the data, along with its own measured data in its memory space, in order to eliminate the overhead of file I/O. After each experiment, the data are post-processed by the clients and are logged as a single event.

4.3.5 Validity of the Experiment Results

The experiments were conducted in the RADS Lab on an isolated measurement network where experiments can run without interference from other applications. As described in Section 4.3.3, the clients repeated the request cycles 3000 times, with a specific set of workload parameters in each experiment to obtain stable experimental data. The experiment was repeated for each set of workload factors, and the statistics were collected and processed to ensure that the final results were within the specified confidence interval for the performance measurements. For the desired performance measure of interest, we achieved a confidence interval of $\pm 2.5\%$ at a confidence level of 95%.

The results are verified using *Little's Law* and the *Utilization Law*. Little's Law is a universally accepted theorem in Queuing theory. It states the relationship among the

number of active clients, the mean response time, and the mean system throughput:
 $N=X*R$. N is the total number of active clients. X is the overall mean system throughput. R is the mean client response time. Utilization Law is another important method for verifying the test data: $U=X*D$. U is the CPU utilization of a given process. X is the throughput of that given server. D is the mean CPU time in seconds per service request. The final results of the experiments were found to be satisfactory by using both verification methods.

Chapter 5 System Performance Comparison

The results of experiments are presented in this chapter, as well as the in-depth system performance analysis and comparison. The experimental results are presented as a set of graphs that show the relationship between one of the performance measures (plotted along the Y axis) and the number of clients (plotted along the X axis), with all other constant parameters stated in the figure caption. The performance analysis focuses on the effects of the different parameters on the performances of the different architectures. There are three modes of configuration. An *All Wired Mode* implies that connection among clients, servers, and the PP-daemons are through fixed 100 Mbps wired links. The *Wireless Client Only Mode* means that only clients are on 1 Mbps/10 Mbps wireless links, whereas the PP-daemon and the servers are connected to each other via 100 Mbps wired networks. The *All Wireless Mode* means that every component of the system is communicating through 1 Mbps/10 Mbps wireless links.

In most cases the analysis of experimental results is done in two steps. The latency of the system as captured in the response time achieved with $N=1$ is used to compare the performances of the architectures at low system load. Both response time and system throughput are used in the performance comparison for medium to high system load. Multiple levels are used for a factor that is varied in an experiment. In most cases, the

graphs corresponding to only one level is presented in the main text. Graphs that correspond to the other levels are included in an appendix.

5.1 Effect of Multi-Threading

A set of experiments was designed to identify the bottleneck devices in the four different systems (the HPP, the FPP, the SPP, and the Sequential system) and show the impact of Multi-threading. The workload parameters are shown in Table 5-1.

Table 5 - 1: Parameters of the Experiment Investigating the Effect of Multi-Threading

Parameters	Level of values
<i>Number of clients (N)</i>	<i>1, 2, 4, 6, 8, 10</i>
<i>Service Times (Exponential distribution)</i>	$S_A=S_C=50\text{ ms}$ $S_B=S_D=55\text{ ms}$
<i>Message size (L)</i>	<i>50 Bytes</i>
<i>Bandwidth (Bw)</i>	<i>1 Mbps, 100 Mbps</i>
<i>Drop rate (D)</i>	<i>0</i>
<i>PP-daemon threading</i>	<i>Single threaded, Multithreaded</i>

5.1.1 Single Threaded Systems

The experimental results for the four systems are presented below. The network connections between the clients, the PP-daemon and the servers are the same during one experiment: either 1 Mbps wireless links or 100 Mbps wired networks.

Low System Load:

As shown in Table 5-2, Figure 5-1, and Figure 5-2, at $N=1$ the SPP system

demonstrates the best latency, followed by the HPP and the FPP systems. The traditional sequential system shows the worst latency. This is due to the concurrent server invocation deployed in the three parallel systems. The SPP system produces a performance improvement of over 100%, compared to the conventional sequential system. Because there is no queuing delay at $N=1$, the latency is primarily determined by the number of messages exchanged per client request cycle among the three parallel systems.

Table 5 - 2: Latency of Single Threaded Systems at $N=1$ ($L=50$ bytes, $D=0\%$, $S_A=S_C=50$ ms, $S_B=S_D=55$ ms)

Bw (Mbps)	Sequential (ms)	HPP (ms)	FPP (ms)	SPP (ms)
1 (All Wireless Mode)	202.615	145.119	147.383	90.431
100 (All Wired Mode)	116.901	85.239	94.431	80.664

The SPP system requires six messages to be exchanged per client cycle: two messages between the client and the SPP-daemon; two messages for the request and reply between the SPP-daemon and each server. There are two server invocations in one client cycle, so a total of four messages are exchanged between the SPP-daemon and the servers. The HPP system requires an exchange of ten messages per client cycle: two messages between the HPP-daemon and the client for the client getting the two server handles from the HPP-daemon. Four messages are sent between the client and

each server: the first two messages verify the handle with the server, and the other two messages consist of the service request and the result reply [FRE-03]. There are two server requests in one client cycle, so a total of eight messages pass between the client and the servers. The FPP system requires eleven messages to be exchanged per client cycle: one “*send oneway*” message from the client to the FPP-daemon, two “*send oneway*” forwarding messages to the two servers, and four messages exchanged between each of the two servers and the client. In the conventional sequential system, twelve messages are exchanged in each client cycle: the client can only get the server handles one by one from the CORBA Naming Service, and there is a total of four messages between the client and the CORBA Naming Service. As described earlier, four messages are exchanged between the client and each of the two servers. The number of messages used in each interaction architecture is shown in Table 5-3 below.

Table 5 - 3: Number of Message Exchanges per Client Cycle

Interaction Architecture	Number of Message Exchanges
Sequential	12
HPP	10
FPP	11
SPP	6

The SPP system has the fewest message exchanges, because the SPP-daemon invokes the servers using the handle in its own name space. The verifying handle step is only performed the first time it invokes the server and skipped in the subsequent

invocations. In the other three systems, the client must verify the handle each time it obtains information from the CORBA Naming Service or from the request messages. On systems with a limited number of servers, by carefully designing the PP-daemon management domain, the SPP-daemon can eliminate the overhead of verifying the handles and keep the communication channels with the servers open. The SPP system achieves the best performance by using the lowest number of messages per client cycle. However, the FPP system performs the worst among the three parallel systems, as it uses the highest number of messages per client cycle. Through the use of concurrent server invocations, however, the FPP system performs better than the Sequential system.

Bw=1 Mbps, SA=SC=50 ms, SB=SD=55 ms, L=50 bytes, D=0%

Figure 5 - 1: The Response Time of the Single Threaded Systems with Service Time 50, 55 ms in the All Wireless Mode

Bw=1 Mbps, SA=SC=50 ms, SB=SD=55 ms, L=50 bytes, D=0%

Figure 5 - 2: The Throughput of the Single Threaded Systems with Service Time 50, 55 ms in the All Wireless Mode

Medium to High System Load:

For most of the architectures, as the number of clients increases, the system throughput increases at first and then levels off at $N=8$ (see Figure 5-2). The best performance at higher values of N is demonstrated by the HPP system, followed by the FPP system and the Sequential system. The SPP system, which performs best at $N=1$, performs poorly at a medium to high load. The reason for this is that the single threaded SPP-daemon can only process one job at a time. During the processing of a job, the SPP-daemon remains blocked for a substantial period of time, waiting for the responses from the two servers. Once both results have been received, the

SPP-daemon processes them and sends the final result back to the client, and then picks up the next job for service. As a result, the SPP-daemon process gets saturated at higher loads: it remains busy 100% of the time in processing and waiting for the servers to respond, while the underlying CPU is kept idle (see Figure 5-3). This phenomenon is called software bottlenecking [FON-89, NEI-95], and a software bottleneck is said to have occurred in the SPP-daemon process. The software bottleneck introduces large queuing delays to client requests and leading to poor system performance.

As shown in Figure 5-4, Figure 5-5, and Figure 5-6, the other three systems do not have the software bottleneck at the PP-daemon. Server B process tends to saturate at a high system load and limit system performance.

An interesting observation is that although the Sequential system performs the worst at low load ($N=1$), its throughput improves with the increase of N , as it attains a throughput that is only 11% lower than that achieved by the highest performer, the HPP system at $N=10$ (see Table 5-4). At a high system load, the waiting time of the requests at each server's service queue offsets the advantages of parallel processing. However, the HPP system still achieves the best performance, because it exchanges fewer messages per client cycle.

Bw=1 Mbps, SA=SC=50, 55 ms, SB=SD=55 ms, L=50 bytes, D=0%
 CPU-A=CPU Util. Of Server A & C, Proc-A=Process Util. Of Server A & C
 CPU-SPP=CPU Util. Of SPP-daemon node, Proc-SPP=Process Util. Of SPP-daemon

Figure 5 - 3: CPU and Process Utilization of the Single Threaded SPP System in the All Wireless Mode

Bw=1 Mbps, SA=SC=50, 55 ms, SB=SD=55 ms, L=50 bytes, D=0%
 CPU-A=CPU Util. Of Server A & C, Proc-A=Process Util. Of Server A & C
 CPU-HPP=CPU Util. Of HPP-daemon node, Proc-HPP=Process Util. Of HPP-daemon

Figure 5 - 4: CPU and Process Utilization of the Single Threaded HPP System in the All Wireless Mode

Bw=1 Mbps, SA=SC=50, 55 ms, SB=SD=55 ms, L=50 bytes, D=0%
 CPU-A=CPU Util. Of Server A & C, Proc-A=Process Util. Of Server A & C
 CPU-FPP=CPU Util. Of FPP-daemon node, Proc-FPP=Process Util. Of FPP-daemon

Figure 5 - 5: CPU and Process Utilization of the Single Threaded FPP System in the All Wireless Mode

Bw=1 Mbps, SA=SC=50, 55 ms, SB=SD=55 ms, L=50 bytes, D=0%
 CPU-A=CPU Util. Of Server A & C, Proc-A=Process Util. Of Server A & C

Figure 5 - 6: CPU and Process Utilization of the Sequential System in the All Wireless Mode

Table 5 - 4: System Throughput of Single Threaded Systems at N=10 (L=50 bytes, D=0%, $S_A=S_C=50$ ms, $S_B=S_D=55$ ms)

Bw (Mbps)	Sequential (cycles/sec)	HPP (cycles/sec)	FPP (cycles/sec)	SPP (cycles/sec)
1 (All Wireless Mode)	27.511	30.382	28.297	12.228
100 (All Wired Mode)	27.567	30.717	29.005	12.405

5.1.2 Multi-Threaded Systems

An important observation presented in the previous section is the existence of performance-limiting software bottlenecks. Multi-threading the bottleneck process is an effective method of alleviating the problem [NEI-95]. Figure 5-7 and Figure 5-8 show the performances of the three Multi-threaded PP-daemon-based systems, and the performance of the Sequential system is also included for comparison. Table 5-5 shows the response time in low load conditions. At N=1, the relative performances of the four systems remain the same as in the single threaded systems (shown in Table 5-5). However, the higher process utilization for Server B seems to indicate that the performance bottleneck has shifted from the SPP-daemon to Server B (see Figure 5-9). The utilization of the multithreaded SPP-daemon process cannot be measured and is not shown in Figure 5-9. A set of CPU and process utilization graphs similar to that of the previous single threaded case of the other three systems was observed in the experiments and is not shown.

Bw=1 Mbps, SA=SC=50 ms, SB=SD=55 ms, L=50 bytes, D=0%

Figure 5 - 7: The Response Time of the Multi-Threaded Systems with Service Time 50, 55 ms in the All Wireless Mode

Bw=1 Mbps, SA=SC=50 ms, SB=SD=55 ms, L=50 bytes, D=0%

Figure 5 - 8: The Throughput of the Multi-Threaded Systems with Service Time 50, 55 ms in the All Wireless Mode

Bw=1 Mbps, $S_A=S_C=50, 55$ ms, $S_B=S_D=55$ ms, L=50 bytes, D=0%
 CPU-A=CPU Util. Of Server A & C, Proc-A=Process Util. Of Server A & C
 CPU-SPP=CPU Util. Of SPP-daemon node

Figure 5 - 9: CPU and Process Utilization of the Multi-Threaded SPP System with Service Time 50, 55 ms in the All Wireless Mode

Table 5 - 5: Latency of Multi-Threaded systems at N=1 (L=50 bytes, D=0%, $S_A=S_C=50$ ms, $S_B=S_D=55$ ms)

Bw (Mbps)	Sequential (ms)	HPP (ms)	FPP (ms)	SPP (ms)
1 (All Wireless Mode)	202.615	144.631	146.777	90.329
100 (All Wired Mode)	116.901	85.34	94.686	80.778

At medium to high load conditions, the multithreaded SPP system displays a dramatic improvement in performance in comparison to the single threaded system (see Table 5-6). When the thread handling the server requests for a given job is blocked, the free CPU can be used by other threads to process the requests for other jobs sent by the different clients in the system. The performances of the other three systems remain

unaltered. This is because of the absence of a software bottleneck in the HPP-daemon and FPP-daemon processes in these systems (shown in Figure 5-4 and Figure 5-5). Consequently, no significant improvement is gained from multi-threading the HPP-daemon and the FPP-daemon processes. The SPP system becomes the best performer in all load conditions (see Figure 5-8), followed by the HPP, the FPP, and the Sequential systems. Since multi-threading eliminates the software bottleneck problem effectively, and it can easily be implemented using many programming languages and fully supported by the current Operating Systems, the following experiments use only the multi-threaded systems.

Table 5 - 6: System Throughput of Multi-Threaded Systems at N=10 (L=50 bytes, D=0%, $S_A=S_C=50$ ms, $S_B=S_D=55$ ms)

Bw (Mbps)	Sequential (cycles/sec)	HPP (cycles/sec)	FPP (cycles/sec)	SPP (cycles/sec)
1 (All Wireless Mode)	27.511	30.422	27.864	31.671
100 (All Wired Mode)	27.567	30.784	29.171	31.580

5.2 Effect of Service Times

According to the nature of the service requests, the clients may make different computational demands on the servers. A set of experiments was conducted to investigate the relative performances of the four systems as the service demands increased. The workload parameters used in the experiments are shown in Table 5-7

below.

Table 5 - 7: Parameters of the Experiment Investigating the Effect of Service Demands

Parameters	Level of values
<i>Number of clients (N)</i>	<i>1, 2, 4, 6, 8, 10</i>
<i>Service Times (Exponential distribution)</i>	<i>$S_A=S_C=10\text{ ms}, 50\text{ ms}, 100\text{ ms}.$ $S_B=S_D=11\text{ ms}, 55\text{ ms}, 110\text{ ms}.$</i>
<i>Message size (L)</i>	<i>50 Bytes</i>
<i>Wireless Bw</i>	<i>1 Mbps</i>
<i>Drop rate (D)</i>	<i>0</i>
<i>PP-daemon thread</i>	<i>Multithreaded</i>

5.2.1 Wireless Client Only Mode

Service Time 10, 11 ms:

The effect on the mean client response time and the mean system throughput of server demands was studied at low to high system load conditions.

Low System Load:

As shown in Table 5-8, Figure 5-10 and Figure 5-11, when $N=1$, all the systems experience an increase in response time with the increase of server demands. For any given server demands set, the SPP system exhibits the shortest response time, due to the fact that it requires the fewest message exchanges per client cycle, followed by the HPP and the FPP systems. The SPP and FPP systems require two and nine message exchanges respectively over the wireless links in this mode, but the HPP and Sequential systems carry out all of their message exchanges on the wireless link. That

is why their performances are much lower than those of the other systems. For a given N, the Sequential system has the longest response time due to its sequential processing of the client requests.

Table 5 - 8: Effect of Server Demand on Latency at N=1 (L=50 bytes, Wireless Bw=1 Mbps, Wired Bw=100 Mbps, D=0%, Multi-PP-daemon, Wireless Client Only Mode)

Service Time (ms)	Sequential (ms)	HPP (ms)	FPP (ms)	SPP (ms)
$S_A=S_C=10$ $S_B=S_D=11$	38.027	25.875	32.671	18.529
$S_A=S_C=50$ $S_B=S_D=55$	202.615	144.631	129.504	82.544
$S_A=S_C=100$ $S_B=S_D=110$	229.175	169.828	174.579	159.679

Bw=1 Mbps, $S_A=S_C=10$ ms, $S_B=S_D=11$ ms, L=50 bytes, D=0%

Figure 5 - 10: The Response Time of the Multi-Threaded Systems with Service Time 10, 11 ms in the Wireless Client Only Mode

Bw=1 Mbps, SA=SC=10 ms, SB=SD=11 ms, L=50 bytes, D=0%

Figure 5 - 11: The Throughput of the Multi-Threaded Systems with Service Time 10, 11 ms in the Wireless Client Only Mode

Medium to High System Load:

As the multi-threaded PP-daemons eliminate the software bottleneck problem (see Section 5.1 for details), the multi-threaded SPP system leads all the way, until the servers become saturated. The Sequential system performs comparably to the FPP system in the high load condition, because the extra work performed by the servers in the FPP system (verifying the handle with the client to send back the results) limits the system's capacity. In the HPP and the Sequential systems, verification of the server handles are performed by the clients, so the system scalability is not affected in these two systems. The SPP and the HPP systems show a performance improvement of

47.3% and 9.4% respectively over the Sequential system (see Table 5-9).

Service Time 50, 55 ms:

A similar performance pattern to that of the previous service demand set was observed in this case (see Figure 5-12 and Figure 5-13). However, in a high system load, the SPP system shows a performance improvement of only 14.3% over the Sequential system. The reason for this is that as the service demands increase, the communication time has less impact on the system performance, compared to the impact of the service demands. So the advantage of the SPP system at small service demands is offset by longer service times.

Table 5 - 9: Effect of Server Demand on System Throughput at N=10 (L=50 bytes, Wireless Bw=1 Mbps, Wired Bw=100 Mbps, D=0%, Multi-PP-daemon, Wireless Client Only Mode)

Service Time (ms)	Sequential (cycles/sec)	HPP (cycles/sec)	FPP (cycles/sec)	SPP (cycles/sec)
$S_A=S_C=10$ $S_B=S_D=11$	102.368	112.017	102.216	150.830
$S_A=S_C=50$ $S_B=S_D=55$	27.511	30.422	27.589	31.440
$S_A=S_C=100$ $S_B=S_D=110$	14.226	15.546	15.066	15.794

Service Time 100, 110 ms:

Again, a similar performance pattern to those of the previous service demand sets is observed in this case (see Figure A-1 and Figure A-2 in Appendix A).

Bw=1 Mbps, SA=SC=50 ms, SB=SD=55 ms, L=50 bytes, D=0%

Figure 5 - 12: The Response Time of the Multi-Threaded Systems with Service Time 50, 55 ms in the Wireless Client Only Mode

Bw=1 Mbps, SA=SC=50, 55 ms, SB=SD=55 ms, L=50 bytes, D=0%

Figure 5 - 13: The Throughput of the Multi-Threaded Systems with Service Time 50, 55 ms in the Wireless Client Only Mode

As expected, the performance improvement of the SPP system over the Sequential system slides to 11.0% at high system load. The benefit of parallel processing is evident at low to medium load conditions, but it decreases at high system load. This is due to the waiting time the client requests experience waiting in queues at the servers.

5.2.2 All Wireless Mode

In this mode, all the communications are via wireless links. The experimental results for $N=1$ and $N=10$ are presented in Table 5-10 and Table 5-11. The response time and the throughput graphs are presented in Figure 5-7, Figure 5-8, and Figure A-3 to Figure A-6 in Appendix A. The HPP and the Sequential systems are unaffected by the mode change, since even in the Wireless Client Only Mode, all of their message exchanges are over the wireless links. As expected, system performances deteriorate with an increase in service demands (see Table 5-10 and Table 5-11). However, the overall relative performances are unaltered among the four systems (see Figure 5-7, Figure 5-8, and Figure A-3 to Figure A-6 in Appendix A). The SPP system is still the best performer under all load conditions, followed by the HPP, the FPP and the Sequential systems.

5.3 Effect of Message Length

The message length of the client request and server reply may vary, depending on the

type of the requests and the service provided. This set of experiments is designed to analyze the relative performances of the four systems, with different message length L .

Table 5 - 10: Effect of Server Demand on Latency at $N=1$ ($L=50$ bytes, Wireless Bw=1 Mbps, $D=0\%$, Multi-PP-daemon, All Wireless Mode)

Service Time (ms)	Sequential (ms)	HPP (ms)	FPP (ms)	SPP (ms)
$S_A=S_C=10$ $S_B=S_D=11$	38.027	25.875	33.853	19.422
$S_A=S_C=50$ $S_B=S_D=55$	202.615	144.631	146.777	90.329
$S_A=S_C=100$ $S_B=S_D=110$	229.175	169.828	174.17	162.686

Table 5 - 11: Effect of Server Demand on System Throughput at $N=10$ ($L=50$ bytes, Wireless Bw=1 Mbps, $D=0\%$, Multi-PP-daemon, All Wireless Mode)

Service Time (ms)	Sequential (cycles/sec)	HPP (cycles/sec)	FPP (cycles/sec)	SPP (cycles/sec)
$S_A=S_C=10$ $S_B=S_D=11$	102.368	112.017	101.013	149.147
$S_A=S_C=50$ $S_B=S_D=55$	27.511	30.422	27.864	31.671
$S_A=S_C=100$ $S_B=S_D=110$	14.226	15.546	15.045	15.771

Table 5-12 shows the parameters for the experiment: the message lengths are 50, 1500, and 10K Bytes. As explained in Section 5.1, different systems give rise to different number of message exchanges per client cycle. However, not all the message exchanges carry the service request/reply message stated here. For example, the message exchanges between the client and the HPP-daemon for getting the server

handles do not include service request/reply messages in their communication packets. The steps taken to verify the handle between the client and the server also do not include the service request/reply messages. The SPP system has all six messages carrying the service request/reply messages per client cycle. The FPP system has five messages carrying the service request/reply message per client cycle. The HPP and the Sequential systems each have four messages carrying the service request/reply message per client cycle. When the message length is 50 bytes, the non-service request/reply messages are similar in size to the service request/reply messages. However, as the message length increases, the difference between the two types of messages shows more impact on the system performance.

Table 5 - 12: Parameters of the Experiment Investigating the Effect of Message Length

Parameters	Level of values
<i>Number of clients (N)</i>	<i>1, 2, 4, 6, 8, 10</i>
<i>Service Times (Exponential distribution)</i>	$S_A=S_C= 50\ ms$ $S_B=S_D= 55\ ms.$
<i>Message size (L)</i>	<i>50, 1500, 10K Bytes</i>
<i>Wireless Bw</i>	<i>1 Mbps</i>
<i>Drop rate (D)</i>	<i>0</i>
<i>PP-daemon thread</i>	<i>Multithreaded</i>

5.3.1 Wireless Client Only Mode

Message Length of 10K Bytes:

The impact of message length on the mean client response time and the mean system throughput is investigated in low to high system load conditions.

Low System Load:

As shown in Table 5-13, Figure 5-14 and Figure 5-15, when $N=1$, all systems experience an increase in response time with the increase of the message length. For any given message length, the SPP system exhibits the shortest response time, due to the fact that it requires the fewest message exchanges per client cycle over the wireless link, followed by the FPP and the HPP systems. In this mode the SPP and the FPP systems require two and three message exchanges respectively to carry the service request/reply messages over the wireless links, but the HPP and the Sequential systems carry out all of their message exchanges over the wireless link. This puts the performance of the SPP system in the lead. The Sequential system has the longest response time, due to its sequential processing of client requests and longest communication time per client cycle.

Table 5 - 13: Effect of Message Length on Latency at $N=1$ ($S_A=S_C=50$ ms, $S_B=S_D=55$ ms, Wireless Bw=1 Mbps, Wired Bw=100 Mbps, $D=0\%$, Multi-PP-daemon, Wireless Client Only Mode)

Message Length (Bytes)	Sequential (ms)	HPP (ms)	FPP (ms)	SPP (ms)
50	202.615	144.631	129.504	82.544
1500	211.002	159.605	159.253	141.295
10K	420.507	323.152	304.055	258.835

Figure 5 - 14: The Response Time of the Multi-Threaded Systems with L=10 KB in the Wireless Client Only Mode

Figure 5 - 15: The Throughput of the Multi-Threaded Systems with L=10 KB in the Wireless Client Only Mode

Medium to High System Load:

Because the multi-threaded PP-daemons eliminate the software bottleneck problem (see Section 5.1 for details), the SPP system leads all the way, until the server is saturated. The Sequential system performs comparably with the HPP system in high load conditions. This is because both of them require four message exchanges to carry the service request/reply message and the communication time is the major component of the response time with such a high L. Consequently, the SPP and the FPP systems show a performance improvement of 102.7% and 15.5% respectively over the Sequential system (see Table 5-14). The communication time dominates the system performances in this case.

Table 5 - 14: Effect of Message Length on System Throughput at N=10 ($S_A=S_C=50$ ms, $S_B=S_D=55$ ms, Wireless Bw=1 Mbps, Wired Bw=100 Mbps, D=0%, Multi-PP-daemon, Wireless Client Only Mode)

Message Length (Bytes)	Sequential (cycles/sec)	HPP (cycles/sec)	FPP (cycles/sec)	SPP (cycles/sec)
50	27.511	30.422	27.589	31.440
1500	23.331	26.772	23.920	31.323
10K	5.462	5.913	6.307	11.071

Message Length of 1500 Bytes:

As shown in Figure B-1 and Figure B-2 in Appendix B, the relative performances of the architectures are similar to that of the architectures observed with the other message lengths. The Sequential system performs comparably with the FPP system under a high load condition, because of the extra work performed by the servers in the

FPP system (verifying the handle with the client to send back the results) limits system capacity. However, the SPP and the HPP systems show a performance improvement of 34.3% and 14.7% over the Sequential system respectively, because the SPP system utilizes the advantage of the shorter communication time. Whereas the benefit of the parallel processing emerges to give the HPP system better performance over the Sequential system.

Message Length of 50 Bytes:

The results for this case were presented earlier in Section 5.2.1. The relative performances of the architectures are similar to that of the architectures observed with the other message lengths (see Figure 5-12 and Figure 5-13). However, at high system loads, the SPP system shows a performance improvement of only 14.3% over the Sequential system. The reason is that as the message length decreases, the communication time has a lesser impact on the system performance compared to the impact of the service demands. So the advantage of the SPP system is smaller with a low message length.

5.3.2 All Wireless Mode

In this mode, all the communications are via wireless links. The experimental results for $N=1$ and $N=10$ are presented in Table 5-15 and Table 5-16. The response time and

the throughput graphs are presented in Figure 5-7, Figure 5-8, Figure 5-15, Figure 5-16, and Figure B-3, Figure B-4 in Appendix B. The HPP and the Sequential systems are unaffected by the mode change, since even in the Wireless Client Only Mode, all of their message exchanges are over the wireless links. As expected, system performances deteriorate with an increase in message length (see Table 5-15 and Table 5-16). However, the overall relative performances are unaltered among the four systems when the message length is 50 bytes and 1500 bytes (see Figure 5-7, Figure 5-8, and Figure B-3, Figure B-4 in Appendix B). However, the situation is very different when the message length is 10K bytes and is discussed next.

Table 5 - 15: Effect of Message Length on Latency at N=1 ($S_A=S_C=50$ ms, $S_B=S_D=55$ ms, Wireless Bw=1 Mbps, D=0%, Multi-PP-daemon, All Wireless Mode)

Message Length (Bytes)	Sequential (ms)	HPP (ms)	FPP (ms)	SPP (ms)
50	202.615	144.631	146.777	90.329
1500	211.002	159.605	176.027	148.943
10K	420.507	323.152	428.158	485.335

Table 5 - 16: Effect of Message Length on System Throughput at N=10 ($S_A=S_C=50$ ms, $S_B=S_D=55$ ms, Wireless Bw=1 Mbps, D=0%, Multi-PP-daemon, All Wireless Mode)

Message Length (Bytes)	Sequential (cycles/sec)	HPP (cycles/sec)	FPP (cycles/sec)	SPP (cycles/sec)
50	27.511	30.422	27.864	31.671
1500	23.331	26.772	23.603	30.923
10K	5.462	5.913	6.290	6.449

Message Length of 10K Bytes:

This is a typical communication-bound system scenario: the communication time dominates the performances of the systems, while the server process utilizations and CPU utilizations are low (see Figure 5-18). The wireless transmission capacity limits the overall system capacity in this case.

At a low system load ($N=1$), the response time mainly depends on how many message exchanges are required to carry the 10KB service request/reply message. The SPP system produces the worst performance, with six service request/reply message exchanges, followed by the FPP, the Sequential and the HPP systems in ascending order.

At medium system load ($N=4$), the FPP and SPP systems improve and perform better than the other two systems (see Figure 5-17). The reason for this is that in the HPP and the Sequential systems, the communication load concentrates on only one wireless link, which is between the client and the servers, while the communication load is distributed over three wireless links in the FPP system (one between the client and the FPP-daemon; the second between the client and the servers; and the third between the FPP-daemon and the servers). Moreover, in the SPP system, the communication load is distributed over two wireless links (one between the client and the SPP-daemon; the

other between the SPP-daemon and the servers). This gives the FPP system the best performance, followed by the SPP, the HPP and the Sequential systems.

At high system load ($N=10$), the SPP system takes the lead, followed by the FPP, the HPP and the Sequential systems (see Table 5-16). Again the performance of the FPP system in the high load condition is affected by the extra work performed by the servers in verifying the handle with the client to send back the results and by the server waiting for the acknowledgement that the client has received the results in the FPP system. As a result, the system throughput is reduced in the FPP system.

Bw=1 Mbps, SA=SC=50 ms, SB=SD=55 ms, L=10K bytes, D=0%

Figure 5 - 16: The Response Time of the Multi-Threaded Systems with L=10 KB in the All Wireless Mode

Bw=1 Mbps, SA=SC=50 ms, SB=SD=55 ms, L=10K bytes, D=0%

Figure 5 - 17: The Throughput of the Multi-Threaded Systems with L=10 KB in the All Wireless Mode

Bw=1 Mbps, SA=SC=50, 55 ms, SB=SD=55 ms, L=10K bytes, D=0%
 CPU-A=CPU Util. Of Server A & C, Proc-A=Process Util. Of Server A & C

Figure 5 - 18: CPU and Process Utilization of the Multi-Threaded Systems with L=10 KB in the All Wireless Mode

5.4 Effect of Network Bandwidth

The network environment in which the systems reside may vary, depending on the type of the network technology used. Currently, 3G wireless technologies can achieve 1 Mbps to 2 Mbps bandwidth; Wireless LAN technology (IEEE802.11) can achieve up to 11.2 Mbps bandwidth; and Wired Ethernet LAN networks typically have a bandwidth of 100 Mbps. This set of experiments is designed to analyze the relative performances of the four systems using different Bandwidth settings. Table 5-17 shows the experiment parameters: the network bandwidths used were 1, 10 and 100 Mbps, to reflect the effects of the different network technologies. Note that performances of the *All Wireless Mode* and the *Wireless Client Only Mode* are presented in Section 5.4.1 and Section 5.4.2 respectively. In order to have a complete understanding of the effect of network bandwidth on performance, the performances of *All Wired Mode* systems using 100 Mbps links are included in each section.

Table 5 - 17: Parameters of the Experiment Investigating the Effect of Network Bandwidth

Parameters	Level of values
<i>Number of clients (N)</i>	<i>1, 2, 4, 6, 8, 10</i>
<i>Service Times</i> <i>(Exponential distribution)</i>	$S_A=S_C= 50\ ms$ $S_B=S_D= 55\ ms.$
<i>Message size (L)</i>	<i>50 Bytes</i>
<i>Wireless Bw/Wired Bw</i>	<i>1 Mbps, 10 Mbps/ 100 Mbps</i>
<i>Drop rate (D)</i>	<i>0</i>
<i>PP-daemon thread</i>	<i>Multithreaded</i>

5.4.1 All Wired Mode and Wireless Client Only Mode

Network Bandwidth of 100 Mbps (All Wired Mode):

The mean client response time and the mean system throughput are investigated at low to high system load conditions.

Low System Load:

As shown in Table 5-18, Figure 5-19 and Figure 5-20, when $N=1$, all the systems experience an increase in response time with a decrease in network bandwidth. For any given network bandwidth, the SPP system exhibits the shortest response time because it requires the fewest message exchanges per client cycle, followed by the HPP and the FPP system. The Sequential system has the longest response time, due to its sequential processing of client requests. As the bandwidth increases, the performances of all the systems improve, due to the reduction in communication time (see Table 5-18).

Table 5 - 18: Effect of Network Bandwidth on Latency at $N=1$ ($S_A=S_C=50$ ms, $S_B=S_D=55$ ms, $L=50$ Bytes, $D=0\%$, Multi-PP-daemon, All Wired Mode and Wireless Client Only Mode)

Bandwidth (Mbps)	Sequential (ms)	HPP (ms)	FPP (ms)	SPP (ms)
1 (Wireless Client Only Mode)	202.615	144.631	129.504	82.544
10 (Wireless Client Only Mode)	119.669	88.055	98.028	81.33
100 (All Wired Mode)	116.901	85.34	94.686	80.778

Medium to High System Load:

As the multi-threaded PP-daemons eliminate the software bottleneck problem (see Section 5.1 for details), the SPP system leads all the way until the servers are saturated. The performance of the Sequential system becomes closer to that of the FPP system in a high load condition, because the extra work performed by the servers in the FPP system (verifying the handle with the client to send back the results) limits system capacity. The SPP and the HPP systems show a performance improvement of 14.6% and 11.7% respectively over the Sequential system (see Table 5-19).

Table 5 - 19: Effect of Network Bandwidth on System Throughput at N=10 ($S_A=S_C=50$ ms, $S_B=S_D=55$ ms, $L=50$ Bytes, $D=0\%$, Multi-PP-daemon, All Wired Mode and Wireless Client Only Mode)

Bandwidth (Mbps)	Sequential (cycles/sec)	HPP (cycles/sec)	FPP (cycles/sec)	SPP (cycles/sec)
1 (Wireless Client Only Mode)	27.511	30.422	27.589	31.440
10 (Wireless Client Only Mode)	27.563	30.768	28.920	31.530
100 (All Wired Mode)	27.567	30.784	29.171	31.580

Network Bandwidth of 10 Mbps:

In this case the relative performances of the architectures are similar to those observed at a bandwidth of 100 Mbps (see Figure C-1 and Figure C-2 in Appendix C). Moreover, the system throughput for any given system is comparable to that achieved with a 100 Mbps bandwidth.

Bw=100 Mbps, SA=SC=50 ms, SB=SD=55 ms, L=50 bytes, D=0%

Figure 5 - 19: The Response Time of the Multi-Threaded Systems with Bw=100 Mbps in the All Wired Mode

Bw=100 Mbps, SA=SC=50 ms, SB=SD=55 ms, L=50 bytes, D=0%

Figure 5 - 20: The Throughput of the Multi-Threaded Systems with Bw=100 Mbps in the All Wired Mode

Network Bandwidth of 1 Mbps:

The results for this case were presented earlier in Section 5.2.1. Again, the relative performances of the architectures are similar to those observed at a bandwidth of 10 Mbps (see Figure 5-12 and Figure 5-13). Moreover, the response times at a low system load show the benefits of the communication time saved and concurrency in service in the SPP system. At low system load, the decreased network bandwidth severely hampers the performance of the Sequential system, which requires the most message exchanges per client cycle. At high system load, this effect is reduced by the waiting time in the server's service queues.

5.4.2 All Wired Mode and All Wireless Mode

The experiments carried out with a network bandwidth of 100 Mbps correspond to the All Wired Mode and do not have the Wireless Client Only Mode and All Wireless Mode. In the All Wireless Mode with network bandwidths of 10 Mbps and 1 Mbps, all the communications are over wireless links. The experimental results for $N=1$ and $N=10$ are presented in Table 5-20 and Table 5-21. The response time and the throughput graphs are presented in Figure 5-7, Figure 5-8, and Figure C-3, Figure C-4 in Appendix C. The HPP and the Sequential systems are unaffected by the mode change, since even in the Wireless Client Only Mode all of their message exchanges are over wireless links. As expected, system performances improve with an increase in

network bandwidth (see Table 5-20 and Table 5-21). However, the overall relative performances remain unaltered among the four systems (see Figure 5-7, Figure 5-8, and Figure C-3, Figure C-4 in Appendix C). The SPP system is still the best performer under all load conditions, followed by the HPP, the FPP and the Sequential systems. However, as network bandwidth decreases, the performance improvement of the three parallel systems over the Sequential system tends to increase. This further proves that the three parallel systems benefit from both the reduced communication time and parallel processing.

Table 5 - 20: Effect of Network Bandwidth on Latency at N=1 ($S_A=S_C=50$ ms, $S_B=S_D=55$ ms, $L=50$ Bytes, $D=0\%$, Multi-PP-daemon, All Wired Mode and All Wireless Mode)

Bandwidth (Mbps)	Sequential (ms)	HPP (ms)	FPP (ms)	SPP (ms)
1 (All Wireless Mode)	202.615	144.631	146.777	90.329
10 (All Wireless Mode)	119.669	88.055	98.061	81.647
100 (All Wired Mode)	116.901	85.34	94.686	80.778

Table 5 - 21: Effect of Network Bandwidth on System Throughput at N=10 ($S_A=S_C=50$ ms, $S_B=S_D=55$ ms, $L=50$ Bytes, $D=0\%$, Multi-PP-daemon, All Wired Mode and All Wireless Mode)

Bandwidth (Mbps)	Sequential (cycles/sec)	HPP (cycles/sec)	FPP (cycles/sec)	SPP (cycles/sec)
1 (All Wireless Mode)	27.511	30.422	27.864	31.671
10 (All Wireless Mode)	27.563	30.768	27.922	31.160
100 (All Wired Mode)	27.567	30.784	29.171	31.580

5.5 Effect of Drop Rate

As wireless technologies evolve, the performances of wireless networks are getting better and better. However, the wireless link packet drop rate is still a factor worthy of consideration, due to the radio transmission nature of the wireless environment. This set of experiments is designed to analyze the relative performances of the four systems with different drop rates. Table 5-22 shows the experimental parameters: drop rates are 0% and 2%. Because D affects the TCP level packets, the number of TCP packets used by each of the interaction architectures is computed. Note that these numbers are different from those presented in Table 5-3 that correspond to higher-level messages. The FPP system requires eighteen packet exchanges on the TCP level per client cycle, which is the largest of the four systems. The Sequential and HPP systems require sixteen and fourteen packet exchanges respectively at the TCP level per client cycle. The SPP system still only requires six packet exchanges at the TCP level per client cycle. The reason for this is that each time the client uses a handle obtained from outside its own memory space, it must first set up a communication channel, which requires two TCP level packet exchanges. In addition, the “*send oneway*” call used in the FPP system also requires a TCP level packet returned to acknowledge that the client request packet has been received.

5.5.1 Wireless Client Only Mode

Drop Rate of 2% :

The influence of the wireless link packet drop rate on the mean client response time and the mean system throughput is investigated in low to high system load conditions.

The wireless link packet drop rate is only introduced on the wireless link in this mode.

We assume that the wired parts of the networks do not drop any packet.

Table 5 - 22: Parameters of the Experiment Investigating the Effect of the Drop Rate

Parameters	Level of values
<i>Number of clients (N)</i>	<i>1, 2, 4, 6, 8, 10</i>
<i>Service Times (Exponential distribution)</i>	$S_A=S_C= 50\ ms$ $S_B=S_D= 55\ ms.$
<i>Message size (L)</i>	<i>50 Bytes</i>
<i>Wireless Bw</i>	<i>1 Mbps</i>
<i>Drop rate (D)</i>	<i>0%, 2%</i>
<i>PP-daemon thread</i>	<i>Multithreaded</i>

Low System Load:

As shown in Table 5-23, Figure 5-21 and Figure 5-22, when $N=1$, all the systems experience an increase in response time with an increase in the wireless link packet drop rate. For both values of D (0% and 2%), the SPP system exhibits the shortest response time, due to the fact that it requires the fewest packet exchanges over the wireless link per client cycle, followed by the FPP and the HPP systems. The Sequential system has the longest response time, due to its greater number of packet

exchanges and sequential processing of the client requests. At $D=2\%$ and $N=1$, there is a dramatic improvement in the relative performance of the SPP architecture over the Sequential architecture.

Table 5 - 23: Effect of Drop Rate on Latency at $N=1$ ($S_A=S_C=50$ ms, $S_B=S_D=55$ ms, Wireless Bw=1 Mbps, Wired Bw=100 Mbps, L=50 Bytes, Multi-PP-daemon, Wireless Client Only Mode)

Drop Rate (D)	Sequential (ms)	HPP (ms)	FPP (ms)	SPP (ms)
0%	202.615	144.631	129.504	82.544
2%	906.363	833.063	810.676	91.104

Medium to High System Load:

Since the multi-threaded PP-daemons eliminate the software bottleneck problem (see Section 5.1 for detail), the SPP system leads all the way until the servers are saturated. Surprisingly, the performance of the FPP system in the high load condition becomes extremely bad. Because 12 of 18 TCP level packet exchanges are between the servers and the client in the FPP system, the servers are blocked much longer with the introduction of packet drops, and system throughput is dramatically reduced. However, the SPP and the HPP systems show a performance improvement of 222.6% and 12.8% respectively over the Sequential system.

Bw=1 Mbps, SA=SB=50 ms, SB=SD=55 ms, L=50 bytes, D=2%

Figure 5 - 21: The Response Time of the Multi-Threaded Systems with D=2% in the Wireless Client Only Mode

Bw=1 Mbps, SA=SC=50 ms, SB=SD=55 ms, L=50 bytes, D=2%

Figure 5 - 22: The Throughput of the Multi-Threaded Systems with D=2% in the Wireless Client Only Mode

Table 5 - 24: Effect of Drop Rate on System Throughput at N=10 ($S_A=S_C=50$ ms, $S_B=S_D=55$ ms, Wireless Bw=1 Mbps, Wired Bw=100 Mbps, L=50 Bytes, Multi-PP-daemon, Wireless Client Only Mode)

Drop Rate (D)	Sequential (cycles/sec)	HPP (cycles/sec)	FPP (cycles/sec)	SPP (cycles/sec)
0%	27.511	30.422	27.589	31.440
2%	9.734	10.984	3.886	31.403

Drop Rate of 0%:

The results for this case were presented earlier in Section 5.2.1. The relative performances of the architectures are similar to those observed at D=2% (see Figure 5-12 and Figure 5-13). The response times at low system load show the benefit of the saved communication time and of concurrent service in the SPP system. The decreased drop rate also boosts the performances of the other three systems, and causes the performance gap between them and the SPP system to decrease. The FPP system demonstrates a much better performance in comparison to D=2%.

5.5.2 All Wireless Mode

In the All Wireless Mode, all the communications are over wireless links. The experimental results for N=1 and N=10 are presented in Table 5-25 and Table 5-26. The response time and the throughput graphs are presented in Figure 5-7, Figure 5-8, and Figure D-1, Figure D-2 in Appendix D. The HPP and the Sequential systems are unaffected by the mode change, since even in the Wireless Client Only Mode all of their message exchanges are over wireless links. As expected, system performances

decrease with an increase in drop rate (see Table 5-25 and Table 5-26).

Table 5 - 25: Effect of Drop Rate on Latency at N=1 ($S_A=S_C=50$ ms, $S_B=S_D=55$ ms, Wireless Bw=1 Mbps, L=50 Bytes, Multi-PP-daemon, All Wireless Mode)

Drop Rate (D)	Sequential (ms)	HPP (ms)	FPP (ms)	SPP (ms)
0%	202.615	144.631	146.777	90.329
2%	906.363	833.063	816.248	116.266

However, the overall relative performances are unaltered among the four systems for both D=0% and D=2% (see Figure 5-7, Figure 5-8 for D=0%, and Figure D-1, Figure D-2 in Appendix D for D=2%). The SPP system is still the best performer under all load conditions. For D=2%, the FPP system performs the worst at high system load. However, with D=2%, the performance improvement of the SPP system over the Sequential system tends to increase. This confirms that the SPP system benefits from both the reduced communication time and parallel processing. This set of experiments shows that the wireless link packet drop rate severely affects the performances of the middleware based systems, but by using the proposed SPP system, high performance can be achieved even in an unreliable wireless environment.

Table 5 - 26: Effect of Drop Rate on System Throughput at N=10 ($S_A=S_C=50$ ms, $S_B=S_D=55$ ms, Wireless Bw=1 Mbps, L=50 Bytes, Multi-PP-daemon, All Wireless Mode)

Drop Rate (D)	Sequential (cycles/sec)	HPP (cycles/sec)	FPP (cycles/sec)	SPP (cycles/sec)
0%	27.511	30.422	27.864	31.671
2%	9.734	10.984	3.802	30.856

Chapter 6 Conclusions

6.1 Introduction

This thesis proposed three CORBA-based parallel client-middleware-server interaction architectures. The performance prototypes of the three proposed architectures and the conventional sequential architecture were designed, implemented and thoroughly analyzed. The prototype systems were implemented in the RADS lab using E2A ASP 5.1, a COTS CORBA platform from IONA. The relative performances of the four systems were investigated under different workload conditions. The experimental results were presented in Chapter 5. A summary of the observations is presented, followed by the final conclusions and the directions for future work.

6.2 Summary

6.2.1 The Impact of Multi-Threading

The effect of multi-threading the PP-daemon process on system performance was presented in Section 5.1. With single threaded PP-daemons, the SPP-daemon process becomes the software bottleneck in the SPP system. The other three systems do not suffer from the software bottleneck problem (see Figure 5-4 to Figure 5-6). The single-threaded SPP system, which performs best under a low system load, performs worst at a medium to high system load, due to the limited processing capacity of the

single threaded SPP-daemon process in the system. By multi-threading the PP-daemon processes, the software bottleneck problem in the SPP system is effectively solved. By incorporating multithreading, the performance of the SPP system increases by 154.7% at high system load. Moreover, the SPP system becomes the best performer under all load conditions. The performances of the other three systems remain unaltered, because they never had the software bottleneck problem.

6.2.2 The Impact of Service Demands

A detailed discussion of the effect of service demands on performance is presented in Section 5.2. In both of the Wireless Client Only Mode and the All Wireless Mode, the mean response time and the overall mean system throughput for all the architectures deteriorate as the service demands increase. This is expected because it takes longer for the servers to process each request when the service demands are higher. The multi-threaded SPP system performs best under all load conditions and all the service demands experimented with, followed by the HPP, the FPP and the Sequential systems. The three parallel architectures benefit from the concurrent server invocations at a low system load. In addition to the concurrency, due to the shortened communication time per client cycle (fewer messages exchanged per cycle), the SPP and the HPP systems produce a superior performance in comparison to the Sequential system at medium to high system load.

6.2.3 The Impact of Message Length

The results of experimenting with the effects of Message Length are presented and analyzed in Section 5.3. As with the effect of the service demands, the mean response time and the overall system throughput deteriorate as the message length increases. However, the relative performances of the architectures under different system load conditions change only for an extremely high message size. With a small to medium message size, the SPP system led all the way from low to high system load in both modes, followed by the HPP, the FPP and the Sequential systems. The SPP system produced the best performance with a high message size (10K bytes) in Wireless Client Only Mode, followed by the FPP, the HPP, and the Sequential systems. However, with high message size in the All Wireless Mode, the HPP system performed best under a low system load; the FPP system performed best under a medium system load; and the SPP system performed best under a high system load. This is because of the distribution of the large size message transmissions over several wireless links (see Section 5.3.2 for a detailed discussion). All the systems experienced a network bottleneck, as network communication capacity dominates system scalability.

6.2.4 The Impact of Network Bandwidth

A detailed discussion of the effects of network bandwidth on system performance is

presented in Section 5.4. As network bandwidth decreased, all four systems experienced a decline in system performance. Again, however, the SPP system demonstrated a superior performance over all other systems from low to high system loads, in both modes, with any network bandwidth settings experimented with. The SPP architecture was followed by the HPP, the FPP and the Sequential systems. All the three parallel architectures outperformed the conventional Sequential system in these experiments. The comparable performances gained in the 10 Mbps wireless LAN setting and the 100 Mbps wired network setting, and the reasonable performances even with the 1 Mbps bandwidth setting, demonstrated the promising future of wireless computing.

6.2.5 The Impact of Drop Rate

A discussion of the effect of the wireless link packet drop rate is presented in Section 5.5. The performances of all four systems deteriorate as the wireless link packet drop rate increased. This was due to the prolonged communication time caused by packet loss and retransmissions during each client cycle. Again, the SPP system performed best out of the four systems, under all load conditions, in both modes. Moreover, the deterioration of its performance (28.7% under a low system load and 2.6% under a high system load) when the wireless link packet drop rate was changed from 0% to 2% is very small compared to all the other systems (over 340% under a low system load

and over 170% under a high system load). The FPP system, which performed second under a low system load, performed the worst under a high system load, due to the extra work carried out by its servers (see Section 5.5 for details) when packets were lost. The HPP system outperformed the Sequential system under all load conditions. The 650% performance improvement of the SPP over the Sequential system under a low system load at $D=2\%$ was very impressive.

6.3 Conclusions

In this research, the software bottleneck problem was identified in the single-threaded SPP architecture; other architectures are observed to be free of this problem. Multi-threading the PP-daemon processes effectively eliminated the software bottleneck problem. All three suggested parallel architectures outperformed the conventional sequential architecture in most cases, due to the concurrent server invocations and the fewer message exchanges per client cycle. The SPP architecture demonstrated best performance under most circumstances, except when the system load was low to medium and very high message size was used in the All Wireless Mode. Moreover, the client code in the SPP architecture is the slimmest, which is best for the resource-constrained wireless devices (limited battery life, less powerful CPU, small memory space, etc). The client codes in the other architectures are relatively bulky compared to that of the SPP architecture.

Note that the source code of the E2A ASP 5.1 platform used in this research is not accessible, and a further performance improvement may be gained if the CORBA middleware source code becomes accessible. Given access to the source code, the daemon of any given architecture can be integrated with the CORBA Naming Service. Such integration is expected to give rise to a performance improvement for the parallel interaction architectures.

6.4 Future Work

The following directions for future research are found to be interesting and worthwhile.

- Using an open source CORBA middleware to implement the three parallel architectures by integrating the PP-daemons into the CORBA Naming Service and comparing their performances warrant further investigation.
- A factor-at-a-time approach was used in the experiments. Using an experimental design that can capture the effect of parameter combinations on performance can lead to interesting insights.
- Load balancing is an important issue in the distributed computing domain. How the suggested parallel architectures can be augmented to perform load-balancing tasks in a larger distributed system is a question which requires answering.

- The wireless environment was emulated using NIST Net in this research.

Experiments conducted in a real wireless environment can provide more accurate information with regard to evaluating the performances of the suggested architectures.

- The performances of the client-server systems are also influenced by scheduling policies. Scheduling techniques for achieving desired Quality of Service need further investigation.

References:

- [ABD-97] Istabrak N. Abdul-Fatah, "Performance of CORBA-Based Client-Server Architectures", *M.Eng. Thesis*, Carleton University, Ottawa, Canada, April 1997.
- [ABD-02] Istabrak N. Abdul-Fatah, Shikharesh Majumdar, "Performance of CORBA-Based Client-Server Architectures", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 13, No. 2, February 2002, pp. 111 – 127.
- [ADW-01] Sandeep Adwankar, "Mobile CORBA", in *Proceedings of the 3rd International Symposium on Distributed Objects and Applications (DOA '01)*, Rome, Italy, Sept. 2001, pp. 52 –63.
- [AHM-01] Imran Ahmad, Shikharesh Majumdar, "Achieving High Performance on CORBA-Based Systems with Limited Heterogeneity", in *Proceedings of IEEE International Symposium on Object Oriented Real-Time Computing (ISORC 2001)*, Magdeburg, Germany, April 2001, pp. 350-359.
- [ALE-00a] Markus Aleksy, Axel Korthaus, "A CORBA-Based Object Group Service and a Join Service Providing a Transparent Solution for Parallel Programming", in *Proceedings of International Symposium on Software Engineering for Parallel and Distributed Systems*, Limerick, Ireland, June 2000, pp. 123 –134.
- [ALE-00b] Markus Aleksy, Axel Korthaus, "Implementation Techniques and an Object Group Service for CORBA-Based Applications in the Field of Parallel Processing", in *Proceedings of the Seventh International Conference on Parallel and Distributed Systems*, Iwate, Japan, July 2000, pp. 65 –72.
- [ALE-01] Markus Aleksy, Martin Schader, "A Scalable Architecture for Parallel CORBA-based Applications", in *Proceedings of the International Conference on Parallel and Distributed Computing and Systems*, ACTA Press, Anaheim USA, August 2001, pp. 200-205.
- [DEN-01] Alexandre Denis, Christian Perez, Thierry Priol, "Portable Parallel CORBA Objects: An Approach to Combine Parallel and Distributed

Programming for Grid Computing”, in *Proceedings of European Conference on Parallel Computing (Euro-Par 2001)*, Manchester UK, Aug. 2001, LNCS 2150, pp. 835-844.

- [DHO-01] Dominique Dhoutaut, David Laiymani, “A CORBA-based architecture for parallel applications: experimentations with the W Z matrix factorization”, in *Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid*, Brisbane, Australia, May 2001, pp. 646 -651.
- [FRE-03] Aleksandr Frezinskiy, *Personal Communications*, IONA Technologies PLC, May 2003.
- [FON-89] Michael J. Fontenot, “Software Congestion, Mobile Servers, and the Hyperbolic Model”, *IEEE Transactions on Software Engineering*, Vol. 15, No. 8, August 1989, pp. 947-962.
- [GIL-01] Christopher D. Gill, David L. Levine and Douglas C. Schmidt, “ The Design and Performance of a Real-Time CORBA Scheduling Service”, *International Journal of Time-Critical Computing Systems, special issue on Real-Time Middleware*, Vol. 20, No. 2, Mar 2001, pp. 117 – 154.
- [HAA-99] Mads Haahr, Raymond Cunningham, Vinny Cahill, “Supporting CORBA Applications in a Mobile Environment”, in *Proceedings of the fifth International Conference on Mobile Computing and Networking (Mobi-Com'99)*, ACM, Seattle, USA, August 1999, pp. 36-47.
- [HAA-01] Mads Haahr, Raymond Cunningham, Vinny Cahill, “Towards a Generic Architecture for Mobile Object-Oriented Applications”, *IEEE workshop on Service Portability and Virtual Customer Environments*, San Francisco, USA, Dec. 2000, pp. 91 –96.
- [HEN-99] Michi Henning, Steve Vinoski, “Advanced CORBA Programming with C++”, *Addison Wesley*, 1999.
- [ION-02] IONA Technologies PLC, “Orbix E2A | APPLICATION SERVER PLATFORM CORBA Programmer’s Guide, C++”, Version 5.1, April 2002, available at <http://www.iona.com/support/docs/index.xml>.
- [JAI-91] Raj Jain, “The Art of Computer Systems Performance Analysis--

Techniques for Experimental Design, Measurement, Simulation, and Modeling”, *John Wiley & Sons, Inc.*, 1991.

- [KAM-00] Tsunehiko Kamachi, Thierry Priol, Christophe Rene, “Data Distribution for Parallel CORBA Objects”, in *Proceedings of European Conference on Parallel Computing (Euro-Par 2000)*, Munich Germany, August 2000, LNCS 1900, pp. 1239 - 1249.
- [KEA-97] Katarzyna Keahey, Dennis Gannon, “PARDIS: A Parallel Approach to CORBA”, in *Proceedings of the 6th IEEE International Symposium on High Performance Distributed Computation*, Portland, USA, August 1997, pp. 31-39.
- [LIL-98] Mika Liljeberg, Kimmo Raatikainen, etc. “Using CORBA to Support Terminal Mobility”, in *Proceedings of Global Convergence of Telecommunications and Distributed Object Computing (TINA 97)*, Nov 1997, pp. 59 - 67
- [MAS-02] Cecilia Mascolo, Licia Capra, Wolfgang Emmerich, “Mobile Computing Middleware”, in *Proceedings of the NETWORKING 2002 Conference*, Pisa, Italy, May 2002, LNCS 2497, pp. 20 - 58.
- [NEI-95] Douglas J. Neilson, C. Murray Woodside, Dorina C. Petriu, Shikharesh Majumdar, “Software Bottlenecking in Client-Server Systems and Rendezvous Networks”, *IEEE Transactions on Software Engineering*, Vol. 21, No. 9, September 1995, pp. 776 - 782.
- [NIS-02] National Institute of Standards and Technology, “NIST Net Home Page”, 2002, available at <http://snad.ncsl.nist.gov/itg/nistnet/>.
- [OMG-01] Object Management Group, “Data Parallel Processing”, November 2001, available at http://www.omg.org/documents/specialized_corba.htm.
- [OMG-02] Object Management Group, “Common Object Request Broker Architecture (CORBA) Specification Version 3.0.2”, December 2002, available at http://www.omg.org/documents/corba_spec_catalog.htm.
- [OMG-02a] Object Management Group, “The Real Time CORBA Specification”, Version 1.1, August 02, available at http://www.omg.org/documents/corba_spec_catalog.htm.

- [OMG-03] Object Management Group, "Wireless Access and Terminal Mobility in CORBA specification Version1.0", March 2003, available at http://www.omg.org/documents/corba_spec_catalog.htm.
- [PRI-99] Thierry Priol, Christophe Rene, Guillaume Alleon, "Programming SCI Clusters using Parallel CORBA Objects", in *Scalable Coherent Interface*, LNCS 1734, Springer 1999, pp. 333 - 348.
- [PRI-00] Thierry Priol, Christophe Rene, Guillaume Alleon, "Code Coupling using Parallel CORBA Objects", in *Proceedings of IFIP TC2/WG2.5 Working Conference on the Architecture of Scientific Software*, Ottawa, Canada, October 2000, pp. 105-118.
- [PYR-00] Irfan Pyarali, Carlos O'Ryan, Douglas C. Schmidt, Nanbor Wang and Vishal Kachroo, Aniruddha Gokhale, Alexander Arulanthu, "Applying Optimization Principle Patterns to Design Real-time ORBs", *IEEE Concurrency*, Vol. 8, No. 1, January-March 2000, pp. 16-25.
- [RED-02] Redhat, "Red Hat Linux 7.3 Reference Guide", 2002, available at <http://www.redhat.com/support/resources/howto/rhl73.html>.
- [RUG-00a] Rainer Ruggaber, Jochen Seitz, M. Knapp, " Π^2 - a Generic Proxy Platform for Wireless Access and Mobility in CORBA", in *Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing (PODC'2000)*, Portland, USA, July 2000, pp.191-198.
- [RUG-00b] Rainer Ruggaber, Jochen Seitz, "Using CORBA Applications in Nomadic Environments", in *Proceedings of the Third IEEE Workshop on Mobile Computing Systems and Applications*, Monterey, USA, Dec. 2000, pp.161 -170.
- [SCH-95] Douglas C. Schmidt, Steve Vinoski, "Introduction to Distributed Object Computing", C++ Report, Vol. 7, No. 1, January 1995, pp. 8-10.
- [SCH-97] Douglas C. Schmidt, Aniruddha Gokhale, Timothy H. Harrison and Guru Parulkar, "A High-Performance End System Architecture for Real-Time CORBA", *IEEE Communications Magazine*, Vol. 35, No. 2, Feb 1997, pp. 72 - 77.

- [SCH-98] Douglas C. Schmidt, "Evaluating Architectures for multithreaded Object Request Brokers," *Communications of the ACM*, Vol. 41, No. 10, Oct 1998, pp. 54 – 60.
- [SCH-98a] Douglas C. Schmidt, David L. Levine, Sumedh Mungee, "The Design of the TAO Real-Time Object Request Broker", *Computer Communications*, Vol. 21, No. 4, April 1998, pp. 294 – 324.
- [SCH-00] Douglas C. Schmidt and Fred Kuhns, "An Overview of the Real-time CORBA Specification", *IEEE Computer Special Issue on Object-Oriented Real-time Distributed Computing*, Vol. 33, No. 6, June 2000, pp. 56 - 63.
- [SHE-00] E-Kai Shen, Shikharesh Majumdar, Istabrak Abdul-Fatah, "The Performance of Adaptive Middleware Systems", in *Proceedings of ACM Principles of Distributed Computing Conference (PODC)*, Portland USA, July 2000, pp. 233 - 241.
- [TRI-98] Sebastiano Trigila, Paul Reynolds, Kimmo Raatikainen, Berny Wind, "Mobility in Long-term Service Architectures and Distributed Platforms", *IEEE Personal Communications Magazine*, Vol. 5, No. 4, August 1998, pp. 44 - 55.
- [VIN-93] Steve Vinoski, "Distributed Object Computing with CORBA", C++ Report, Vol.5, No.6, July/August 1993, pp.32-38.
- [VIN-97] Steve Vinoski, "CORBA: Integrating Diverse Applications within Distributed Heterogeneous Environments", *IEEE Communications Magazine*, Vol. 35, No. 2, February 1997, pp.46 - 55.
- [WU-01] Wai-Keung Wu, Shikharesh Majumdar, "Engineering CORBA-Based Systems for High Performance", in *Proceedings of International Conference on Parallel Processing*, Vancouver Canada, Aug. 2002, pp. 473 - 482.

Appendix A: Supplementary Results of the Experiments Investigating the Effect of Service Demand

Bw=1 Mbps, SA=SC=100 ms, SB=SD=110 ms, L=50 bytes, D=0%

Figure A - 1: The Response Time of the Multi-Threaded Systems with Service Time 100, 110 ms in the Wireless Client Only Mode

Bw=1 Mbps, SA=SC=100 ms, SB=SD=110 ms, L=50 bytes, D=0%

Figure A - 2: The Throughput of the Multi-Threaded Systems with Service Time 100, 110 ms in the Wireless Client Only Mode

Bw=1 Mbps, SA=SB=10 ms, SB=SD=11 ms, L=50 bytes, D=0%

Figure A - 3: The Response Time of the Multi-Threaded Systems with Service Time 10, 11 ms in the All Wireless Mode

Bw=1 Mbps, SA=SC=10 ms, SB=SD=11 ms, L=10 bytes, D=0%

Figure A - 4: The Throughput of the Multi-Threaded Systems with Service Time 10, 11 ms in the All Wireless Mode

Bw=1 Mbps, SA=SC=100 ms, SB=SD=110 ms, L=50 bytes, D=0%

Figure A - 5: The Response Time of the Multi-Threaded Systems with Service Time 100, 110 ms in the All Wireless Mode

Bw=1 Mbps, SA=SC=100 ms, SB=SD=110 ms, L=50 bytes, D=0%

Figure A - 6: The Throughput of the Multi-Threaded Systems with Service Time 100, 110 ms in the All Wireless Mode

Appendix B: Supplementary Results of the Experiments Investigating the Effect of Message Length

Bw=1 Mbps, SA=SC=50 ms, SB=SD=55 ms, L=1500 bytes, D=0%

Figure B - 1: The Response Time of the Multi-Threaded Systems with L=1500 Bytes in the Wireless Client Only Mode

Bw=1 Mbps, SA=SC=50 ms, SB=SD=55 ms, L=1500 bytes, D=0%

Figure B - 2: The Throughput of the Multi-Threaded Systems with L=1500 Bytes in the Wireless Client Only Mode

Bw=1 Mbps, SA=SC=50 ms, SB=SD=55 ms, L=1500 bytes, D=0%

Figure B - 3: The Response Time of the Multi-Threaded Systems with L=1500 Bytes in the All Wireless Mode

Bw=1 Mbps, SA=SC=50 ms, SB=SD=55 ms, L=1500 bytes, D=0%

Figure B - 4: The Throughput of the Multi-Threaded Systems with L=1500 Bytes in the All Wireless Mode

Appendix C: Supplementary Results of the Experiments Investigating the Effect of Network Bandwidth

Bw=10 Mbps, SA=SC=50 ms, SB=SD=55 ms, L=50 bytes, D=0%

Figure C - 1: The Response Time of the Multi-Threaded Systems with Bw=10 Mbps in the Wireless Client Only Mode

Bw=10 Mbps, SA=SC=50 ms, SB=SD=55 ms, L=50 bytes, D=0%

Figure C - 2: The Throughput of the Multi-Threaded Systems with Bw=10 Mbps in the Wireless Client Only Mode

Bw=10 Mbps, SA=SC=50 ms, SB=SD=55 ms, L=50 bytes, D=0%

Figure C - 3: The Response Time of the Multi-Threaded Systems with Bw=10 Mbps in the All Wireless Mode

Bw=10 Mbps, SA=SC=50 ms, SB=SD=55 ms, L=50 bytes, D=0%

Figure C - 4: The Throughput of the Multi-Threaded Systems with Bw=10 Mbps in the All Wireless Mode

Appendix D: Supplementary Results of the Experiments Investigating the Effect of Drop Rate

Bw=1 Mbps, SA=SB=50 ms, SB=SD=55 ms, L=50 bytes, D=2%

Figure D - 1: The Response Time of the Multi-Threaded Systems with D=2% in the All Wireless Mode

Bw=1 Mbps, SA=SC=50 ms, SB=SD=55 ms, L=50 bytes, D=2%

Figure D - 2: The Throughput of the Multi-Threaded Systems with D=2% in the All Wireless Mode