REARCHITECTURING AN ELECTRONIC WARFARE SYSTEM BASED ON SERVICE ORIENTED ARCHITECTURE

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

BAKİ ERZURUMLU

IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS FORTHE DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF INFORMATION SYSTEMS

NOVEMBER 2008

Approval of the Graduate School of Informatics

Prof. Dr. Nazife Baykal
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Yasemin Yardımcı
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Aysu Betin Can
Supervisor

Examining Committee Members

Assoc. Prof. Dr. Erkan Mumcuoğlu	(METU, II)	____________________
Assist. Prof. Dr. Aysu Betin Can	(METU, II)	____________________
Onur Aktuğ (MSc.)	(ASELSAN)	____________________
Assoc. Prof. Dr. Onur Demirörs	(METU, II)	____________________
Assist. Prof. Dr.Tuğba Taşkaya Temizel	(METU, II)	____________________

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name	**: Baki ERZURUMLU**
Signature	**: ________________**

ABSTRACT

REARCHITECTURING AN ELECTRONIC WARFARE SYSTEM BASED ON SERVICE ORIENTED ARCHITECTURE

Erzurumlu, Baki
M. S., Department of Information Systems
Supervisor: Assist. Prof. Dr. Aysu Betin Can

November 2008, 66 pages

In this work an electronic warfare system is restructured to service oriented architecture. Service Oriented Architecture (SOA) is a paradigm that realizes rapid and low cost system development. The most important characteristics of SOA are standard based interoperability, which allows services developed on different platforms to run together, and dynamic composition via discovery, which provides dynamic composition of application at runtime using the existing services.

The old warfare system that was developed by ASELSAN Inc. contained embedded software and was designed using traditional object oriented techniques. In this thesis, we have extracted services out of the system and restructured the warfare system based on service oriented principles.

In this thesis, we have focused on the dramatic effect of reusability when SOA is introduced to the electronic warfare system. To understand the effect of service orientation, the new system is evaluated in terms of line of code, memory consumption and extra CORBA interface communication overhead.

Keywords: SOA, interoperability, CORBA, reusability

ÖZ

SERVİS YÖNELİMLİ MİMARİ PRENSİPLERİNE GÖRE ELEKTRONİK HARP SİSTEMİNİN YENİDEN MİMARİ YAPILANDIRILMASI

Erzurumlu, Baki
Yüksek Lisans , Bilişim Sistemleri Bölümü
Tez Yöneticisi: Yrd. Doç. Dr. Aysu Betin Can

Kasım 2008, 66 Sayfa

Bu tez çalışmasında bir elektronik harp sistemi servis yönelimli mimariye (SYM) uygun olarak yeniden yapılandırılmıştır. SYM hızlı ve düşük maliyetli sistem geliştirmeyi gerçekleştiren bir değerler dizisidir. SYM'nin en önemli karakteristikleri, servislerin farklı platformlarda geliştirilmesine imkân sağlayan standart tabanlı birlikte işlevsellik ve çalışma zamanında mevcut servislere göre uygulamanın dinamik olarak oluşumunu sağlayan keşif yoluyla dinamik oluşumdur. ASELSAN tarafından geliştirilmiş olan ve içerisinde gömülü yazılımları içeren eski elektronik harp sistemi geleneksel nesne eğilimli tekniklere göre tasarlanmıştır. Bu tezde, sistem dışarısına servis çıkartılarak elektronik harp sistemi SOA prensiplerine göre yeniden yapılandırılmıştır.

Bu tezde, Elektronik Harp Sistemine SYM’nin dâhil edilmesinin yeniden kullanılabilirliğe olan etkisi üzerine durulmuştur. Servis yöneliminin etkisinin anlaşılması için yeni sistem kod satır sayısı, bellek kullanımı ve ekstra CORBA ara yüzünün getirmiş olduğu ek yükler bakımından değerlendirilmiştir.

Anahtar Kelimeler: SYM, birlikte işlevsellik, CORBA, yeniden kullanılabilirlik

To My Parents and My Lovely Sister

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my supervisor Assistant Professor Aysu Betin Can for his guidance, advice, criticism, encouragement, endless patience and insight throughout the completion of the thesis.

I am indebted to all of my friends and colleagues for their support and encouragements. I am also grateful to ASELSAN Inc. for the facilities that made my work easier.

My family, no words can help me to express my feelings, but at least I can say that I am grateful to you for the life you provide to me.

TABLE OF CONTENTS

ABSTRACT IV

ÖZ VI

DEDICATION VIII

LIST OF TABLES XII

LIST OF FIGURES XIII

LIST OF ABBREVIATIONS XIV

CHAPTER

1. INTRODUCTION 1

2. LITERATURE SURVEY 4

  2.1 SERVICE ORIENTED ARCHITECTURE 4

    2.1.1 Service Orientation Paradigm 4

    2.1.2 SOA Defined 5

    2.1.3 SOA Collaborations 5

    2.1.4 Service 7

    2.1.5 SOA and Quality Attributes 9

    2.1.6 Web Service 11

  2.2 COMMON OBJECT REQUEST BROKER ARCHITECTURE (CORBA) 14

    2.2.1 CORBA Event Service and Notification Service Specification 17

  2.3 SERVICE EXTRACTION 18

3. DESIGN AND IMPLEMENTATION 21

  3.1 CURRENT ARCHITECTURE 22

    3.1.1 Application Software 25

3.1.2 Embedded Software 26
3.2 PROBLEMS IN THE CURRENT ARCHITECTURE 27
3.3 REFACTORED ARCHITECTURE 28
3.3.1 Service Implementation 28
3.3.2 Solving the Multiple User Problem 30
3.3.3 The System Architecture After Refactoring 36
4. EVALUATION AND DISCUSSION 39
4.1 REUSABILITY ANALYSIS 39
4.1.1 Interface Metrics for Reusability 39
4.1.2 Service Usage Measurements 43
4.2 CODE MEASUREMENT RESULTS 44
4.3 PERFORMANCE COMPARISON OF CORBA AND WS 45
4.4 MEMORY CONSUMPTION MEASUREMENTS 46
4.5 COMMUNICATION PERFORMANCE EXPERIMENTS 50
4.5.1 Experiment 1 52
4.5.2 Experiment 2 54
4.5.3 Experiment 3 56
4.5.4 Experiment 4 58
4.5.5 Experimental Results 60
5. CONCLUSION 61
REFERENCES 63

LIST OF TABLES

Table 2-1 Evolution strategies 19
Table 4-1 Tactical Record Service Interface Metrics 42
Table 4-2 Service Usage Code Measurements.......... 43
Table 4-3 Service Usage Effort Measurements 43
Table 4-4 Code Measurement Result.......... 44
Table 4-5 The measured performance on a local machine 46
Table 4-6 The measured performance on the networked machine 46
Table 4-7 Transferred data in experiments 52
Table 4-8 Results for Current Architecture (Single Computer).......... 53
Table 4-9 Results for Refactored Architecture (Single Computer) 56
Table 4-10 Results for Current Architecture (Multiple Computer) 58
Table 4-11 Results for Refactored Architecture (Multiple Computer).......... 59

LIST OF FIGURES

Figure 2-1 Collaborations in a service-oriented architecture 6
Figure 2-2 Web service collaboration 13
Figure 2-3 Idl Definition 16
Figure 2-4 A request passing from client to object implementation 16
Figure 3-1 Logical System Architecture 21
Figure 3-2 Current Software Architecture 23
Figure 3-3 StartRecord Message Sequence Diagram for Current Software 24
Figure 3-4 Applications and System Software 25
Figure 3-5 Observer Pattern Structure 32
Figure 3-6 Refactored Architecture 36
Figure 3-7 StartRecord Message Sequence Diagram after Service Included 38
Figure 4-1 Curent Application Software Memeroy Consumption 47
Figure 4-2 Tactical record service memory consumption 48
Figure 4-3 Refactored application software memory consumption 49
Figure 4-4 Elapsed Time Calculation Sequence Diagram 51
Figure 4-5 Experiment-1 Testbed Configuration 53
Figure 4-6 Results for Current Architecture (Single Computer) 54
Figure 4-7 Experiment-2 Testbed Configuration 55
Figure 4-8 Results for Refactored Architecture (Single Computer) 56
Figure 4-9 Experiment-3 Testbed Configuration 57
Figure 4-10 Results for Current Architecture (Multiple Computer) 58
Figure 4-11 Experiment-4 Testbed Configuration 59
Figure 4-12 Results for Refactored Architecture (Multiple Computer) 60

LIST OF ABBREVIATIONS

APP: Arguments Per Procedure

ARS: Argument Repetition Scale

BPEL: Business Process Execution Language

COMPOSE: Component-Oriented Software Engineering

CORBA: Common Object Request Broker Architecture

COTS: Commercial-off-the-shelf

DAC: Distinct Argument Count

DAR: Distinct Argument Ratio

DCOM: Distributed Component Object Model

DII: Dynamic Invocation Interface

DSI: Dynamic Skeleton Interface

ESB: Enterprise Service Bus

GPS: Global Positioning System

GUI: Graphical User Interface

HTTP: Hypertext Transfer Protocol

IDL: Interface Definition Language

INS: Inertial Navigation System

IOR: Interoperable Object Reference

IR: Interface Repository

JacORB: Java Object Request Broker

OA: Object Adapter

OMG: Object Management Group

ORB: Object Request Broker

RMI: Remote Method Invocation

SOA : Service Oriented Architecture

SOAP:	Simple Object Access Protocol
TAO:	The ACE ORB
UDDI:	Universal Description, Discovery and Integration
W3C:	The World Wide Web Consortium
WS:	Web Service
WSDL:	Web Services Description Language
XML:	The Extensible Markup Language

CHAPTER 1

INTRODUCTION

Over the last few years, electronic warfare systems have grown in number of functionalities [1], leaving companies to handle increasingly complex system architectures. Traditional system architectures have reached to the limit of their capabilities, while traditional requirements of electronic warfare systems persist.

In order to a system be rapid in production and to have low cost, the system should have the following characteristics [2]:

- Adaptability to various changes,
- Enhancement of system quality,
- Reduction of operation and maintenance cost

In the 21st century, based on the Internet popularization, SOA attracts attention as it realizes above characteristics [3].

SOA is a design framework for the construction of software systems by "union of services". A service is a software unit that can process assigned functionalities in a stand-alone manner and which can be requested by standardized procedures. Each service runs on heterogeneous environment, namely different platforms, operating systems and programming languages. Hence, the service should be easily added or replaced or re-used. The granularity of the service differs according to functionality that it implements. There are several types of services which are [4]:

1 - Re-use of existing systems such as an application program on the legacy system or a package,

2 – Commercially universal services provided over internet,

3 - Services provided from third party organizations,

4 - Newly developed, application specific services.

As in the other business fields, service-oriented architecture has emerged as a solution to the complex requirements of electronic-warfare systems. As ASELSAN Inc., has been producing electronic warfare systems for years, we have decided to the design functionalities, which have almost the same requirements, as a separate services which guides us to SOA. The main reason of choosing SOA as the implementation architecture is the foreseen improvements in code reusability and maintainability. By SOA usage we plan to increase the code reusability and decrease the implementation duration.

The aim of this research is investigating the affects of SOA on the architecture of Electronic Warfare Projects' software in terms of reusability and performance.

In this thesis, a common functionality of electronic warfare systems developed in ASELSAN Inc., "Tactical Record" feature is implemented as a service. Several experiments are performed in order to evaluate the contribution of service-oriented architecture to the system software. More specifically, the thesis presents the measurements about the effect of service orientation on reusability. In addition to reusability, time and memory consumption measurements are presented to elaborate the overhead that the service orientation has brought in performance. Finally, some code and implementation measurements are made in order to see benefits of SOA from the programmer’s perspective.

This thesis is organized as follows: In Chapter 2, a literature survey on SOA and CORBA is presented. The current architecture, which is the system software under investigation before restructuring, and the refactored architecture are detailed in Chapter 3. The details of the extracted service are discussed in the same chapter. Chapter 4 presents an evaluation of the current and the refactored architecture. Finally, in Chapter 5, the presented work is concluded.

CHAPTER 2

LITERATURE SURVEY

In this chapter, Service Oriented Architecture is reviewed. After presenting SOA, CORBA and Web Services technologies which are appropriate to implement a SOA are defined.

2.1 SERVICE ORIENTED ARCHITECTURE

This section briefly defines the concepts, principles and technologies of service-orientation and service-oriented architecture.

2.1.1 Service Orientation Paradigm

According to [5], "Service Oriented Architecture (SOA) is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains and implemented using various technology stacks". SOA describes a set of patterns and methodologies for developing loosely coupled, business-aligned services for the separation of concerns between description, implementation, and binding.

2.1.2 SOA Defined

There are several definitions of SOA existing in the literature; some of them focus on technical perspective, some take a business perspective and a few focuses on architectural perspective. The World Wide Web Consortium (W3C) technically focuses and defines SOA as “A set of components which can be invoked, and whose interface descriptions can be published and discovered” [6]. This definition of SOA underlines the technical edge of architecture, whereas architecture is to be taken as a paradigm and a set of practices.

From an architectural perspective, SOA is defined as architecture for an application which is developed using a set of services. SOA describes system functionality as a set of shared, reusable services [7]. But, it is not just a system which is constructed as a set of services. A system developed using SOA could still include code which implements functionality specific to that system.

2.1.3 SOA Collaborations

The most fundamental form of SOA contains three components; a Service Consumer, a Service Provider and a Service Registry as shown in Figure 2-1. These are interacting with each other to supply/perform automation.

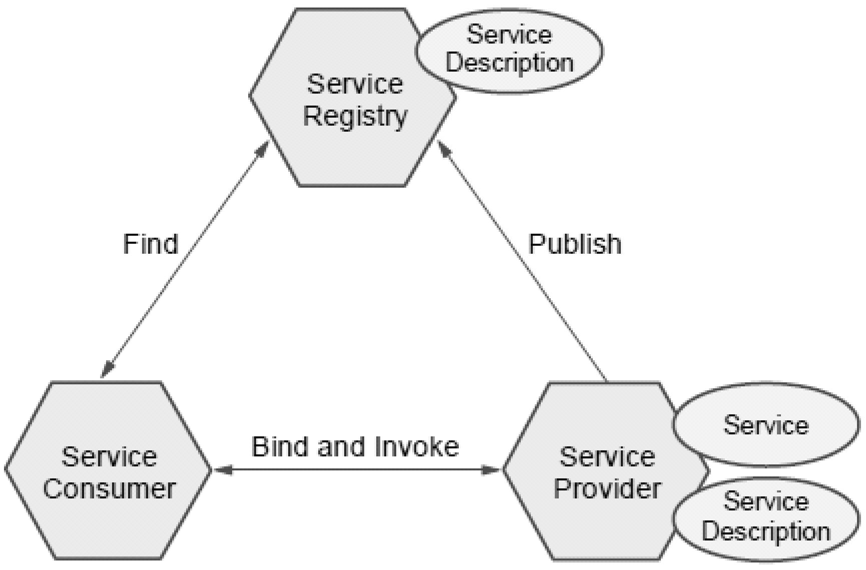


Figure 2-1 Collaborations in a service-oriented architecture [8]

In [2] roles in a service-oriented architecture are summarized as:

- Service Consumer: The service consumer is an application, a software component or another service which needs a service. It initiates the query of the services in the repository, binds to the service over a broker, and triggers the service functionality. The service consumer consumes the service according to the service interface.

- Service Provider: The service provider is a network-addressable entity which performs requests from consumers. It publishes its services and interface contract to the service repository so that the service consumer can invoke and access the service.

- Service Registry: A service registry is used for service discovery. It includes a service repository which consists of available services and allows for the lookup of service provider interfaces to interested service consumers.

Each component in the SOA can play one or more of the three roles of service provider, consumer and registry.

The operations in a service-oriented architecture are:

- Publish: A service description must be published to the registry so that it can be discovered and invoked by a service consumer.

- Find: A service consumer identify a service by querying the service registry for a service that meets its requirements.

- Bind an Invoke: After recovering the service description, the service consumer invokes the service according to the information in the service description.

2.1.4 Service

In a SOA, services are the building blocks from which an application or system is developed. "A service can be defined as an implementation of a well-defined piece of business functionality, with a published interface that is discoverable and can be used by service consumers when building different applications and business processes" [9]. As seen in the above description, the technology used to develop the service, such as a programming language or operating system, does not form the definition of a service. As a result of not being described in terms of platform specifications (operating system, programming language), we can say that interoperability is provided in SOA.

Although no official principles are defined for SOA there exists accepted set of principles. These can be summarized as follows [2]:

- Services are reusable: Services are designed and developed to support reuse
- Services share a formal contract: For services to interact they only need to share a formal contract that describes each service and defines the terms of information exchange
- Services are loosely coupled: SOA is a loosely coupled architecture because it separates the interface from the implementation. Services share interface contract, not implementation. Runtime discovery decreases the dependency between service producers and consumers and makes a SOA system even more loosely coupled.
- Services abstract underlying logic: The only part of a service provider which is visible to the outside world is the service interface. So a service interface encapsulates the service implementation.
- Services are compassable: Service may form other services. This brings to build services that have different functionalities from the same set of smaller services. So we can say that this principle increases reusability.
- Services are autonomous: Services have distinct boundaries. They should be stand-alone and should not depend on the state of other services or functions.
- Services are stateless: Services do not have to manage state information, since that can prevent their ability to remain loosely coupled.

- Services are discoverable: Services should have human-readable interface contracts in order to be discovered by programmers and service consumers.

2.1.5 SOA and Quality Attributes

Some quality attributes that may be important in helping SOA to achieve an organization’s business aims and their impacts are as follows:

2.1.5.1 Interoperability:

“Interoperability refers to the ability of a collection of communicating entities to share specific information and operate on it according to an agreed-upon operational semantics” [10]. Enhancement of interoperability is the most important benefit of SOA.

The use of communication standards (Web Service, CORBA, RMI, DCOM), SOA provides good interoperability as it allows services to interact each other which are implemented in different languages and deployed on different platforms.

2.1.5.2 Performance:

According to the [9] “Performance is related to response time (how long it takes to process a request), throughput (how many requests overall can be processed per unit of time), or timeliness (ability to meet deadlines, i.e. to process a request in a deterministic and acceptable amount of time).” SOA have an unfavorable impact on the performance of a system because of the network delays and the overhead caused by brokers which manage communication. Require of communication over the network increases the reaction time.

In order to meet the systems performance requirements, both the service consumer and providers' system architecture must be designed and evaluated with awareness [9].

On the positive side, SOA service providers can be forwarded from location to location without affecting service consumers. Thus, this location transparency improves the total throughput and accessibility of the system.

2.1.5.3 Extensibility:

Extending applications by adding additional services or implementing additional functionalities into existing services is supported in SOA. Extensibility for architecture is very important because requirements of the systems are continually changing and evolving in modern software development processes. Therefore, the service interface must be designed and evaluated carefully by the developers to make sure that it can be extended without causing a major impact on the service consumers [9].

2.1.5.4 Testability:

"Testability is the degree to which a system or service facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met"[11]. Using SOA negatively impact the testability because of the complexity of the testing of services which are distributed across a network.

It is very difficult to test services which are provided by external organizations. If access to the source code of these services is not possible, and if they implement runtime discovery of services it is not possible to identify which services are used until a system executes. Therefore it is nearly impossible to test these services [9].

2.1.6 Web Service

The W3C's Web Services Architecture Working Group has jointly come to agreement on the following working definition of a Web service:

"A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards" [6].

Web services are technology which is well suited to apply a service-oriented architecture. Web services are self-describing and modular applications that execute business logic as services can be published, discovered, and invoked over the network. Web services can be developed as loosely coupled application components using any programming language, or any operating system based on XML standards. This feature provides the delivery of applications as a service accessible to anyone, anytime, at any location, and using any platform.

Web services can be published, discovered, and invoked over the computer network. The standards required to do so are [6]:

- Simple Object Access Protocol: SOAP is a standard for exchanging XML-based messages over a computer network, using HTTP. SOAP composes the foundation layer of the Web services and provides a basic messaging framework, which more abstract layers can build on.

- Web Service Description Language: WSDL is the standard format for describing a Web service. A WSDL describes how to access a Web service and what functionalities it will execute, and serves as an interface contract between the service provider and service consumer.

- Universal Description, Discovery, and Integration: UDDI is a XML-based protocol that provides a distributed directory that enables business to list them on the Internet and discover other Web services.

Some of the key features of Web services are:

- Self-contained.
- Self-describing
- Modular.
- Published, located, and invoked across the Web.
- Language independent and interoperable.
- Inherently open and standards based.
- Dynamic
- Composable

Figure 2-2 shows a typical Web service collaboration that is based on the SOA model shown previously in Figure 2-1.

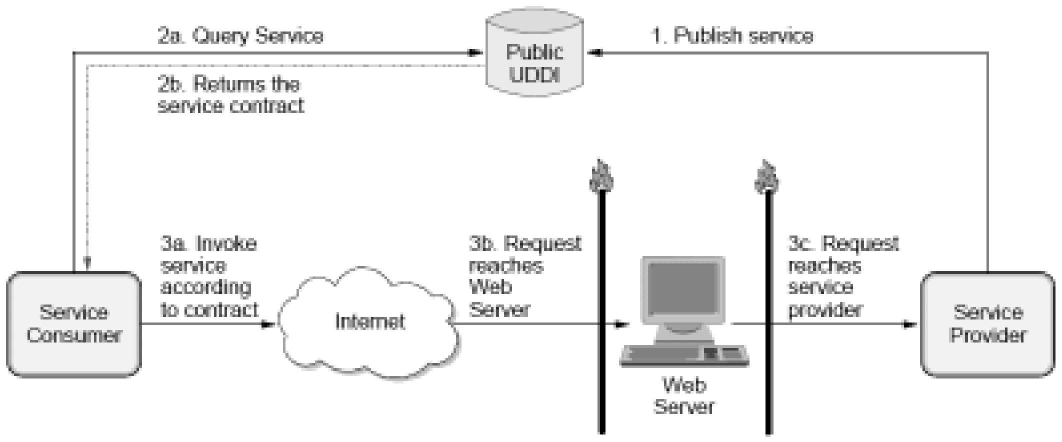

2a. Query Service
Public UDDI
1. Publish service
2b. Returns the service contract
3a. Invoke service according to contract
Internet
3b. Request reaches Web Server
3c. Request reaches service provider
Service Consumer
Web Server
Service Provider

Figure 2-2 Web service collaboration

It is important that Web services are not the only technology which can be used to apply a service-oriented architecture. Many other technologies such as CORBA, RMI exist which can be used to develop SOA.

2.1.6.1 Web Service-based Event Notification

Web services-based event notification systems provide connection between event driven architecture, which is a set of design methodologies for loosely-coupled system architectures based on event notifications, and SOA. Features of both the event notification mechanisms and the Web services technologies are merged in these systems. WS technologies are used to deliver event notifications and manage subscriptions in WS-based event notification systems.

Three similar specifications are proposed for Web services-based event notification systems. The first is WS-Events which was the earliest one and proposed by HP. The second is WS-Eventing [12] which has a broader vendor support (Microsoft, IBM, Sun and CA) and the latest version was released in August 2004. The last specification, WS-Notification [13, 14], is supported by IBM and Globus Alliance.

This specification was approved as an OASIS standard in October 2006. WS-notification is more complex than other standards [15]. Therefore, it has three individual specifications. These specifications are: WS Base Notification, WS Brokered Notification and WS Topics.

WS-Eventing and WS-BaseNotification, which describes basic interactions between the notification consumer and producer, have similar architecture and functionalities. They both support the Publish/Subscribe paradigm, in this paradigm a subscriber subscribes to one or more events and an event producer publish events. Publish/Subscribe architectures are same in these two specifications. However, at the SOAP message level; they are incompatible with each other.

WS-Eventing allows defining only one filter which is a default Xpath filter. WS-BaseNotification defines three types of filters which are TopicExpression, ProducerProperties and MessageContent. A notification consumer can use one or all of these filters.

As discussed before, there is no commonly proposed specification for the Web services event notification systems. Different specifications have been proposed in this area “specifically WS-Notification and WS-Eventing are two major initiatives.” [15]. Unfortunately, because of the incompatibility among these specifications, it is not possible to guarantee the interoperability of two event notification mechanisms in two different systems, although they both use Web service technologies.

2.2 COMMON OBJECT REQUEST BROKER ARCHITECTURE (CORBA)

Common Object Request Broker Architecture (CORBA) [16] is an Object Management Group (OMG) standard for distributed object computing (DOC). OMG was founded in 1989 to develop, accept and support specifications for developing applications in distributed heterogeneous environments. CORBA is one of the first specifications that have been adopted by OMG. TAO [17] and JacORB [18] are among the most widely used open-source CORBA implementations.

The main motivation of CORBA is the object invocation where the objects may deploy locally or remotely. A CORBA based application from any vendor, on any operating system, programming language and network can communicate with another CORBA based application. CORBA is a client-server model for distributed computing, and is formed from six main components:

- Object Request Broker (ORB) Core
- Interface Repository
- Dynamic Invocation Interface (DII)
- Dynamic Skeleton Interface (DSI)
- Object Adapters (OA)
- Interface Definition Language (IDL)

For each object type, an interface is defined in an IDL file. The IDL interface definition is independent of programming language and operating system, but maps to most of the popular programming languages like C, C++, and Java etc. The IDL file is compiled to generate client stubs and server skeletons for a given language. Figure 2-3 illustrates the idl definition which contains a structure definition and a single interface.

```
struct TimeOfDay {
    short   hour;       // 0 - 23
    short   minute;     // 0 - 59
    short   second;     // 0 - 59
};

interface Time {
    TimeOfDay   get_gmt();
};
```

Figure 2-3 Idl Definition

The communication between clients and objects is established by a component called ORB. When a client wants to invoke an operation on an object that is in the server, the ORB is used by the client to specify the required operation and marshal (serialize) the arguments that will be sent. When the invocation request reaches the server, the same interface is used to unmarshal the arguments. After performing the requested operation, the results are marshaled according to the interface and sent to the requesting client. The last step of the remote operation call is the unmarshaling (read - deserialize) of the result. In Figure 2.4, a sample remote operation call is illustrated.

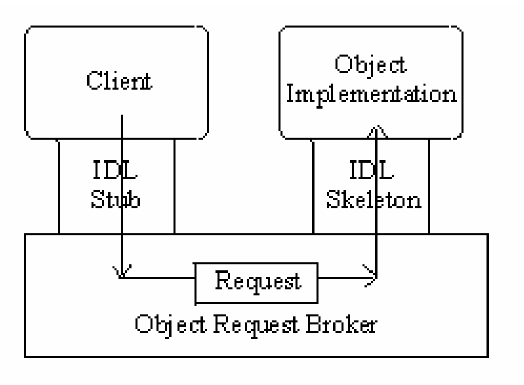


Figure 2-4 A request passing from client to object implementation [19]

2.2.1 CORBA Event Service and Notification Service Specification

CORBA describes Event services and Notification services which decouple the communication between CORBA objects and allow exchange of events in an asynchronous format. These services permit multiple suppliers to communicate with multiple consumers asynchronously and without knowing each other. These services apply publish/subscribe paradigm and specifications of these services describe both the interfaces and infrastructures for CORBA notification systems [20] Push model and the Pull model are two different models that can be used to operate these services. In the push model, the provider sends an event into the event channel and these events are delivered by the event channel to all registered consumers. In the pull model, the consumer requests an event from the event channel by invoking a pull operation.

The CORBA Event Service specification [21] was first introduced in March, 1995. CORBA Event Service defines a basic mechanism for event notification but it has several limitations. These limitations are:

- Event filtering is not supported; an event consumer receives all events on channel.
- There is no hierarchical structure for events.
- Different qualities of service (QoS) are not supported.

The CORBA Notification Service Specification [22] is developed to solve the limitations of the CORBA Event Service. The major aim of the CORBA Notification Service is to improve the CORBA Event Service by supporting event filtering and QoS. Consumers of the CORBA Notification Service subscribe to events they are interested in by associating filter object. The CORBA notification service specification defined “Structured Events”, which is useful for effective filtering, supplies a well-defined data structure to map a generic event to a well structured event. CORBA Notification specification defines 13 QoS properties. Each channel, each connection, and each message can be configured to support QoS.

2.3 SERVICE EXTRACTION

Several methodologies and strategies have defined for migrating existing systems to SOA. Brief summaries of two approaches, that address different problems, are discussed in the proceeding part of this section. First the Renaissance [23] method, which takes a reengineering approach to maintenance, is explained. And then Component-Oriented Software Engineering (COMPOSE)[24] method, which is a service oriented process for building applications using software components, is briefly described.

Renaissance

Renaissance method presents a set of maintenance strategies which put reengineering above replacement [23]. One of the reasons that makes reengineering to be used instead of replacement is reengineering of the system eliminates the high cost and risks that replacement brings.

Renaissance covers continued maintenance, reengineering, and system replacement. Reengineering is a general term and renaissance method defines four types of reengineering. Six evolution strategies are as shown in Table 2-1:

Table 2-1 Evolution strategies [23]

Continued Maintenance	Accommodating change in a system, without radically changing its structure, after it has been delivered and deployed.
Revamp	Transforming a system by modifying or replacing its user interfaces. The internal workings of the system remain intact, but the system appears to have changed to the user.
Restructure	Transforming a system's internal structure without changing any external interfaces.
Rearchitecture	Transforming a system by migrating it to a different technological architecture
Redesign for Reuse	Transforming a system by redeveloping it, using some of the legacy system components.
Replace	Totally replacing a system

COMPOSE

There are obvious similarities between components and services, and because of these similarities, a process for evolving an existing system using commercial-off-the-shelf (COTS) components might be a proper choice for application to evolve systems to SOA. According to [24] COMPOSE method is used to “evolve a legacy freight tracking system so that it supported the demanding requirements of the company's larger customers”.

COMPOSE uses the concepts of service providers and service consumers as an integral model of the system being developed. The services, that will meet the requirements of the system, are gathered by mapping the existing components and the functional requirements. By this way, COMPOSE models an existing system with a series of refined sub-systems that provide and consume services. The resulting model can then be used as a roadmap for incremental evolution.

COMPOSE doesn’t explicitly address the entire business context of the proposed activity and this is the potential weakness of applying COMPOSE [24].

These approaches present some perspectives on the migration-to-SOA and they address many of the important issues. For a business which migrates to SOA, must be ready to divide the existing systems to granular services, because these systems support the core business processes. The enterprise service model is the definition of the key processes for business, after it is mapped onto existing functionalities. Renaissance provides some important pointers for determining the feasibility of reengineering an existing system into services. If developers have used the enterprise service architecture, which is a plan for the organisation’s business services bus, as a supplementary input to the Renaissance process, it might provide useful information into the service creation. COMPOSE could be used to model an enterprise service architecture and map service definitions onto components that can deliver those services.

CHAPTER 3

DESIGN AND IMPLEMENTATION

The restructured project, namely ELECTRO-WAR, is a military electronic warfare project which is designed for mission and time critical operations. It is simply formed of application software running on a PC, embedded software running on embedded target boards and hardware which the algorithms are running on. The logical architecture of ELETRO-WAR is given in Figure 3-1.

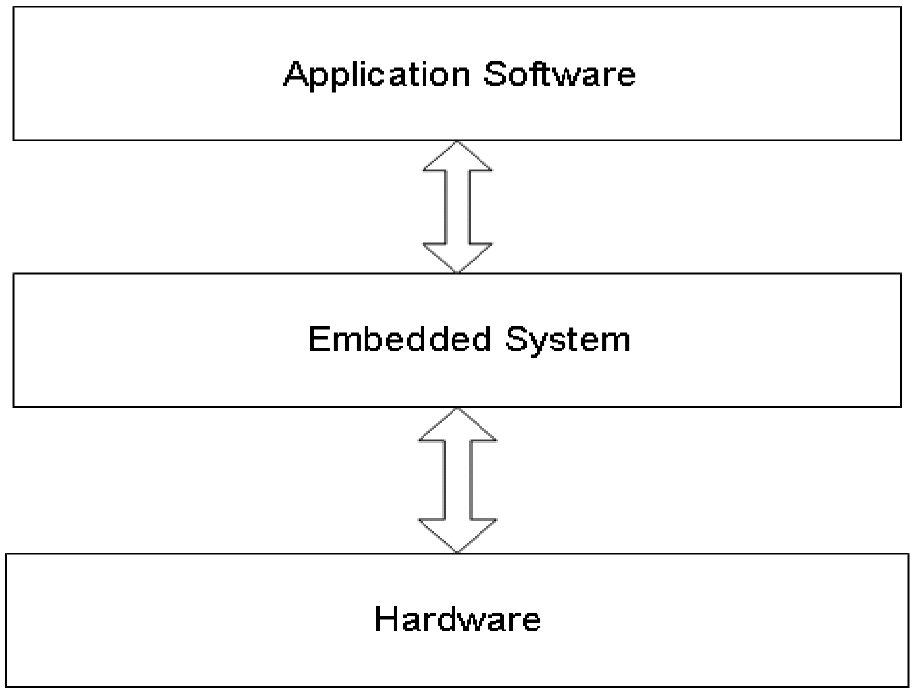


Figure 3-1 Logical System Architecture

In this chapter the current and refactored software architecture of ELECTRO-WAR are explained in detail.

3.1 CURRENT ARCHITECTURE

The current software architecture involves application software, embedded software and algorithm software. Application software is deployed on desktop PC and includes different types of modules which are customized based on project requirements. Embedded software runs on embedded target boards and manages the system scenarios and controls the hardware via special device drivers.

Figure 3-2 illustrates the ELECTRO-WAR's current architecture drawn by using Unified Modeling Language (UML) Deployment Diagram. The ELECTRO-WAR software is designed as a three-tier model: The Application Software which is implemented in JAVA runs on a Windows Operating System, the Control Software which is implemented in C++ runs on the real time operating system VxWorks[25] and the algorithm software which is implemented in C runs on MCOS.[26]

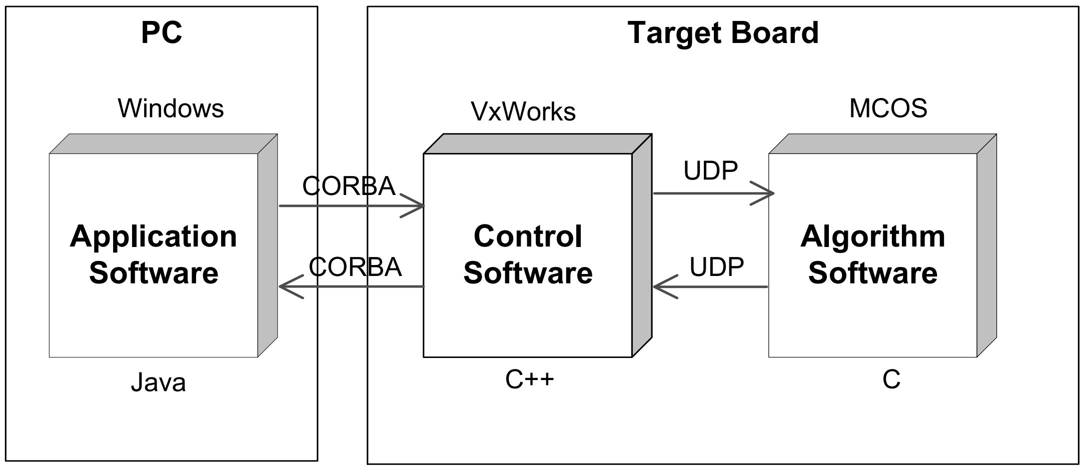


Figure 3-2 Current Software Architecture

One of the main functions of the ELECTRO-WAR project is the tactical record facility. It can simply be explained as the facility of recording the target's characteristics. Because the project is classified, the implementation details of this functionality can not be written in this thesis work.

To describe the system operation and communication between the software blocks of ELECTRO-WAR, the message lifecycle of StartRecord message is given in Figure 3-3 as an example. To start a new record, operator pushes the start record button on the GUI, then GUI sends StartRecord message to the embedded control software. When StartRecord message reaches to the control software, it sends this message to the relevant algorithm software. If algorithm software starts the record correctly it will send RecordStarted message to the control software. Right after the control software taking the RecordStarted message, it sends the acknowledgment message to the GUI. Then GUI shows the "record started" information to the operator.

The details of the application and embedded software of ELECTRO-WAR project are given in the proceeding sections.

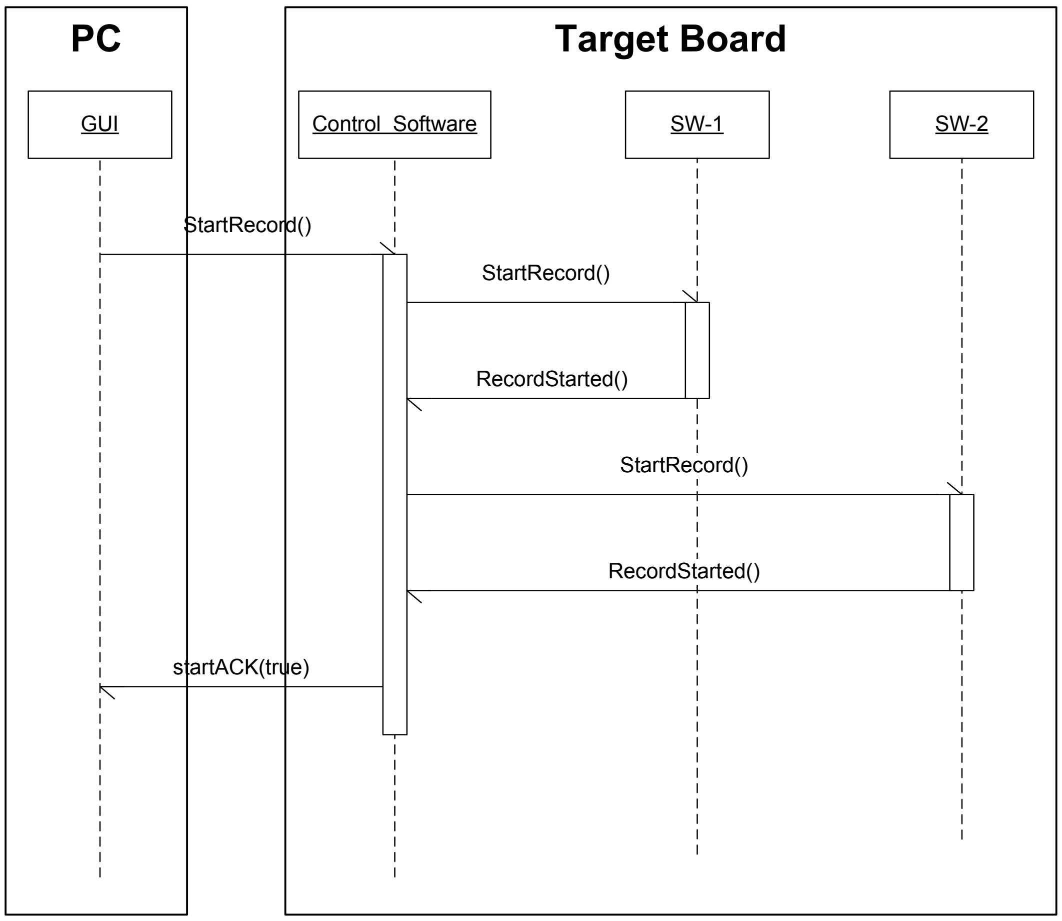


Figure 3-3 StartRecord Message Sequence Diagram for Current Software

3.1.1 Application Software

"Application software is a subclass of computer software that employs the capabilities of a computer directly and thoroughly to a task that the user wishes to perform"[27] It is designed for end users so it is also called end-user programs. As seen on Figure 3-4 application software sits on top of system software because it is unable to run without the operating system and system utilities.

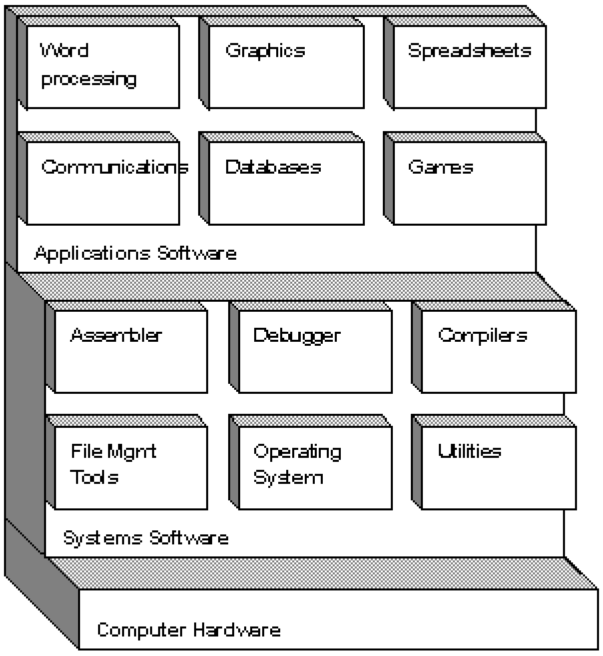


Figure 3-4 Applications and System Software

ELECTRO-WAR's application software includes database programs, map applications, analysis tools and various drivers which are necessary for communication with the peripheral hardware Global Positioning System (GPS), Inertial Navigation System (INS), and Power Distribution Unit). It manages the upper level system scenarios by interacting with the other system software and hardware drivers.

ELECTRO-WAR’s application software communicates with the embedded software over early determined interfaces. It collects the user input via the graphical user interface (GUI), creates messages after evaluating these inputs and sends them to Control Software over a CORBA interface. After receiving the message that is sent by the Control Software it shows the related messages to the operator in a suitable format. Moreover it serves a graphic output screen to the user. The graphics are plotted by using the incoming data from the embedded software.

3.1.2 Embedded Software

“An embedded system is a physical system that employs computer control for a specific purpose. Unlike a general purpose computing system, an embedded system does one or a few predefined tasks. Embedded systems do not provide standard computing services and they usually form a part of a larger system.” [28]

A typical embedded system has a central processing unit (CPU), a main memory unit (MMU) and its peripherals such as device drivers, converters and interfaces. ELECTRO-WAR’s embedded software is responsible for managing the lover level system scenarios. It is composed of a control and algorithm software.

The control software of ELECTRO-WAR project is responsible for controlling the hardware in order to accomplish a specific mission. It communicates low-level algorithm software and hardware drivers via its interfaces. It is implemented in C++ language.

The algorithm software of ELECTRO-WAR project is composed of multiple functional blocks, each of which is responsible for running a single algorithm. It interacts just with the control software via its UDP interface. The control messages and their results are sent over this communication interface.

As nearly all embedded software, the ELECTRO-WAR project's embedded software has some distinguishing characteristics such as:

- Designed for specific purposes.
- Offers computer control.
- Generally designed for usually mission and time critic purposes.
- Offers predictable delays.
- Have timing constraints.

In the next section the problems observed in the current architecture of ELECTRO-WAR project are explained.

3.2 PROBLEMS IN THE CURRENT ARCHITECTURE

Even if the current architecture works properly, it needs refactoring for giving reaction to the changing requirements as soon as possible. Moreover it should also be refactored in order to serve the requirements of new short-timed projects. Problems in the current architecture are summarized as follows:

- Because of the inflexible formation of the current architecture, it is not possible to reuse the features, which are common nearly in all electronic warfare projects, without recoding them.
- As the programmers have different perspectives; same requirements are implemented in a different manner in different projects.

- In the current architecture common features are not distributed so it is not possible to use these features in another application which is located on another PC or on the same PC.
- As the common features are built in to the projects when they are not designed according to SOA, these common features will not be available for multi-user at the same time.

3.3 REFACTORED ARCHITECTURE

As discussed in the previous section, the current architecture of ELECTRO-WAR project has some problems. In order to solve these problems we decided to implement the tactical record function of the software as a service. With this implementation we planned to make the software more flexible and improve the system performance.

In this section implementation of observer pattern in CORBA and the implemented service are discussed.

3.3.1 Service Implementation

The application software of ELECTRO-WAR project is designed based on Model View Controller architectural pattern. This pattern isolates the business logic from user interface so it is easy to modify the business rules without affecting the user interface. Thus extracting the tactical record functionality implementation from the application software does not require much labor.

The tactical record feature is selected to be a service and extracted from the application software. This makes the application software to become a service consumer to the Tactical Record service. Tactical record functionality has been implemented four times in different projects in ASELSAN Inc. In these different projects the functionality of this feature is common. So it is not difficult to determine the requirements of the Tactical Record Service.

Implementation details, class diagrams, sequence diagrams and business logic of the tactical record service are not mentioned in this thesis because of the classification. Only general properties of the tactical record service are mentioned in the proceeding part of this section.

Tactical report service has an interface with service consumers and embedded control software over CORBA. Web service or Enterprise Service Bus (ESB) is not selected because of performance factor. Web service and all ESBs are using XML format in communication. The use of XML as the data representation format creates extra overhead in the communication and processing of data. XML messages can be 10 to 20 times larger than the equivalent binary format, so sending them over a network takes longer. Because XML uses a text format, it has to be processed before any operation is executed. XML processing consists of at least three separate operations (parsing, validation and transformation), all of which are CPU and memory related operations [29].

Tactical record service has a configuration file which is written in XML format. The configuration file is as follows:

```
...
<System>
          <SystemNo>1</SystemNo>
          <Language>EN</Language>
</System>
<FolderInfo>
          <tdgb>/archive/test/</tdgb>
          <tddb>/archive/test/</tddb>
          <iorFolder>D:/berzurumlu/ior/</iorFolder>
</FolderInfo>
<Corba>
          <ControlSWSource>NamingService</ControlSWSource>
</Corba>
...
```

System no, language of the service, folder information (archive and ior folder) and the source of the Control Software CORBA object (naming service or ior) can be configured by using this file.

In order to avoid recoding, the common features are implemented as services. Because of the service providers have to be multiple service consumers, there will be a mechanism to notify consumers about changes in the service provider. To solve this multiple user problem we used a different mechanism. This mechanism is discussed in the next section.

3.3.2 Solving the Multiple User Problem

In order to notify multiple users about a state change in a service, there should be a mechanism which provides a one-to-many relationship between the service provider and consumers. Because of the communication performance overhead WS is not selected as communication type; therefore, WS-based event notification solutions can not be used. CORBA notification service is not selected because in this service clients cannot subscribe for notification of particular service change also clients cannot specify the delivery order and delivery interval so a new notification mechanism is constructed using the Observer Pattern.

Since the communication model between the tactical record service and the application software is structured around the observer pattern, a detailed discussion of this pattern is given in this section.

The Observer pattern “defines[s] a one-to-many dependency between objects so that when one object [the subject] changes state, all its dependents [its observers] are notified and updated automatically.” [30]. Observer generally are used in event-driven systems, such as OS and GUI implementations ,where objects need to react to state changes elsewhere in the system (e.g., external input) without knowing when these events might occur. The key objects in this pattern are subject and observer. A subject may have any number of dependent observers. All observers are notified whenever a change occurs in the subject’s state. In response, each observer will query the subject to synchronize its state with the subject's state.

The structure of the pattern is illustrated in Figure 3-5.

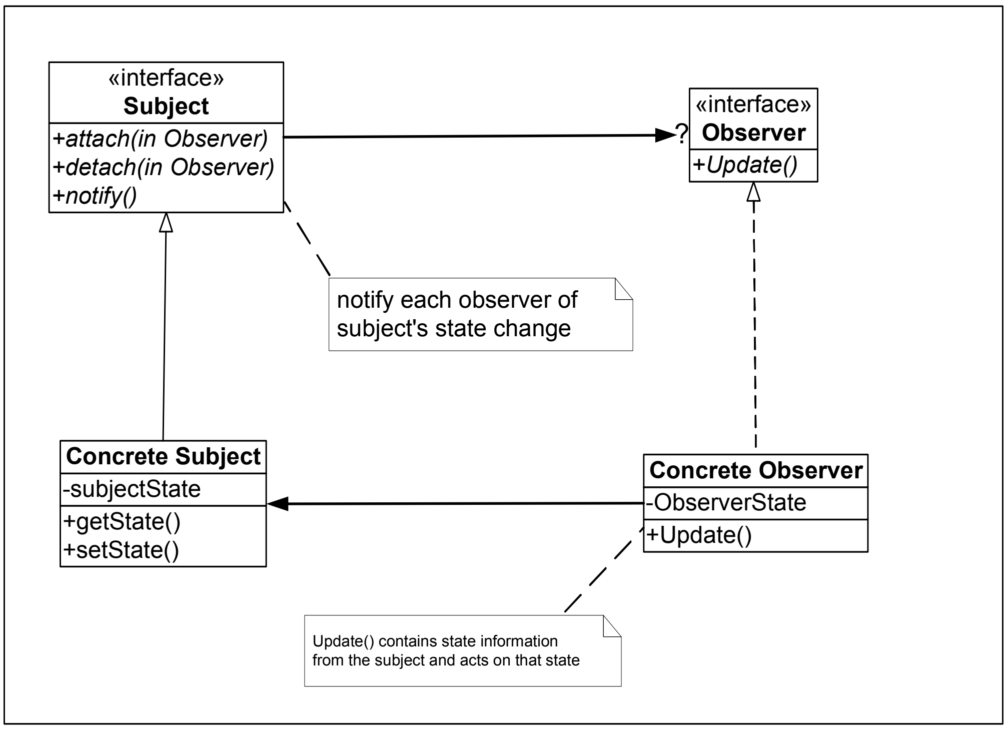


Figure 3-5 Observer Pattern Structure[30]

The participants of the structure are as follows:

Subject: Subject provides an interface for attaching and detaching Observer objects. Any number of Observer objects may observe a subject.

Observer: Observer defines an updating interface for objects that should be notified of changes in a subject.

ConcreteSubject: ConcreteSubject sends a notification to its observers when its state, which ConcreteObserver objects interested in changes.

ConcreteObserver: ConcreteObserver is the observing object which implements the Observer updating interface to keep its state consistent with the subject's.

ConcreteSubject is responsible for the notification of its observers in the case of change that may affect the observers' current state. After this notification ConcreteObserver may query the ConcreteSubject which causes the notification for more information. This information then used by the ConcreteObserver to update its state.

Since CORBA is chosen as the communication protocol, concrete subject and the concrete observers communicate over CORBA protocol. Details of the observer pattern implementation in CORBA are given in the proceeding part of this section.

A part of the IDL file is as follows:

```
interface RecordObserver
{
...
        oneway void updateRecordStarted();
        oneway void updateRecordStopped();
        oneway void updateDiscFull();
...
}
interface RecordInterface
{
...
        oneway void attach(in RecordObserver observer);
        oneway void detach(in RecordObserver observer);
        oneway void StartRecord();
        oneway void StopRecord();
...
}
```

Two different interfaces are defined in this IDL file. One is RecordObserver and the other is RecordInterface.

RecordObserver Interface

RecordObserver interface defines an updating interface for service consumers that should be notified of changes in a service provider. The implementation details of these updating methods in the service provider are as follows:

```
protected void update(RecordObserver observer)
    {
        switch (stateUpdated)
        {
         ...
            case RECORD_STARTED:
                ((RecordObserver ) observer).updateRecordStarted();
                break;
            case RECORD_STOPPED:
                ((RecordObserver ) observer).updateRecordStopped();
                break;
            case DISC_FULL:
                ((RecordObserver ) observer).updateDiscFull();
                break;
          ...
        }
    }
```

When record started message reaches the service provider, it notifies its early attached consumers using updateRecordStarted message. The usages of other messages are similar.

RecordInterface Interface

RecordInterface defines capabilities of the service this means that service providers use the service provider functionalities over this interface. Methods of this interface can be separated into two groups. One group is for observable methods (attach (), detach ()) and the remaining part is functionality methods. Functionality methods are not discussed in this thesis because these methods give information about the operational concepts of the electronic warfare system. Details of the observable methods as follows:

Attach()

Attach method attaches new service consumer to the service provider. After consumer attaches to the provider, it begins to take notify messages which are defined in RecordObserver interface. A service consumer has to be implementing the RecordObserver interface in order to attach to the provider. Usage of the attach method by the consumer is as follows:

```
...
org.omg.CORBA.Object object = null;
       try
       {

           object = CORBAServer.getObjectFromNS("TacticalRecord-1");
           tacticalRecord = TacticalrecordInterfaceHelper.narrow(object);
           if (tacticalRecord._non_existent())
              throw new Exception();
           TacticalRecordObserver observer =
       TacticalRecordObserverHelper.narrow(CORBAServer.getPOA().servant_t
o             _reference(new TacticalRecordObserverPOATie(this)));

            tacticalRecord.attach(observer);

       } catch (Exception e)
       {
            e.printStackTrace();
       }
...
```

Detach()

Detach method detaches the attached service consumer from the service provider.

3.3.3 The System Architecture After Refactoring

Figure 3-6 illustrates the ELECTRO_WAR's new architecture, which is a service oriented architecture. The system now has one extra tier: Service tier which runs on windows and implemented in Java. Tactical record service is accessed via network and it supports multi-user. In this architecture application software is fully decoupled from the tactical record implementation. Accessing the records is available via the service component only.

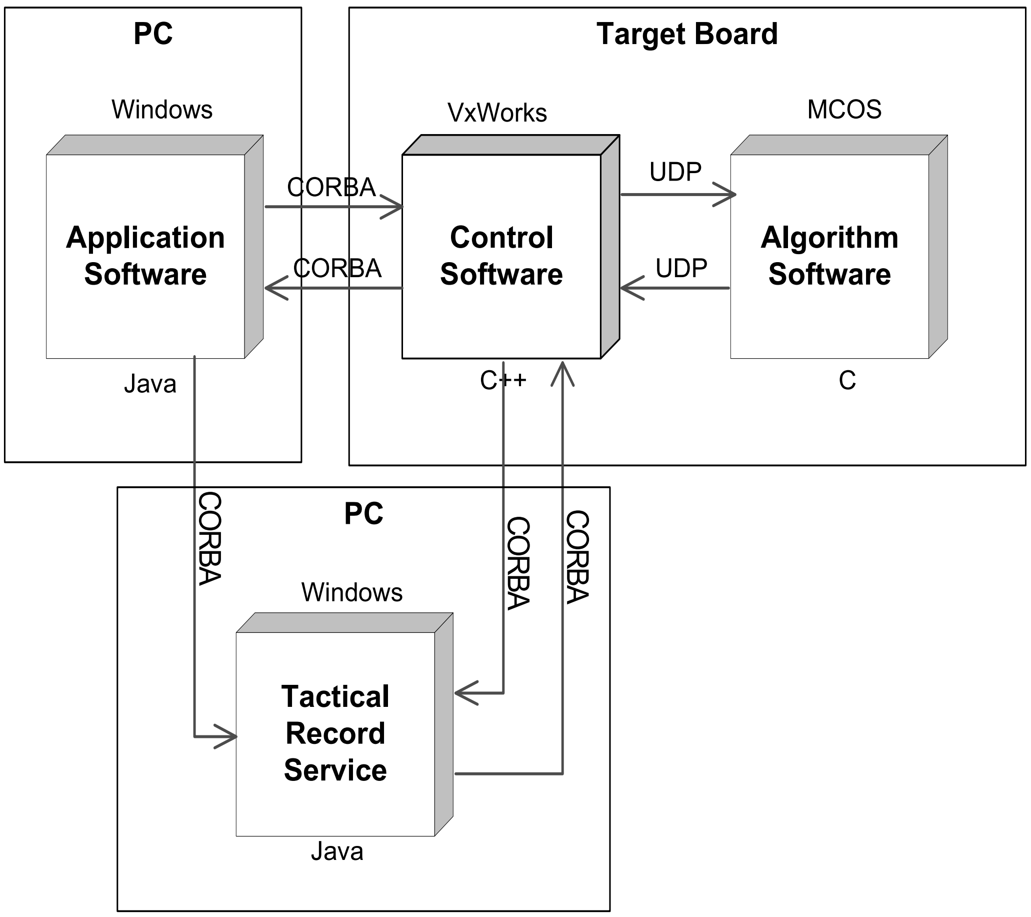


Figure 3-6 Refactored Architecture

Figure 3-7 illustrates a UML sequence diagram for StartRecord Message after the service is included. In this new architecture, application software has to be attached to the tactical record service to keep the track of the changes in the service. To start a new record, one of the operators pushes the start record button on the GUI, then GUI sends StartRecord message to the tactical record service. In proper circumstances tactical record software sends the StartRecord message to the embedded control software. When StartRecord message reaches to the control software, it sends this message to the relevant algorithm software. If algorithm software starts the recording correctly, it will send RecordStarted message to the control software. Right after the control software taking the RecordStarted message it sends the acknowledgment message to the tactical record service. Then tactical record service notifies all early attached service consumers.

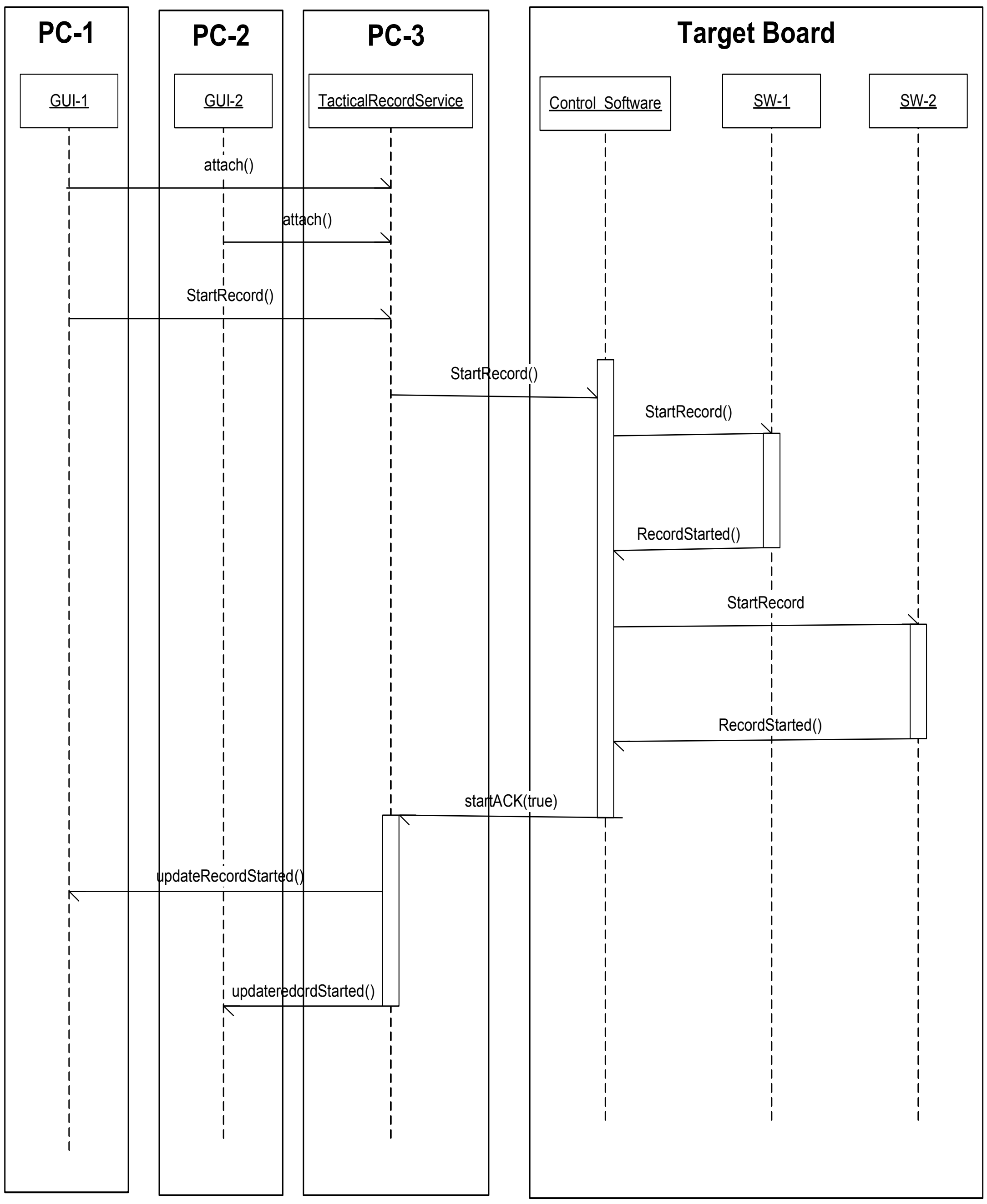


Figure 3-7 StartRecord Message Sequence Diagram after Service Included

CHAPTER 4

EVALUATION AND DISCUSSION

This chapter first presents reusability analysis of the tactical record service. Then, after discussing the testing environment, the results of the performance evaluations of the tactical record service for 1, 10,100 and 1000 records are presented. Memory consumptions and, comparison results of the CORBA and Web Service are also discussed in this chapter.

4.1 REUSABILITY ANALYSIS

It is necessary to measure the reusability of services in order to analyze the reusability factor. In this thesis, two different metrics are used to measure the reusability introduced by the use of the tactical record service.

4.1.1 Interface Metrics for Reusability

According to [31] understandability of service interfaces is a major quality that affects the reusability of a service. In this section a set of interface metrics are analyzed for measuring understandability and reusability of tactical record service. These metrics are arguments per procedure (APP), distinct argument ratio (DAR), and argument repetition scale (ARS) [31]. Mean values of the reference services which are taken from [31] and p-values [32] of the interface metrics are used to evaluate the interface metrics of the tactical record service. Interfaces of 12 reference services were chosen to provide empirical data and variety for analyzing of the metrics.

P-value: P value is associated with a test statistic. It is "the probability, if the test statistic really were distributed as it would be under the null hypothesis, of observing a test statistic [as extreme as, or more extreme than] the one actually observed " [32].

A p-value of .05 or less rejects the null hypothesis "at the 5% level" that is, the statistical assumptions used imply that only 5% of the time would the supposed statistical process produce a finding this extreme if the null hypothesis were true.

5% and 10% are common significance levels to which p-values are compared.

Arguments Per Procedure (APP): This metric uses two properties of a given interface. Procedure Count (η_p) is the total count of procedures that are publicly declared by an interface. Argument Count (η_a) is the total count of arguments of the publicly declared procedures. Mean size of procedure declarations of an interface is measured in APP, and is described as:

$$APP = \frac{n_a}{n_p}$$

“It is believed that procedures with fewer arguments are easier to understand, and so will be easier to reuse. It follows that component interfaces with lower APP will tend to have better reusability”. [31]

Distinct Argument Ratio (DAR): Distinct Argument Ratio (DAR) is a derivative metric which is intended to be size independent. DAR is defined as:

$$DAR = \frac{DAC}{n_a}$$

where η_a is the Argument Count of the interface and Distinct Argument Count(DAC), defined as

$$DAC = |A|$$

where *A* is the set of name-type pairs used as arguments in an interface.

Enhancement of the service reusability will be provided by declaring arguments consistently .It can be said that, interfaces with lower DAC and DAR are declared more consistently.

Argument Repetition Scale (ARS): The Argument Repetition Scale (ARS) metric is defined as:

$$ARS = \frac{\sum_{a \in A} |a|^2}{n_a}$$

where A is the set of name-type pairs, |a| is the count of procedures in which argument name-type a is used, and η_a is the Argument Count of the interface [31].

Greater ARS shows more consistent interface that leads us to better reusability.

Results: Table 4-1 illustrates average values and standard deviations of the reference service interfaces metrics; and the interface metrics values of the tactical record service and the p-values of these metrics.

Table 4-1 Tactical Record Service Interface Metrics

	Reference Services		Tactical Record Service	
	Average	Standard Deviation	Metric value	p-values
Distinct Argument Ratio (DAR)	0,37	0,093	0,32	0,055
Arguments Per Procedure(APP)	1,92	0,78	1,29	0,042
Argument Repetition Scale (ARS)	8,64	4,88	7	0,1

Statistical p-value calculations show that for each metrics the difference between the tactical record service and reference architectures is considered to be statistically significant.

The tactical record service has lower APP than average value of the reference services. In tactical record service every procedure contains maximum 2 arguments and that decreases the APP value of the tactical record service. And lower APP increases the reusability of the tactical record service.

Average DAR value of the reference services is greater than the tactical record service DAR value and it shows that the tactical record service arguments are declared more consistently. And consistency tends to lead to easier understanding and reusing.

The ARS value of the tactical record service is smaller than average ARS value of the reference service. This result shows that procedures in the tactical record service have less repetition in their arguments. Lower ARS is not good for the reusability of the services but it is not expected that smaller interfaces have larger ARS because larger interfaces will generally have more arguments and contains more repetitions than the small interfaces.

4.1.2 Service Usage Measurements

In this section reusability of the service is analyzed in terms of effort (man-hour) which takes to use the tactical record service and code which have to be implemented to consume the tactical record service.

Table 4-2 Service Usage Code Measurements

	Line of Code	Number of Class
Tactical Record Service	8546	123
Tactical Record Service Consumer	695	5

Table 4-2 shows that tactical record service has 8546 lines code 123 classes. On the other hand, 695 lines code and 5 classes is enough to use the tactical record service in an application. This means that new electronic warfare project in ASELSAN, which applies SOA and has to implement tactical record functionality, implements only 695 lines code to supply the tactical record requirements and this is nearly %8 of the total functionality code. So it brings %92 code reusability.

Table 4-3 Service Usage Effort Measurements

	Man-hour
Tactical Record Functionality	920
Tactical Record Service	245
Tactical Record Consumer	43

During the development of the ELECTRO-WAR application, the software team used extreme programming (XP). The tactical record functionality is implemented by four people in six iterations. For each iteration, necessary measurements are taken so it is not difficult to calculate elapsed time for implementing the tactical record functionality in ELECTRO-WAR. Table 4-3 illustrates that implementing the tactical record functionality takes 920 hours in ELECTRO-WAR. Refactoring the ELECTRO-WAR and extracting tactical record service took 245 hours and it is nearly the 25% of the previous implementation in ELECTRO-WAR. This effort is spent *once* and includes design, implementation, unit tests and documentation. But only in 43 hours consumers can begin to use tested and documented tactical record service. This 43 hours effort includes only service usage and integration tests. Service discovery and requirement analysis need extra effort. Even though this extra effort increases the usage time, there is a big difference between using the tactical record service and implement tactical record functionality one more time. Thus, the total gain only for man-hour is nearly 95%.

4.2 CODE MEASUREMENT RESULTS

Table 4-4 illustrates the line of code and the number of class measurement results of the application software and the tactical record service.

Table 4-4 Code Measurement Result

	Line Of Code	Number Of Class
Current Application Software	105151	1497
TacticalRecord Service	8546	123
New Application Software	100120	1446

It is expected that the line of code difference between the current application software and the new application software to be nearly equal to the line of code of the tactical record service. But the measurement results show that this is not the case. And also the number of class measurements shows the similar behavior as the line of code measurement results. Factors that cause these results are summarized as follows:

- Some classes are used both in the application and service software. When we implement the common functionalities as in the current architecture, the commonly used classes are coded just once. However when we implement the common functionalities as services we are obliged to code those common classes both in the application and the service software.

- In the current architecture, since the common functionalities are hard coded within the application software, these functionalities have just one CORBA interface, with control software. However, when we pick the common functionalities out of the current architecture and implement them as services, the newly implemented service becomes to have two CORBA interfaces, both with the control and application software. This results in an increase in the number of classes and line of code.

4.3 PERFORMANCE COMPARISON OF CORBA and WS

Table 4-5 shows the average time of 10 repetitions of the method calls on a local machine. All methods include 1000 invocations with different parameters. These experiments show that a web service has more overheads than CORBA. In a web service (WS), XML messages are used and these messages 10 to 20 times larger than the equivalent binary format, so sending them over a network takes longer. Because of XML uses a text format, it has to be processed before any operation is executed. Another factor of the slow WS is that SOAP has one more network layer than CORBA. [29, 33]

Table 4-5 The measured performance on a local machine [33]

	Time (milisecond)				
	int	char	Byte	Char Array	Struct
CORBA	495,6	526,4	523,3	629,7	498,6
WS	15164	11700	11344	13736	11198

Table 4-6 shows the average time of 10 repetitions of the method calls on networked computer. All methods include 1000 invocations with different parameters. At the networked machines, the technologies additionally have the network overheads. In the case of a Web Service usage, since the used procedures are similar, the timing overheads are more or the less same to those caused of a local machine's procedure calls. In fact these overheads consist only of the network latency. [33]

Table 4-6 The measured performance on the networked machine [33]

	Time (milisecond)				
	int	char	Byte	Char Array	Struct
CORBA	1396,1	1458,1	1415	1458,2	1418,2
WS	14500	11403	11352	13156	11469

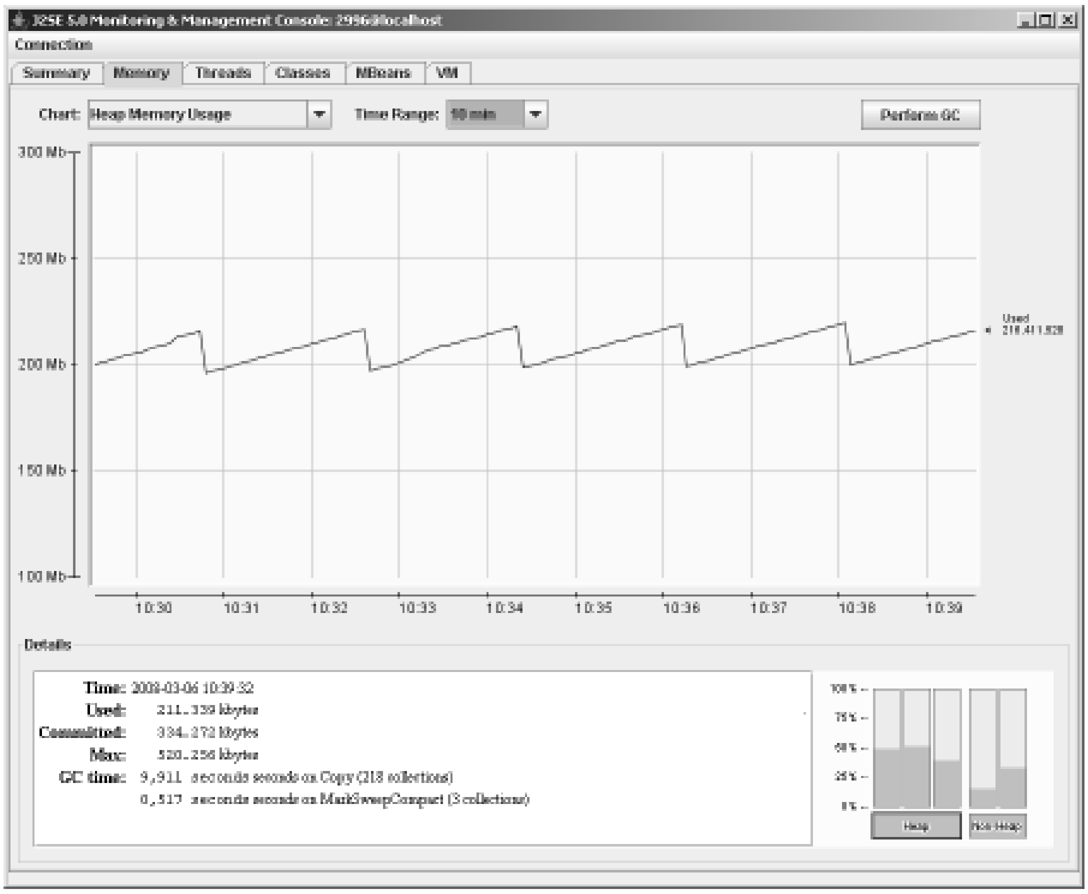
As WS is slower, consume more memory, more network bandwidth, and more CPU cycles than CORBA, CORBA is selected as a communication protocol.

4.4 MEMORY CONSUMPTION MEASUREMENTS

In this part memory consumptions of the service and the application software are discussed. Measurements are taken by JConsole [34] which is a JMX-compliant graphical tool for monitoring a Java virtual machine.

Current application software:

Figure 4-1 illustrates the memory consumption of the current application software. The average consumption is about 210 MByte. Garbage Collector causes instantaneous decreases of the memory consumption.

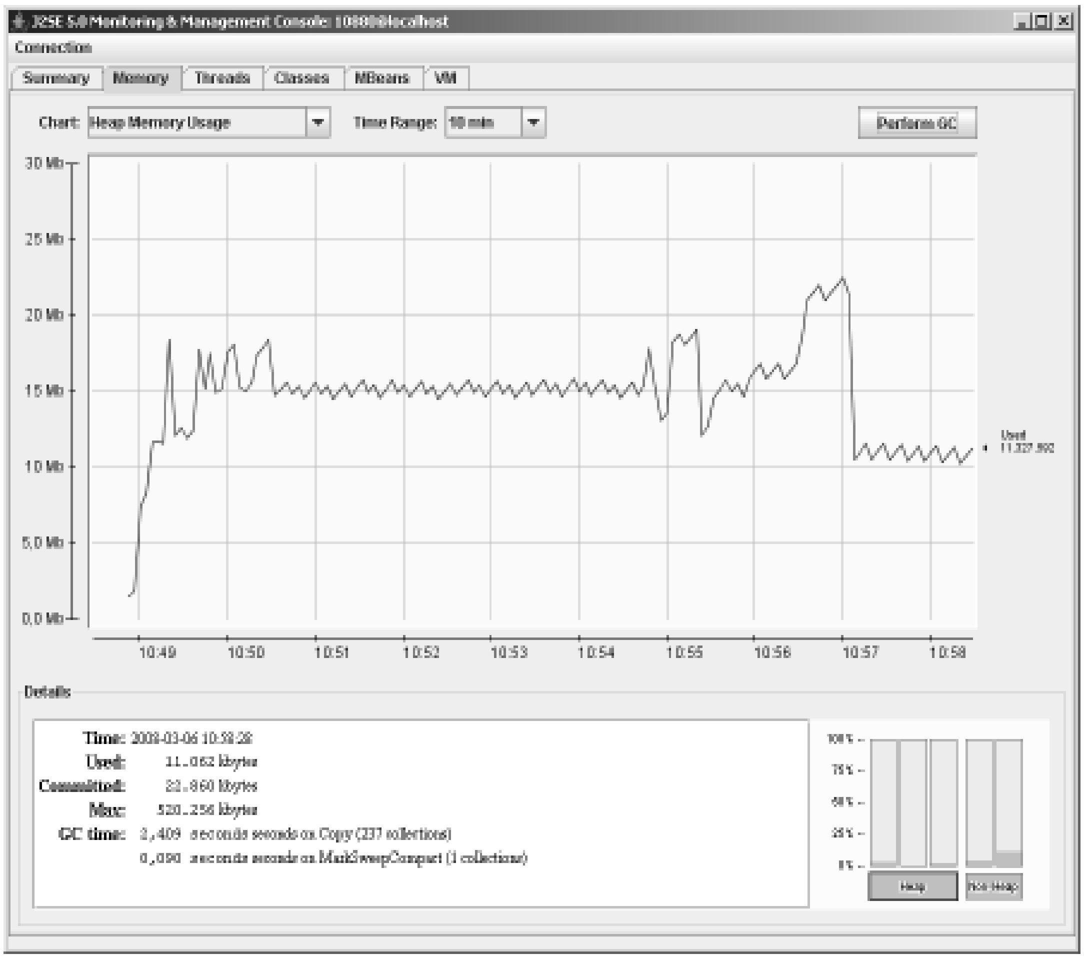


```
      Time: 2008-03-06 10:39:32
      Used:    211.339 kbytes
 Committed:    334.272 kbytes
       Max:    520.256 kbytes
   GC time:  9,911  seconds seconds on Copy (218 collections)
             0,517  seconds seconds on MarkSweepCompact (3 collections)
```

Figure 4-1 Curent Application Software Memeroy Consumption

Tactical Record Service:

Figure 4-2 illustrates the memory consumption of the current application software. In idle case tactical record service consumes 15 MByte memory space. But when the service effectively used it consumes 23 MByte memory space.

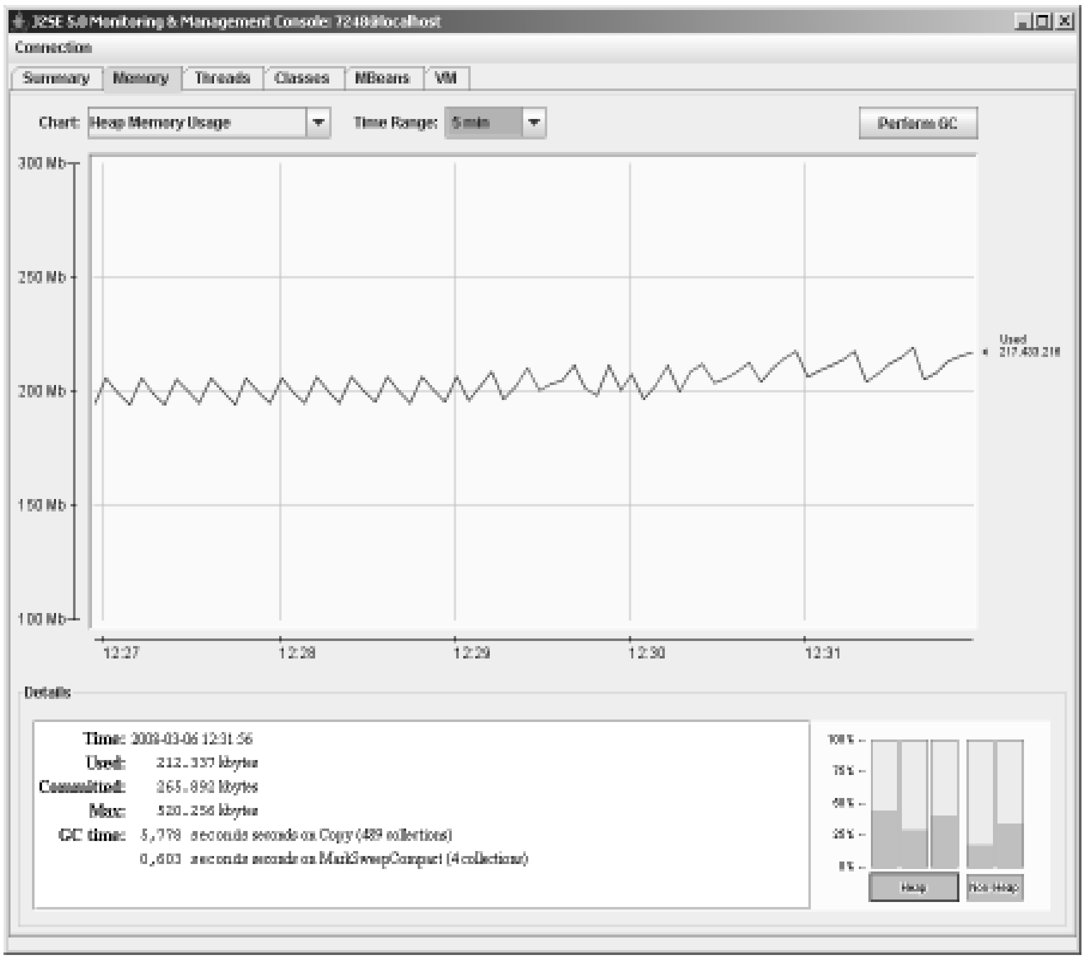


```
     Time: 2008-03-06 10:58:28
     Used:      11.062 kbytes
Committed:      22.860 kbytes
      Max:     520.256 kbytes
  GC time:  2,409  seconds seconds on Copy (237 collections)
            0,090  seconds seconds on MarkSweepCompact (1 collections)
```

Figure 4-2 Tactical record service memory consumption

Refactored application software:

Figure 4-3 illustrates the memory consumption of the refactored application software. The average consumption is about 200 MByte. After extracting tactical record service the memory consumption of the application software decreases to 10MByte. Reduction of the memory consumption increases the system performance.



```
      Time: 2008-03-06 12:31:56
      Used:    212.337 kbytes
 Committed:    265.892 kbytes
       Max:    520.256 kbytes
   GC time:  5,778  seconds seconds on Copy (489 collections)
             0,603  seconds seconds on MarkSweepCompact (4 collections)
```

Figure 4-3 Refactored application software memory consumption

4.5 COMMUNICATION PERFORMANCE EXPERIMENTS

In the experiments that are given in detail in the following sections, all Intel based PC's consisting of single 2.60 GHz processor, 1.0 GB RAM, running Windows XP Service Pack 2 that are connected by 100 Mbps Fast Ethernet has been used. At the tables in the following sections, data represents the elapsed time between the calling time of the selectDisc message and returning time of the getRecords message which is illustrated in Figure 4-4.

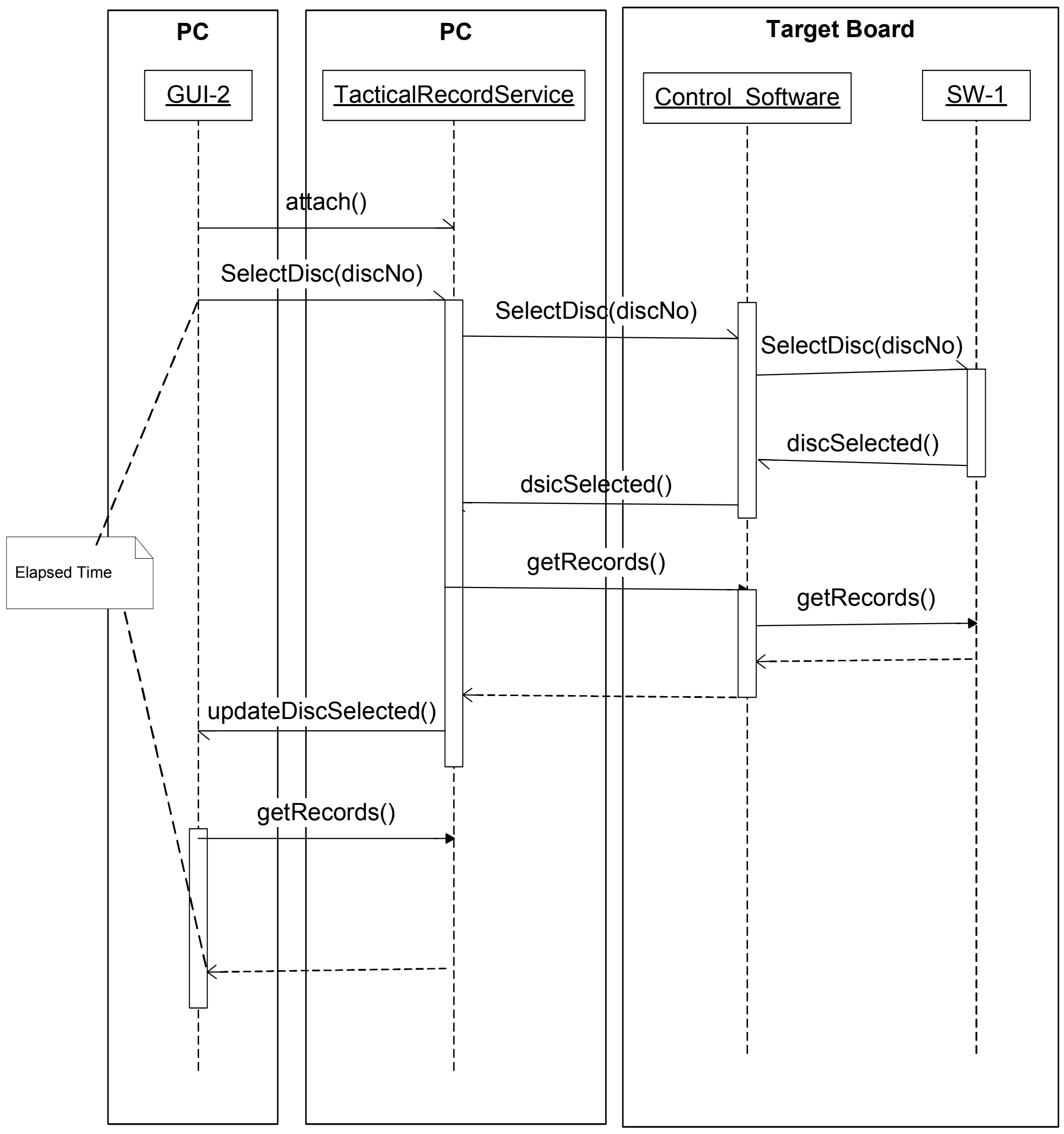


Figure 4-4 Elapsed Time Calculation Sequence Diagram

For the experiments, two different strategies have been followed: (a) the service provider and the service consumer codes have been executed on a single computer and (b) the execution has been distributed on different computers one of which is the service provider and the other one is the service consumer. In each experiment elapsed time data has been obtained using getCurrentTimeMillis() method of Java spec as follows:

```
...
long startTime = System.currentTimeMillis();
// code for disc selection and record request

long finishTime = System.currentTimeMillis();
long elapedTime = finishTime-startTime;
...
```

For each experiment elapsed time measurements are taken 5 repetitions for 1, 10,100 and 1000 records. Table 4-7 shows the amount of data transferred for different numbers of records.

Table 4-7 Transferred data in experiments

Number of Records	Total data transferred (byte)
1	9040
10	89368
100	892888
1000	8928092

4.5.1 Experiment 1

In this experiment, current application software and control software simulator deployed into the same computer and elapsed time has been evaluated for this case.

The testbed is illustrated in Figure 4-5. In Table 4-8 and Figure 4-6 the elapsed time results have been stated for 5 repetitions.

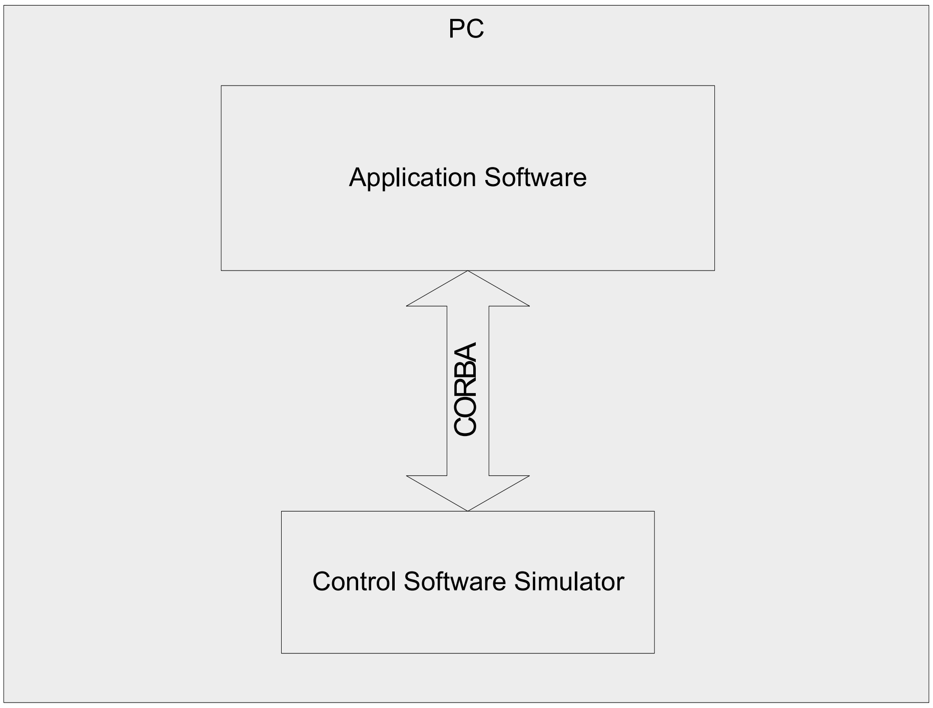


Figure 4-5 Experiment-1 Testbed Configuration

Table 4-8 Results for Current Architecture (Single Computer)

	Elapsed Time (Milisecond)				
1 Record	9	10	5	7	11
10 Record	16	16	13	15	16
100 Record	93	78	94	98	79
1000 Record	1328	1203	1187	1219	1437

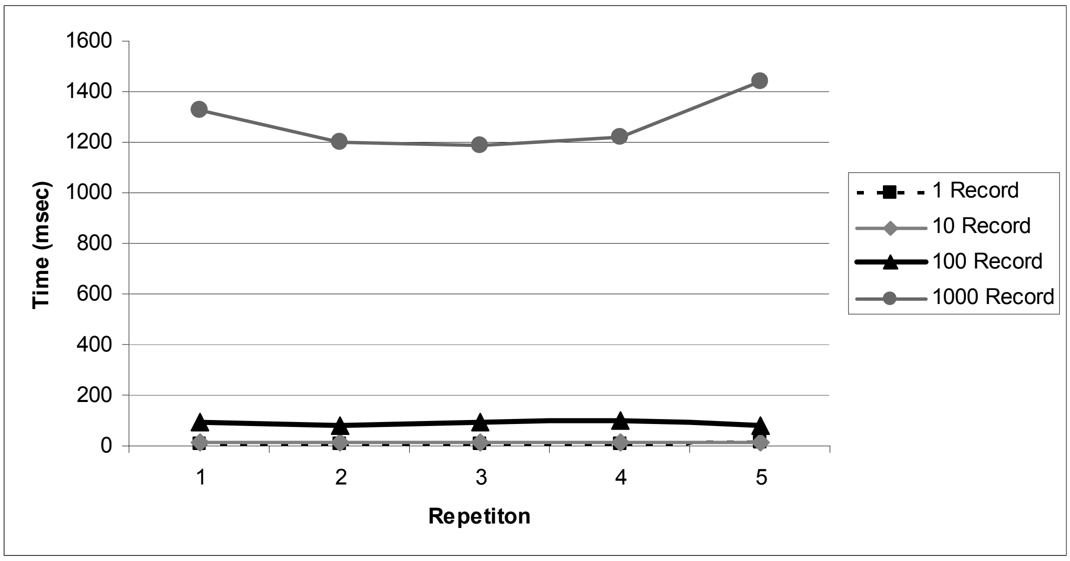


Figure 4-6 Results for Current Architecture (Single Computer)

4.5.2 Experiment 2

In this experiment, refactored application software, tactical record service and the control software simulator deployed into the same computer and elapsed time has been evaluated for this case. The testbed is illustrated in Figure 4-7. In Table 4-9 and Figure 4-8 the elapsed time results have been stated for 5 repetitions.

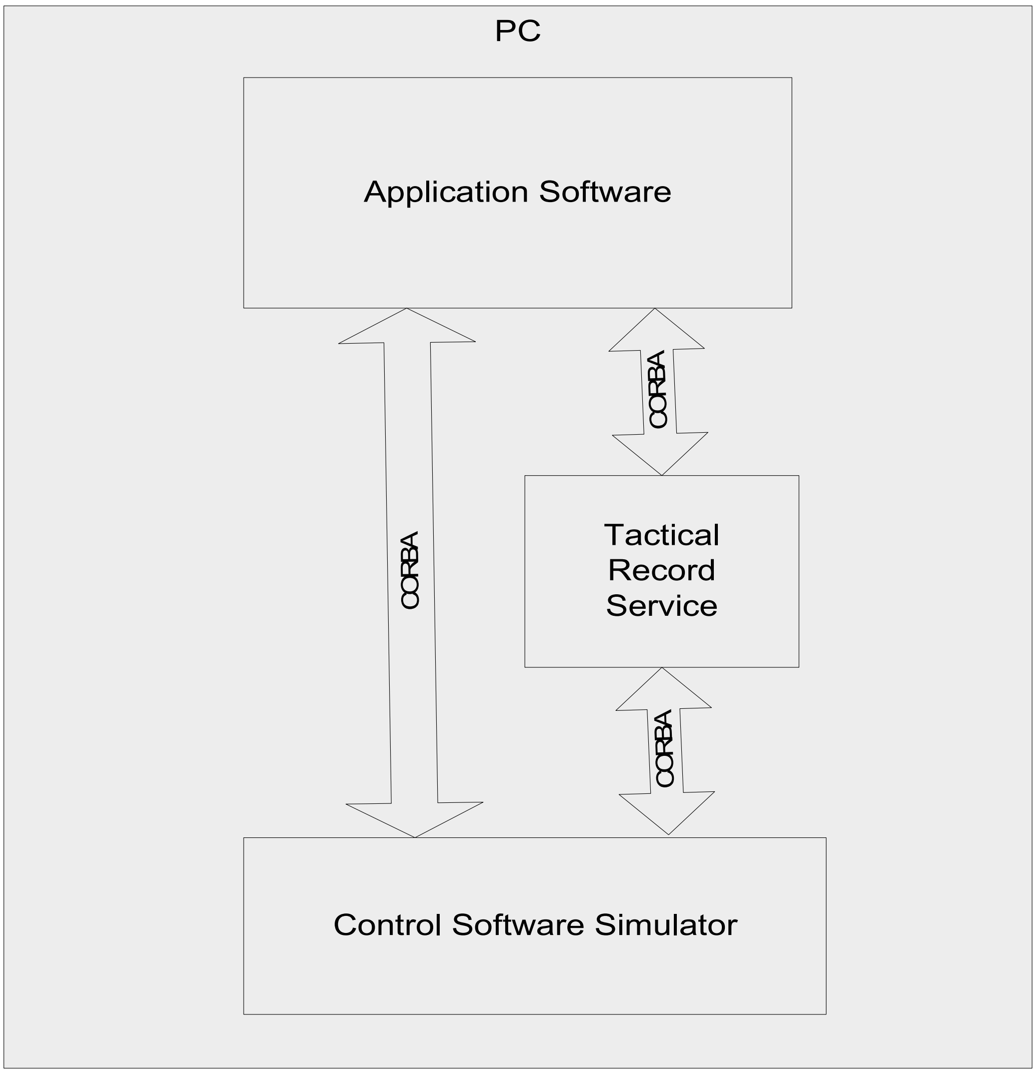


Figure 4-7 Experiment-2 Testbed Configuration

Table 4-9 Results for Refactored Architecture (Single Computer)

	Elapsed Time (Milisecond)				
1 Record	15	16	20	16	16
10 Record	31	40	32	31	31
100 Record	282	225	203	266	240
1000 Record	2063	2093	2062	1957	2102

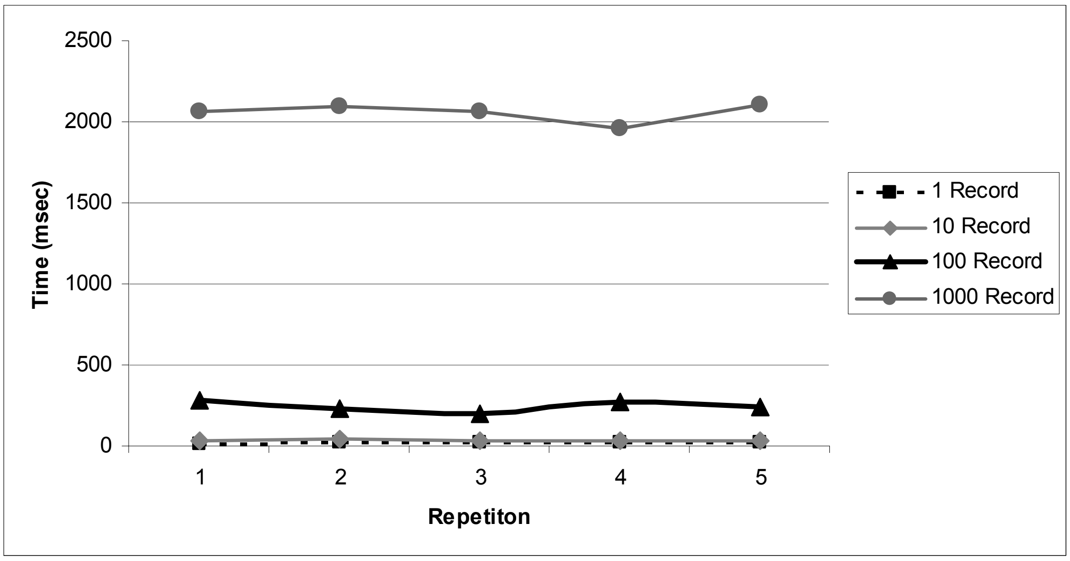


Figure 4-8 Results for Refactored Architecture (Single Computer)

4.5.3 Experiment 3

In this experiment, current application software and the control software simulator deployed into separate computers and elapsed time has been evaluated for this case.

The testbed is illustrated in Figure 4-9. In Table 4-10 and Figure 4-10 the elapsed time results have been stated for 5 repetitions.

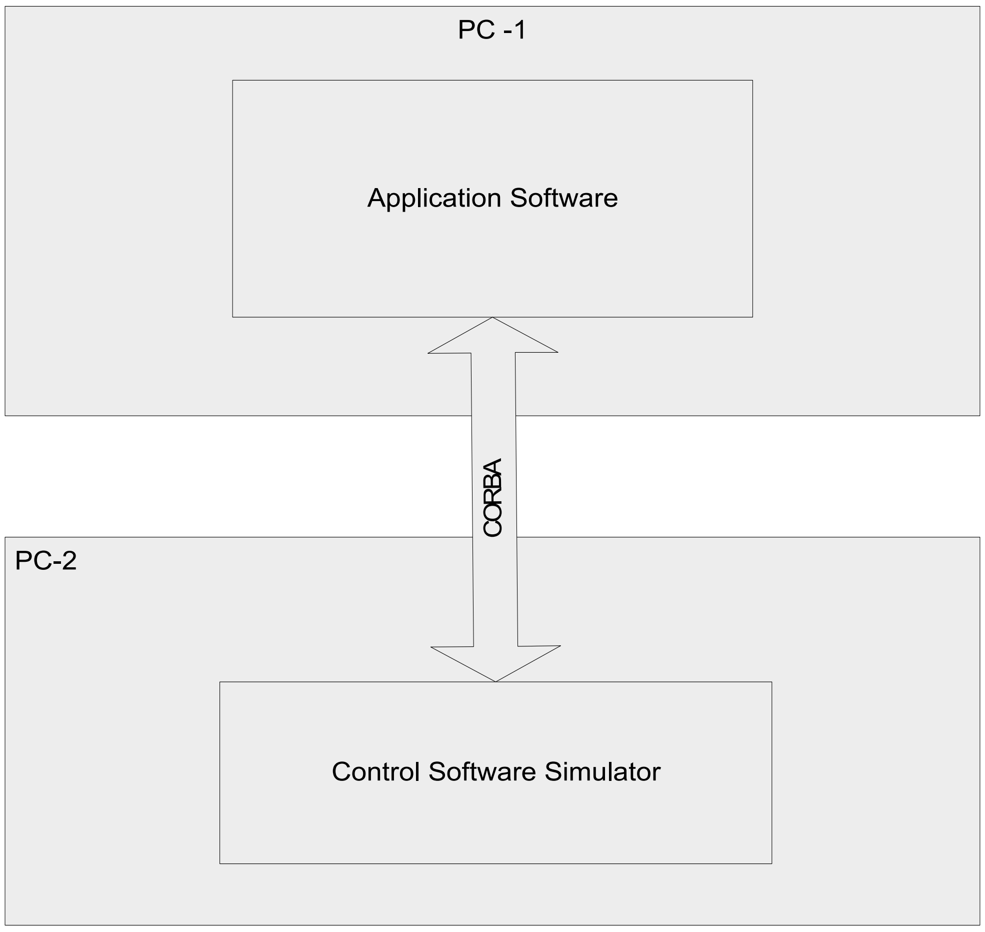


Figure 4-9 Experiment-3 Testbed Configuration

Table 4-10 Results for Current Architecture (Multiple Computer)

	Elapsed Time (Milisecond)				
1 Record	19	20	17	15	23
10 Record	52	49	58	60	64
100 Record	141	156	175	140	141
1000 Record	1704	1688	1750	1735	1715

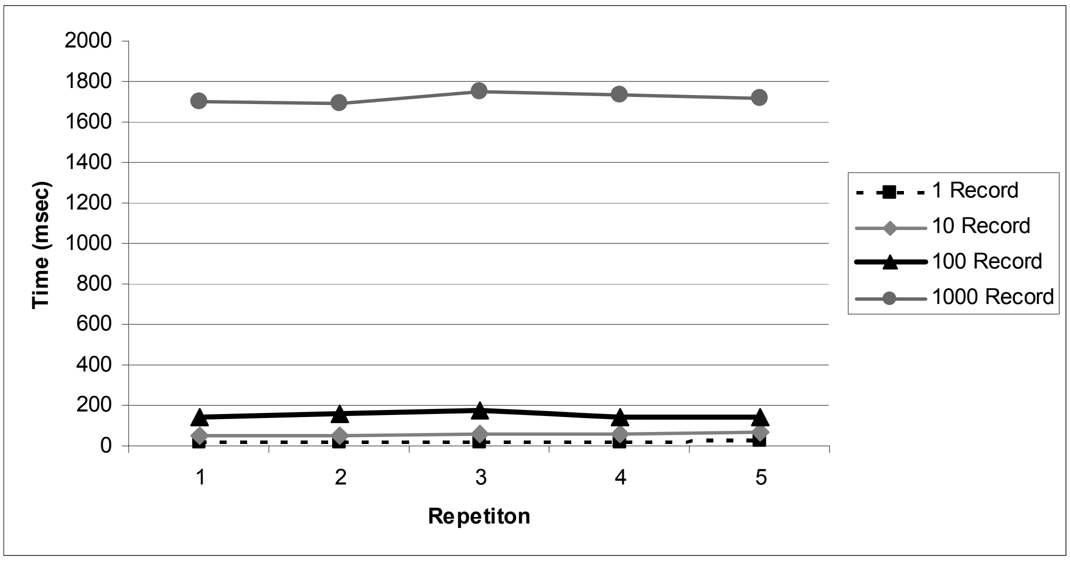


Figure 4-10 Results for Current Architecture (Multiple Computer)

4.5.4 Experiment 4

In this experiment, refactored application software deployed on PC-1 and tactical record service,the control software simulator deployed on PC-2 and elapsed time has been evaluated for this case. The testbed is illustrated in Figure 4-11. In Table 4-11 and Figure 4-12 the elapsed time results have been stated for 5 repetitions.

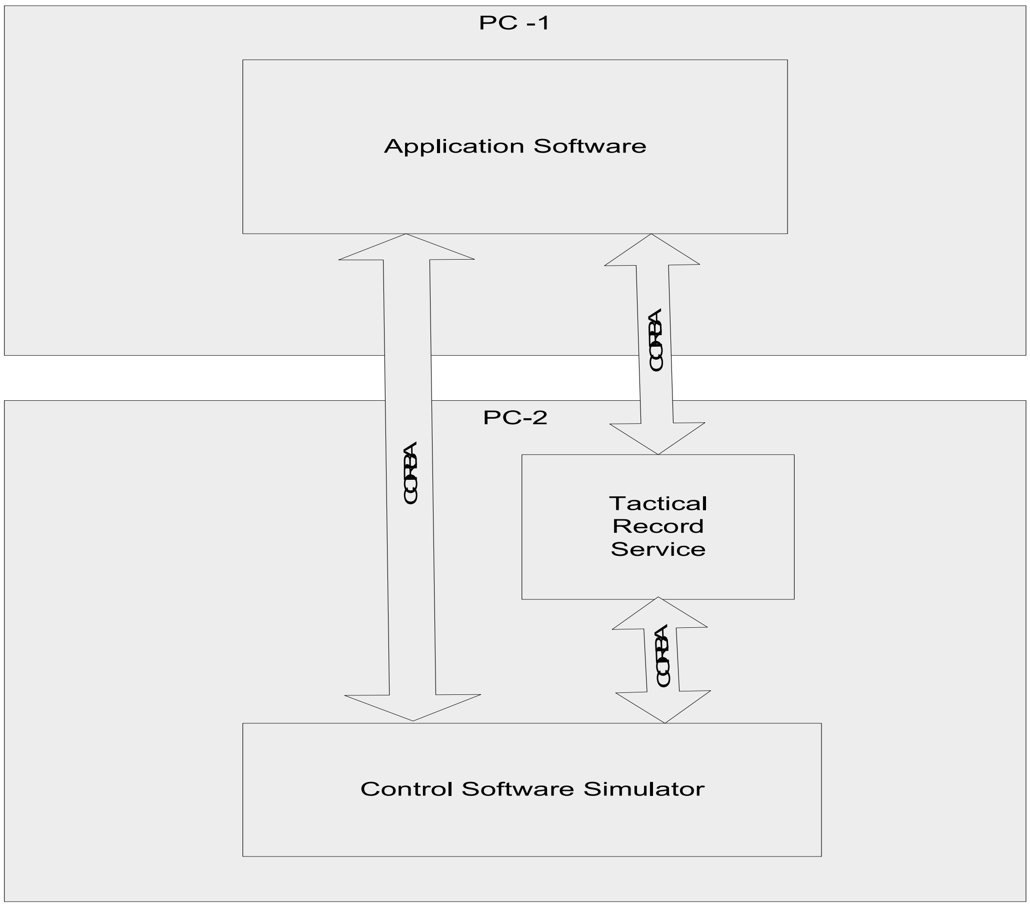


Figure 4-11 Experiment-4 Testbed Configuration

Table 4-11 Results for Refactored Architecture (Multiple Computer)

	Elapsed Time (Milisecond)				
1 Record	98	78	94	104	90
10 Record	134	141	158	157	144
100 Record	422	453	422	390	469
1000 Record	2468	2125	2281	2296	2171

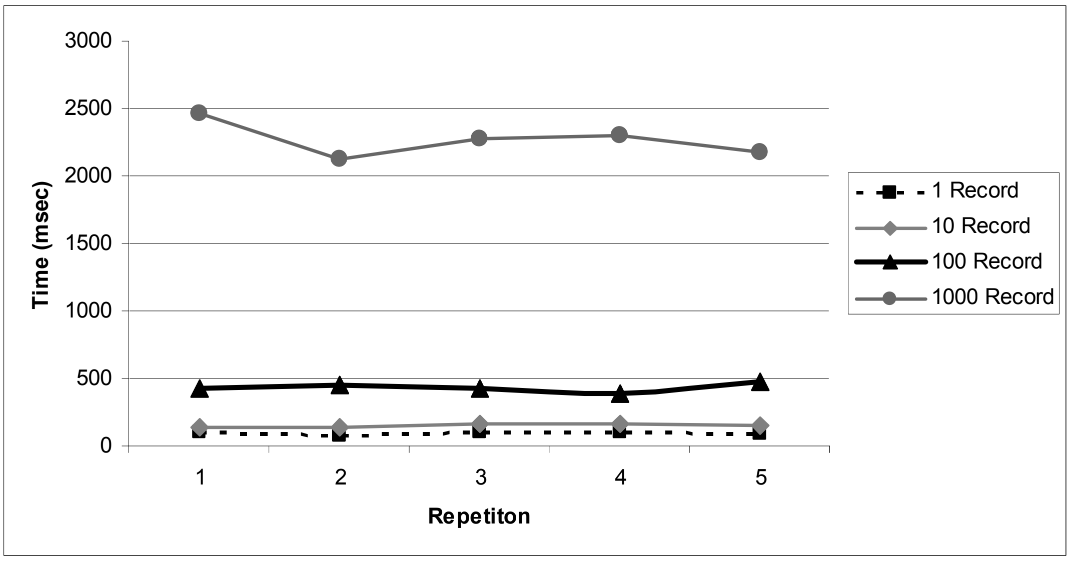


Figure 4-12 Results for Refactored Architecture (Multiple Computer)

4.5.5 Experimental Results

Experiments show that elapsed time results with refactored architecture are greater than the current architecture. This result is expected because in the current architecture data transferred to only one CORBA interface, but in the refactored architecture data transferred to two CORBA interfaces. This extra one CORBA interface causes the time consumption difference. The biggest time difference is measured in single computer case for 1000 records (9MByte total data transfer). This difference is about 800 milliseconds and this is acceptable for 9MByte data transfer.

Because of the network latency, multiple computer case timing elapsed time results are greater than single computer cases. But for one record transfer state the elapsed time difference between the multiple and single computer is negligible because transferred data for one record does not create big time differences.

CHAPTER 5

CONCLUSION

In this thesis, refactoring of an electronic warfare system based on service oriented architecture has been presented. The “Tactical Record” functionality is extracted from electronic warfare system application software and implemented as a service. Using this service, a series of performance and reusability evaluations have been performed.

Considering the thesis work, the following improvements can said to be achieved by refactoring the current architecture of the examined ELECTRO-WAR project according to the SOA principles.

When the tactical record feature of ELECTRO-WAR project is implemented as a service, porting this feature becomes easier. Moreover picking out this feature from the application software makes the programmer’s life easier in the modification of the software by replacement of existing services. Also it provides rapid and low cost system development by combination of implemented services.

As our basic focus is to implement a common functionality as a service in order to achieve reusability, the test results proved us to be successful. The procedure and argument names in the service interface is understandable and consistent, as stated in [38], that result can yield the conclusion that the service is reusable. As explained in the Evaluation section, the other criteria for measuring reusability is service usage metrics, like lines of code needed to be implemented and the work need to be done in a project to use the developed service. As our tests show, the implementation of SOA saved nearly 95% of the efforts for the same functionality to be implemented using traditional techniques.

Performance measurements show that memory consumption has been decreased in the refactored architecture of ELECTRO-WAR project and it affects the overall system performance positively. It is observed that an extra CORBA interface inserted in the refactored architecture affects the communication performance negatively. Experimental results show that these affects are negligible for 1, 10 records and much significant for 100, 1000 records transfer.

As a future study, the contribution of services can be analyzed according to other quality factors such as maintainability, testability.

REFERENCES

[1] J. M. Sullivan, "Revolution or evolution? The rise of the UAVs," in *Technology and Society, 2005. Weapons and Wires: Prevention and Safety in a Time of Fear. ISTAS 2005. Proceedings. 2005 International Symposium on*, 2005, pp. 94-101.

[2] T. Erl, *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*: Prentice Hall, 2005.

[3] K. Channabasavaiah and K. Holley, "Migrating to a service-oriented architecture," IBM, 2004.

[4] D. Krafzig, K. Banke, and D. Slama, *Enterprise SOA: Service-Oriented Architecture Best Practices*: Prentice Hall, 2004.

[5] OASIS, "Reference Model for Service Oriented Architecture 1.0," 2006.

[6] W3C, "Web Services Architecture," 2004.

[7] "Architecture classification for SOA-based applications," in *Object and Component-Oriented Real-Time Distributed Computing, 2006. ISORC 2006. Ninth IEEE International Symposium on*, 2006, p. 8 pp.

[8] L. KRISHNAMURTHY, "Comparative Assessment of Network-Centric Software Architectures," 2006.

[9] L. O'Brien Lero, P. Merson, and L. Bass, "Quality Attributes for Service-Oriented Architectures," in *Systems Development in SOA Environments, 2007. SDSOA '07: ICSE Workshops 2007. International Workshop on*, 2007, pp. 3-3.

[10] L. Brownsword and D. Carney, "Current Perspectives on Interoperability," 2004.

[11] "IEEE standard computer dictionary. A compilation of IEEE standard computer glossaries," *IEEE Std 610,* 1991.

[12] D. Box and L. Felipe, "Web Services Eventing," 2004.

[13] OASIS, "Web Services Brokered Notification 1.2," S. S. Dave Chappell, Ed., 2004.

[14] OASIS, "Web Services Base Notification 1.3," 2006.

[15] H. Yi and G. Dennis, "A Flexible and Efficient Approach to Reconcile Different Web Services-based Event Notification Specifications," in *Web Services, 2006. ICWS '06. International Conference on*, 2006, pp. 735-742.

[16] OMG, "Common Object Request Broker Architecture (CORBA/IIOP)," 2004.

[17] I. Object Computing, "The ACE ORB (TAO)," 2007.

[18] JacORB, http://www.jacorb.org/.

[19] OMG, "Common Object Request Broker Architecture: Core Specification," 2004.

[20] R. E. Gruber, B. Krishnamurthy, and E. Panagos, "CORBA Notification Service: design challenges and scalable solutions," in *Data Engineering, 2001. Proceedings. 17th International Conference on*, 2001, pp. 13-20.

[21] OMG, "CORBA Event Service Specification," 2004.

[22] OMG, "CORBA Notification Service Specification," 2004.

[23] I. Warren and J. Ransom, "Renaissance: a method to support software system evolution," in *Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings. 26th Annual International*, 2002, pp. 415-420.

[24] G. Kotonya and J. Hutchinson, "Viewpoints for Specifying Component-Based Systems," 2004.

[25] W. River, "VxWorks," 2008.

[26] Sesa, "MCOS," 2005, http://www.sesa.es/en/mcos.htm.

[27] G. Walters, *The Essential Guide to Computing*: Prentice Hall, 2001.

[28] S. Agrawal and P. Bhatt, "Real-time Embedded Software Systems," 2001.

[29] N. A. B. Gray, "Comparison of Web Services, Java-RMI, and CORBA service implementations."

[30] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*: Addison-Wesley, 1994.

[31] M. A. S. Boxall and S. Araban, "Interface metrics for reusability analysis of components," in *Software Engineering Conference, 2004. Proceedings. 2004 Australian*, 2004, pp. 40-51.

[32] R. Thisted, "What is a P-value?," 1998.

[33] K. SeongKi and H. Sang-Yong, "Performance comparison of DCOM, CORBA and Web service," 2006.

[34] S. Microsystems, "Using JConsole to Monitor Applications."

[35] I. Wong-Bushby, R. Egan, and C. Isaacson, "A Case Study in SOA and Re-architecture at Company ABC," in *System Sciences, 2006. HICSS '06. Proceedings of the 39th Annual Hawaii International Conference on*, 2006, pp. 179b-179b.