A SERVICE ORIENTED REFLECTIVE WIRELESS MIDDLEWARE

A SERVICE ORIENTED REFLECTIVE WIRELESS MIDDLEWARE

by

Bora Yurday

A thesis submitted to

the Graduate Institute of Sciences and Engineering

of

Fatih University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Engineering

June 2006
Istanbul, Turkey

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

Prof. Dr. Bekir Karlık
Head of Department

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

Assoc. Prof. Dr. Haluk Gümüşkaya
Supervisor

Examining Committee Members

Prof. Dr. Kemal Fidanboylu	
Assoc. Prof. Dr. Haluk Gümüşkaya	
Assist. Prof. Dr. Tuğrul Yanık	

It is approved that this thesis has been written in compliance with the formatting rules laid down by the Graduate Institute of Sciences and Engineering.

Assist. Prof. Dr. Nurullah Arslan
Director

Date
June 2006

SERVICE ORIENTED REFLECTIVE WIRELESS MIDDLEWARE

Bora Yurday

M. S. Thesis - Computer Engineering
June 2006

Supervisor: Assoc. Prof. Dr. Haluk Gümüşkaya

ABSTRACT

The role of middleware has become increasingly important in mobile computing, where the integration of different applications and services from different wired and wireless businesses and service providers exist. The requirements and functionalities of the wireless middleware can be achieved by Service Oriented Computing which can be an ideal paradigm for mobile services. Reflective middleware responses are optimized to changing environments and requirements. In this thesis a Service Oriented Reflective Wireless Middleware (SORWIM) is proposed. It provides a set of services for efficient and reliable information discovery and dissemination in ad hoc mobile environments. One of the primary goals of this research is to investigate how the construction of mobile services can benefit from the Service-Oriented paradigm.

Keywords: Wireless Middleware, Service Oriented Architecture, Service Orchestration.

.

SERVİS ODAKLI AKSETTİRİCİ KABLOSUZ ARAKATMAN

Bora Yurday

Yüksek Lisan Tezi – Bilgisayar Mühendisliği
Haziran 2006

Tez Yöneticisi: Doc. Dr. Haluk Gümüşkaya

ÖZ

Kablolu ve kablosuz servis sağlayıcıların ürettiği program ve servisleri entegre eden mobil programlamadaki arakatmanın (middleware) rolü artan bir öneme sahiptir. Kablosuz ara katmanın ihtiyaç ve fonksiyonları mobil servisler için ideal bir yaklaşım olan servis odaklı mimariyle sağlanabilir. Değişen çevre ve ihtiyaçlar aksettirici (Reflective) arakatman tarafından optimize edilir. Bu çalışmada servis odaklı aksettirici kablosuz arakatman (SORWIM) sunulmaktadır. mobil ortamlardaki bilgiyi keşfetmek ve erişmek için etkin ve güvenilir servisler sağlar. Bu çalışmanın öncelikli amaçlarından biri servis odaklı yaklaşımdan yararlanarak mobil servislerin nasıl oluşturulacağının araştırılmasıdır.

Anahtar Kelimeler: Ağsız Arakatman, Servis Odaklı Mimari, Servis Orkestrasyonu

ACKNOWLEDGEMENT

I would like to express my gratitude to my supervisor Assoc. Prof. Dr Haluk Gümüşkaya for his immense help in planning and executing the works and insight throughout the research.

I am very grateful to my friend Ömer Dogan and research assistant Engin Tozal for their valuable suggestions and comments.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Middleware is distributed software that sits above the operating system and below the application layer and abstracts the heterogeneity of the underlying environment (Schantz and Schmidt, 2001). It provides integration and interoperability of applications and services running on heterogeneous computing and communications devices, and simplifies distributed programming. Middleware can be decomposed into multiple layers such as host infrastructure middleware (JVM, Adaptive Communication Environment (ACE)), distribution middleware (RMI, CORBA, DCOM, SOAP), common middleware services (J2EE, .NET), and domain-specific middleware services (developed for particular domains, such as telecom, e-commerce, health care, process automation, or aero-space) (Schmidt,2002). Conventional middleware technologies, such as CORBA and RMI have been designed and used successfully with fixed networks. These middleware platforms are not appropriate for mobile computing, because they are too big and inflexible for mobile devices. There are significant challenges to design and optimize middleware for mobile computing. It is therefore essential to devise new middleware solutions and capabilities to fulfill the requirements of emerging mobile technologies.

Service-Oriented Computing (SOC) (Papazoglou and Georgakopoulos, 2003) is a distributed computing paradigm based on the Service-Oriented Architecture (SOA) (Papazoglou and Heuvel, 2005), which is an architectural style for building software applications that use services. SOC and SOA are not completely new concepts; other distributed computing technologies like CORBA and RMI have been based around similar concepts.

SOA and SOC are merely extensions of the existing concepts and new technologies, like XML, and Web Services, are being used to realize platform independent distributed systems. The SOA appears to be an ideal paradigm for mobile services. However, it is currently focused only on enterprise and business services. In addition, most of SOA research has been focused on architectures and implementations for wired networks. There are many challenges that need to be addressed by wireless middleware based on SOA. Wireless middleware will play an essential role in managing and provisioning service-oriented applications. In this thesis a Service Oriented Reflective Wireless Middleware (SORWiM) is proposed. It provides a set of services for efficient and reliable information discovery and dissemination in ad hoc mobile environments. One of the primary goals of this research is to investigate how the construction of mobile services can benefit from the Service-Oriented paradigm by mapping of wireless middleware functionalities as stateless services on SOA.

This thesis is structured as follows. In Chapter 2, the background of the middleware and the problems in wireless middleware are introduced. Service Oriented Architecture and its approach to wireless middleware are also explained here. In Chapter 3, the architecture and the design issues of basic services and the implementation details of SORWiM are given, In Chapter 4 how composite services are orchestrated using these basic services are presented. Finally in Chapter 5, some of our findings are summarized.

CHAPTER 2

BACKGROUND AND RELATED WORK

2.1 MIDDLEWARE

It is almost certain that telecommunication networks will be based on the Internet Protocol in the near future. As a network layer protocol the IP alone is not sufficient—not even with suitable Internet transport protocols such as TCP, UDP and RTP. In the Internet community a larger number of activities is going on to specify new protocols and application programming interfaces in order to provide a complete networking solution. The construction of distributed systems is a difficult task for programmers, which can be simplified with the use of middleware. Middleware can be seen as a common development and runtime environment that enables the connection of components written in different languages and running on different operating systems (Nunn, 2005).

2.1.1 Requirements of the Middleware

The following requirements were proposed by (Emmerich, 2005): network communication, coordination, reliability, scalability and heterogeneity.

Network Communication: The network communication of different hosts requires the transformation of the complex data structures into a suitable format, which can be transmitted using transport protocols. Providing uniform, standard, high-level interfaces to the application developers and integrators, so that applications can be easily composed, reused, ported and made to interoperate.

Coordination: Coordination is required to control multiple communication points, which exist in distributed systems. Synchronization is useful during the communication of the concurrent components on the same host. There are several ways to achieve synchronization. If the component that asks some service from another component remains unblocked and can continue to perform its operations, then this way is called deferred synchronous (if service request was initiated by client) or asynchronous (if initiated by server).

Reliability: Distributed system implementations need to include error detection and correction mechanisms to liquidate the unreliability's caused by errors.

Scalability: Scalability defines how well hardware or a software system can adapt to increased demands. The main task is to provide the changes in distributed systems without changing its architecture or design. In order to achieve this task, it is desired that middleware respects different dimensions of transparency like Access, Location, Migration, Replication transparencies handled by the middleware.

Heterogeneity: The different components of distributed systems have to be resolved by the middleware.

2.1.2 Middleware Layers

According to Schmidt (Schantz and Schmidt, 2001), middleware can be decomposed into multiple layers as shown in Fig.2.1. In the following these layers will be explained.

Host Infrastructure Middleware: Host infrastructure middlewares such as JVM (Java Virtual Machine), Adaptive Communication Environment (ACE) encapsulate and enhance native operating system communication and concurrency mechanisms to create portable and reusable network programming components.

Distribution Middleware: Distribution middlewares like RMI, CORBA and SOAP define higher-level distributed programming models whose reusable APIs and mechanisms automate and extend the native operating system network programming capabilities encapsulated by host infrastructure middleware.

Common Middleware Services: Common middleware services such as Sun's J2EE, Microsoft's .NET define higher-level domain-independent components that allow application developers to concentrate on programming application logic, without having to write the code needed to develop distributed applications, instead using lower-level middleware features directly.

Domain-specific Middleware Services: Developed for particular domains, such as telecom, e-commerce, health care, process automation, or aerospace.

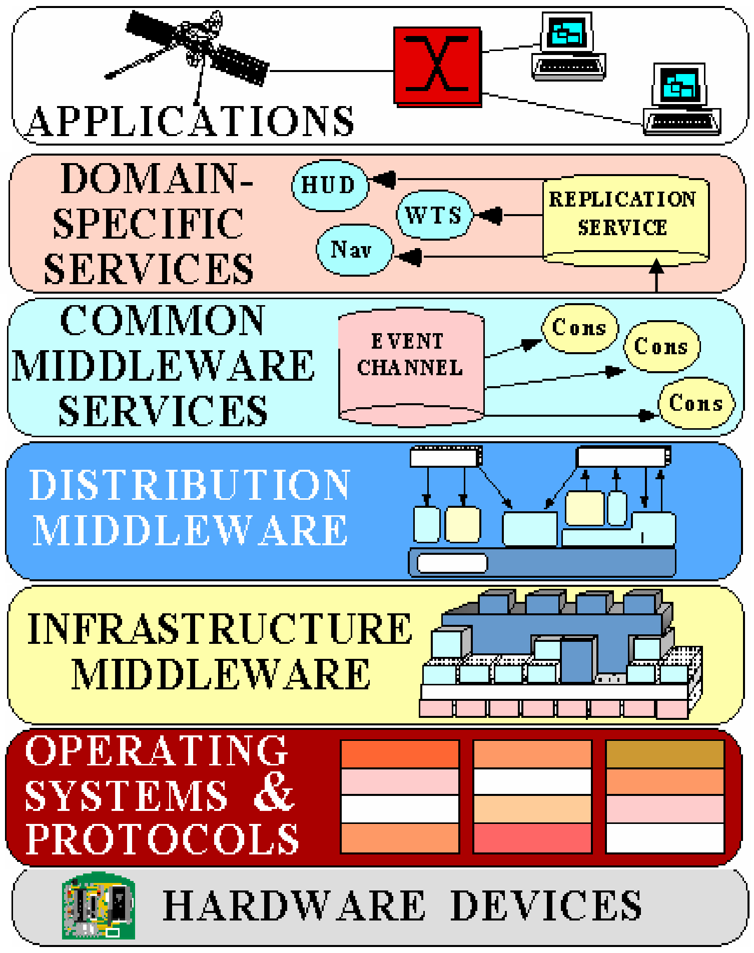


Fig. 2.1 Middleware layers.

2.2 WIRELESS MIDDLEWARE FOR MOBILE COMPUTING

Limited resources, heterogeneity, and a high degree of dynamism are the most common properties that usually exist in mobile devices such as pocket PCs, PDAs, sensors, phones, and appliances. Although limited resource availability varies from device to device; mobile devices generally don't have powerful CPUs, large amount of memory and high-speed I/O and networking compared to desktop PCs. Different hardware and software platforms, operating systems imply changes in some parameters such as byte ordering, byte length of standard types, and communication protocols. The degree of dynamism present in ubiquitous computing does not exist in traditional servers and workstations. A PDA, for example, interacts with many devices and services in different locations, which implies many changing parameters such as the type of communication network, protocols, and security policies. Therefore, because we can't predict all possible combinations, the software running on a PDA device must be able to adapt to different scenarios to cope with such dynamism.

All these properties affect the design of the wireless middleware infrastructure required for mobile computing. Conventional middleware platforms are not appropriate, because, first of all, they are too big and inflexible for mobile devices (Mascolo et al, 2002), (Vaughanl, 2004). There are some studies for improving the performance of conventional middleware platforms : CORBA (CORBA, 2005) and RMI (RMI, 2005) like RAPP (RAPP, 1999), ALICE, Embedded ORBs, Wireless RMI, Mobile RMI (Mobile RMI ,2005). These studies aim to improve poor performance and problems of conventional middleware platforms across wireless networks.

A wireless middleware should be lightweight as it must run on hand-held, resource-scarce devices. Conventional middleware platforms expect static connectivity, reliable channels, and high bandwidth that are limited in resource-varying wireless networks. Wireless middleware should support an asynchronous form of communication, as mobile devices connect to the network opportunistically and for short periods of time. It should be built with the principle of awareness in mind, to allow its applications to adapt its own and the middleware behavior to changes in the context of execution, so as to achieve the best quality of service and optimal use of resources. Hiding network topologies and other deployment details from distributed applications becomes both harder and undesirable since applications and middleware should adapt according to changes in location, connectivity, bandwidth, and battery power.

New wireless network middleware is required to increase performance of applications running across potentially mixed wireless networks (from 3G to WLANs), supporting multiple wireless devices, providing continuous wireless access to content and applications, as well as to overcome periods of disconnection and time-varying bandwidth delivery. Wireless middleware could also ensure end-to-end security and dependability from handheld devices to application servers.

We cannot customize existing middleware platforms manually for every particular device. It is not flexible or suitable for coping with dynamic changes in the execution environment. Reflective middleware presents a comprehensive solution to deal with ubiquitous computing (Roman et al, 2001). The concept of ubiquitous computing was introduced in the early 90's (Weiser, 1991). Mark Weiser from Xerox PARC expressed the goal as to achieve the most efficient kind of technology that is essentially invisible to the user, to make computing as ordinary as electricity. In the beginning the focus was on small special purpose devices, network protocols, interaction substrates, and new styles of applications. Reflective middleware system responses are optimized to changing environments or requirements, including mobile interconnections, power levels, CPU/network bandwidth, latency/jitter, and dependability needs. Reflection is the ability of a program to observe and possibly modify its structure and behavior. In the SORWiM architecture, reflection was used in the service level. A fully reflective middleware was not implemented initially; instead we worked on reflective services that take decisions according to context information.

Principles and guideless for designing middleware for mobile computing have been published in literature (Mascolo et al, 2002), (Vaughanl, 2004), (Roman et al, 2001), (Mascolo et al, 2004), and some wireless middleware projects have been developed for some specific areas, such as sensor networks (Heinzelman et al, 2004), (Yu et al, 2004). A few researchers have also published service oriented computing imperatives for wireless environments (Sen et al, 2005), (Thanh and Jørstad, 2005). To the best of our knowledge, there are no or a few real implemented wireless middleware platforms based on SOA.

2.3 Research Areas

In this section we identify the key research areas for the future software systems enabling seamless service provisioning in heterogeneous, dynamically varying computing and communication environments. These areas are not orthogonal; same or similar research items and issues appear in more than one research area. We have divided the research space into three key research areas based on (K Raatikainen, 2001). The areas are reconfigurable applications, environment monitoring and mobile distributed information base.

2.3.1 Reconfigurable Applications

Situations, in which a user moves with its end-device and uses information services, are challenging. Moreover, there is not static binding between middleware and the application; not even in the case of multi-mode access devices that can handle several access technologies including wireless LAN, short-range radio, and packet radio. It must be possible to move a service session (or one end-point of a service session) from one device to another.

In these situations the partitioning of applications and the placement of different co-operating parts is a research challenge. The support system of a nomadic user must distribute, in an appropriate way, the parts among the end-user system, network elements and application servers. In addition, when the execution environment changes in an essential and persistent way, it may be beneficial to redistribute the co-operating parts. The redistribution or relocation as such is technically quite straightforward but not trivial. On the contrary, the set of rules that the detection of essential and persistent changes is based on is a challenging research issue.

Another research issue of fundamental importance in distribution is fault-tolerance. Replication, which is a commonly used method to achieve fault-tolerance in traditional distributed systems, is not sufficient alone. The baseline applications must remain operational, at least in a tolerable manner, even if some services of the underlying execution environment cannot be utilized.

2.3.2 Environment Monitoring

Adaptability is one of the fundamental requirements in nomadic computing. The basic principle of adaptability is simple. When the circumstances change, then the behavior of an application changes according to the desires of user preferences. Environment monitoring is one of the fundamental enablers of adaptive applications. The three primary issues are discovery (which equipment are available), service location (which services are available), and available capabilities (computing power, various storage capabilities, available capacity on communication paths).

2.3.3 Mobile Distributed Information Base

File and information synchronization between different devices is already available but in quite primitive forms. A single information base for a user—possibly different views for her different roles—and for multiple user groups is a fundamental enabler for seamless reconfiguration of the end-user system for a mobile user and for seamless user roaming from one role to another one.

The mobile distributed information base should provide consistent, efficiently accessible, reliable and highly available information base. This implies a distributed and replicated world-wide "file system" that also supports intelligent synchronization of data after disconnections. Shared access and support of transactional operations also belong to the list of requirements.

2.4 Related Works

In this section we will review study on current research activities in various research areas of software systems. Below we briefly summarize our major findings in the Web.

Endeavour Expedition: The Endeavour Expedition at the University of California in Berkeley (Endeavour Expedicion, 2002) is a collection of projects that examines various aspects of ubiquitous computing. The goal is to enhance human understanding through the use of information technology, by making it dramatically more convenient for people to interact with information, devices, and other people.

Ninja: Enabling Internet-scale Services from Arbitrarily Small Devices that develops a software infrastructure to support scalable, fault-tolerant and highly-available Internet-based applications (S.D. Gribble et al., 2001).

Iceberg: An Internet-core Network Architecture for Integrated Communications that is seeking to meet the challenge for the converged network of diverse access technologies with an open and composable service architecture founded on Internet-based standards for flow routing and agent deployment (H.J. Wang et al., 2000).

OceanStore: An Architecture for Global-Scale Persistent Storage that is designed to span the globe and to provide continuous access to persistent information (J. Kubiatowicz et al., 2000).

Telegraph: An adaptive dataflow system that allows people and organizations to access, combine, analyze, and otherwise exploit data wherever it resides (J. M. Hellerstein et al., 2000).

Oxygen: In the MIT the corresponding project is called Oxygen (MIT Project Oxygen, 1999). The Oxygen project targets in the means of turning a dormant environment into an empowered one that allows the users to shift much of the burden of their tasks to the environment. The project is focusing on eight enabling technologies: new adaptive mobile devices, new embedded distributed computing devices, networking technology needed to support those devices, speech access technology, intelligent knowledge access technology, collaboration software, automation technology for everyday tasks, and adaptation methods. (M. Dertouzos, 1999).

FCE Group: The Future Computing Environments (FCE) Group at Georgia Tech has addressed the problems in building intelligent and interactive human-centric systems that support and augment our daily lives rely on the concepts of ubiquitous and aware computing. The group is attempting to break away from the traditional desktop interaction paradigm and move computational power into the environment that surrounds the user. The research challenge not only involves distributing the computation and networking capabilities, but also includes providing a natural interface to the user. It also aims at providing knowledge about the user and the environment that surrounds the user.The Future Computing Environments (FCE) Group at Georgia Tech is working to build interactive environments to augment daily activity. The research method is application-oriented, meaning that they identify the everyday activity to support before considering how to augment the environment. The mission is to identify, investigate, and invent technologies and environments that can be prototyped quickly and evaluated in real-life situations. (A.K. Dey, 2001).

Portolano: The project in the University of Washington at Seattle is called Portolano . The project has three main areas of interest: infrastructure, distributed services, and user interfaces. An essential research area is data-centric routing that facilitates automatic data migration among applications. Context aware computing, which attempts to coalesce knowledge of the user's task, emotions, location, and attention, has been identified as an important aspect of user interfaces. Task-oriented applications encounter infrastructure challenges including resource discovery, data-centric networking, distributed computing and intermittent connectivity (M. Esler at al., 1999).

2K and Gaia: In the University of Illinois at Urbana-Champaign the research project in this area is 2K: A Component-Based, Network-Centric Operating System for the next Millennium (2K). The 2K is an open source, distributed adaptable operating system. The project integrates results from research on adaptable, distributed software systems; mobile agents and agile networks to produce open systems software architecture for accommodating change. The architecture is realized in the 2K operating system. It manages and allocates distributed resources in order to support a user in a distributed environment. The basis for the architecture is a service model in which the distributed system customizes itself. The objective is to achieve a better fulfillment of user and application requirements. The architecture encompasses a framework for architectural-awareness so that the architectural features and behavior of a technology are reified and encapsulated within software. Adaptive system software, which is aware of the architectural and behavioral aspects of a technology, specializes the use of these technologies to support applications forming the basis for adaptable and dynamic QoS, security, optimization, and self-configuration (M. Roman et al, 2000).

MosquitoNet: The Mobile Computing Group at Stanford University (MosquitoNet) has developed the Mobile People Architecture (MPA) that addresses the challenge of finding people and communicating with them personally, as opposed to communicating merely with their possibly inaccessible machines. The main goal of the MPA is to put the person, not the device that the person uses, at the endpoints of communication session. The architecture introduces the concept of routing between people by using the Personal Proxy. The proxy has a dual role: as a Tracking Agent, the proxy maintains the list of devices or applications through which a person is currently accessible; as a Dispatcher, the proxy directs communications and uses Application Drivers to massage communication bits into a format that the recipient can see immediately. It does all this while protecting the location privacy of the recipient from the message sender and allowing the easy integration of new application protocols (P. Maniatis et al, 1999).

PIMA: The PIMA project (Pima Project) at the IBM T.J. Watson Research Center has developed a new application model for pervasive computing. The model is based on the following three principles. A device is a portal into a space of applications and data, not a repository of custom software managed by the user. An application is a means to perform a task, not a piece of software written to exploit capabilities of a device. The computing environment is the information-enhanced physical surrounding of a user, not a virtual space to store and run software. Based on those principles the following research challenges were identified : development of a programming model that identifies abstract interaction elements, specifying an abstract service description language, creating a task-based model for program structure, and creating a navigation model; building a development environment that supports the programming model above; developing specification languages for applications in terms of requirements, and for devices in terms of capabilities; developing mediating algorithms to negotiate a match between application requirements and device capabilities; run-time detection of changes in available resources and redistribution of computation; handling temporary disconnections; and enhancing current techniques of failure detection and recovery.

2.5 The Last Words on Related Work

The benefits of middleware software are obvious. The most significant advantage when compared to a pure IP-based socket programming approach is in the improved programming model. The middleware solutions are usually based on object-oriented programming and method invocations. The invocations are based on strongly typed interfaces that provide both compile and run time error checking. They also hide many implementation details. Therefore, middleware-based application development is much faster than the Internet based one. The fundamental problem of the current middleware specifications is that they only take advantage of a narrow subset of useful Internet protocols. The current middleware specifications were born in a time when the Internet protocols were a synonym of the TCP/IP transport. Later they have developed solutions of their own for Quality-of-Service, directory, discovery, and so on, independently from each other and from the IETF specifications. The fundamental research challenge is the question of how the developments in Internet protocols and in different middleware solutions could be harmonized. By harmonization we mean two things. Firstly, we need to solve the problem of incorporating evolving Internet solutions of Quality-of-Service, mobility, discovery, and security into the existing middleware specifications without breaking those specifications. Secondly, we need to find solutions to how different middleware solutions can become interoperable in the sense that components of an application can be executed on different middleware platforms. Section 2.6 (Service Oriented Architecture) outlines new framework for the middleware specification.

.

2.6 SERVICE ORIENTED ARCHITECTURE

Service-Oriented Computing (SOC) (Papazoglou and Georgeakopoulos, 2003) is a distributed computing paradigm based on the Service-Oriented Architecture (SOA) (Papazoglou and Heave, 2005), which is an architectural style for building software applications that use services. SOA and SOC are merely extensions of the existing concepts and new technologies, like XML, and Web Services, are being used to realize platform independent distributed systems. Service can be described as self-contained, stateless business function which accepts one or more requests and returns one or more responses through a well-defined, standard interface.

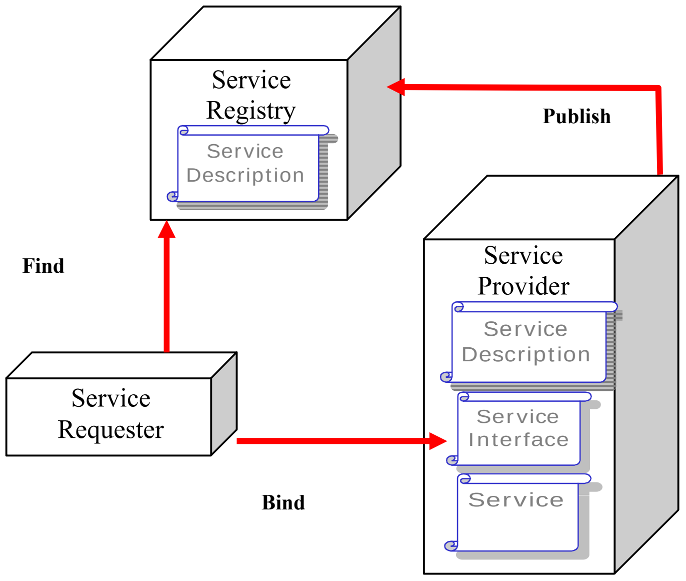


Fig. 2.2 Service oriented architecture.

As shown in Fig. 2.2 the service requester searches for the services in the service registry, the place where available services are listed and that allows providers to advertise their services and requesters to lookup and query for services. The service requesters bind with the service provider through the interfaces.

Design Philosophy of SOA

• SOA requires that functions, or services, are defined by a description language

• SOA-based service is self-contained (the service maintains its own state).

• SOA-based services are platform-independent

• SOA-based services can be dynamically located, invoked and (re-)combined.

Extending SOA

Service oriented architecture implements concepts such as service management, composition and Basic operations. as shown in Fig 2.3 (Papazoglou and Heuvel, 2005)

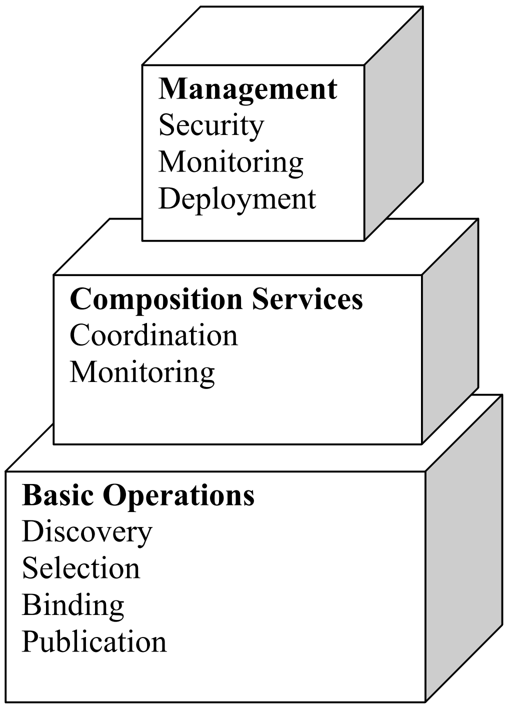


Fig. 2.3 Extending SOA.

Basic Operations: It defines an interaction between software agents as an exchange of messages between service requesters (clients) and service providers. These interactions involve the publishing, finding and binding of operations.

Composition Services: SOA creates a collection of services that can communicate with each other using service interfaces to pass messages from one service to another, coordinating and monitoring an activity between one or more services.

Management: Service management role is to manage operations in the service platform, like the deployment of services or monitors the correctness and overall functionality of orchestrated services.

2.7 WEB SERVICES

Web services are a programming technology for distributed systems whose goal is to ensure interoperability between software applications running on a variety of platforms and/or frameworks by utilizing the technologies of the World Wide Web.

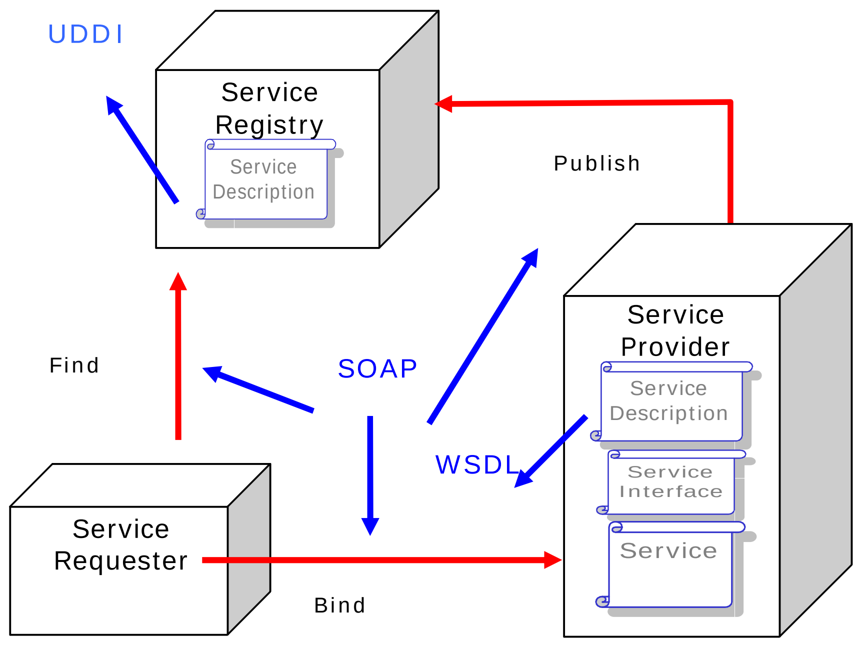


Fig. 2.4 Web service architecture

Web services as shown in Fig 2.4 promote a service-oriented programming style which is usually characterized in terms of publishing, finding and binding cycle.

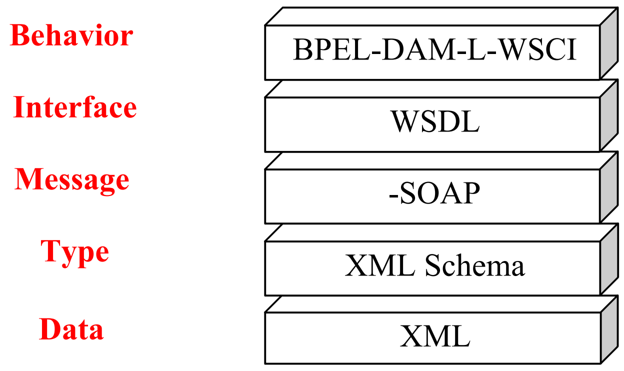


Fig. 2.5 Web service standards

Web service standards as shown in Fig. 2.5 consist of Simple Object Access Protocol (SOAP) which is an XML-based messaging protocol defining standard mechanism for remote procedure calls. The Web Service Description Language (WSDL) (Thanh and Jørstad, 2005) defines the interface and details service interactions. The Universal Description Discovery and Integration (UDDI) protocol supports publication and discovery facilities (Box et all, 2000). Finally, the Business Process Execution Language for Web Services (BPEL4WS) (Chinnici et al, 2002) is exploited to produce a service by composing other services.

2.4.1 Simple Object Access Protocol (SOAP)

SOAP is a lightweight, XML-based protocol for the exchange of information in a distributed environment. SOAP provides a standard packaging structure for transporting XML documents over a variety of standard Internet technologies, including SMTP, HTTP, and FTP. It also defines encoding and binding standards for encoding non-XML RPC invocations in XML for transport. SOAP provides a simple structure for doing RPC. A SOAP message consist of three parts, a SOAP envelope, an optional SOAP header and SOAP body as shown in Fig 2.6.

```
<soap:Envelope    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"    >
<soap:Header>            <!--  header  element(s)  here  -->       </soap:Header>
<soap:Body>               <!-- body element(s) here -->              </soap:Body>
</soap:Envelope>
```

Fig. 2.6 A SOAP schema.

SOAP Envelope: The envelope is the top element of the XML document.

SOAP Header: The header element is the optional element that can appear as a direct sub-element of the envelope element. The header element may contain elements used for transaction, authentication or other information.

Soap Body: The body contains the SOAP message request or response details. It also contains one way message details or fault details.

2.4.2 Web Service Description Language (WSDL)

A WSDL document defines services as collections of network endpoints, or ports. In WSDL, the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions: messages, which are abstract descriptions of the data being exchanged, and port types which are abstract collections of operations. The concrete protocol and data format specifications for a particular port type constitute a reusable binding.

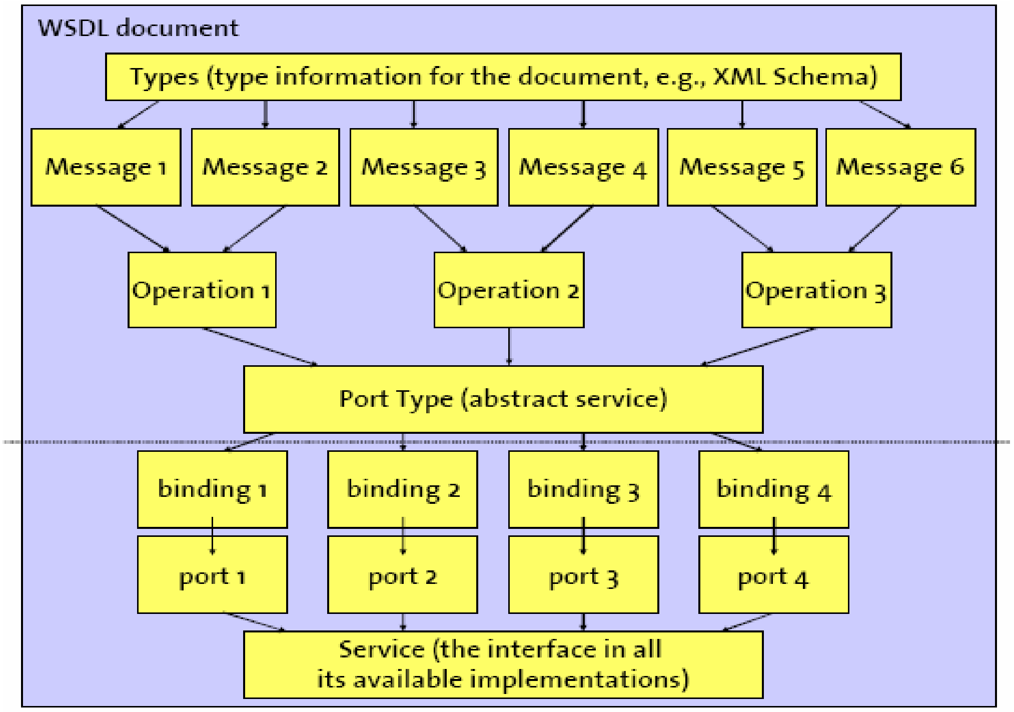


Fig. 2.7 WSDL Structure.

As shown in Fig. 2.7 WSDL document uses the following elements in the definition of network services

Types: a container for data type definitions using some type system such as XSD

Message: an abstract, typed definition of the data being communicated.

Operation: an abstract description of an action supported by the service.

Port Type: an abstract set of operations supported by one or more endpoints.

Binding: a concrete protocol and data format specification for a particular port type.

Port: a single endpoint defined as a combination of a binding and a network address.

```
<?xml version="1.0" encoding="UTF-8"?>
    <wsdl:part name="from" type="xsd:string"/>
  </wsdl:message>
  <wsdl:message name="sendMessageResponse">
    <wsdl:part name="sendMessageReturn" type="xsd:string"/>
  </wsdl:message>
  <wsdl:message name="getMessageResponse">
    <wsdl:part name="getMessageReturn" type="xsd:string"/>
  </wsdl:message>
  <wsdl:message name="sendMessageRequest">
    <wsdl:part name="to" type="xsd:string"/>
    <wsdl:part name="from" type="xsd:string"/>
    <wsdl:part name="subject" type="xsd:string"/>
    <wsdl:part name="content" type="xsd:string"/>
    <wsdl:part name="type" type="xsd:string"/>
  </wsdl:message>
  <wsdl:portType name="MessagingService">
    <wsdl:operation name="getMessage" parameterOrder="from">
      <wsdl:input message="impl:getMessageRequest" name="getMessageRequest"/>
      <wsdl:output message="impl:getMessageResponse" name="getMessageResponse"/>
    </wsdl:operation>
    <wsdl:operation name="sendMessage" parameterOrder="to from subject content type">
      <wsdl:input message="impl:sendMessageRequest" name="sendMessageRequest"/>
      <wsdl:output message="impl:sendMessageResponse" name="sendMessageResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="MessagingServiceSoapBinding" type="impl:MessagingService">
    <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="getMessage">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="getMessageRequest">
      </wsdl:input>
      <wsdl:output name="getMessageResponse">
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="sendMessage">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="sendMessageRequest">
      </wsdl:input>
      <wsdl:output name="sendMessageResponse"></wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="MessagingServiceService">
    <wsdl:port binding="impl:MessagingServiceSoapBinding" name="MessagingService">
      <wsdlsoap:address location="http://localhost:8083/ServiceModule/services/MessagingService"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Fig. 2.8 WSDL Example: Messaging Service In SORWiM

2.4.3 Universal Description, Discovery, and Integration (UDDI)

UDDI (Box et all, 2000) provides a worldwide registry of web services for advertisement, discovery, and integration purposes. The service provider can register three types of information in a UDDI registry. These three types of information are commonly referred to as "white pages", "yellow pages" and "green pages" information in the literature:

- White pages provide basic contact information about a company, such as the business name, address and contact information.

- Yellow pages describe a business service using different categorizations. (such as being in the manufacturing or software development business).

- Green pages provide technical information on the behaviors and supported functions of a business service hosted by a business. This may cover phone-based services such as call centers, E-mail based services such as technical support for a product, fax-based services such as a fax to E-mail service, etc.

CHAPTER 3

DESIGN AND IMPLEMANTATION OF BASIC SERVICES IN SORWiM

The main requirements of SORWiM are to overcome heterogeneity of mobile devices / platforms and to handle adaptation to mobile platforms with lightweight reusable service containers provided by Service Oriented Platform. Flexible discovery mechanism for the registered services of SORWiM is maintained by UDDI which provides a worldwide registry of web services for advertisement, discovery, and integration purposes.

In mobile systems the mobile nodes need to be aware of changes happening in the environment, such as user location, time of day, nearby people and devices, and user activity. Generally, this kind of information is called context. Context-aware services can be described as finding the relation between the context information collected by individual mobile nodes and adapting the way they behave according to the current context.(Papazoglou and Georgakopoulos, 2003). That is also known as reflection. Reflection is the ability of a program to observe and possibly modify its structure and behavior. In the SORWiM architecture, reflection was used in the service level. A fully reflective middleware was not implemented initially; instead we worked on reflective services that take decisions according to context information. In middleware the functionality of the services are separated by layers. In SORWiM by the help of SOC paradigm the functionalities can be separated by stateless services as shown in Fig 3.1. These services are, Event (Notification), Messaging, Location, and Redirection.

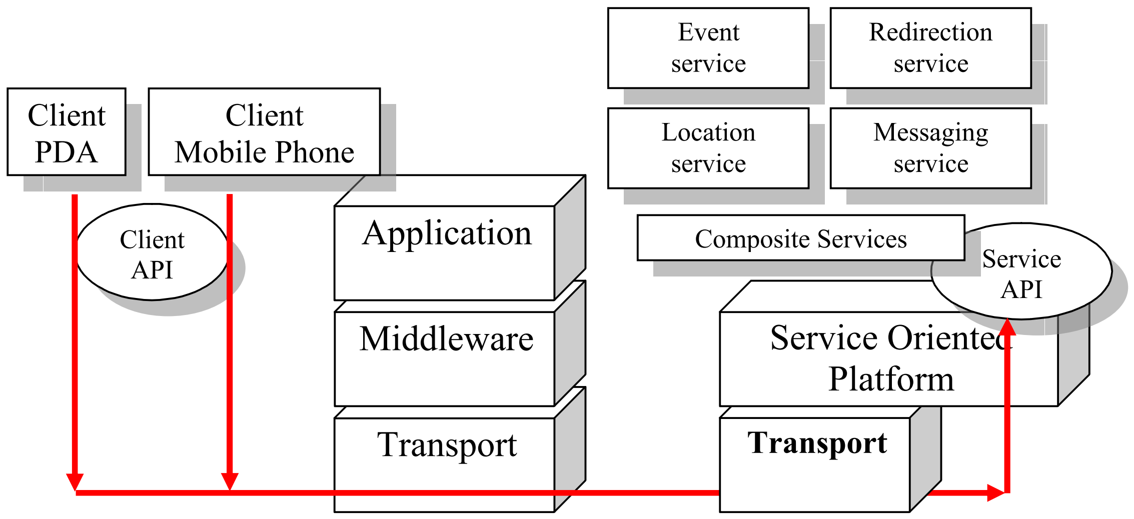


Fig. 3.1 High level system architecture.

Event Service provides the inference schema for profiles, notification (Alert) and Information queries.

Messaging Service has the role for creation of different types of messages such as RPC, SMTP, and SMS. In the project RPC based communication is implemented.

Location Service refers to the problem of updating and searching the current locations of multiple mobile nodes.

Redirection Service is responsible for client redirection and server reference translation at certain points.

These are some of the first important and basic services in a typical wireless environment. We have also developed some example composite services using these basic services in our implementation.

The mobile client application TestPad, which aims to test the basic services, was written using the PocketBuilder, the Sybase's the rapid application development tool for building mobile and wireless applications running on Microsoft Pocket PC devices (Gamma et al, 1995). In the middleware implementation we used the Apache Axis which is a proven service oriented platform to develop Java web services (Schmidt et al, 2000).

3.1 COMPONENT ARCHITECTURE

Component based architecture of SORWiM is divided into four logical layers, Sensor, Client, Sever and Storage layers as shown in Fig.3.2.

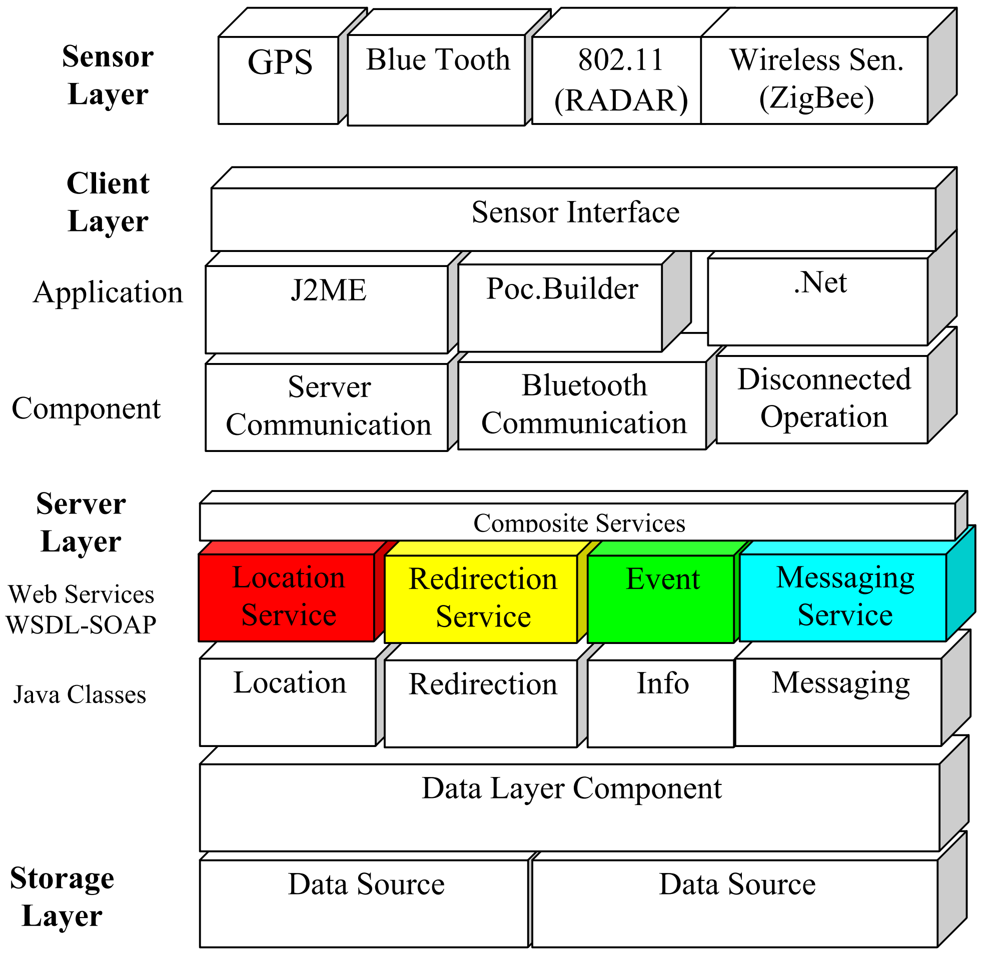
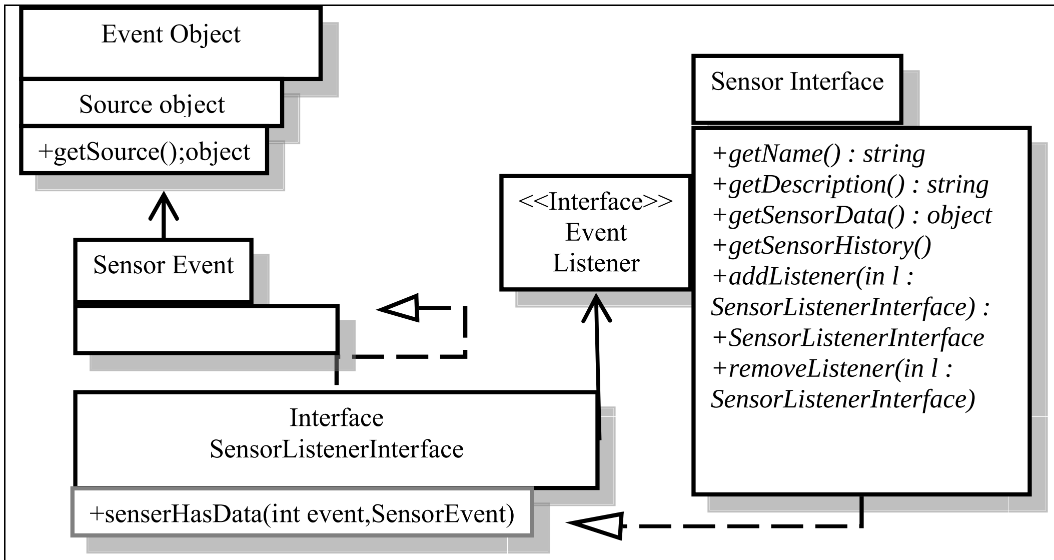


Fig. 3.2 Component architecture.

Sensor Layer: Sensors can be integrated with mobile devices (gps in a pda), or external (a Bluetooth enabled GPS unit), for determining the location indoor with the help of WLAN access points, or zigbee enabled wireless sensors. A sensor in a sensor layer can through the sensor interface make context information available to applications as shown in Fig. 3.3.


Event Object
Source object
+getSource();object
Sensor Event
<<Interface>> Event Listener
Sensor Interface
+getName() : string
+getDescription() : string
+getSensorData() : object
+getSensorHistory()
+addListener(in l : SensorListenerInterface) :
+SensorListenerInterface
+removeListener(in l : SensorListenerInterface)
Interface SensorListenerInterface
+senserHasData(int event,SensorEvent)

getName() returns the sensor name may be implemented in the class implementing the interface.

getDescription() returns the sensor description

getSensorData() If the sensor is able to produce sensor data on request this method should return a data object.

getData() methods may be implemented in the class implementing this interface.

getSensorHistory() If the sensor is able to produce a sensor history this method should return the history in a vector.

addListener(SensorListenerInterface) / removeListener(SensorListenerInterface) The sensor must implement a publish/subscribe system for SensorEvents. Subscription is handled through SensorListenerInterfaces.

Fig. 3.3 Sensor layer.

The Client Layer: The component based design is used in the Client Layer for using reusable components like disconnected operation and server communication. The heterogeneous applications communicate with SOAP messages through the service interfaces described by WSDL.

The Server Layer: The Server layer is divided into core components implementing functionality for services-application. The framework includes number of components implementing functionality for the location, redirection, information, messaging services. The applications are implemented as Java Web services and provide their interfaces with SOAP to the Terminal layer. One benefit of using Web services is that the service interfaces can be described in the WSDL and the service interface descriptions can be automatically generated from remote Java interfaces when services are deployed on the server.

The Storage Layer: The data layer supports a basic set of methods for storing, retrieving, querying, and deleting objects from the database.

3.2 IMPLEMANTATION ARCHITECTURE

The architecture of SORWiM is shown in Fig.3.3 where each service is depicted by a WSDL document, which describes service accessing rules. Web services are registered to the central UDDI database. The client searches the UDDI to find out the service it needs, fetches the WSDL file, and generates the stub with the WSDL to stub code generator provided by the web service toolkit, and starts calling remote methods.

Design patterns codify design expertise, thus providing time-proven solutions to commonly occurring software problems in certain contexts, (Andrews et al, 2003). We extensively used the GoF design patterns (W3C, 2002) in the implementation of the SORWiM architecture and its subsystems.

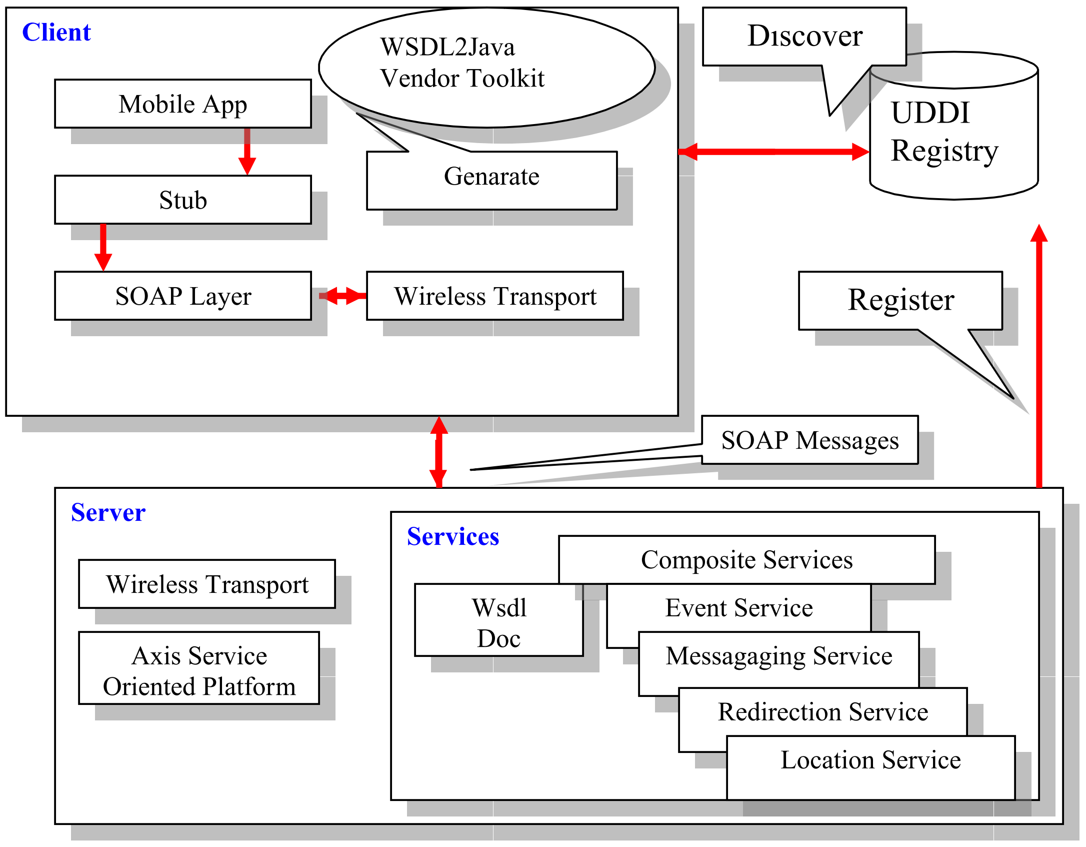


Fig. 3.3 Implementation architecture.

3.3 PACKAGES

The package diagrams of SORWiM basic services are shown in Fig.3.4 package describes a stateless service which can communicate with other services through service interfaces. That is there is no dependency between services which is one of the primary goals of SOA. By defining relations between services we create composite services that form new application scenarios in our wireless framework.

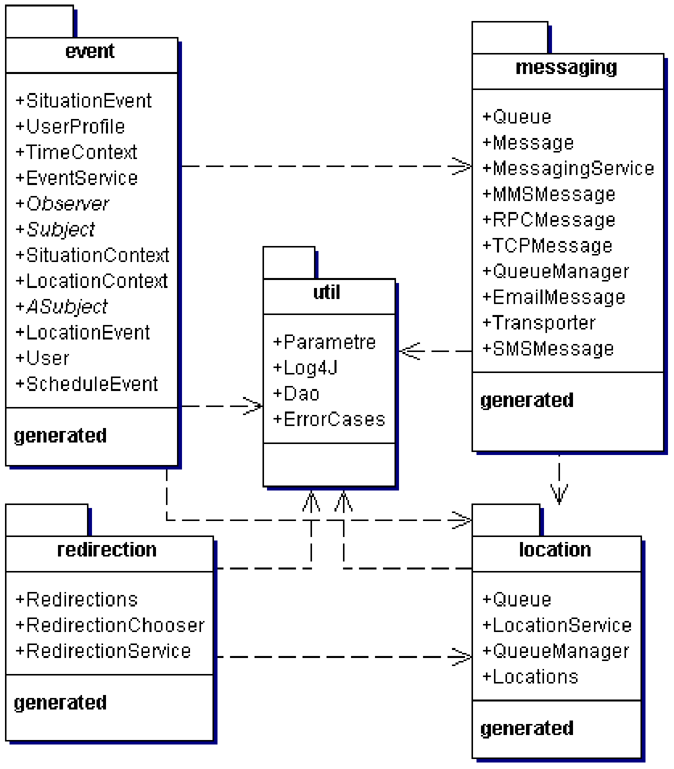


Fig. 3.4 The UML package diagrams of SORWiM basic mobile services.

We need an util package for database access, initializing parameters, logger, and error cases. And we just need a single instance of these classes to exist in the system. For ensuring each of these classes has one instance and provide a global point of access to it, singleton pattern will be used in the implementation.

3.3.1 Event Service

The EventService as shown in Fig 3.5 is one the first basic services of SORWIM. It provides an interface schema for four main functions, authentication, profiles and preferences, notification (alert), and system and user information the methods are shown in Fig 3.6.

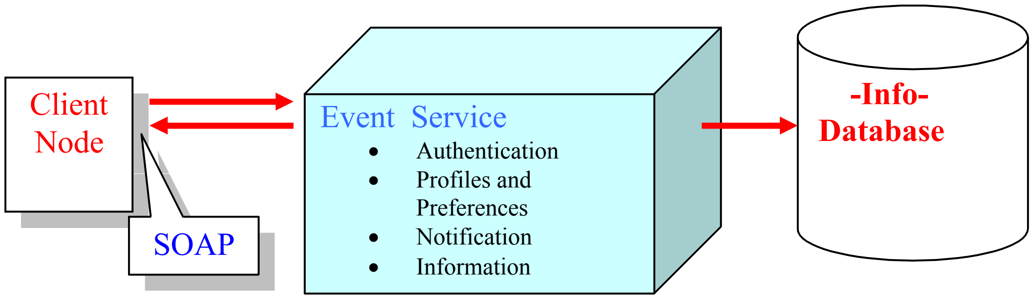


Fig. 3.5 Event service.

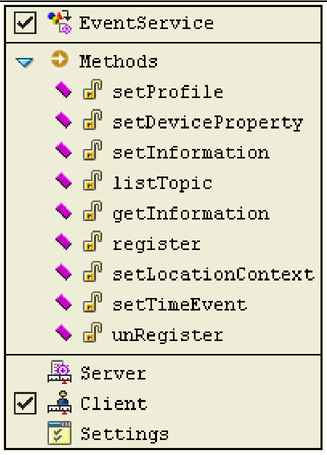
 EventService Methods setProfile setDeviceProperty setInformation listTopic getInformation register setLocationContext setTimeEvent unRegister Server Client Settings **MobileServices implemented in the Axis 1.2 Framework**	**Authentication**: *register / unRegister* mobile user logs in /out the system **Profile Preferences**: *setProfile* detailed user profile and preferences is set here. **Notification**: *setDeviceProperty* Device info such as battery power, memory, and communication bandwidth *setLocationContext*: Location info *setTimeEvent*: Time based-notification event **System and User Information**: *getInformation/ listTopic* get system and user information such as profiles, preferences/list a topic

Fig. 3.6 Event service implementation.

In the event service, context based publication is used for notifications. Publisher publish context event on time, location, situation based. Subscribers provide a filter based on the context event.

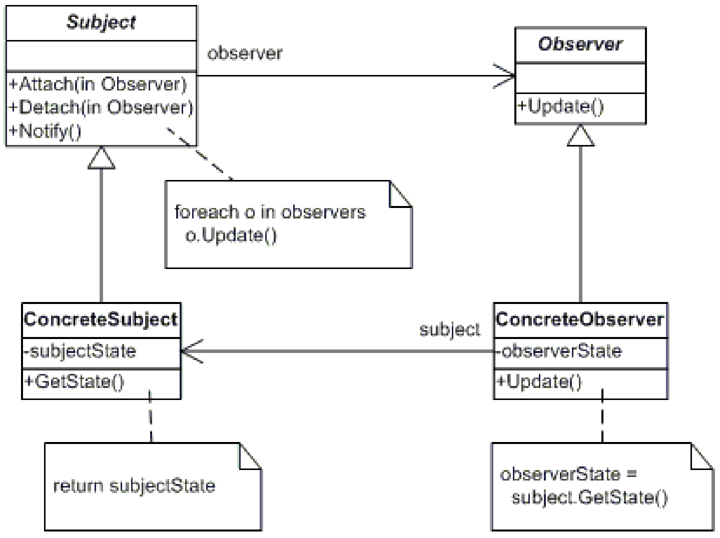


Fig. 3.7 Observer pattern.

A behavioral pattern, the observer pattern as shown in Fig. 3.7, can be used in the implementation of the notification engine. This pattern defines a one-to-many dependency between a subject object and any number of observer objects so that when the subject object changes state, all its observer objects are notified and updated automatically.

Observer Pattern based Event service as shown Fig. 3.8; the users (publishers) publish events with LocationContext, SituationContext and TimeContext. The Subscribers are notified by invoking notifyObserver. The users state is updated by the events set in events Classes (ScheduleEvent, SituationEvent, LocationEvent)

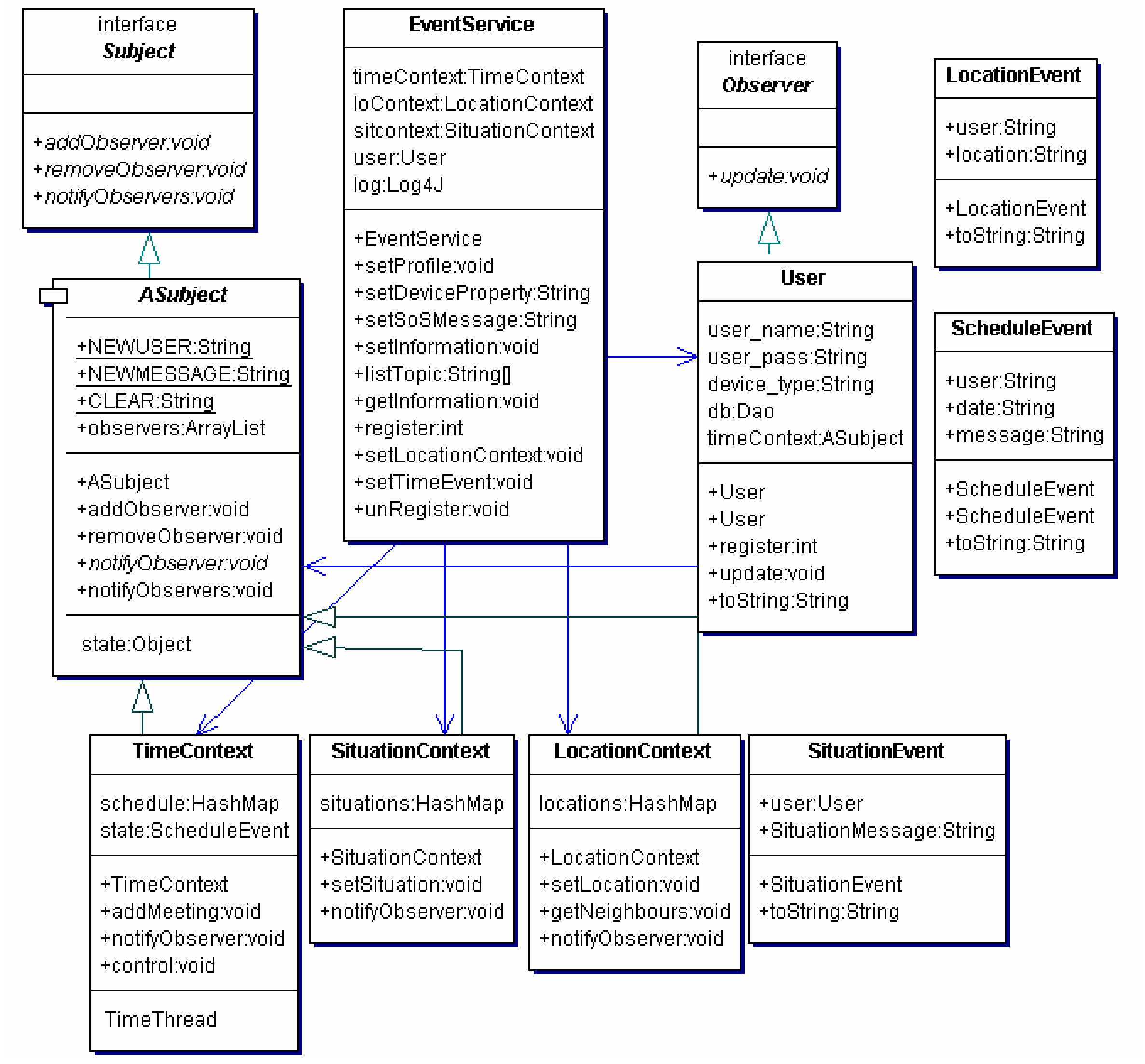


Fig. 3.8 Observer pattern based event service

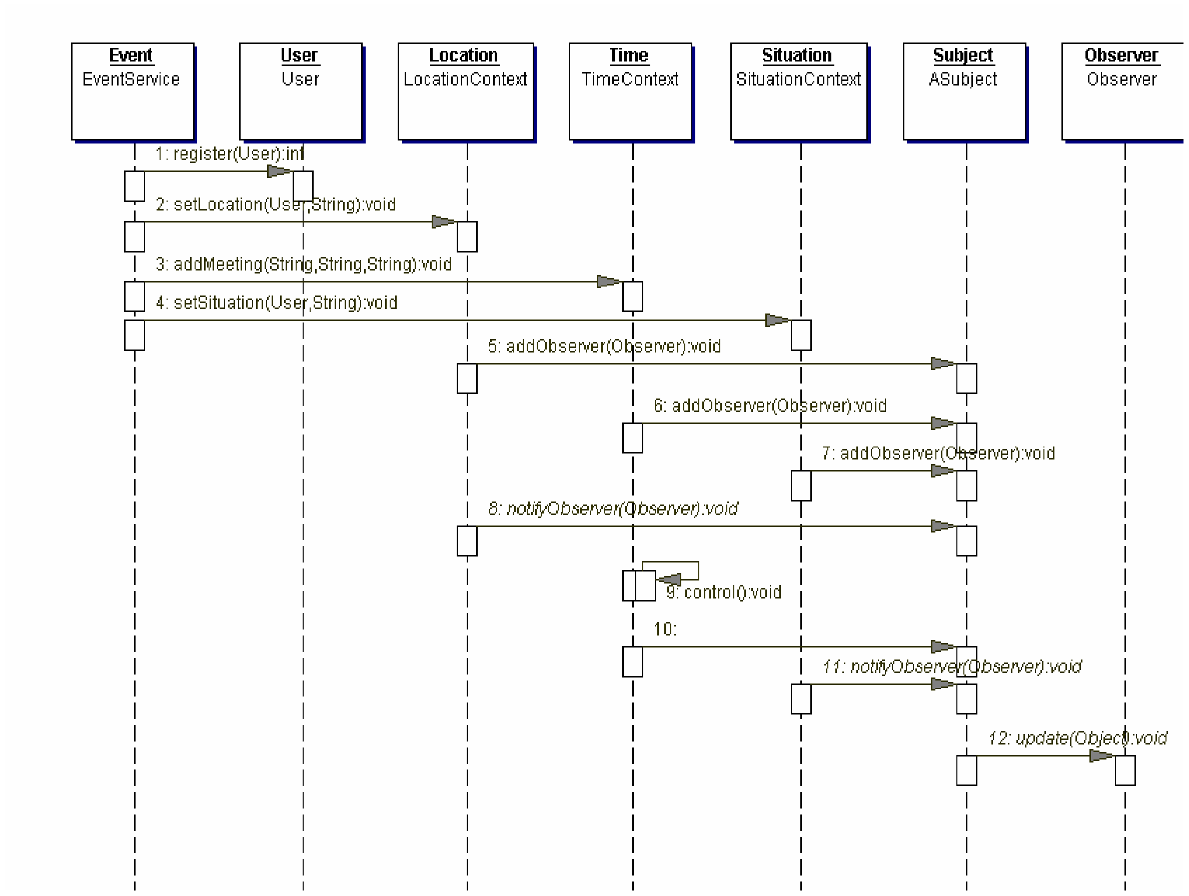

Event
EventService
User
User
Location
LocationContext
Time
TimeContext
Situation
SituationContext
Subject
ASubject
Observer
Observer
1: register(User):int
2: setLocation(User,String):void
3: addMeeting(String,String,String):void
4: setSituation(User,String):void
5: addObserver(Observer):void
6: addObserver(Observer):void
7: addObserver(Observer):void
8: notifyObserver(Observer):void
9: control():void
10:
11: notifyObserver(Observer):void
12: update(Object):void

Fig. 3.9 Flow chart event service.

As shown in Fig. 3.8 whenever user is registered, reference the User Object that implements from Observer Interface. Each user act as an observer. Then the user sets the context information by the help of the objects *LocationContext*, *SituationContext*, *TimeContext* which extends *ASubject*. Context objects have the rules of notification of time, location, and situation. *ASubject* collects the subject information and has the role of invoking *notifyObserver* method which updates the state of each observer.

3.3.2 Messaging Service

The Messaging Service package as shown in Fig.3.9 has basic messaging services to create, send, receive and read XML based messages for mobile applications running on different platforms. This service can send messages using RPC, SMTP, SMS types.

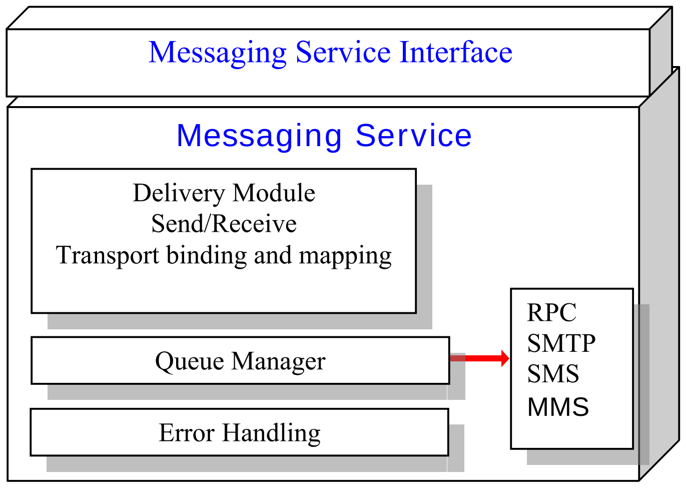


Fig. 3.10 Messaging service.

The class diagram of the MessagingService is shown in Fig.3.11. We used the Factory design pattern for creating message types. The Factory Pattern deals with the problem of creating objects without specifying the exact class of object that will be created. In addition to the factory pattern, by using the reflection API, the messaging service can decide at runtime which class (message type) is to instantiate.

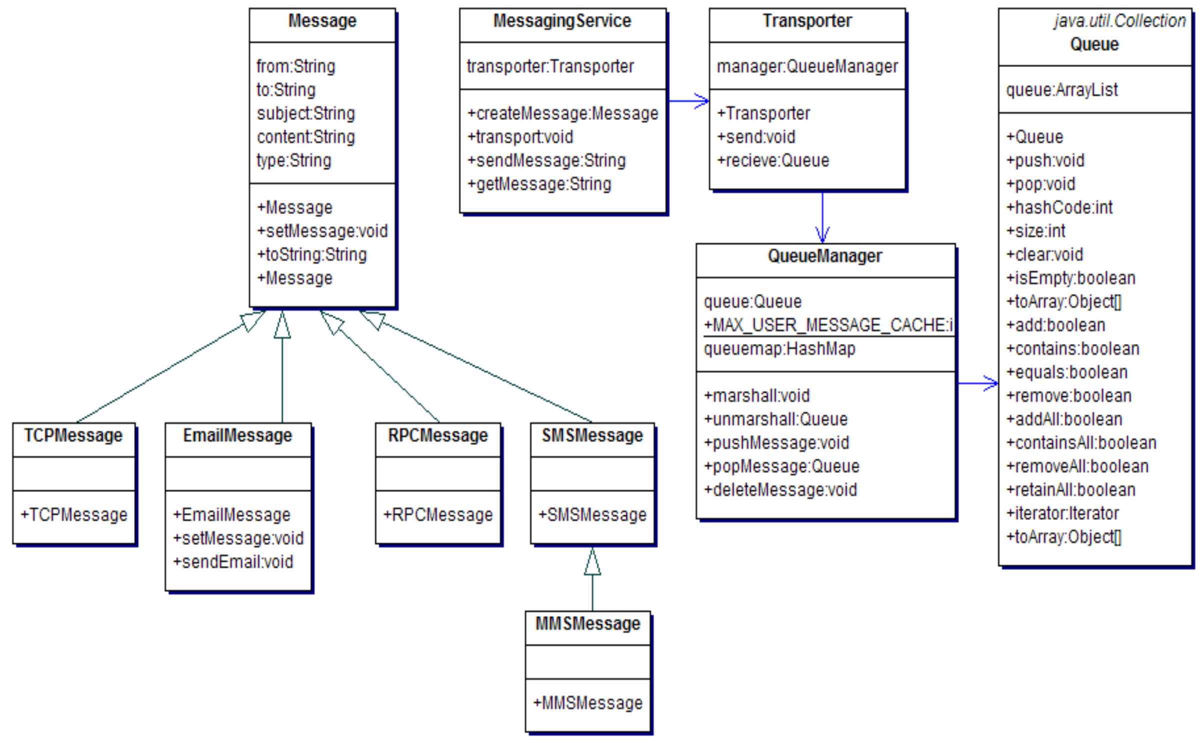
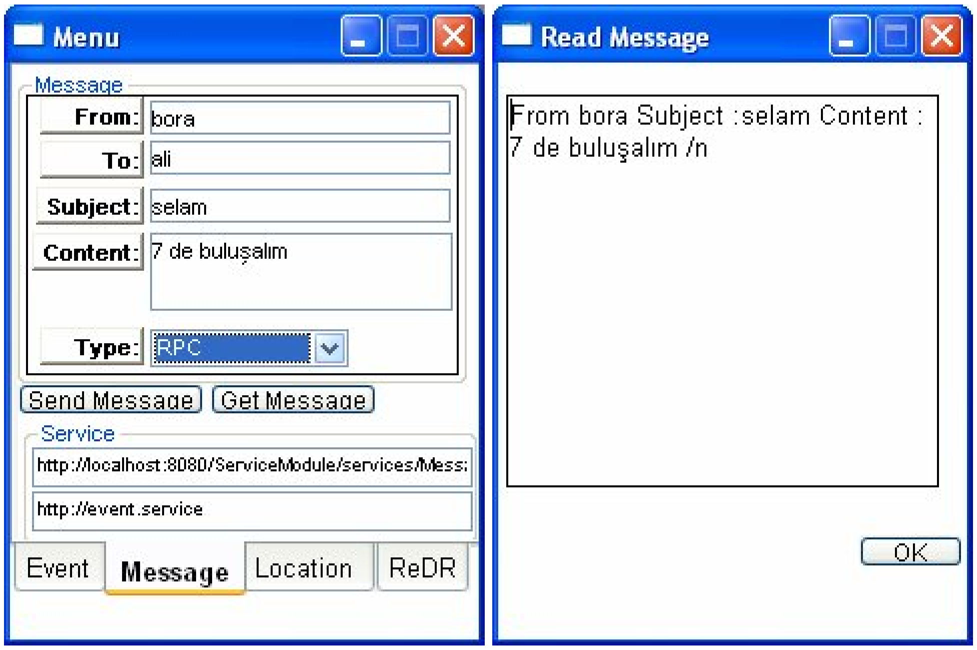


Fig. 3.11 The class diagram of the MessagingService.

The asynchronous communication is provided by the message oriented middleware approach of SORWiM using a messaging queue structure as shown in Fig. 3.11. The MessagingService class is the proxy class for the messaging service package. In order to test the services in SORWiM. TestPad is implemented with PocketBuilder. RPC based messaging is tested as shown in Fig.3.12



From: bora
To: ali
Subject: selam
Content: 7 de buluşalım
Type: RPC

http://localhost:8080/ServiceModule/services/Mess:
http://event.service

From bora Subject :selam Content :
7 de buluşalım /n

Fig. 3.12 TestPad messaging service.

3.3.3 Location Service

The LocationService handles the problem of updating and searching the current locations of mobile nodes in a wireless network. This service is also used by other services for giving decisions and providing active context-awareness that autonomously changes the application behavior according to the sensed location information.

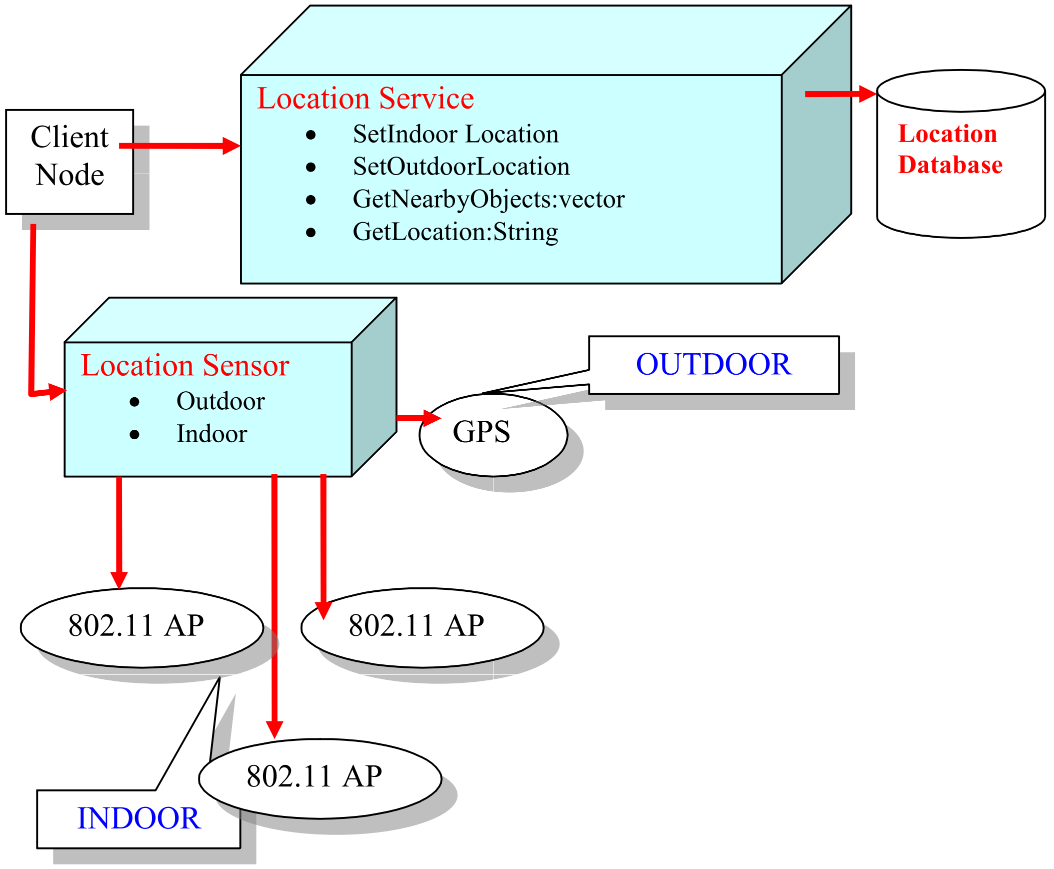
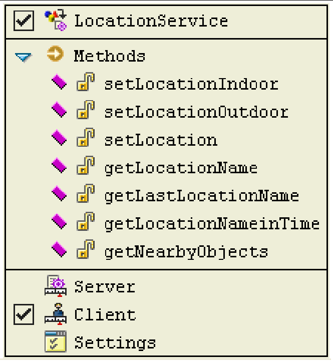


Fig. 3.13 Location service.

This study can be integrated with another study, WiPoD (Wireless Position Detector), which locates and tracks a user having an IEEE 802.11 supported device across the coverage area of a WLAN (PocketBuilder, 2006).. The location information for the SORWiM indoor applications will be provided by WiPoD.



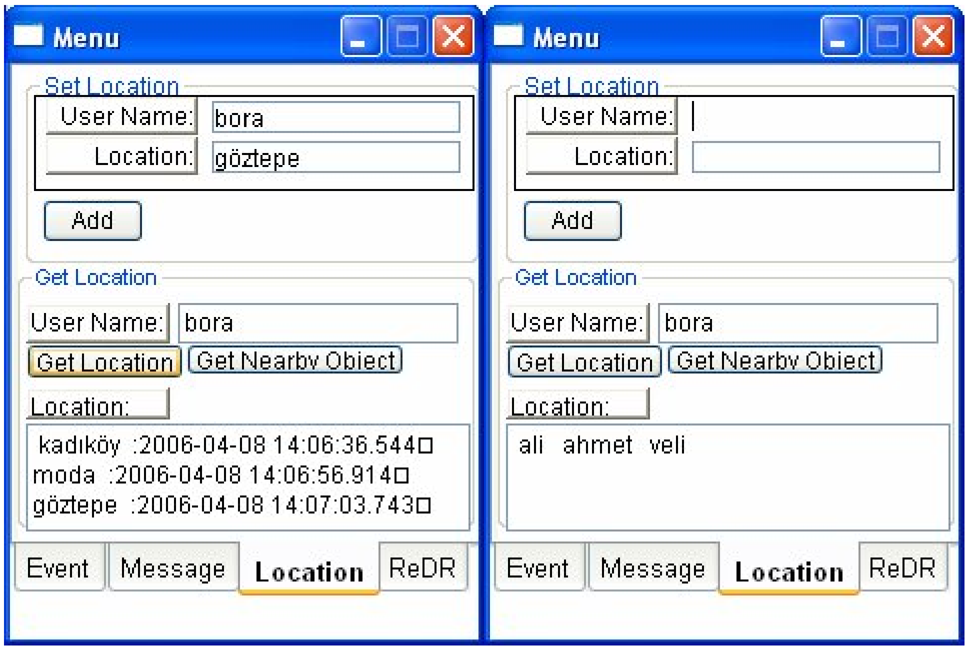
```
☑ LocationService
▽ Methods
    setLocationIndoor
    setLocationOutdoor
    setLocation
    getLocationName
    getLastLocationName
    getLocationNameinTime
    getNearbyObjects
  Server
☑ Client
  Settings
```

Location

- setLocation methods for different location types, indoor, outdoor
- getLocationName/setLocationName
- getLocationNameinTime: specify the location of the user according the time slices

getNearbyObjects: get nearby objects

Fig. 3.14 Location Service in Axis



User Name: bora
Location: göztepe

User Name: bora
Location:
```
kadıköy :2006-04-08 14:06:36.544□
moda :2006-04-08 14:06:56.914□
göztepe :2006-04-08 14:07:03.743□
```

User Name:
Location:

User Name: bora
Location:
```
ali  ahmet  veli
```

Fig. 3.15 TestPad Location Service

With the help of TestPad we test the method of LocationService as shown in Fig.3.14. We query the location of the user in time and near by objects of that user.

3.3.4 Redirection Service

The redirection service is responsible for client redirection to different servers and server reference translation at certain points. There are two main problems dealt with in redirection: address migration and data replication as shown Fig 3.15. Server translation and redirection can be done transparently or none transparently. The redirection service needs to be aware of the state of connectivity and the location of the client at any point in time.

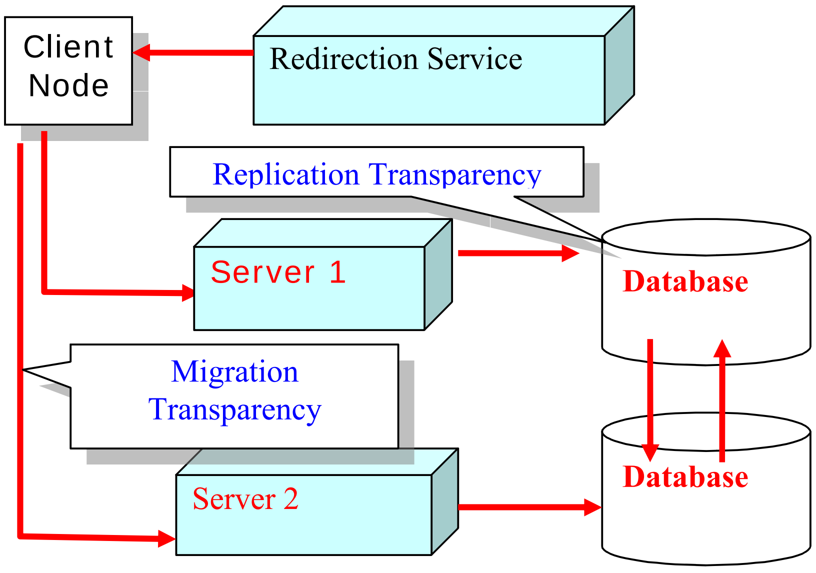


Fig. 3.16 Redirection service.

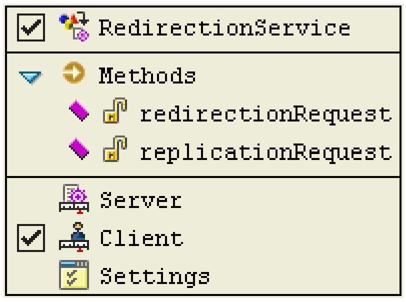
	**RedirectionService**
	redirectionRequest(String username)
	replicationRequest(String username,String newEndPoint)

Fig. 3.17 Redirection Service in Axis.

The steps of redirection:

- Request- sent from a client to a server to signal that the client wishes to carry out a request on the server.
- Response sent from the server to a client in response to a request.
- Redirection – Sent from a server to a client signaling that the client should send its request to another server whose reference will be embedded in redirection response.

Migration transparency allows components to change their location, which is not seen by a client requesting these components. In an asynchronous replication one server acts as the master, while one or more other servers act as slaves. The master server writes updates to log files, and maintains an index of the files to keep track of log rotation. These logs serve as records of updates to be sent to any slave servers. The slave receives any updates that have taken place, and then blocks and waits for the master to notify it of new updates. These replication functions are provided by the replication servers of different vendors.

CHAPTER 4

SERVICE ORCHESTRATION WITH COMPOSITE SERVICES

4.1 Service Orchestration

Complex mobile services are created by aggregating the functionality provided by each independent service. Typically, a composite web service specification is executed by a single coordinator node as shown as Service Orchestration component in Fig 4.1 It receives the client requests, makes the required data transformations and invokes the component web services. Composite services provide standardized interfaces for state transfer between basic services.

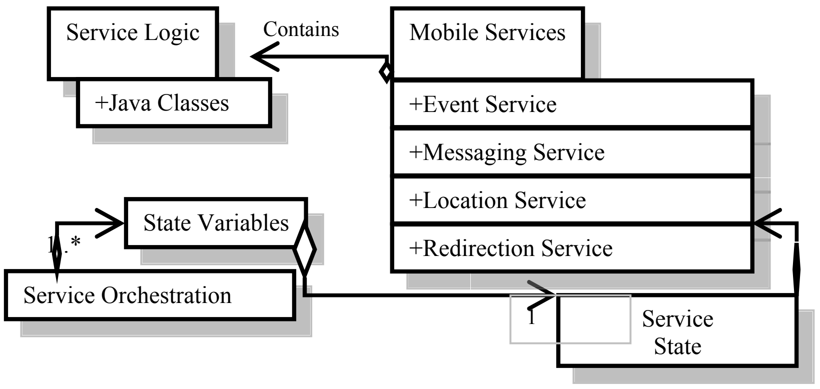


Fig. 4.1 Service orchestration.

Service Orchestration: This describes how Web services can interact at the message level, including the business logic and execution order of the interactions.

State Variables: The input and output parameters of the service.

Service State: Services are self-contained and required to maintain their own state. Services should not be dependent on the state of other services.

Mobile services: The overall representation of the service, which consists of one or several service logic components.

Service Logic: the executable code

4.2 Service Orchestration in SORWIM

Fig. 4.2 shows two example composite services. The Location Based Redirection Composite Service (LBRCS) maintains transparent redirection between the indoor and outdoor servers according to changes in the mobile location using basic Location and Redirection services. The Context Aware Notification Delivery Composite Service provides (CANDCS) notification messages that are created by the middleware based the context information sensed by mobile devices using three basic services.

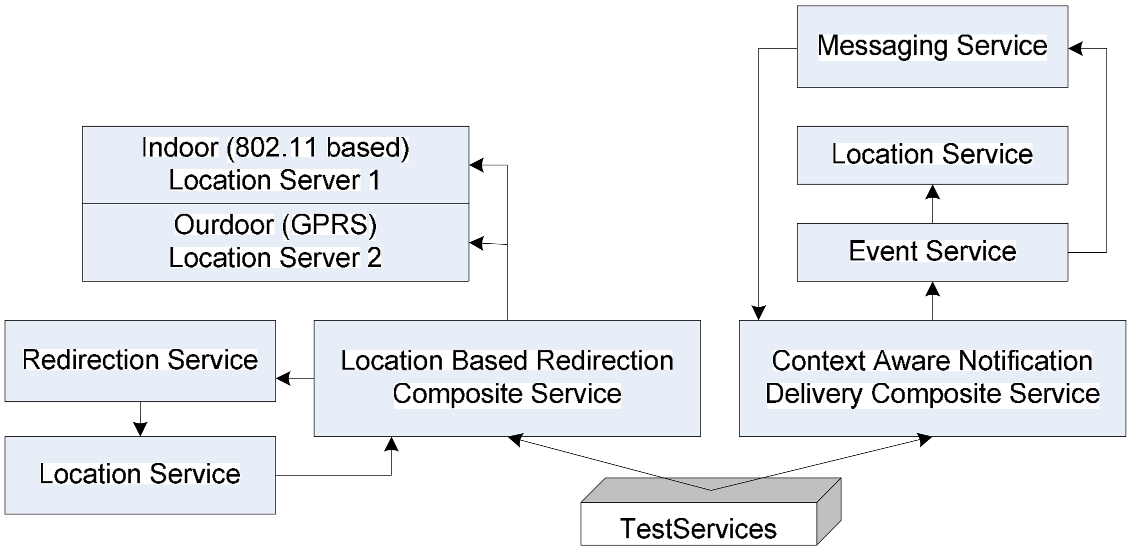
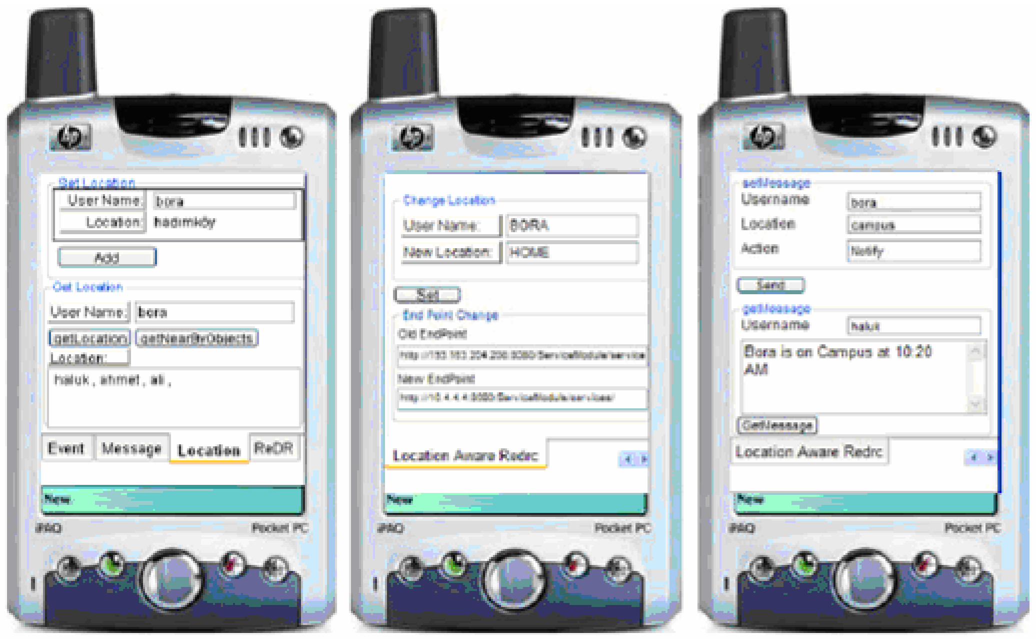

Messaging Service
Indoor (802.11 based) Location Server 1
Ourdoor (GPRS) Location Server 2
Location Service
Event Service
Redirection Service
Location Based Redirection Composite Service
Context Aware Notification Delivery Composite Service
Location Service
TestServices

Fig. 4.2 Service composition in SORWIM.

LBRCS maintains transparent redirection according to changes in the mobile location using basic Location and Redirection services. This service first obtains the device location (Location Service) and redirects to the available addresses (Redirection Service). We will give a scenario to explain how LBRCS can be used in a mobile application. In this scenario, there are three locations. These are the user's home and user's office which both are covered by a Wireless LAN, and the outdoor location between the home and office where GPRS connection is available. In all these three locations the user can connect to these networks using Pocket PCs or laptops. When the user has left his/her office and moved out from the office WLAN coverage area, his mobile device keeps connected to SORWIM through the GPRS connection. The WLAN-to-GPRS handover is generally initiated by the mobile user. In our approach, the mobile user sets the location information based on the local measurement of WLAN signal quality. When LBRCS decides that a WLAN-to-GPRS handover is required, it sends a handover-required message and an available endpoint (IP, port number, and service name) of the server to the mobile device. The mobile user then performs a handover and switches from the WLAN connection to the GPRS connection. When the user arrives in home, and discovers a wireless access point, the mobile device sends the location information to LBRCS through the GPRS connection.

SORWIM then sends the response of the endpoint of the server that is accessible in the office. The user performs a handover again and switches from GPRS to the WLAN connection again. The TestPad screen for this last scenario is shown in Fig 5 (b). The user sets the new location as Home and gets a response message from SORWIM. The response is a new connection end point for the WLAN network. The old and new end points are also shown in the lower part of the screen in Fig 5 (b). CANDCS which composes the services of Messaging, Event and Location, notifies all registered users who are interested in a specific event. Imagine the following scenario using this composite service: Haluk is waiting for Bora's arrival and types in his Pocket PC as "notify me as soon as Bora appears on the campus". This message is sent to SORWIM using the Pocket PC's window shown in the upper part of the screen in Fig 5 (c). When Bora arrives and enters the campus, immediately a notification is sent to Haluk's Pocket PC as a message or SMS as shown in the lower part of the screen in Fig 5 (c)The Location Based Redirection Composite Service maintains transparent redirection according to changes in the mobile location using basic Location and Redirection services. The solution can be seen as two main steps –obtain mobile device location (Location service) and - redirect to the available addresses (Redirection Service).

In an other scenario, Context aware notification delivery service can be used as emergency service. When a user is in a emergency position. He/She send emergency message to the middleware .The users in the same region get this emergency mes-sage with the user‘s positions in time as shown in Context Aware notification delivery services role is the decision making for request similar to “I want to be registered for notification about whether Person X is in Y location .The event service provide interface for registering the notifications. The location service serve for locations and can query the near by objects as as shown in Fig. 4.3. And the messaging service that has the ability of sending different type of messages as shown in is responsible for delivering the notification messages.


Set Location
User Name: bora
Location: hadimköy
Get Location
User Name: bora
Location:
haluk , ahmet , ali ,

Change Location
User Name: BORA
New Location: HOME
End Point Change
Old EndPoint
http://193.163.204.200:8080/ServiceModule/service
New EndPoint
http://10.4.4.4:8080/ServiceModule/services/

setMessage
Username bora
Location campus
Action Notify
getMessage
Username haluk
Bora is on Campus at 10:20 AM

Fig. 4.3 Basic services (a) and composite services (b) (c) TestPad screens

CANDCS makes a decision using three basic services for requests similar to "I want to be registered for notifications about whether person X is in Y location". The Event Service provides an interface for registering the notifications. The Location Service gives location information and can query the nearby objects as shown in Fig. 4.3 (a). The Messaging Service that has the ability of sending different types of messages is responsible for delivering the notification messages.

4.3 Performance

In this section we report on comparisons of Web Service and Java RMI solutions for messaging service in SORWIM. Web Services differ from distributed object technologies like RMI, CORBA by not accepting the concept of an object reference; instead a service is defined simply by an end-point that supports various operations.(N.A.B. Gray,2002) . Auto generated client-side stubs approach in Web Services that is very similar to Java-RMI clients. Typically starts with an interface definition for the service. A client-side stub is auto-generated from this interface.

There are a number of research studies for the performance evaluations of web services and SOAP compared to middleware technologies, including Java RMI, CORBA. In these studies, a comparison is made based on the performance of a web service implementation that calls a simple method and an RMI, CORBA, or other implementation that calls the same method (K. P. Birman, 2004) , (D.Davis et al, 2002) , (C.Damerey et al, 2005) , (J.Elfwing et al, 2004) , (M. B. Juric et al, 2004) .

We performed similar benchmarking tests to measure the performance of SORWiM to see if its performance is acceptable for a wireless middleware. The technologies used in this study were: Jakarta Tomcat 5.0.25, Axis 1.2, Java RMI from Java 1.4 SDK. The server specification is Intel Pentium M Processor 1.6 GHz 591 Mhz. 504 MB RAM ,the latest Ethereal tool (WireShark) network traffic analysis program. The client is an HP IPAQ h6340 Pocket PC. The Messaging Service in SORWiM is implemented using two middleware technologies, RMI and Web Service. The remote interface for performance tests is given in Fig. 4.4.

```
public interface MessagingService extends java.rmi.Remote {

   public java.lang.String getMessage(java.lang.String from) throws
java.rmi.RemoteException;

   public java.lang.String sendMessage(java.lang.String to, java.lang.String from,
java.lang.String subject, java.lang.String content, java.lang.String type) throws
java.rmi.RemoteException; }
```

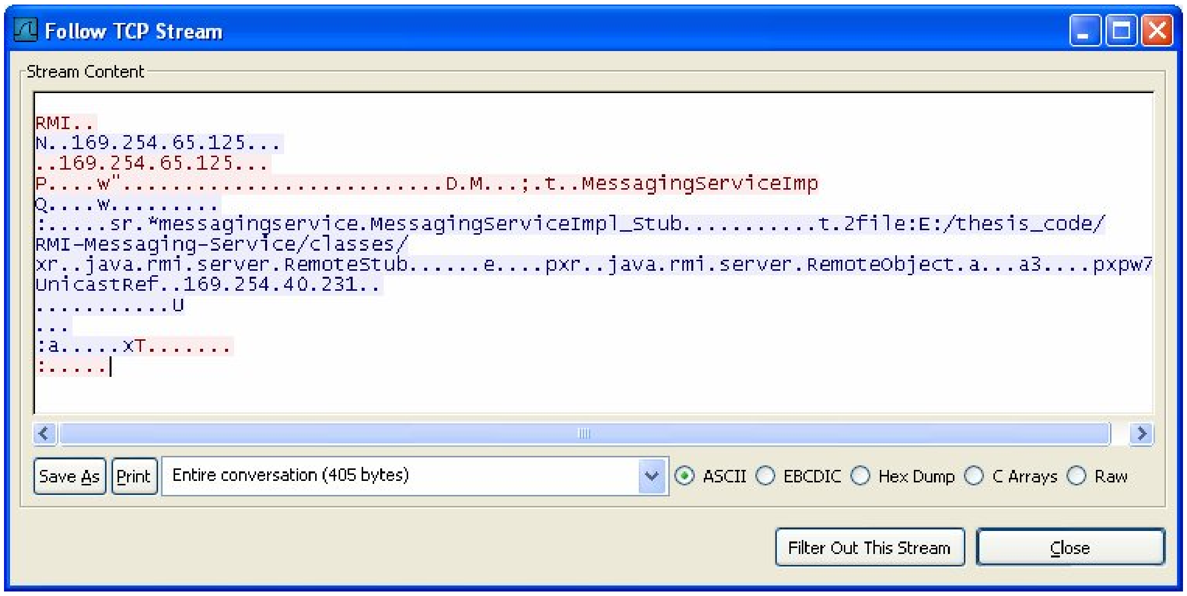
Fig. 4.4 The messaging service interface

The data shown in Table 1 show relative performances for two implementations of the "messaging service" with a request for a single call. For the first call the web service performance is better than RMI .

Method	**Call**	WS packet number ans size	**CPU Time Client** [ms] Web Services	RMI packet number ans size	**CPU Time Client** [ms] RMI
sendMessage	1	8 /1953 bytes	330	43 / 4268 bytes	5975

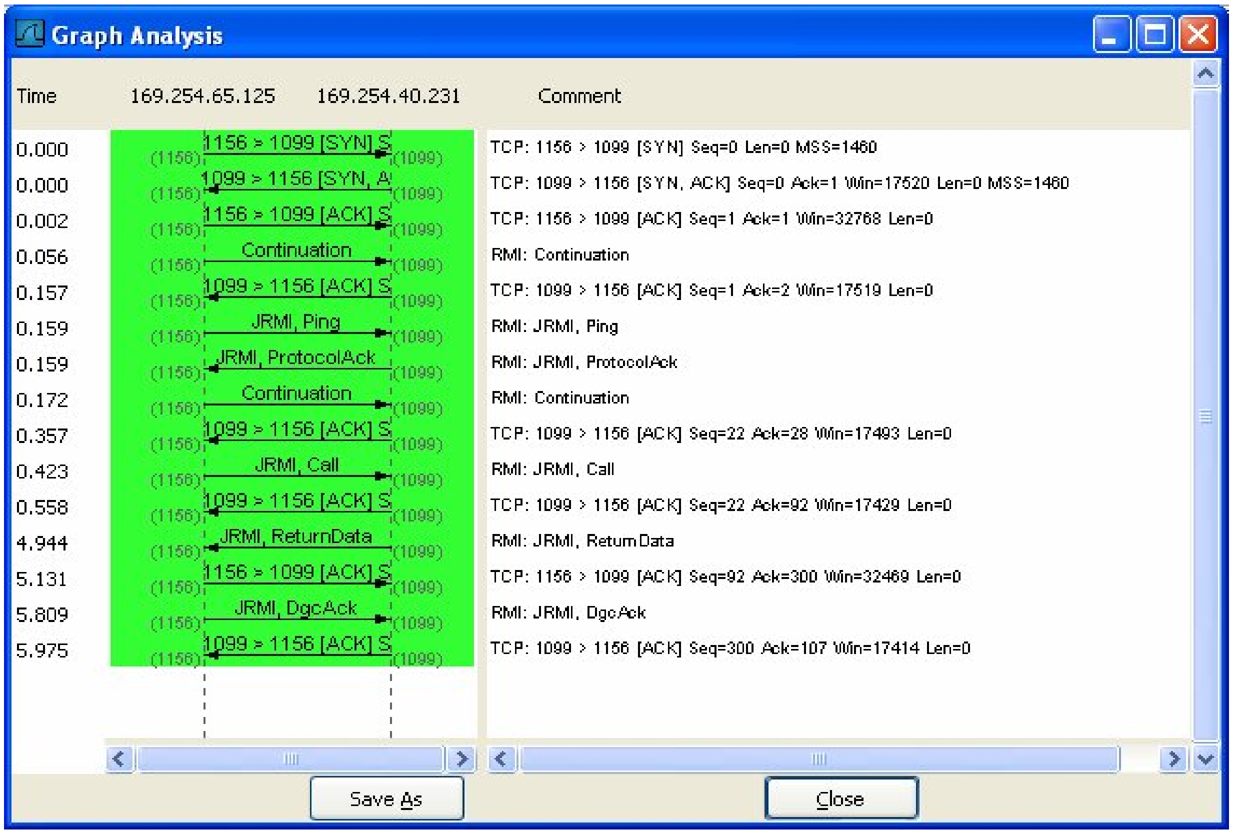
Table 1

The Java-RMI solution has the most complex mechanism ; establishment of a connection to the rmiregistry and submission of lookup request ,establishment of a connection to an HTTP server and posting of a request to download a class file which TCP stream is shown in Figure 4.5, and finally exchanges with the actual server. The graph analysis show that RMI registry cost 101 ms and connection the server costs 185 ms and download stub cost 4396 ms. These costs are the reasons of the poor performance as shown in Fig. 4.6



```
RMI..
N..169.254.65.125...
..169.254.65.125...
P....w"........................D.M...;.t..MessagingServiceImp
Q....w.........
:.....sr.*messagingservice.MessagingServiceImpl_Stub..........t.2file:E:/thesis_code/
RMI-Messaging-Service/classes/
xr..java.rmi.server.RemoteStub......e....pxr..java.rmi.server.RemoteObject.a...a3....pxpw7
UnicastRef..169.254.40.231..
...........U
...
:a.....xT.......
:.....
```

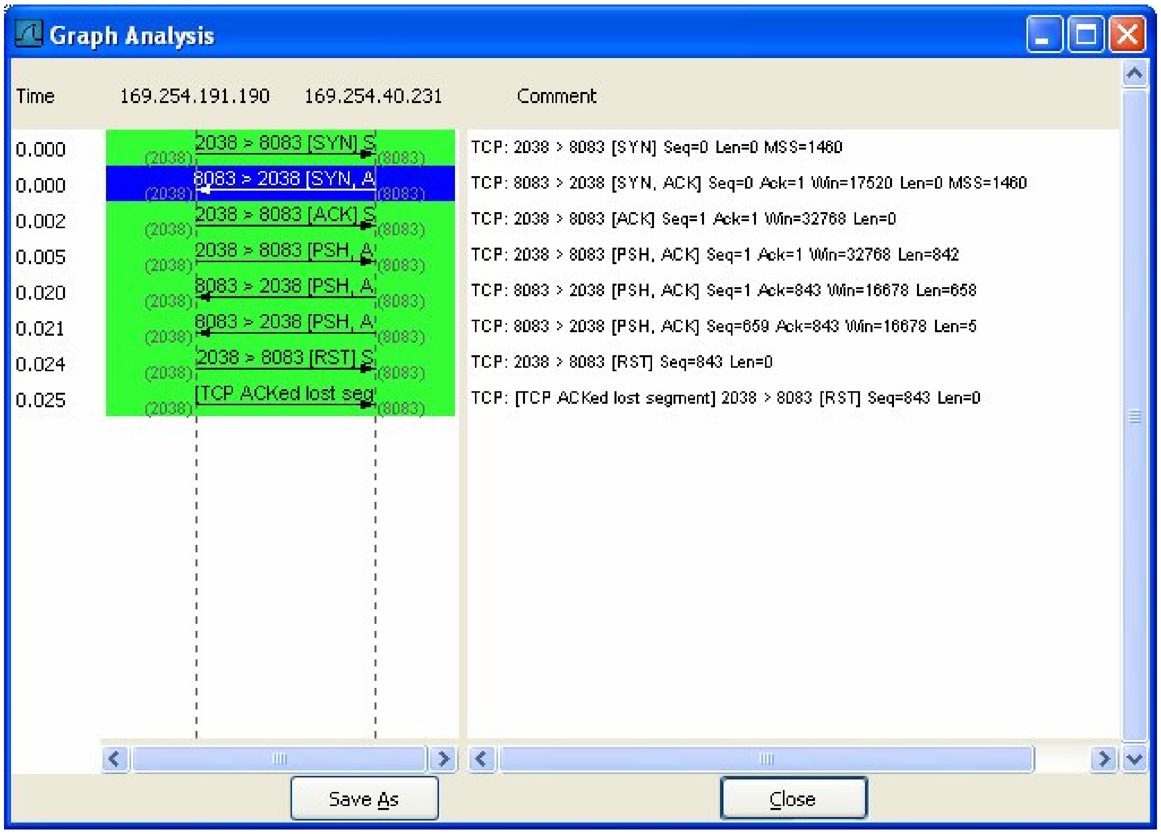
Fig. 4.5 The TCP stream of remote stub



```
Time     Comment
0.000    TCP: 1156 > 1099 [SYN] Seq=0 Len=0 MSS=1460
0.000    TCP: 1099 > 1156 [SYN, ACK] Seq=0 Ack=1 Win=17520 Len=0 MSS=1460
0.002    TCP: 1156 > 1099 [ACK] Seq=1 Ack=1 Win=32768 Len=0
0.056    RMI: Continuation
0.157    TCP: 1099 > 1156 [ACK] Seq=1 Ack=2 Win=17519 Len=0
0.159    RMI: JRMI, Ping
0.159    RMI: JRMI, ProtocolAck
0.172    RMI: Continuation
0.357    TCP: 1099 > 1156 [ACK] Seq=22 Ack=28 Win=17493 Len=0
0.423    RMI: JRMI, Call
0.558    TCP: 1099 > 1156 [ACK] Seq=22 Ack=92 Win=17429 Len=0
4.944    RMI: JRMI, ReturnData
5.131    TCP: 1156 > 1099 [ACK] Seq=92 Ack=300 Win=32469 Len=0
5.809    RMI: JRMI, DgcAck
5.975    TCP: 1099 > 1156 [ACK] Seq=300 Ack=107 Win=17414 Len=0
```

Fig. 4. 6 Graph Analysis Of RMI Connection

The typical illustrative Web Service application has a client connect to a service, submit a single request for data, and terminate as shown in Fig 4.7. In any case, for such applications, Web Services technologies perform well in comparison with the distributed object systems.



Time	169.254.191.190	169.254.40.231	Comment
0.000	(2038) 2038 > 8083 [SYN] S	(8083)	TCP: 2038 > 8083 [SYN] Seq=0 Len=0 MSS=1460
0.000	(2038) 8083 > 2038 [SYN, A	(8083)	TCP: 8083 > 2038 [SYN, ACK] Seq=0 Ack=1 Win=17520 Len=0 MSS=1460
0.002	(2038) 2038 > 8083 [ACK] S	(8083)	TCP: 2038 > 8083 [ACK] Seq=1 Ack=1 Win=32768 Len=0
0.005	(2038) 2038 > 8083 [PSH, A	(8083)	TCP: 2038 > 8083 [PSH, ACK] Seq=1 Ack=1 Win=32768 Len=842
0.020	(2038) 8083 > 2038 [PSH, A	(8083)	TCP: 8083 > 2038 [PSH, ACK] Seq=1 Ack=843 Win=16678 Len=658
0.021	(2038) 8083 > 2038 [PSH, A	(8083)	TCP: 8083 > 2038 [PSH, ACK] Seq=659 Ack=843 Win=16678 Len=5
0.024	(2038) 2038 > 8083 [RST] S	(8083)	TCP: 2038 > 8083 [RST] Seq=843 Len=0
0.025	(2038) [TCP ACKed lost seg	(8083)	TCP: [TCP ACKed lost segment] 2038 > 8083 [RST] Seq=843 Len=0

Fig. 4.7 Graph Analysis Of Web Service Connection.

Method	Call	CPU Time Server [ms] Web Services	CPU Time Client [ms] Web Services	CPU Time Server [ms] RMI	CPU TimeClient [ms] RMI
getMessage	100	3216	44320	5238	9120
sendMessage	100	4247	53540	6116	11170

Table 2

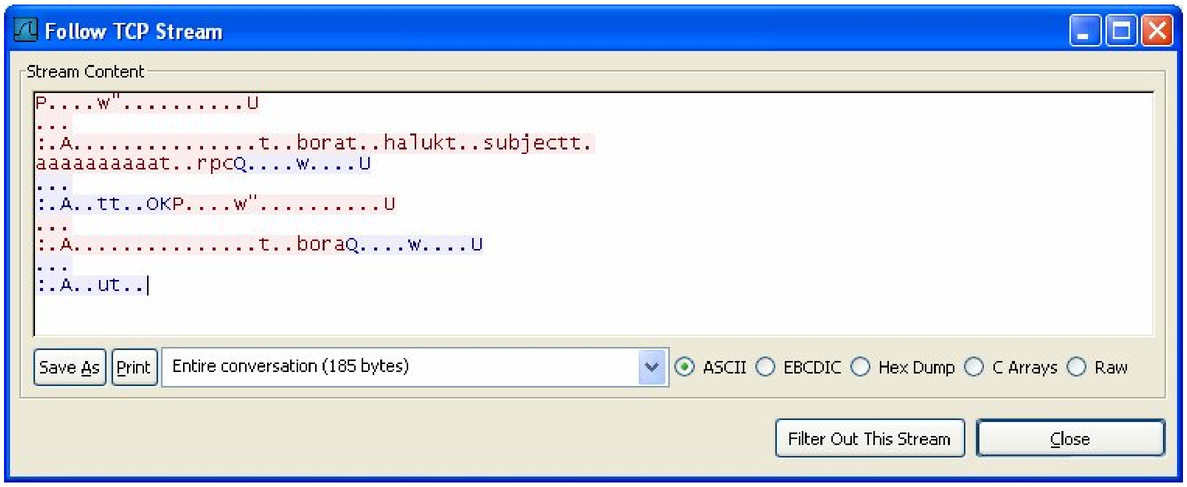
Our test results and other research studies showed in Table 2 that web services implementations performed slower than a Java RMI implementation.

The large amount of XML metadata contained in SOAP messages as shown in Fig 4.8 is the main reason that Web services will require more network bandwidth and CPU times than RMI as shown in Fig. 4.9. We have concluded that although the web service implementations are slower, the performance is acceptable for many real time operations of a wireless middleware. The performance can be made better and improved if the following issues are known and taken into consideration in middleware server designs.

All numeric and other data in web services are converted to text. Meta-data, defining structure, are provided as XML mark-up tags. XML parsers allow client and server implementations to construct their distinct but equivalent representations of any data structures The use of HTTP, and XML text documents, supports increased interoperability but also represents a significant increase in run-time cost for web service solutions as compared with Java-RMI solutions. The XML formatted documents are inherently more voluminous than the binary data traffic of the other approaches. More data have to be exchanged across the network, and more control packets are required.

When considering performance alone, web services provide value when the over-head of parsing XML and SOAP is outweighed by the business logic or computation performed on the server. Although web services generally don't provide value in performance, but do provide a convenient way to provide user interface, automatic firewall protection (because they use HTTP for transport and HTTP traffic can normally pass through firewalls), mobility and heterogeneity of applications, transparent proxies, and thin clients. The most natural designs for distributed objects are easy to use but scale poorly, whereas web services have good scaling properties.

The performance studies in literature generally measure RPC-style communication and do not consider the possibilities of document-oriented designs that demonstrate the strengths of web services. Web services are not intended to be used RPC-style like other distributed object technologies. Web services provide a literal encoding that can be used in a document-centric paradigm rather than an RPC-centric one. If the document-centric nature of web services is used in implementations, web services can outperform other traditional implementations when compared with an RPC-centric approach. According to .(N.A.B. Gray, 2002) the pure-object RMI or CORBA implementation is faster for small batches of documents and low-latency networks, but performance degrades rapidly with larger batches and higher latency. The web services have a high initial cost, but show little or no change with larger batches. Higher latency creates a greater initial cost, but performance is still independent of batch size. As latency increases, the performance benefits of the document-oriented approach increase significantly. This is relevant when in some real world wireless communication scenarios, latency may even be minutes, or hours, as for disconnected or asynchronous operations.

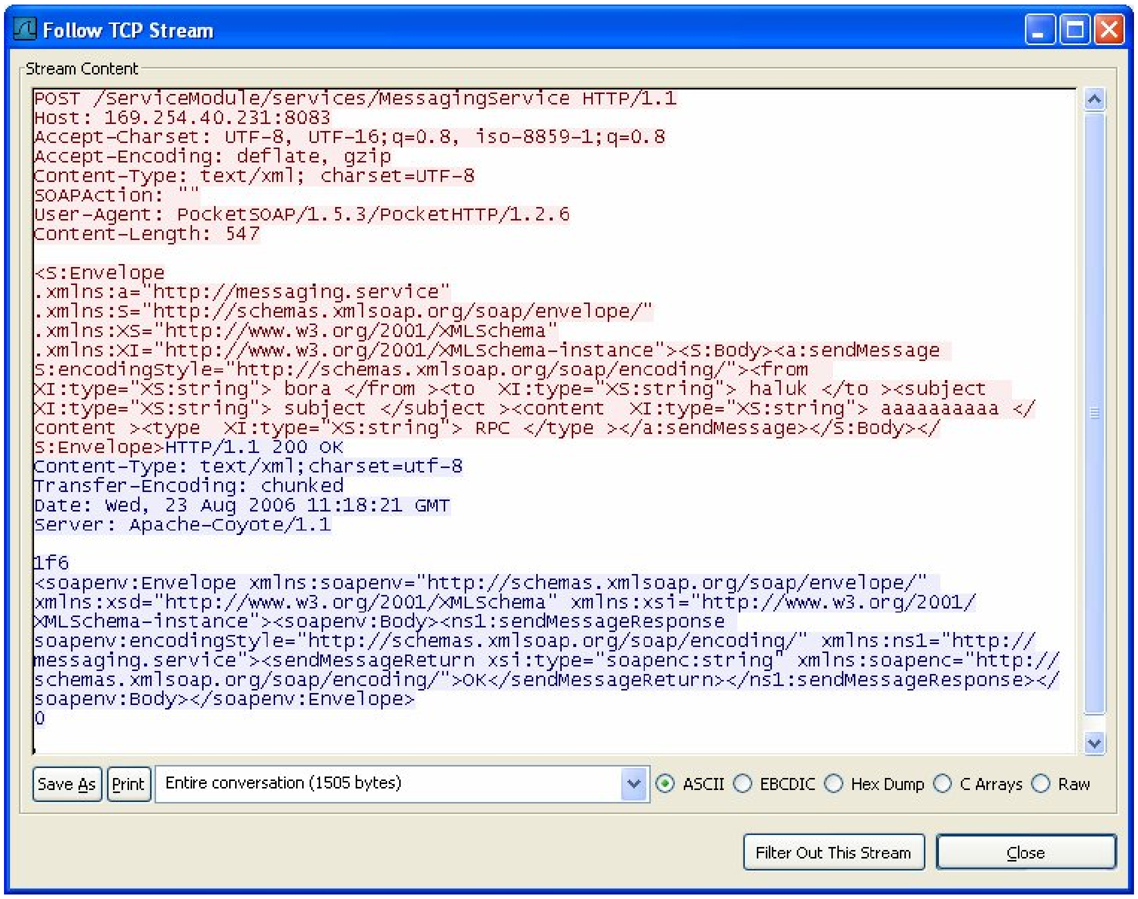


```
P....w"..........U
...
:.A...............t..borat..halukt..subjectt.
aaaaaaaaaat..rpcQ....w....U
...
:.A..tt..OKP....w"..........U
...
:.A...............t..boraQ....w....U
...
:.A..ut..
```

Entire conversation (185 bytes)

[✓ ASCII] EBCDIC Hex Dump C Arrays Raw

Fig. 4.8 TCP Stream of RMI



```
POST /ServiceModule/services/MessagingService HTTP/1.1
Host: 169.254.40.231:8083
Accept-Charset: UTF-8, UTF-16;q=0.8, iso-8859-1;q=0.8
Accept-Encoding: deflate, gzip
Content-Type: text/xml; charset=UTF-8
SOAPAction: ""
User-Agent: PocketSOAP/1.5.3/PocketHTTP/1.2.6
Content-Length: 547

<S:Envelope
.xmlns:a="http://messaging.service"
.xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
.xmlns:XS="http://www.w3.org/2001/XMLSchema"
.xmlns:XI="http://www.w3.org/2001/XMLSchema-instance"><S:Body><a:sendMessage
S:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"><from
XI:type="XS:string"> bora </from ><to  XI:type="XS:string"> haluk </to ><subject
XI:type="XS:string"> subject </subject ><content  XI:type="XS:string"> aaaaaaaaaa </
content ><type  XI:type="XS:string"> RPC </type ></a:sendMessage></S:Body></
S:Envelope>HTTP/1.1 200 OK
Content-Type: text/xml;charset=utf-8
Transfer-Encoding: chunked
Date: Wed, 23 Aug 2006 11:18:21 GMT
Server: Apache-Coyote/1.1

1f6
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance"><soapenv:Body><ns1:sendMessageResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns:ns1="http://
messaging.service"><sendMessageReturn xsi:type="soapenc:string" xmlns:soapenc="http://
schemas.xmlsoap.org/soap/encoding/">OK</sendMessageReturn></ns1:sendMessageResponse></
soapenv:Body></soapenv:Envelope>
0
```

Entire conversation (1505 bytes)   [✓ ASCII] EBCDIC Hex Dump C Arrays Raw

Fig. 4.9 TCP Stream of web service.

CHAPTER 5

CONCLUSION

The increasing diversity of devices-terminals, network elements and the application servers imply that different middleware solutions will be in use. In near future a single dominant middleware platform which provides the interpretability between existing middleware platforms and between parts of an application running on different middleware platforms is needed. With SOC benefits, the requirements and the functionalities of the wireless middleware solutions may be achieved. In this thesis the first designs and issues of a SOA based reflective wireless middleware, SORWIM, were presented with the help of location, messaging, event (notification) and redirection services .the key issue problems in the wireless environment are tried to be solved.

We would like to work on the research and development problems of different mobile applications such as location based services (LBS) based on service oriented computing using the basic and composite services provided by the middleware. The currently SORWIM has only four basic services and two composite services which can be used as building blocks of such wireless applications. We need develop some more new basic and composite services for more complex mobile applications and LBS. The integration of SORWIM and WiPoD will be our next project. The WiPoD client application currently has a decentralized architecture and does not need a server. It will be connected to the SORWIM server to provide the indoor location information for the mobile user

The performance evaluation and runtime environments such as operating systems, required programs and libraries, memory sizes, CPU powers and I/O requirements of mobile devices to run SOA based typical mobile applications will be also analyzed and presented in the future.

REFERENCES

Andrews R.,"Business Process Execution Language for Web Services (BPEL4WS)", *Version 1.1. Technical Report* 2003

Birman K. P., "Like it or not, web services are distributed objects," *Commun. ACM*, vol. 47, no. 12, pp. 60–62, 2004.

Box D. ,Ehnebuske D. , Kakivaya G. , Layman A., Mendelsohn M., Nielsen N., ," Simple Object Access Protocol (SOAP) 1.1." *WRC Note*, available at http://www.w3.org/TR/2000/NOTE-SOAP-2000058/ 2000

Chinnici R., Gudgina M., Moreau J.,Weerawarana S ., ,"Web Service Description Language (WSDL)", version 1.2. Technical Report 2002

Davis D. , Parashar M. , "Latency performance of SOAP implementations," *IEEE Cluster Computing and the Grid*, 2002.

Demarey C., Harbonnier H., Rouvoy R., and Merle P., "Benchmarking the round-trip latency of various java-based middleware platforms,"Studia Informatica Universalis Regular Issue, vol. 4, no. 1, p. 724, May 2005, iSBN: 2912590310.

Dertouzos M., "The Oxygen Project," Scientific American, Vol. 281 No. 2, August 1999, 52-63.

Dey A.K., "Understanding and Using Context," *Personal and Ubiquitous Computing*, 2001.

Elfwing R., Paulsson U., Lundberg L., "Performance of SOAP in web service environment compared to CORBA." in APSEC. *IEEE Computer Society*, 2002, pp. 84–.

Emmerich W.,"Software Engineering and middleware:A roadmap.", *Communications of the ACM,* pages 117 -129 2000

Endeavour Expedicion: Charting the Fluid Information Utility, http://endeavour.cs.berkely.edu/.

Esler M., "Next Century Challenges: Data-Centric Networking for Invisible Computing," in Proc. MobiCom '99, August 1999, 256-262.

FCE: Future Computing Environments, http://www.cc.gatech.edu/fce/

Gamma E. , Helm R., Johnson R., Vlissides J., , "Design Patterns, Elements of Reusable Object-Oriented Software." *Addison-Wesley* 1995

Gray N.A.B.,"Comparison of Web Services, Java-RMI, and CORBA service implementations" 2002, available at : http://mercury.it.swin.edu.au/ctg/AWSA04/Papers/gray.pdf

Gribble S.D. , "The Ninja Architecture for Robust Internet-Scale Systems and Services," to appear in a Special Issue of Computer Networks on Pervasive Computing.

Gümüşkaya H., Hakkoymaz H. ," WiPoD Wireless Positioning System Based on 802.11 WLAN Infrastructure.", *Proceedings of the Enformatika*, Vol. 9 (2005) 126–130

Heinzelman W. B., Murphy A. L., Carvalho H. S. , Perillo M. ,"A. Middleware to Support Sensor Network Applications.", *IEEE Network*, January/February 6 – 14 2004

Hellerstein J. M. , Avnur R. , "Eddies: Continuously Adaptive Query Processing," *in Proc. ACM SIGMOD 2000 Conference*.,2000

Hellerstein. J. M., "Adaptive Query Processing: Technology in Evolution," *IEEE Data Engineering Bulletin*, 2000.

JAVA RMI , available at http://java.sun.com/products/jdk/rmi/

Juric M. B. , Kezmah B. , " Java RMI, RMI tunneling and Web services comparison and performance analysis"*ACM ,2004*

Kubiatowicz J., "OceanStore: An Architecture for Global-Scale Persistent Storage," *Proc. ASPLOS 2000*, November 2000.

Maniatis P. , "he Mobile People Architecture," ACM Mobile Computing and Communications Review, July 1999.

Mascolo C. , Capra L. ,Emmerich W. ,"Mobile Computing Middleware. Lecture Notes In Computer Science, Advanced Lectures on Networking", *Vol. 2497. Springer-Verlag* 20–58 2002

Mascolo C., Capra L. , Emmerich W.,"Principles of Mobile Computing Middleware.", *In Middleware for Communications Wiley* 2004

Mindreef Coral ,available at: http://www.mindreef.com/

MIT Project Oxygen, http://www.oxygen.lcs.mit.edu/.

Mobile RMI , available at https://www.cs.tcd.ie/publications/tech-reports/reports.05/TCD-CS-2005-13.pdf

MosquitoNet: The Mobile Computing Group at Stanford University, http://mosquitonet.stanford.edu/index.html.

Nunn R. ,"Distributed.Software Architectures Using Middleware", Available at: http://www.cs.ucl.ac.uk/staff/W.Emmerich/lectures/3C05-02-03/aswe18-essay.pdf

Papazoglou M. , Georgakopoulos D. ," Service-Oriented Computing.", *Communications of the ACM.* Vol. 46, No. 10 2003

Papazoglou M. P. , Heuvel W. J. ,"Service Oriented Architectures: Approaches, Technologies and Research Issues.", *The VLDB Journal* 2005 Available at: http://infolab.uvt.nl/pub/papazogloump-2005-81.pdf

PocketBuilder: http://www.sybase.com/products/developmentintegration/pocketbuilder

Portolano: An Expedition into Invisible Computing, http://portolano.cs.washington.edu/.

Raatikainen K, " Functionality Needed in Middleware for Future Mobile Computing Platforms,"*Citeseer*,2001 available at : http://citeseer.ist.psu.edu/raatikainen01functionality.html

Roman M. , Campbell R.H., "Gaia: Enabling Active Spaces," in Proc. ACM SIGOPS European Workshop, Kolding, Denmark, September 2000.

Roman M., Kon F. , Campbell R. ,"Reflective Middleware: From your Desk to your Hand.", *IEEE Communications Surveys*, 2 (5) 2001

Schantz R., Schmidt D., "Middleware for Distributed Systems: Evolving the Common Structure for Network-Centric Applications.", *In Encyclopedia of Software Engineering John Wiley & Sons, Inc*., New York 2001

Schmidt D., "Middleware for Real-Time and Embedded Systems.", *Communications of the ACM*, Vol. 45, No: 6 June 2002

Schmidt D., Stal M., Rohnert H., Buschmann F., ,"Pattern-Oriented. Software Architecture: Patterns for Concurrent and Networked Objects." *John Wiley & Sons, Inc*., New York 2000

Sen R., Handorean R. , Roman G., Gill C.," Service Oriented Computing Imperatives in Ad Hoc Wireless Settings. Service-Oriented Software System Engineering: Challenges And Practices",, *Idea Group Publishing*, Hershey, USA, April, 247 – 269 2005

Thanh D., Jørstad I. ," A Service-Oriented Architecture Framework for Mobile Services", *IEEE Proceedings of the Advanced Industrial Conference on Telecommunications/Service Assurance with Partial and Intermittent Resources Conference/ELearning on Telecommunications Workshop*, September 2005

The PIMA Project: Platform-Independent Model for Applications, http://www.research.ibm.com/PIMA/.

The Reactive Adaptive Proxy Placement Architecture (RAPP) , available at http://www.comp.lancs.ac.uk/~adrian/Papers/seitz-proxyarch-im99.pdf

Vaughan-Nichols S.J. ,"Wireless Middleware: Glue for the Mobile Infrastructure.", *IEEE Computer,* Vol. 37, No. 5 18–20 2004

W3C. UDDI Technical White Paper. Technical Report 2000

Wang H.J. , "An Internet-core Network Architecture for Integrated Communications," *IEEE Personal Communications*, August 2000.

Weiser M., "Some Computer Science Issues in Ubiquitous Computing," Communications of the ACM, July 1993, 74-84.

Weiser M., "The Computer for the Twenty-First Century," Scientific American, September 1991, 94-104.

Yu Y., Krishnamachari B., PrasannaIssues V., ,"Designing Middleware for Wireless Sensor Networks.", *IEEE Network*, January/February 15 – 21 2004