PROACTIVE CACHING SCHEDULER FOR WEB SERVICES

by

Mehmet Güveniş

B.S., Computer Engineering, Eastern Mediterranean University, 2004

Submitted to the Institute of Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering
Boğaziçi University
2007

PROACTIVE CACHING SCHEDULER FOR WEB SERVICES

APPROVED BY:

Assoc. Prof. Dr. Can Özturan ……
(Thesis Supervisor)

Dr. Haluk Bingöl ……

Assist. Prof. Dr. Güray Yılmaz ……

DATE OF APPROVAL: 24.01.2007

ACKNOWLEDGEMENTS

I would like to thank my thesis advisor Assoc. Prof. Dr. Can Özturan for his contribution to my education, for showing me useful advices and keeping me motivated throughout my thesis study.

I thank to Dr. Haluk Bingöl and Assist. Prof. Dr. Güray Yılmaz for participating in my thesis jury and giving me feedback.

Finally, I would like to thank to my family for their supports and patience that made this study possible.

ABSTRACT

PROACTIVE CACHING SCHEDULER FOR WEB SERVICES

Web services are one the most promising technologies in distributed computing. Platform independence and easy interoperability features of web services solved many problems faced with earlier distributed computing technologies. As a result, web services are essential for communication of objects in distributed world. Nowadays, the main motivation regarding to web services is composing them to accomplish business applications. There are many web service composition languages in the literature. On those languages, Business Process Execution Language (BPEL) is the most effective composition language to provide interaction among web services.

Despite platform independence and easier interoperability features, web services face many performance and availability problems. There are two main reasons for this: The first reason is the propagation delay between clients and server. The second reason is extensive use of XML in web service operations. Thus, in this study, we aimed to solve performance and availability problems of web services that BPEL processes use in business applications by applying a caching mechanism. Our main contribution is to apply a proactive caching mechnism to web services, so generated cache can be used for future operations. Our Proactive Caching Scheduler and Daemon Proactive Caching Scheduler performs caching of web services whose binding information is known. One of the most important aspects of our study is to apply efficient caching without sacrificing overheads on client side. By this way, we aimed to decrease the load in web services since it is not necessary to invoke a web service if its cache is not expired. Our research concludes with experimental tests taken from real life scenarios. The experiments help us to better understand the applicability and efficiency of our approach.

ÖZET

WEB SERVİSLERİ İÇİN PROAKTİF ÖNBELLEKLEYİCİ VE PROGRAMCI

Web servisleri günümüzde kullanılan en önemli dağıtık yazılım mimarisidir. Web servislerinin platformdan bağımsız olması ve kolay entegrasyonu, kendinden önce var olan dağıtık yazılım mimarilerinde karşılaşılan sorunları çözmede etkili olmuştur. Sonuç olarak, web servisleri dağıtık mimarinin vazgeçilmez parçası olmuştur. Günümüzde, web servisleri ile ilgili ana motivasyon, iş akışlarında kullanılmak üzere değişik web servislerini birleştirme ve entegrasyonunu sağlamaya yöneliktir. Bu konuda en etkili web servis entegrsayonu Business Process Execution Language (BPEL) dili ile sağlanabilir.

Web servisleri, platformdan bağımsız ve kolay entegre olabilmelerine karşın, performans ve kullanılabilirlik açısından birçok problemlere sahiptirler. Bu konunun iki ana nedeni vardır. Birinci neden, web servislerine erişmede istemci sunucu arasında olan network trafiğidir. İkinci neden ise, web servisleri işlemlerinde XML bazlı protokollerin ve dillerin kullanılmasıdır. Bu yüzden, bu tezde, BPEL işlemlerinde kullanılan web servislerinin performans ve kullanılabilirlik sorunlarına çözüm bulmayı amaçladık. Bu çözüm için de katkımız, proaktif önbellekleme kullanarak, ileride gereken web servis sonuçlarını önbellekte tutmak olmuştur. Bu yüzden oluşturduğumuz Proaktif Önbellekleyici ve Hayali Proaktif Önbellekleyici ve Programcı, ulaşım bilgileri bilinen web servislerine etkili önbellekleme sağlamayı amaçlamıştır. En önemli amaçlarımızdan biri de, önbelleklemeyi sağlarken istemci tarafında performans kaybına sebep olmamak idi. Bu sayede, web servislerinin yükünü azaltmaya çalıştık. Tezin sonunda yaptığımız deneysel testler, yaklaşımımızın etkisini ölçmeye yardımcı olmuştur.

TABLE OF CONTENTS

ABSTRACT.....iv
ÖZET.....v
LIST OF FIGURES.....viii
LIST OF TABLES.....x
LIST OF ABBREVIATIONS.....xi
1. INTRODUCTION.....1
1.1. Motivation.....1
1.2. Outline.....2
2. PREVIOUS WORK 4
2.1. Introduction to Web Services 4
2.1.1. General Definition and Characteristics of Web Services.....4
2.1.2. Web Services vs. Other Distributed Technologies.....6
2.1.3. Web Services and Service Oriented Architecture.....8
2.1.3.1. XML.....8
2.1.3.2. SOAP.....9
2.1.3.3. WSDL.....9
2.1.3.4. UDDI.....10
2.1.4. Web Services Composition.....10
2.1.4.1. The Need to Compose Web Services.....11
2.1.4.2. Web Services Composition Standards.....12
2.1.4.3. Web Service Orchestration Engines.....17
2.2. Web Service Caching……19
2.2.1. The Importance of Web Service Caching.....19
2.2.2. Important Factors in Web Service Caching.....20
2.2.3. Existing Work In Web Service Caching……...22
2.3. Summary…….....29

3. OVERVIEW OF OUR ARCHITECTURE…….31
3.1. Platforms and Tools Used in Proposed Architecture……...31
3.2. Proposed Architecture……..32
4. PCS PRE-PROCESSOR……..36
4.1. Transformation of BPEL Documents……...36
4.2. Generating Executables of Transformed BPEL Documents…….41
5. PCS AND DAEMON PCS……...42
5.1. Proactive Caching Scheduler (PCS)…….42
5.1.1. Usage of WSIF in BPEL Processes and PCS……....42
5.1.2. Working Principle of PCS…….45
5.2. Daemon Proactive Caching Scheduler (PCS)……....48
5.2.1. Working Principle of Daemon PCS……....48
6. EXPERIMENTAL TESTS AND RESULTS……...51
6.1. US Weather Forecast Web Service……...52
6.2. Turkish Republic Central Bank Currency Exchange Web Service……..53
6.3. Turkish Republic Identity Web Service……....54
6.4. E-Bay Item Information Web Service……...56
7. CONCLUSION..……...58
7.1. Overview……...58
7.2. Future Work……..59
APPENDIX A: SOFTWARE CD……..60
REFERENCES……...61

LIST OF FIGURES

Figure 2.1. Orchestration and Choreography.....13

Figure 2.2. A schematic view of a BPEL process.....15

Figure 2.3. Example "Hello World" BPEL process.....17

Figure 2.4. Axis flow architecture.....26

Figure 2.5. High level view of the self-pruning cache object.....27

Figure 2.6. Overview of Content Delivery Network (CDN).....28

Figure 3.1. Generating executable of BPEL document…….....33

Figure 3.2. Execution of BPEL processes with PCS and Daemon PCS.....34

Figure 4.1. Initial BPEL document.....38

Figure 4.2. Transformed BPEL document.....40

Figure 5.1 Data types used in WSDL document of currency converter web service 43

Figure 5.2 Extracting Currency Converter Web Service Parameters in PCS.....44

Figure 5.3 WSIF Binding for currency converter web service on WSDL document of PCS.....45

Figure 5.4 Service Definition for PCS in WSDL document of PCS…….....45

Figure 5.5 Proactive Caching Scheduler (PCS) mechanism.....48

Figure 5.6 Daemon PCS mechanism.....50

Figure 6.1 Analyses of US Weather Forecast Web Service.....53

Figure 6.2 Analyses of Turkish Republic Central Bank Currency Exchange Web Service.....54

Figure 6.3 Analyses of Turkish Republic Identity Web Service.....56

Figure 6.4 Analyses of eBay Item Information Web Service.....57

LIST OF TABLES

Table 6.1 Test Environment Used In Experimentation.....51

Table 6.2 US Weather Forecast Web Service Input and Output Data.....52

Table 6.3 Turkish Republic Central Bank Currency Exchange Web Service Input and Output Data.....54

Table 6.4 Turkish Republic Identity Web Service Input and Output Data.....55

Table 6.5 E-Bay Item Information Web Service Input and Output Data.....56

Table 6.6 Summary of Complete Experimentation.....57

LIST OF ABBREVIATIONS

API	Application Programming Interface
ASP	Active Server Pages
B2B	Business to Business
BPEL	Business Process Execution Language
BPEL4WS	Business Process Execution Language for Web Services
BPWS4J	Business Process Execution Language for Web Services JavaTM Run Time
CDN	Content Delivery Networks
CORBA	Common Object Request Broker Architecture
DCOM	Distributed Component Object Model
DOM	Document Object Model
EAI	Enterprise Application Integration
EJB	Enterprise Java Beans
HTTP	Hypertext Transport Protocol
IBM	International Business Machines Corporation
IDE	Integrated Development Environment
J2EE	Java2 Platform Enterprise Edition
JAXB	Java Architecture for XML Binding
JAXP	Java API for XML Processing
JAXR	Java API for XML Registries
JAX-RPC	Java API for XML-Based RPC
JCA	Java Cryptography Architecture

JMS	Java Message Service
MD5	Message Digest 5
OMG	Object Management Group
PCS	Proactive Caching Scheduler
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SAAJ	SOAP with Attachments API for Java
SAX	Simple API for XML
SMTP	Simple Mail Transport Protocol
SOA	Service – Oriented Architecture
SOAP	Simple Object Transport Protocol
SQL	Structured Query Language
tModel	Type Model
TTL	Time to Live
TTR	Time to Refresh
UDDI	Universal Description Discovery Integration
WSCI	Web Service Choreography Interface
WSDL	Web Services Description Language
WSFL	Web Services Flow Language
WSIF	Web Services Invocation Framework
XLANG	XML Business Process Language
XML	Extensible Markup Language

1. INTRODUCTION

1.1. Motivation

Web services are one of the most popular and widely used applications in distributed computing. Web service technologies are based on XML (Extensible Mark-up Language), so it is possible to develop applications that all kinds of platforms and systems can use. Earlier versions of distributed object technologies had problems of integration, scalability and generic use. On the other hand, platform and implementation independence of web services solved the problems of integration, scalability and generic use of distributed object computing. Web service technologies evolved in recent years and are essential structures in many applications of business world. Thus, composing web services and maintaining interaction between composed web services has become very important for accomplishing business goals. There are several web service composition languages in the literature and it is stated that Business Process Execution Language for Web services (BPEL4WS) is the most promising and efficient web services composition language [1].

Despite their popularity and implementation independence, serious performance and availability problems are faced in web services. There are two main reasons for performance and availability problems in web services. First reason is propagation delays between client and servers, especially when the web service is not located on LAN (Local Area Network) of clients. The second reason is the performance degradation due to XML. Since XML processing is an expensive operation in terms of time and memory, it influences the performance of web service communication.

In this research, we aimed to solve the performance and availability problems of certain web services. Our main idea is to develop a caching mechanism to web services that BPEL processes use in business applications. In our architecture, BPEL processes are deployed in Oracle BPEL Process Manager and they are executed in Oracle BPEL Process Server. Our proposed architecture supports web services whose binding information is stated in a configuration file. Our architecture includes a Proactive Caching Scheduler Mechanism

(PCS) which uses this configuration file to decide whether to invoke a web service or apply caching based on the expiration information of corresponding web service. Furthermore, our architecture has a Daemon Proactive Caching Scheduler (Daemon PCS) which puts web services in a schedule and invokes the web services to refresh the cache as needed. Our architecture also includes a mechanism that keeps statistical information of previously invoked web services and this information is used to prune unimportant web service results when the cache size exceeds a threshold. Our contribution includes solution of two problems: Performance and availability. We aimed to solve performance problems of web service invocations since invocation data reside on cache and it is not needed to invoke a web service every time. Furthermore, our architecture solves the availability problems of web services since it provides a fault tolerance mechanism in case the web services may not be reached due to communication problems. Our proposed architecture is tested and its feasibility is analyzed with real life scenarios.

1.2. Outline

In Chapter 2, previous work related with web services and caching concept in web services is presented. In the first section, fundamental elements of web services are discussed. Then, usage of these elements in our architecture is stated. After that, the importance and necessity of composing several web services are emphasized. In the second section, previous work of different web service caching techniques are presented. These techniques are compared with our proposed architecture to reflect the applicability of our approach.

In Chapter 3, overview of our architecture is presented. First, platforms and tools used when designing our proposed architecture is stated. Then, brief explanation of our architecture is made. At the first stage, it is explained how to prepare BPEL processes to be able to interact with our proposed architecture. Then, core elements of our proposed architecture is explained briefly. The core elements of our architecture are elaborated in Chapter 4 and Chapter 5.

In Chapter 4, the importance and implementation details of PCS Pre-processor are explained which transforms BPEL documents in such a way that they are able to interact with our PCS. A simple example is given for the transformation of BPEL documents. Then, the deployment of BPEL processes to Oracle BPEL Process Server is explained.

In Chapter 5, caching operations of our architecture are explained. Our caching architecture consists of two parts. The first part is the PCS which interacts with BPEL processes and decides whether to apply caching or web service invocation. The second part is the Daemon PCS which schedules web services and refreshes the cache. The working principle of PCS and Daemon PCS is explained in detail. Then, some problems faced during caching is described and solution for these problems is emphasized.

In Chapter 6, experimental tests are made using real life scenarios to examine the applicability and feasibility of our approach. At the end, the efficiency of our approach is proven since there are significant performance improvements in processing.

In Chapter 7, the research is concluded by presenting the summary of work done and its contributions to the section. Then, planned future work is discussed to improve and extend the applicability of our approach.

2. PREVIOUS WORK

In this chapter, previous work related to web services and caching concept in web services is explained. The chapter consists of two sections. In the first section, a brief introduction to web services is made. First, definition and general characteristics of web services are explained. Then, web services and previous distributed computing technologies are compared to reflect the advantages of web services. After that, web service components are introduced. Then, the needs of web service composition and its standards are introduced. Moreover, the effects of web service components and web service composition in building our caching architecture are specified. In the second section of the chapter, the concept of caching in web services is explained. First, the motivation for caching in web services is mentioned. Then, some important factors that should be considered in web service caching are discussed. After that, different web service caching strategies are mentioned and the impacts of these strategies in web service based applications are emphasized. Moreover, existing web service caching strategies and our proposed caching architecture is compared.

2.1. Introduction to Web Services

2.1.1. General Definition and Characteristics of Web Services

Web services are programmable collections of operations that are accessible over a network. Communication with web services is made through standardized XML messaging on a platform and implementation independent manner [2]. It is not necessary to worry about the implementation details of web services when using them. Web services support true service oriented architecture and they have open, Internet-oriented standards based interfaces.

Web services has several characteristics. Web services need not to be deployed on World Wide Web necessarily; they also can operate on Local Area Networks, on the same

operating system of clients, or on shared memory between processes that reside on the same computer [3].

Another important characteristic is that a web service's implementation and deployment platform details are not specific to a particular program. A web service has a declared API and some kind of invocation mechanism (network protocol, or data encoding schemes, and so on), so all clients use this web service through those declared API and invocation mechanism [3].

Web services are XML based. Web service communication, service description and discovery are based on XML. Since XML is a universal language, it eliminates the problems of operating system, programming language, networking or platform binding of protocols. Thus, it becomes very easy to provide a real interoperable system with XML [4].

Web services are loosely-coupled. In other words, the consumer of the web service is not tied to that web service directly. The web service interface can change over time but this does not affect the consuming ability of clients to a web service. Thus, it can be said that adaptation of changes between web services and clients is done easily with loose coupling [5].

Web services are able to provide both synchronous and asychnronous operations. Synchronous operations require the client to bind to a web service and block until it gets a response from web service before continuing further operations. Asynchronous communication allows client to get information later, so client can continue on further processing after web service invocation. Thus, operations for which performance plays an important role needs asynchronous processing and this feature is avaliable in web services [5].

Web services support both RPCs and document exchange. In the case of RPC, clients are able to invoke procedures defined in web services. In the case of document exchange, communication is performed via XML formatted documents. Thus, it is possible to provide communication by exchanging simple or complex XML documents via web services [5].

2.1.2. Web Services vs. Other Distributed Technologies

Before web services concept existed, there had been several distributed computing technologies about information retrieval and data exchange communication. Those technologies provided the birth and growth of web services, so they are called the building blocks of web services. The most popular and used distributed object models that have been are Microsoft's Distributed Component Object Model (**DCOM**), Object Management Group (**OMG**)'s Common Object Request Broker Architecture (**CORBA**) and JavaSoft's Java/Remote Method Invocation (**RMI**).

Despite robustness and wide use, Distributed Object Computing had many obstacles in terms of scalability, interoperability, coupling and platform independence. These obstacles gave rise to a more independent, scalable and popular technology based simply on XML messaging between disparate clients and servers: *Web services*.

Let's look at the previous Distributed Computing technologies briefly. DCOM [6] was Microsoft COM's technology to work on a distributed network. However, DCOM was not an industrial standard so it could not spread over platforms beyond Windows [7]. Moreover, it was hard to manage the security aspects of DCOM even with two networks which use Windows [8]. Unlike DCOM, CORBA was not a standard for a particular company. It was a standard of OMG's architecture and it was suitable to work on many areas of use. However, it was a very expensive technology and it was very hard to use, so it could not spread over much [9]. Java's RMI was a very popular technology to solve the integration problems of distributed computing since it was suitable to write applications in one language and run them in different platforms [10]. However, it could not manage to solve the problems of specific applications directly [8].

Distributed Object Computing mainly relies on RPC communication. In RPC, two or more processes can establish communication by one process making calls to a method in another process, and another process responding to the first process by returning the computed result to it. This mechanism is easy to understand for programmer, provides synchronization between processes and hides the complexity of the distributed system from user. However, it has certain disadvantages. For example, it provides tight coupling

between client and server, so any failure on either side affects the other side. One site can be blocked in case of a failure or latency of another side. On the other hand, web service infrastructure also provides messaging or document-centric model of distributed computing. Document-centric messaging provides a simpler and more flexible interface [3]. Moreover, document-centric messaging is suitable for loosely-coupled systems, so when new information is added to a web service, the adaptation to this change is done in more automatic way. In this way, loosely coupled business operations are able to interoperate in an automated fashion by simply exchanging messages between each other.

Distributed Object Computing approaches lacked on data encoding. They were capable of dealing with simple data, but it was not possible to map complex data types in applications. The main reason for this was the binary encoding format of data in those approaches. However, web services solve this problem with XML. XML is text-based and has a hierarchical structure. These properties make representation of complex data structures easier than the binary data format used in COM and CORBA [3]. Moreover, validation in XML and XML Schema concepts enable us to describe data types and perform validation on documents through standardized interfaces.

Web services are able to perform transformation of data between both applications and business documents, such that they provide application integration of e-business in a more efficient way. Web services also work on open Internet standards such as HTTP and SMTP, so they can exploit the features of those standards [3].

The other benefit of web services over existing technologies is perfect interoperability of SOA (Service Oriented Architecture). With this interoperability, platform-independent communication model is established. The minimum requirement for communication is the knowledge and ability of constructing and deconstructing SOAP (Simple Object Access Protocol) messages and how to send and receive HTTP transmissions [3].

Since web service communication is based on SOAP and SOAP is able to work on HTTP, it is possible for web services to pass through and operate beyond firewalls without any external adjustment.

2.1.3. Web Services and Service Oriented Architecture

Web service technology is built on Service Oriented Architecture (SOA). SOA provides programs and services to be accessed through various interfaces over a network. In this section, we will discuss the basic elements that construct a web service and SOA.

2.1.3.1. XML

XML (Extensible Mark-up Language) is the main building block of web services and Service Oriented Architecture. Web service communication is done via XML. As it will be explained in the following sections, web service protocols like SOAP, WSDL and UDDI are special XML based structures to define, publish and provide communication in web services.

XML has many aspects that make it indispensable. The most important aspect of XML is that, it is platform and language independent [11]. Thus, web services benefit from this property to provide free distributed systems. Moreover, it is a flexible language, so it is possible to generate different mark-up languages using XML. Since it is a formatted language, it is easy to provide searching of any element in XML. Furthermore, XML documents are sharable, so it is possible to use a data in a format that was created in a different format [12].

XML is used in many parts of our proposed architecture since web service standards are based on XML. Furthermore, XML parser is used for our proposed architecture. There are many XML parsers in the literature. Since Java language is used for the implementation of our architecture, we needed an XML parser in Java. Java has many APIs defined to access and use XML in various types of applications, such as Java Architecture for XML Binding (JAXB) [13], Java API for XML Processing (JAXP) [14], Java API for XML Registries (JAXR) [15], Java API for XML-Based RPC (JAX-RPC) [16] and SOAP with Attachments API for Java (SAAJ) [17]. However, Java's XML DOM API is used as an XML parser for our architecture. The details of used XML parser are explained in Chapter 4.

2.1.3.2. SOAP

SOAP (Simple Object Access Protocol) is a lightweight remote method invocation protocol for the exchange of structured data in distributed computing. SOAP is based on XML and HTTP, which makes it a programming language and platform independent protocol for remote method invocation over the Internet and through firewalls [18].

Each SOAP message is composed of an envelope. The SOAP envelope consists of two parts: SOAP Header, which provides optional attributes about supply directive and control information, transactions and security, and SOAP Body, which is the XML payload of SOAP message that is being sent. [19].

SOAP uses many transport protocols, but HTTP is the most popular protocol in SOAP communication. SOAP envelopes are generally transmitted using HTTP GET and POST messages, and the response SOAP messages are also encoded in HTTP.

In our study, Apache's SOAP implementation is extensively used to invoke several web services that reside in a servlet engine. In our study, we expolited Apache's SOAP implementation's RPCs (Remote Procedure Call) to invoke web services over HTTP.

2.1.3.3. WSDL

WSDL is an XML based language that describes how to interact with a web service. Basically, it defines the message formats and protocol bindings necessary to interact with web services.

WSDL is often used with SOAP and XML Schema to provide web services over the internet. A client program can connect to a web service and read the WSDL of corresponding web service to determine available functions and data structures of such service. It is possible to define any data type in WSDL using XML Schema. After getting desired information from WSDL file, the client is able to call corresponding web service using SOAP [20].

A typical WSDL file contains details about the binding protocol to corresponding web service, host and port number of such service, formats of input and output messages, exceptions that can be thrown and operations that can be performed in corresponding web service.

WSDL is used extensively in this study. Each web service in our architecture has a WSDL file and it defines data structures and binding information of corresponding web services. Thus, once the WSDL file is read by client process, it is possible to communicate with corresponding web services through SOAP.

2.1.3.4. UDDI

UDDI is the registry that is used to record web services and web service descriptions. Web services are registered (published) in UDDI registries, so requesters can find corresponding web services and bind to them [21].

Type Model (tModel) is a mechanism in UDDI to describe and exchange information about a web service, such as service description, or reference to a WSDL file. It is possible to search a specific service using tModel. A business can have several tModels and each tModel can represent a specific web service in order to describe corresponding service.

UDDI is not used in our study since we concentrated on optimizing performance and fault tolerance of web services in which binding information is known.

2.1.4. Web Services Composition

In this section, web service composition is explained. First, the need to compose multiple web services is discussed. After that, two general approaches in web service composition is defined. Then, two most popular web service composition languages (WSCI and BPEL4WS) are explained.

2.1.4.1. The Need to Compose Web Services

As it is mentioned in Section 2.1.2 and Section 2.1.3, web services define a true SOA to provide the interoperation of system in a distributed manner. The interoperability of SOAP, XML, WSDL and UDDI enabled the development of a language and platform independent communication system. Howewer, these protocols enable the development of simple web services. In business world, several web services need to be composed together to run a business operation. B2B (Business to Business) and EAI (Enterprise Application Integration) applications need to use the composition of different web services in business environments. Thus, if the implementation of a business operation involves the invocation of several web services, it is necessary to benefit from web services composition technology to combine the functionality of several web services [22].

Composition of web services provides abstraction in complex applications since it aggregates components of such applications in a progressive manner.

In today's business world, it is very important for business processes to quickly adapt to customer needs and market conditions. This can only be achieved by incorporating new customers, parties, or suppliers in business processes. A single standard is needed to manage both EAI and B2B interactions involving web services. Thus, web services composition aims to provide an open, standards based approach to connect various web services together to create higher level business processes [23]. In this way, business processes can include the functionality of several web services to meet the requiremenets of customers.

In this thesis, proposed framework is completely based on web service composition. Thus, there are several business processes that consist of one or more web services and those web services are interacted with each other to perform a business task.

2.1.4.2. Web Services Composition Standards

There are two main standards for composing multiple web services in business operations. These standards are ***Orchestration*** and ***Choreography***. In this section, these standards are explained by pointing out their differences. Then, we will explain two widely used web service composition languages: WSCI and BPEL4WS.

Orchestration and choreography are two methodologies to perform web service composition. Altough they both provide web service composition, they operate in different logic.

In orchestration, there is a central process that takes control of both internal and external web services and coordinates the executions of web services in the process. The participating web services are independent of each other, so none of the web services knows which business process it is part of. Moreover, none of the web services know which other web services are involved in the process. Only the central process knows these information. Thus, orchestration aims to describe how web services can interact at the message level, and it performs the workflow and execution order of the interactions [24]. An example to web service orchestration language is BPEL4WS (Business Process Execution Language for Web Services).

On the other hand, choreography is a more collaborative approach that allows each web service to know when to execute its corresponding operations and whom it will interact with during the composition process. Thus, choreography is associated with exchange of messages that occur between several web services [24]. An example to web service choreography language is WSCI (Web Services Choreography Interface).

Orchestration differs from choreography in that it is a more centralized approach that has one central party to control the process flow between multiple web services. On the other hand, choreography is more collaborative approach which controls exchange of messages of multiple parties, and there is no centralized party in the conversation [24] [25]. Figure 2.1 shows the graphical representation of orchestration and choreography.

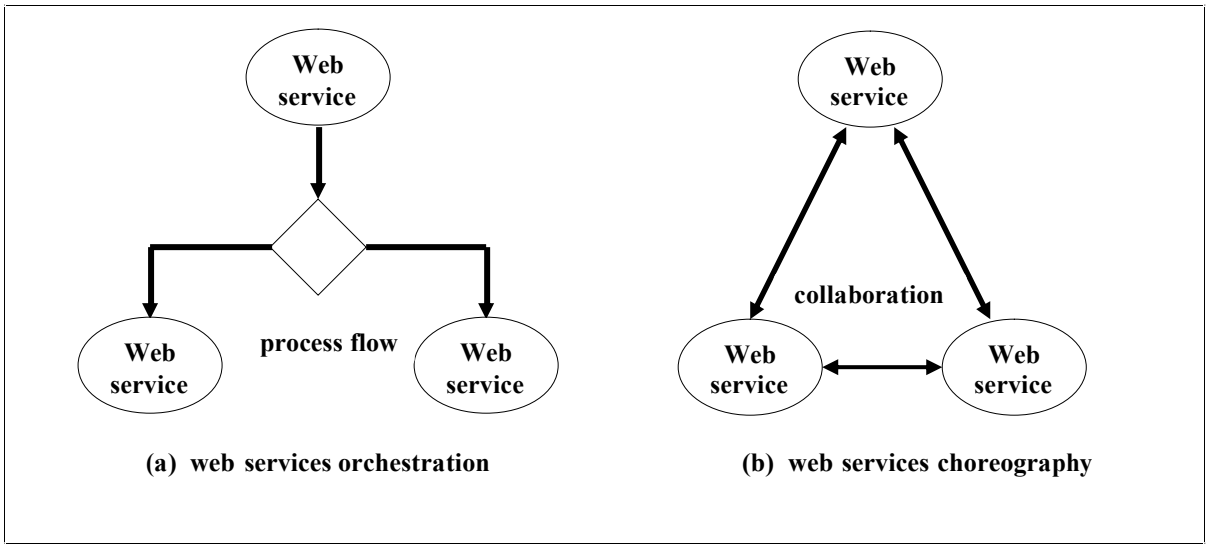

Figure 2.1. Orchestration and Choreography [24]

The Web Service Choreography Interface (WSCI), as stated in [26], is an XML based interface description language that represents the flow of messages between several web services in composed interactions. It aims to define the behaviour of several web services in terms of dependencies by using messages exchanged, correlation, sequencing rules, exception handling and transactions [26]. An important aspect of WSCI is that, it only defines the behavior between different web services, so it does not define an executable business process [24]. Thus, there are several interfaces for each party of WSCI and there is no single process to control the whole interaction [24] [25]. WSCI aims to construct a general choreography interface that provides interoperability between different layers. Thus, WSCI is designed to act as a gateway between different standard layers [27].

BPEL is an XML-based programming language that is primarily used to define business processes. Those business processes are generally cooperated with different web services and BPEL engine provides the cooperation between various business processes and web services. It supports web services technology stack, including SOAP, WSDL, UDDI, WS-Reliable Messaging, WS-Addressing, WS-Coordination and WS-Transaction [28].

BPEL is the composition of IBM's graph-based Web Services Flow Language (WSFL) and Microsoft's block structured XLANG. BPEL4WS supports modeling two types of processes: *executable* and *abstract* processes. An abstract process is a business protocol

specifying the message exchange behavior between different parties without revealing the implementation details of any of them. An executable process specifies the interaction and message exchanges between partners by maintaining information such as the execution order, fault and exception handling mechanisms of interaction [3].

Although it is a new language, BPEL has become the most popular web service composition language in business world since it allows us both to define abstract processes and write executable specifications of processes [29]. Moreover, BPEL has rich tools on which business processes can be executed (by BPEL servers) and developed (by BPEL designers) [30].

BPEL can be used within and between companies. Within companies, BPEL is used to provide EAI in a standard form and extend the integration with existing systems. Between enterprises, BPEL is used to enable easier and more effective integration among different business partners. Thus, BPEL is the key technology in environments where several web services are used in a cooperative manner [30].

With BPEL, it is possible to specify the exact order of web service invocations in a business process. Web service invocations can be either sequentially or parallel. Interaction between different web services can be provided by conditional behavior of BPEL such that, a web service invocation can depend on the result returned from a previous web service's invocation. BPEL also supports variable declaration, looping and exception handling. Thus, it is possible to define a business process in an algorithmic manner in BPEL [30] [31].

A BPEL process looks like another web service that is composed of existing services for its client's perspective. BPEL processes communicate with other processes and web services through *partner links*. Partner links show how to make a bond between a BPEL process and an existing web service [32]. BPEL processes interact with external processes and web services by invoking their corresponding methods that reside in their *port types*. Port types are defined in WSDL document of web services. The schematic view of a BPEL process is shown in Figure 2.2 [30]. As it can be observed from Figure 2.2, the example BPEL process establishes communication with external parties through partner links. So,

each web service interaction needs a partner link to be communicated. Moreover, the corresponding methods of external partners of the process are invoked through their port type structures. So, each BPEL process need to use one or more port types offered by web services to invoke external methods and get corresponding result from web services. Thus, it can be said that partner links are like bridges between BPEL processes and external parties, and port types are like doors of external parties that BPEL processes use for business operations. The order of web service invocations can also be seen in Figure 2.2.

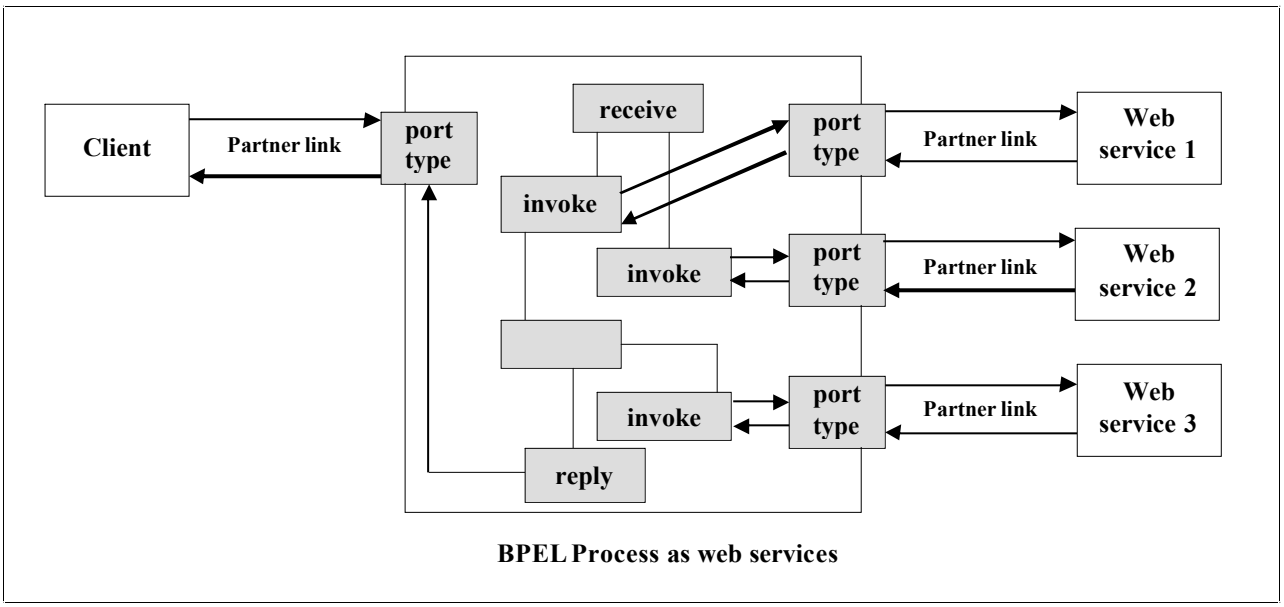

Figure 2.2. A Schematic view of a BPEL process [30]

With BPEL, it is possible to interact with external processes and web services. This is maintained by Web Services Invocation Framework (WSIF). WSIF is a simple Java API for invoking web services. It provides interaction of web services through their WSDL descriptions. Thus, it is possible to provide integration among web services without worrying their implementation details. WSIF is an Apache technology and was originally developed by IBM alphaWorks as a part of its Web Services Toolkit [33]. Thus, WSDL is the core of the integration framework for accessing various web services using many platforms. The only condition is that the corresponding service should be described using WSDL. Since WSDL includes the binding information to a web service, it can include the provider to show how to bind the corresponding service using WSIF. WSIF defines and comes packaged with various providers for various types of languages and protocols [34].

To explain the functionality of BPEL, a simple “Hello World” BPEL process example from Oracle web site’s BPEL designer tutorial [20] is explained. The example shows the basic features of BPEL process, as well as its interaction with a web service. The process receives a request from the client that executes it. Then, it invokes a simple “Hello World” web service. After that, it receives the result returned from web service and returns it back to the client that executed the process. Figure 2.3 depicts the example.

As it can be observed from Figure 2.3, the name of the process and its corresponding namespaces are defined in <*process*> tag. The namespaces specify information such as XML schemas and namespaces of external parties. Two partner links are defined that BPEL process uses to interact with external services or processes. The partner link *partnerLinkA* is for the client that executes the BPEL process, and the partner link *HelloWSPartnerLink* is for the “Hello World” web service. After defining partner links, two variables are defined that are accessible throughout the process. The variable *inputVar* is used as parameter to be sent to “Hello World” web service, and the variable *outputVar* is the output that will be returned from “Hello World” web service. After defining partner links and variables, the invocation of “Hello World” web service is performed. First, the request is taken from the client that executes the process by <*receive*> activity. As it can be observed, the client’s partner link (*partnerLinkA*) and port type (*tns:MyPortType*) are used when receiving the request from client. After the BPEL process receives the request from client, it invokes the “Hello World” web service with the <*invoke*> statement. The BPEL process uses “Hello World” web service’s partner link (*HelloWSPartnerLink*) to establish communication with it. Moreover, the process invokes the operation defined in “Hello World” web service’s port type (*partnerNS:MyPortType*) with input parameter *inputVar* and gets the desired result with output parameter *outputVar*. The web service operation is *HelloOperation*. This means that corresponding web service’s operation named *HelloOperation* is invoked in the when the <*invoke*> statement is executed. Partner links, port types and operations are defined on WSDL structure of “Hello World” web service. Upon getting the desired result from “Hello World” web service, the response is returned back to client that invokes the BPEL process. Interaction of BPEL process between its client and “Hello World” web service is performed in <*sequence*> activity. This specifies that the operations are performed in sequence.

```
<?xml version="1.0" encoding="UTF-8"?>
<process name="HelloBPELProcess"
          targetNamespace="http://www.mycomp.org/HelloWorld"
          xmlns:tns="http://www.mycomp.org/HelloWorld"
          xmlns:partnerNS="http://www.mycomp.org/OrchestrationProject1"
          xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/">
  <!-- Define the partnerLinks -->
  <partnerLinks>
    <!-- PartnerLink specifying the relation with the client invoking this BPEL process. -->
    <partnerLink name="partnerLinkA"
                 partnerLinkType="tns:ClientPartnerLinkType"
                 myRole="ServiceProviderForClient"/>
    <!-- PartnerLink specifying the relation between this BPEL process and Hello Web Service. -->
    <partnerLink name="HelloWSPartnerLink"
                 partnerLinkType="partnerNS:HelloWSPartnerLinkType"
                 partnerRole="HelloServiceProvider"/>
  </partnerLinks>
  <!-- Define variables used in this BPEL process -->
  <variables>
    <variable name="inputVar" messageType="partnerNS:requestMessage"/>
    <variable name="outputVar" messageType="partnerNS:responseMessage"/>
  </variables>
  <!-- The logic for the BPEL process starts here -->
  <sequence>
    <!-- Receive the request. -->
    <receive  name="Start"
              partnerLink="partnerLinkA"
              portType="tns:MyPortType"
              operation="operationA"
              variable="inputVar"
              createInstance="yes">
    </receive>
    <!-- Invoke the Hello Web Service partner. -->
    <invoke  name="InvokePartner"
             partnerLink="HelloWSPartnerLink"
             portType="partnerNS:MyPortType"
             operation="HelloOperation"
             inputVariable="inputVar"
             outputVariable="outputVar"/>
    <!-- Return the response back to the client. -->
    <reply  name="End"
            partnerLink="partnerLinkA"
            portType="tns:MyPortType"
            operation="operationA"
            variable="outputVar"/>
  </sequence>
</process>
```

Figure 2.3. Example “Hello World” BPEL process [20]

2.1.4.3. Web Service Orchestration Engines

In order to execute BPEL processes, an orchestration engine is required. For our architecture, we needed an orchestration engine that deploys BPEL processes in its server

and executes them. There are many BPEL orchestration servers in the literature. These orchestration servers operate either in J2EE and .NET platforms. For example, ActiveBPEL is an open source BPEL orchestration engine implemented in Java. It reads BPEL process definitions (and other inputs such as WSDL files) and creates representations of BPEL processes. The engine takes care of persistence, queues, alarms, and many other execution details [35]. IBM's WebSphere Process Server is another BPEL engine that runs on top of Java EE Platform. It can operate in many platforms such as Linux and z/OS [36]. IBM AlphaWorks BPWS4J is IBM's another BPEL engine implemented in J2EE platform to execute business processes written in BPEL [37]. On the other hand, there are BPEL engines that run on .NET platform such as Microsoft BizTalk Server. Microsoft BizTalk Server enables developers to transfer BPEL documents into a BizTalk orchestration server or vice versa [38]. Moreover, OpenStorm Service Orchestrator [39] is the most widely used orchestration server that is used both in J2EE and .NET platforms.

Oracle BPEL Process Manager is used for deployment and execution of BPEL processes for our proposed architecture. Oracle BPEL Process Manager provides a user-friendly and reliable solution for designing, deploying, and managing BPEL business processes [40] [41]. Oracle BPEL Process Manager executes standard BPEL processes. Moreover, it enables long-running process flows to be maintained in a database, so that these processes can be stored in case of a failure. Oracle's BPEL Process Manager is an open standard that reduces the cost and complexity of deploying and managing business processes by providing audit trails, process history and reporting information. It also provides a flexible binding framework to orchestrate web services, Java/J2EE components, portals, JCA interfaces and JMS (Java Message Service) destinations [40] [42]. Oracle's BPEL Process Manager supports both synchronous and asynchronous messaging and advanced exception management. It is possible to use WSIF (Web Services Invocation Framework) for invoking external Web services with Oracle's BPEL Process Manager. For example, WSIF can be used with Oracle BPEL Process Manager to invoke an EJB (Enterprise Java Bean) or an external web service that returns the social security number for a customer. Thus, external web services or EJBs can be treated as any other service that has a WSDL interface [40].

Oracle's JDeveloper is used to construct example BPEL documents which serve as the input of our architecture. Oracle's JDeveloper is a free IDE used to develop Java and web services [43]. Moreover, it offers rich visual tools to construct BPEL documents. It is also possible to debug, optimize, and deploy Java applications and web services via JDeveloper.

2.2. Web Service Caching

2.2.1. The Importance of Web Service Caching

As it is mentioned in Section 2.1, web service protocols became a key component for implementing distributed applications running on different platforms. Furthermore, composing web services provide business applications to be linked together across enterprise networks. However, like many technologies, web services also suffer from performance and availability problems.

Brian Goodman [44] mentions about three factors that affect the performance and availability of web services. The first factor is the time for a client to make a request to the web service through a distributed network. A web service must serve many clients that make a request to it. The second factor is the time it takes to handle the message. As it is known, web service protocols are based on XML. Web service communication requires parsing of XML documents. XML parsing and schema validation are expensive operations in terms of time and memory. Thus, using XML as a high-level data format creates overheads in terms of application performance [45]. The third factor is the time it takes web service itself to be executed. Since business applications require involvement of several web services, a web service itself may need to call other web services.

Jussi Myllymaki et al. [45] stated that, despite platform and implementation independence of web services, they suffer from performance problems. The performance problems are mainly due to latencies in wide area network connections. Thus, long round-trip propagation delays between client and server causes high latencies for information access in distributed applications [46]. Moreover, the system availability declines as the

layers and elements (such as network routers, gateways, application servers etc.) grow in a network.

As a result, delays in wide area connections and the use of XML as a high-level data format causes performance and availability problems in web services. Caching is an important and efficient mechanism to solve these problems since it prevents full interaction of clients and web services every time information is needed from a web service. Thus, caching strategy provides a universal mechanism for improving performance in web services. Thus, in this thesis, we aimed to construct a generic caching mechanism that can be used in many web services to solve performance and availability problems.

2.2.2. Important Factors in Web Service Caching

Web service caching provides important performance improvements and solves availability problems and the problems of load in servers. However, many factors should be considered when providing caching in web services. This section mentions which factors must be taken into account for an efficient cache mechanism.

Douglas B. Terry [47] mentions that, an efficient caching mechanism should be transparently deployable and generally applicable. Transparent deployment means that the solution must not change the implementation of parts of client, server or the communication protocol between them. Moreover, since the growth of web services is so fast, the solution should not be based on a specific web service. Thus, the solution should be scalable and general enough to be applied to all web services. In our study, web services do not need to be adjusted to our proposed architecture as long as their binding information is known. Thus, our caching mechanism is a generic implementation that is applicable to all web services whose binding information is known.

Matt Powell [48] explains the important factors that should be taken into consideration when building a caching mechanism in web services. It is discussed that dynamism of web services plays an important role in caching. For instance, if the data returned from a web service is always different, it is not helpful to apply caching. However, just because data is dynamic, it does not mean that caching should not be applied to a web service. If even a

portion of the response is relatively static, caching can be used to increase the web server's performance. Our proposed solution maintains the balance of caching and web service invocation according to the dynamism of web services.

In many situations, web services deal with user-specific data. This feature decreases the usefulness of caching, but caching still can provide advantages in these situations. For example, consider a web service that has a number of users. The performance can be improved if the caching is performed for each user, particularly if the user calls the web service multiple times. Thus, we can avoid the web service call every time the corresponding data is needed by multiple users.

Caching operation is not limited to simply caching of responses. It is possible to cache any sort of application data or resources in a web service response. The cached data can be stored in a dataset, and it can be used in further call operations of web services.

Caching can be very useful if it is possible to predict the future operations and resources in a web service interaction. If a web service includes two operations and it is more likely that the second operation will follow the first one, web service can proactively prepare the execution steps of the second operation once the first operation is performed, so once the first operation is performed and second operation is called, the second operation is executed faster.

The expiry duration of web service data is also very important when designing caching strategies. Sometimes it is easy to determine when the data should be removed from the cache since a process that updates data may run at regular intervals. However, there may be situations where data in a web service needs to be updated at relatively random intervals. In both cases, the key is to find out the optimal time interval that the cache will be updated. Finding optimal time provides a balance between harm done getting old data against the performance improvements provided by returning cached data [48].

The caching in web services is done mostly in client-side, so it is important for servers to notify clients when the data will expire. Thus, the web service may add a field to the XML response that specifically states the expiration time of the data. If other pre-built

solutions perform caching, they usually provide mechanisms for indicating an expiration time. For instance, when using HTTP caching, the expiration time of data is specified in HTTP headers, so caching can be applied on client side until the expiry period finishes.

2.2.3. Existing Work In Web Service Caching

There has been an extensive research done about building caching mechanisms on web services. In this section, previous work done on web service caching is discussed, along with some obstacles to applying caching techniques in web services.

HTTP caching is one of the most popular caching mechanisms on Internet. HTTP caching prevents the browser to download a page that was accessed before, provided that the page is not changed. It simply works as follows: The specific web source sends a *Last-Modified* (or *ETag*, which is opaque validation data rather than a date) header to client. Upon downloading, the client also caches the page. When the client requests the page again, it sends an *If-Modified-Since* (or for *ETag*, *If-None-Match*) header with the validation info it got to server. If the header is matched with server, the page is not changed, so server sends back a response code for *Not Modified* with an empty body. This tells the client to use the cached version of the page. If the client's version is out of date, the cache is not used, so the server will render the page [49], [50].

Jussi Myllymaki et al [45] and Matt Powell [48] propose using HTTP caching on web services. It is mentioned in [45] that, since SOAP requests use HTTP as the transport protocol and HTTP caching is done on the Internet, it might be possible to extend the existing HTTP caching mechanism (proxy caches and HTTP headers) with SOAP support. However, it is also mentioned that SOAP uses HTTP POST for transport, so SOAP method call parameters and message headers that contain expiration information are not visible to an HTTP proxy. Thus, the clients are not able to intelligently determine how to cache responses to HTTP POST requests [48]. Thus, despite its benefits, the applicability of using HTTP caching on web services is limited and not adequate.

Cache coherence is an important concept in applying caching not only to web services, but to web. Cache coherence concept is used to deal with integrity problems when several clients use data stored on local caches resource that is shared between several clients [51]. The benefit of cache coherence concept arises when different clients maintain caches of a common memory resource by providing integrity on shared memory resource between multiple clients. Cache coherence models are typically based on one of three mechanisms: expiration times, polling and invalidations [45].

For expiration times, different methods are studied for estimating a document's time-to-live (TTL) or time-to-refresh (TTR) values on the web [52]. By this way, it is concluded that these values can be used to determine the future TTL and TTR values of the corresponding document on web, so cache refresh time of the document can be determined by using these values.

In polling, a client contacts the server periodically, when the document is expired, or when a request is made to determine if a refresh is needed. Some protocols, such as HTTP, have tools such as *If-Modified-Since* request modifier, and those tools provide explicit support for making polling cheaper than a full retrieval of any document [45]. Thus, more frequent polling provides stale data to be refreshed and hence provides stronger coherence. In [53], combinations of client polling and server respond have been studied, and it is concluded that server can pre-emptively respond to client according to some known values, such as TTR of the client. Thus, a pre-emptive refresh of the cache is maintained in the client before the next refresh time for the client.

Invalidation is used to eliminate stale data by requiring that a server inform all clients that cached its data whenever some of the data has changed and must be refreshed or invalidated. It is stated that this method has disadvantages such as requiring the server to do extra bookkeeping and causing extra cost of client-server communication especially when server has many clients to inform [45]. In our architecture, expiration information of web services are kept in a configuration file, so extra client-server communication overhead is not needed. However, our architecture is less dynamic than invalidation technique since expiration information is not taken from server automatically.

Jussi Myllymaki [45] mentions about providing database support for web service caching. In [45], web services are called by stored procedures or user defined functions. Two tables are used in the database for caching: First table stores the most recent cache for web service responses. The second table stores the stale cache in case desired result does not exist in the first table and the corresponding web service could not be called due to a network failure or other reason. Thus, there is a close integration with the database engine, and this close integration permits efficient coupling of database data (relational and XML) with web services and appropriate query planning and execution just like with any other SQL query. Thus, this provides a persistent database cache for web services. Our architecture shows similarities with Myllymaki's approach since cached information is also kept in a database. However, Myllymaki mostly concentrated on getting data even if the corresponding web service is temporarily not available and this caused extra space overhead on client side.

In [48], ASP.NET's caching capabilities are discussed. ASP.NET has a rich cache support and this support can be used to make any job easier when providing caching capabilities for XML web services. Basically, there are three different approaches of caching in ASP.NET:

- **ASP.NET Output Caching:** It is used to cache the output of the particular page, so that this cached output can be used in future requests of the same page. The output cache provides a way to inform ASP.NET that response built for a particular page can be returned to any further requests for that page. Thus, instead of executing the ASPX script for future requests, the response from the previous request is immediately returned.
- **HTTP Response Caching:** It is used to cache the response returned from an HTTP header. Thus, ASP.NET allows user to set the HTTP headers so that client applications and HTTP proxies know how to cache the HTTP response that user sends. However, as mentioned before; since SOAP requests are POSTed on HTTP, this feature is limited.
- **Application Caching:** It provides generic caching capability for a complete ASP.NET application. It is possible to use the cache to store any sort of random data in its collection. It is also possible to set expiration criteria for the data that is stored using application caching.

In [54], a particular form of ASP.NET Output Caching is discussed. It is possible to add a tag to the page to indicate that the corresponding page will be cached. The duration and location of the cache can be specified. The following example illustrates the tag that can be added to the page as follows:

```
<%@OutputCache Duration="60" Location="Any" %>
```

As it is seen, the tag indicates that the corresponding page will be cached for 60 seconds. The location of the cache may be *Any* (by default), *Client*, *Server* or *Downstream* (a proxy before the page reaches to the client).

It is possible to cache an ASP page according to the value of a parameter. As an example, the tag:

```
<%@OutputCache Duration="60" VaryByParam="product_id" %>
```

indicates that caching is applied according to the value of a parameter named “product_id”. For the first load of page, the result is returned from server. For future calls of page; if the value of the parameter “product_id” is not changed within 60 seconds, the result is returned from cache; otherwise, the page is loaded from server.

Brian Goodman [44] mentions about methods of caching to accelerate web services. It discusses the basics for rolling a caching object. In this work, it is discussed to use a Java development environment with Apache’s Axis implementation of the SOAP protocol and a web application server, such as IBM WebSphere. The main goal is to apply caching by independent from web service. Figure 2.4 illustrates the proposed architecture.

Apache’s first SOAP implementation included a provider. The provider is used to decode the SOAP envelope into an object form and method. Then, the decoded object is instantiated and returned result is grouped in a SOAP envelope. In Axis, similar procedure is performed through handlers. Apache’s Axis provides an architecture that there are handlers that are used to define transaction flows. As it is seen from Figure 2.4, flows of request and response are defined through handlers. Each handler can affect the current preceding and following SOAP envelope. These handlers are united to constitute a larger service. The pivot handler in Figure 2.4 is used to indicate the point at which request

becomes a response. In the paper, it is mentioned that the cache can be inserted to the pivot handler.

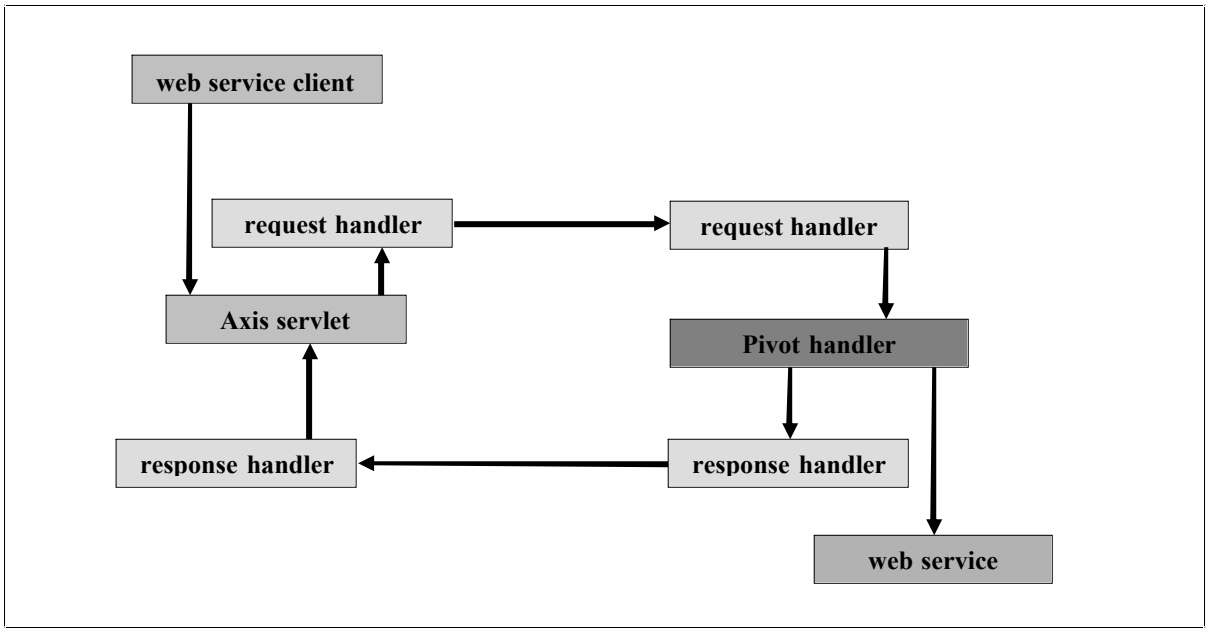

Figure 2.4. Axis flow architecture [44]

The cache object is illustrated on Figure 2.5. It is a self-pruning cache that stores entries, as needed, but prunes them at their expiration times. Depending on the implementation, the cache could afford the ability to keep popular cached objects around longer than the infrequently used ones. In order to cache a response from a web service, it is needed to generate a unique key. The unique key is used to index the cache and it is generated from the service and the arguments. In a web service invocation situation; as soon as there is a unique key, it is checked to see if there is an item in the cache. If so, the cached response is returned. In this case, the SOAP body is cached. If there is no item in the cache, the invocation is made and the result of invocation is stored. In this situation, the cache stores items for a fixed time and there is a thread that repeatedly checks the expiration times of items. If the expiration times of items are finished, the thread will prune the stale items. Like Goodman's approach, our architecture uses unique indexing key generated from web service and its arguments, so each web service call can be differentiated from each other. However, in our architecture, cached values are not pruned in their expiration times. Instead, they are invoked again so the cached information will be

proactively available for future requests. Furthermore, our architecture prunes web service results according to the frequency of their previous invocations.

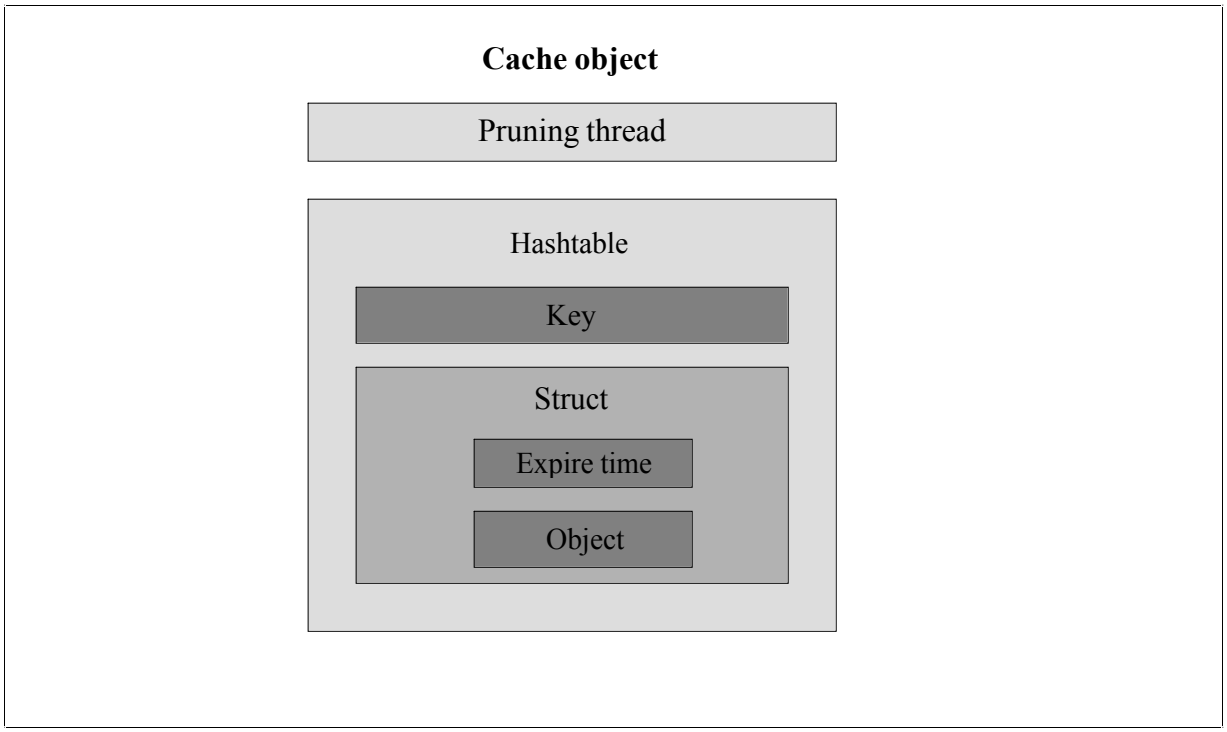

Figure 2.5. High level view of the self-pruning cache object [44]

Athena Vakali and George Pallis [55] cover Content Delivery Networks (CDNs), which is a new trend in web applications and caching concept on web. In CDNs, the content is distributed to cache servers located close to users, resulting in fast, reliable applications and web services for users. There are three parts in CDN topology:

- Surrogate servers (distributed among the world) that cache origin servers' content.
- Routers and network elements that take content requests and forward them to the optimal surrogate server.
- An accounting mechanism that keeps logs about content requests to the origin servers.

In CDN topology, client-server communication is composed of two parts: one is between the client and the surrogate server, and other is between the surrogate server and the origin server. This topology is used to maximize network bandwidth, increase

availability of servers and copy content to provide correctness of data. Figure 2.6 depicts the topology and overview of CDN.

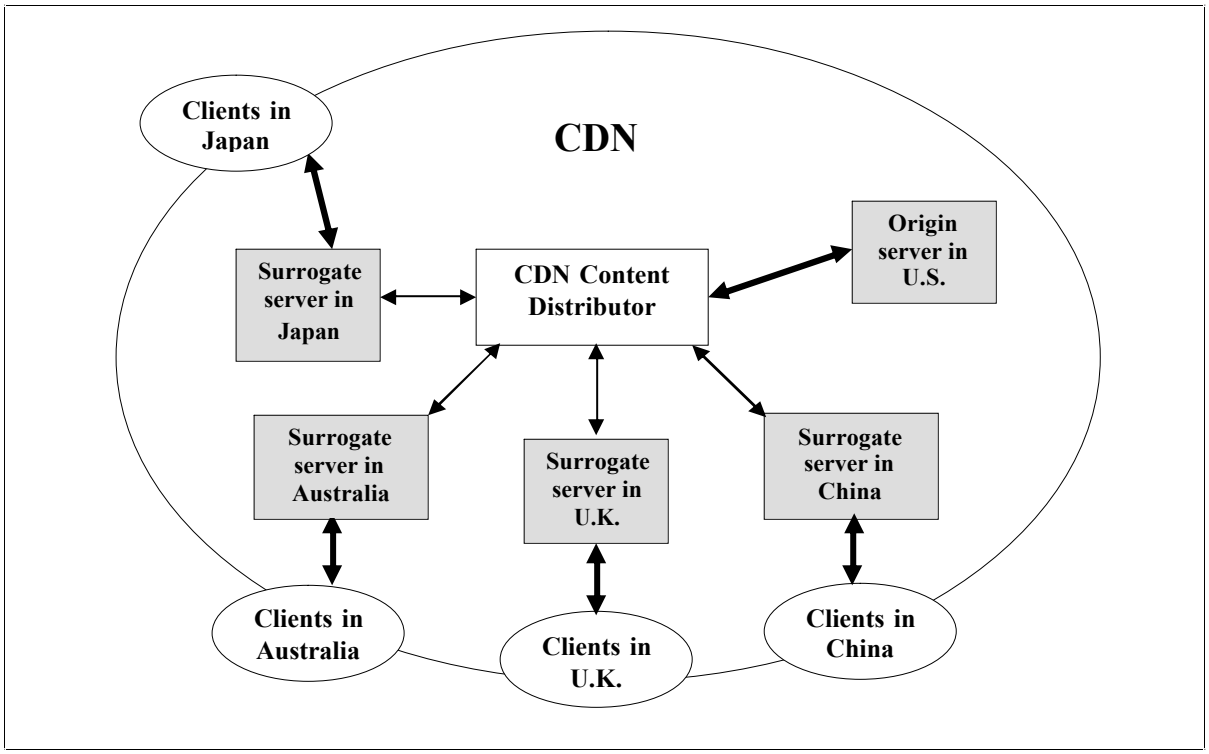

Figure 2.6. Overview of Content Delivery Network (CDN) [55]

There are 3 types of content outsourcing mechanisms in CDNs. These mechanisms are as follows:

- **Cooperative push-based:** Content is proactively pushed from the origin server to surrogate servers. Initially, the content is pre-fetched to surrogate servers and then, surrogate servers cooperate to reduce the replication and update cost. This method is similar to the method used in our caching architecture. However, in our case, web service content is pushed into database. Cooperative push-based approach is more generalized than our approach since it supports many clients to share cached information. However, it is slower than our architecture since it involves the overhead of accessing surrogate servers.
- **Uncooperative pull-based:** Client requests are forwarded to the closest surrogate server. If the requested cache is not found on the surrogate server, the request is forwarded either to a peer surrogate server of the underlying CDN or to the origin server.

- **Cooperative pull-based:** Client requests are forwarded through DNS redirection to their closest surrogate server. If the requested cache is not found, surrogate servers interact with each other to find the intended objects and add them to their caches. Thus, there is a close cooperation among surrogate servers.

Exploiting caching under CDNs is a simple idea since surrogate servers are equipped with caches. Two main ideas have been proposed in [55] about this issue. The first approach is called *Web Prefetching*, in which client's future requests are deduced and pre-fetched to cache. This approach prevents overheads related with bandwidth and reduces a significant part of latency. The second approach is called *Surrogate Server Cache Segmentation*, in which each cache of surrogate server is partitioned logically into several domains to provide a flexible memory management. It is proposed that intelligent cache segmentation on surrogate servers will increase cache hits and reduce accessing costs.

Segura-Devillechaise and Jean-Marc Menaud [56] mentioned using aspect orientation to achieve dynamic adaptation and using this dynamic adaptation to cache web services. The paper concludes that information systems should tailor a cache to their needs and experiments showed that the required adaptation is dynamic, motivated by a variability that could not be anticipated before. Therefore, it is stated that aspect orientation has the potential to support this dynamic adaptation.

Justin Reabow et. al. [57] concentrated on solving availability problems in web services during network disconnections, rather than concentrating on performance problems. In order to perform this, cache architecture is built that accesses and modifies WSDL descriptions of web services to customize a cache control on web services. Moreover, studies have been performed to provide support to mobile users for web services.

2.3. Summary

In this chapter, we tried to emphasize the importance of web services and caching conecpt in web services. In the first section, we stated the most important aspects of web

services that makes them more prefferable compared to previous distributed system technologies. Then, we desribed Service Oriented Architecture (SOA) and its elements that constitute the main parts of web services. Furthermore, we explained the importance of composition of different web service applications to perform business applications. We also explained BPEL4WS, which constitutes the input element of our study. In the second section, we mentioned about caching in web services. We tried to emphasize the importance and need of caching concept in web services. Then, we tried to decide when and where to cache in certain applications. Moreover, we gave previous work done about web service caching in different platforms and compared different approaches with our caching architecture. As a result, it is obvious that caching is an essential operation in Internet-based applications, so web services need caching extremely in large business applications.

3. OVERVIEW OF OUR ARCHITECTURE

In this chapter, our proposed framework is described as an overview. First, features of platforms and tools of our proposed architecture are emphasized. Then, the main parts of our architecture is described.

3.1. Platforms and Tools Used in Proposed Architecture

As it is mentioned in Chapter 2, web services need to interact in distributed applications efficiently. This interaction can be maintained with web service composition. In order to design an efficient and generic caching architecture for web services, we needed a web service composition language that can include involvement of several web services. In Chapter 2, WSCI and BPEL are observed as the most popular web service composition languages. On these two languages, BPEL is chosen for our proposed architecture since it is more popular and a flexible language to define business processes. Furthermore, since web services are platform independent, we needed a web service composition language that can be used on every platform. Since BPEL is completely XML based [58], it can be used on every platform. Thus, the input elements of our proposed architecture are BPEL processes that exploit web services and other BPEL processes to accomplish business operations.

It is mentioned in Chapter 2 that, BPEL documents are XML-based documents that define to run business operations. Moreover, in order to run BPEL documents, we needed an orchestration engine. There are many orchestration engines available and they are briefly mentioned in Chapter 2. On those orchestration engines, Oracle's BPEL Process Manager and Oracle's BPEL Process Server is used to deploy and execute BPEL processes in our architecture. Thus, BPEL documents are deployed on Oracle's BPEL Process Server as executeble processes. Then, those BPEL processes are executed on Oracle's BPEL Process Manager.

3.2. Proposed Architecture

In this section, we describe our proposed architecture. The first step of our architecture is to transform BPEL documents into executable processes and deploy them in Oracle's BPEL Process Server. Then, the core elements of our architecture and their interaction with deployed BPEL processes are explained. There are basically four core elements in our proposed architecture. The first element is the Proactive Caching Scheduler Pre-processor (PCS Pre-processor) which transforms BPEL documents in such a way that they are ready to interact with our caching architecture. The second element is Proactive Caching Scheduler (PCS) which fully interacts with BPEL processes and decides to cache or invoke them. The third element is the Daemon Proactive Caching Scheduler (Daemon PCS) which schedules and applies caching operation to invoked web services. The fourth element is a trace program that analyzes data collected from PCS and Daemon PCS and keeps statistical information about those data.

At first stage of our architecture, BPEL documents are simple XML documents. In order to run a business process, BPEL documents must be transformed into executables and executed in a BPEL engine. Figure 3.1 demonstrates how BPEL documents are transformed into executable processes. At the first stage, we have an XML based BPEL document that performs a specific task. This BPEL document may involve invocation of several other BPEL documents, Enterprise Java Beans (EJBs) and external web services. In order for BPEL documents to interact with our architecture, they must be modified to include elements of our architecture. So, in our architecture, each BPEL document is parsed by our PCS Pre-processor. Our PCS Pre-processor parses the corresponding documents, finds the statements where web service invocations are made, and replaces those invocations with the invocation to our PCS. Thus, instead of calling web services directly, BPEL processes will be able to call our PCS as a web service with specified parameters. PCS Pre-processor is implemented in Java and Java's XML DOM API is used to parse corresponding BPEL documents. Since BPEL documents show the features of XML documents, they could be used as natural XML documents during parsing process. After BPEL processes are parsed by our PCS, they must be compiled and deployed to a BPEL engine for execution. The Oracle BPEL Process Manager includes an Ant utility

called **obant**, which you can use to configure complex compilation and deployment scenarios. Obant is a like an envelope around standard Ant, which sets the environment and then invokes the standard Ant compiler to compile BPEL documents [59]. After corresponding BPEL processes are compiled with obant utility, they are deployed to server as *.jar* files and are ready for execution.

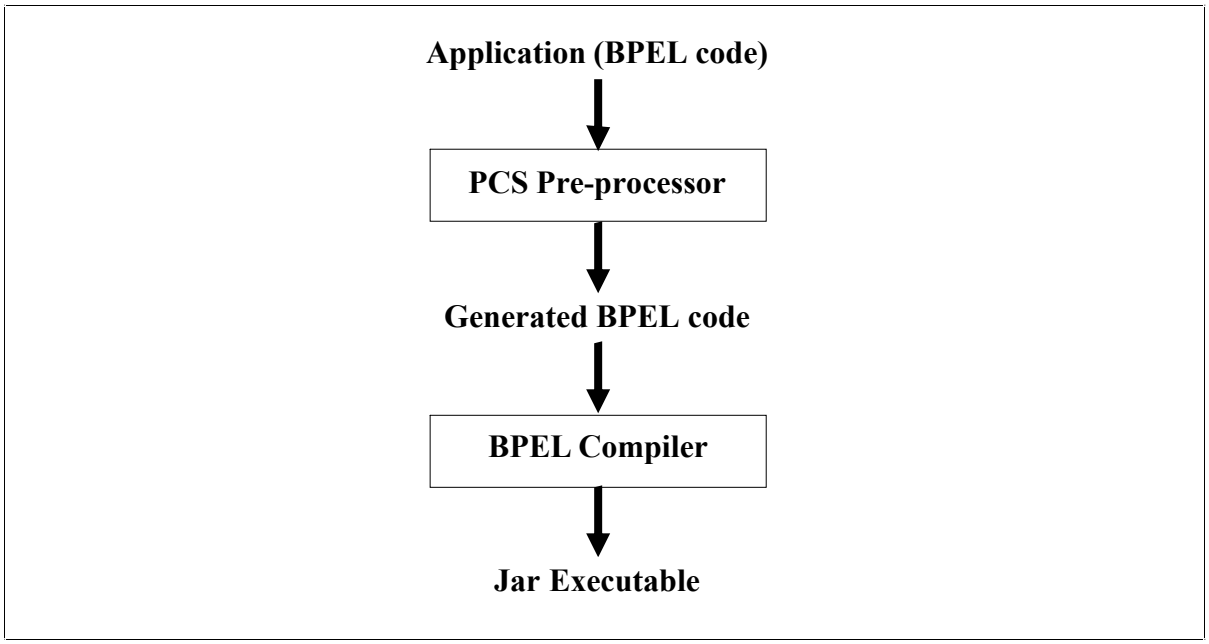

Figure 3.1. Generating executable of BPEL document

Generation of business processes from BPEL documents are depicted on Figure 3.1. The BPEL processes are interacted with our PCS and Daemon PCS as they interact with web services. Figure 3.2 demonstrates the execution of BPEL processes with our architecture. At the top part of the figure, there are BPEL processes which were transformed and deployed on Oracle BPEL Server as explained on Figure 3.1. These processes interact completely with our PCS. Our PCS is invoked in each web service invocation statement of BPEL processes. PCS serves each web service as they are invoked through BPEL processes. PCS consist of procedures. Each procedure serves a different web service. These procedure names and their parameters are defined in WSDL document of our PCS. Procedure names are also defined in WSDL documents of web services. Thus, based on specific web service, corresponding procedure is invoked in PCS with specified parameters instead of invoking web services themselves. PCS checks if corresponding web

services are invoked before. If the web service is not invoked before, it is invoked with specified parameters and its result is stored in cache. If the web service is invoked before, its last invocation time is checked. If the time is expired; the web service cache is expired, so the corresponding web service is called with specified parameters and the result of invocation is stored in cache. If the expiration time is not expired, the result is returned from cache. Thus, PCS serves each web service invocation and determines whether or not the cache has expired or not.

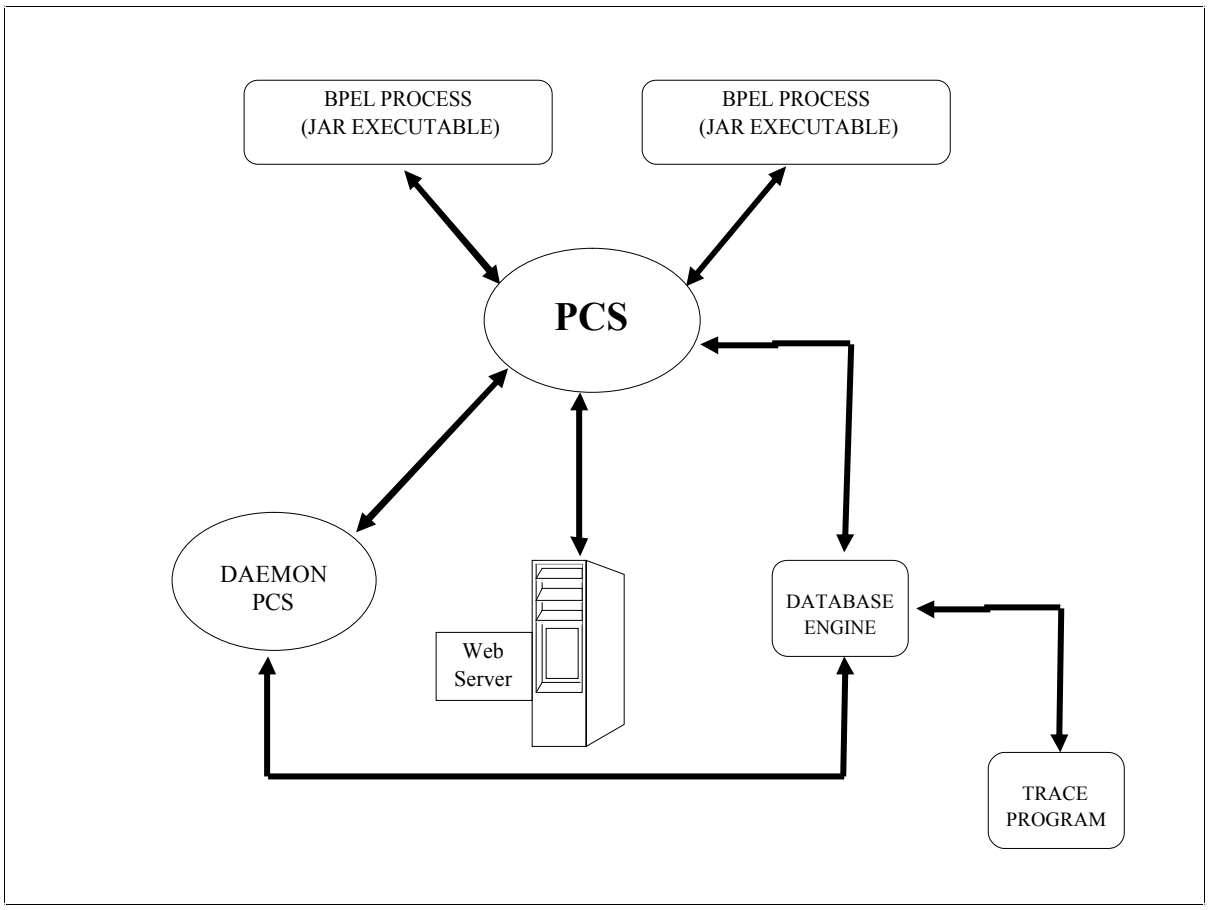

Figure 3.2. Execution of BPEL processes with PCS and Daemon PCS

Daemon PCS is used to perform a schedule of web services and apply caching with respect to the schedule generated. It reads web service expiration information from a configuration file and specifies the times where corresponding web services' cache are expired. Then, using a background thread; it invokes each web service as their expiration

time finishes. This operation is performed periodically, so it is possible to keep fresh information and use it whenever it is needed by our BPEL processes.

Our architecture use a database engine to store cached information from previous invocation of web services, so that cached information can be used as long as they are not stale.

When designing a caching mechanism for web services, we considered that caching web service invocation continuously results overhead in space. In order to solve this problem, we implemented a trace program that keeps statistical information about previously invoked web services. Our trace program utilizes this statistical information to compare the importance of web services based on their invocation frequency. When the size of the cache exceeds a specified threshold, our trace program prunes web services that has low invocation frequency. Thus, we managed to balance the process of caching and disk size optimization throughout the study.

4. PCS PRE-PROCESSOR

This chapter explains the operations of the architecture that are done at compile time. The procedure of how to transform BPEL documents in such a way that they can be able to interact with our PCS is explained. Then, the details of transformation of BPEL documents are emphasized with an example. After that, the procedure of how BPEL documents are transformed into executable processes is explained.

4.1. Transformation of BPEL Documents

BPEL documents are XML-based documents that provide to run a business task. Since a pre-processor provides more transportable code between different machine architectures [60], a pre-processing engine that provides interaction with our PCS is required. To be able to interact with our proposed architecture, we implemented a pre-processor program that traverses BPEL documents and perform some manipulations on their structure.

In order to transform BPEL documents, it is needed to use XML access models to be able to perform traversing, reading and manipulation inside the code. DOM and SAX are two strong models that are language independent and supported by numerous languages to access XML. DOM (Document Object Model) is a model that defines a standard way of accessing and manipulating XML based documents. The DOM presents an XML document as a node tree-structure. The elements, attributes and text are presented as nodes using DOM [61]. SAX (Simple API for XML) is another parser model to access XML and it handles XML information as a stream of data. The stream is unidirectional, so; unlike DOM, it is not possible to re-access previously accessed data without re-parsing. However, since DOM reads all content of the document as a node tree in memory, it is slower than SAX, especially when the corresponding XML document is large.

Our pre-processor is implemented in Java. Java's XML DOM API is used during the construction of pre-processor. The PCS pre-processor traverses each BPEL document, finds the statements where web service invocations are done and replaces those invocations with the invocation to our PCS. Thus; instead of calling external web services, our PCS

acts as a web service itself and it is executed when a web service needs to be invoked in the BPEL process. Figure 4.1 and 4.2 illustrate an example of a currency converter BPEL document.

Figure 4.1 presents the example of the BPEL document before involvement of our PCS Pre-processor. The BPEL document starts with a *<process>* statement. Namespaces of XML schemas and currency converter web service are defined with the *<process>* statement. Then, two partner links are defined: one for the client that executes the BPEL process (*ThesisService*) and other for the currency converter web service (*Currency*). After that, two variables are defined. The variable *CurrencyRequest* is the input variable of currency request web service. The variable *CurrencyResponse* is the output variable that is returned from currency converter web service.

After defining partner links and variables, the web service interaction is done in the *<sequence>* statement. First, the value of the input variable is assigned with *<assign>* tag. For our example, "US DOLLAR" is assigned as the input variable. Then, the web service is invoked. The *<invoke>* statement is used to invoke the intended web service. The input and output variables of *<invoke>* statement are *CurrencyRequest* and *CurrencyResponse* variables respectively. The partner link, port type and operation name of *<invoke>* statement are partner link, port type and operation of currency converter web service respectively. Since partner links describe how to logically tie a BPEL process to an existing web service, it can be said that the *<invoke>* statement is logically tied to currency converter web service. Since the port type and operation are port type and operation of currency converter web service, the BPEL process will use the port type and operations defined in currency converter web service. As a result, it can be said that BPEL process defined on Figure 4.1 will invoke currency converter web service on the *<invoke>* statement.

```
<?xml version="1.0" encoding="UTF-8"?>
<process   name="Thesis"
           xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
           xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
           xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
           xmlns:cur="http://oracle.com/service/currency/"
           xmlns:prc="http://oracle.com/bpel/Thesis/"
           xmlns:xsd="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://oracle.com/bpel/Thesis/">

    <partnerLinks>
       <partnerLink  myRole="ThesisService"
                     name="Thesis"
                     partnerLinkType="prc:ThesisLT"
                     partnerRole="ThesisCallBackService"/>

       <partnerLink  name="Currency"
                     partnerLinkType="cur:CurrencyLT"
                     partnerRole="CurrencyService"/>
    </partnerLinks>

    <variables>
       <!-- input for the Currency web service -->
       <variable messageType="cur:CurrencyRequestMessage"
                 name="CurrencyRequest"/>
       <!-- output from the Currency web service -->
       <variable messageType="cur:CurrencyResponseMessage"
                 name="CurrencyResponse"/>
    </variables>

    <sequence>
       <pick createInstance="yes">
          <onMessage  operation="thesisoperation"
                       partnerLink="Thesis"
                      portType="prc:ThesisPT"
                      variable="FactorialRequest">

             <sequence>
                <assign name="assign1">
                   <copy>
                      <from  expression="string('US DOLLAR')"/>
                      <to  part="myInput"
                           query="/myInput/cur:MyCurrency"
                           variable="CurrencyRequest"/>
                   </copy>
                </assign>

                <invoke  inputVariable="CurrencyRequest"
                         outputVariable="CurrencyResponse"
                         operation="currencyoperation"
                         partnerLink="Currency"
                         portType="cur:CurrencyPT"/>
             </sequence>
          </onMessage>
       </pick>
    </sequence>
</process>
```

Figure 4.1. Initial BPEL document

Figure 4.2 illustrates the transformed BPEL document after PCS Pre-processor is executed. The changed and added parts are shown in bold. At the first step of transformation process, our PCS web service's namespace is added to the namespace

declarations part of the document, so this namespace can be used in the usage of our PCS throughout the BPEL document. Then, partner link information of our PCS is added to the partner links section of the document since the BPEL process will use our PCS in web service invocations. The core part of transformation is done in the <*invoke*> statement of the BPEL process. The <*partnerlink*> element of <*invoke*> statement changes with the partner link of our PCS, because the corresponding BPEL document will be logically tied to our PCS. The <*portType*> element is also changed and it represents the port type of our PCS, because PCS's operations will be used after the invocation process. It is important to notice that the <*operation*> element is not changed since our PCS has the operation named *currencyoperation* in the port type of its WSDL document. Thus, based on the *currencyoperation* operation of *CachingSchedulerPT* port type in *CachingScheduler* partner link, our PCS web service is invoked instead of the currency converter web service. The input parameter of invocation is the *CurrencyRequest* variable. At the end of the invocation, the result is returned asynchronously to the process and it will be assigned to the *CurrencyResponse* output variable.

After PCS Pre-processor traverses the web service invocations and replaces them with the invocation to PCS, it adds an additional <*invoke*> element to the BPEL document. This <*invoke*> element invokes *ProcessCounter* operation of PCS. Notice that the input and output variables (*ProcessCounterRequest* and *ProcessCounterResponse*) for *ProcessCounter* operation are also defined at the variables section of the BPEL document. Moreover, the name of the BPEL document is assigned to the input variable (*ProcessCounterRequest*) just before invocation of *ProcessCounter* operation to be sent as an input parameter. The *ProcessCounter* operation of PCS stores the statistical information about previous web service invocations. It stores which web services are invoked at what order in which BPEL processes. Thus, it becomes possible to measure the invocation frequency of previously invoked web services. In order to exploit this statistical information, a trace program is implemented that read these data and prune less frequently invoked web service information from cache when the cache size exceeds a specified threshold. This is very important because it is not possible to keep cached information forever. With the trace program, we aimed to minimize the overheads in disk capacity. Moreover, when the cache size is too large, the time to fetch data from cache can be long

and caching would not be utilized effectively. Thus, we aimed to keep the web service invocations that are more frequently invoked. By this way, the cache size is kept balanced.

```
<?xml version="1.0" encoding="UTF-8"?>
<process   name="Thesis"
           xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
           xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
           xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
           xmlns:cur="http://oracle.com/service/currency/"
           xmlns:prc="http://oracle.com/bpel/Thesis/"
           xmlns:xsd="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://oracle.com/bpel/Thesis/"
           xmlns:cacsch="http://oracle.com/service/cachingscheduler">

    <partnerLinks>
       <partnerLink  myRole="ThesisService"
                     name="Thesis"
                     partnerLinkType="prc:ThesisLT"
                     partnerRole="ThesisCallBackService"/>

       <partnerLink  name="Currency"
                     partnerLinkType="cur:CurrencyLT"
                     partnerRole="CurrencyService"/>

       <partnerLink  name="CachingScheduler"
                     partnerLinkType="cacsch:CachingSchedulerLT"
                     partnerRole="CachingSchedulerService"/>
    </partnerLinks>

    <variables>
       <!-- input for the Currency web service -->
       <variable messageType="cur:CurrencyRequestMessage"
                 name="CurrencyRequest"/>
       <!-- output from the Currency web service -->
       <variable messageType="cur:CurrencyResponseMessage"
                 name="CurrencyResponse"/>
       <!-- input for the ProcessCounter operation of PCS -->
       <variable messageType="prcnt:ProcessCounterRequestMessage"
                 name="ProcessCounterRequest"/>
       <!-- output from the ProcessCounter  operation of PCS -->
       <variable messageType="prcnt:ProcessCounterResponseMessage"
                 name="ProcessCounterResponse"/>
    </variables>
    <sequence>
       <pick createInstance="yes">
          <onMessage   operation="thesisoperation"
                       partnerLink="Thesis"
                       portType="prc:ThesisPT"
                       variable="FactorialRequest">
             <sequence>
                <assign name="assign1">
                   <copy>
                      <from  expression="string('US DOLLAR')"/>
                      <to   part="myInput"
                            query="/myInput/cur:MyCurrency"
                            variable="CurrencyRequest"/>
                   </copy>
                </assign>
```

Figure 4.2 Transformed BPEL document

```
                <invoke   inputVariable="CurrencyRequest"
                          outputVariable="CurrencyResponse"
                          operation="currencyoperation"
                          partnerLink="CachingScheduler"
                          portType="cacsch: CachingSchedulerPT"/>

                <!—Assign the name of BPEL process to input parameter -->
                <assign name="Process_Counter_Assign">
                   <copy>
                      <from  expression="string('Thesis.bpel')"/>
                      <to  part="myInput"
                           query="/myInput/prcnt:ProcessName"
                           variable="ProcessCounterRequest"/>
                   </copy>
                </assign>

                <invoke  inputVariable="ProcessCounterRequest"
                          operation="processcounteroperation"
                          outputVariable="ProcessCounterResponse"
                          partnerLink="CachingScheduler"
                          portType="cacsch:CachingSchedulerPT"/>
             </sequence>
          </onMessage>
       </pick>
    </sequence>
</process>
```

Figure 4.2. Transformed BPEL document (continued)

4.2. Generating Executables of Transformed BPEL Documents

After the corresponding BPEL documents are transformed by PCS Pre-processor, they need to be compiled and deployed to server as executable processes. Oracle BPEL Process Manager's **obant** utility is used to compile BPEL documents and deploy them to Oracle BPEL Process Server. The obant is a wrapper around standard Ant utility, which sets the environment and then invokes the standard Ant Java task [59]. After that, corresponding BPEL processes can be executed in Oracle BPEL Process Server.

5. PCS AND DAEMON PCS

In this chapter, the core part of proposed architecture is described. First, web services Invocation Framework (WSIF) is explained that provides invocation of web services on BPEL processes. Then, usage of WSIF in our architacture is explained. After that, the logic behind PCS and its functionality over BPEL processes is stated. The proposed operations and behaviour of PCS according to web service expiration times are explained. Then, the function and necessity of Daemon PCS is explained. By this way, the effectness of our architecture when executing BPEL-based business applications is emphasized.

5.1. Proactive Caching Scheduler (PCS)

5.1.1. Usage of WSIF in BPEL Processes and PCS

As it is stated in Chapter 4, BPEL processes are parsed by PCS Pre-processor to be able to interact with our PCS and deployed to Oracle BPEL Process Server. Then, the PCS web service starts to operate when a web service invocation is occurred in BPEL processes. Invocation of PCS in BPEL is accomplished by **WSIF** (**W**eb **S**ervices **I**nvocation **F**ramework).

In our architecture, WSIF is closely integrated with WSDL structures of web services that will be invoked on BPEL processes. The only condition to use WSIF is to know the WSDL structures of web services. In our architecture, BPEL processes invoke our PCS as a web service, so by knowing WSDL structures of our PCS, WSIF is used to access the data types and binding information of our PCS. Thus, it becomes possible to invoke our PCS through WSIF.

In order to invoke our PCS through BPEL processes, we need data defined in BPEL processes to send variables to our PCS and get corresponding results back to BPEL processes. However, since BPEL documents are XML based and our PCS is Java based, we needed a mapping between XML and Java. To provide this mapping, automated Java to

XML bindings are used to convert XML Schema types to Java types automatically. For this purpose, Oracle BPEL Process Manager supports default Java-to-XML bindings through the use of *XML façades* [34] for WSIF. XML façades are a set of Java interfaces and classes which you can access and modify XML data stored in BPEL variables in an easy way using *get/set* methods. In this manner, it is not required to change XML content directly since the XML is hidden behind the façade and you can manipulate the data through regular Java interfaces. This concept is known as **XML serialization**. The idea behind XML façades is to provide support for mapping of data types between XML and other high level languages [34]. In our architecture, data types defined in WSDL structure of our PCS are mapped to their equivalent Java types by producing Java classes from XML schemas.

Most of the simple data types can be mapped to either primitive or object types from XML to Java. In addition to this, complex types can also be mapped to their corresponding Java types. In this study, many considered web services' WSDL documents have complex data types defined in their <*types*> element. For example, below fragment shows the data types used on the WSDL structure of the currency converter web service:

```
<types>
   <xs:schema  elementFormDefault="qualified"
               targetNamespace="http://oracle.com/service/currency/">

      <xs:complexType name="CurrencyInput">
         <xs:sequence>
            <xs:element name="MyCurrency" type="xs:string" />
             <xs:element name="Name_of_BPEL" type="xs:string" />
             <xs:element name="Name_of_WS" type="xs:string" />
         </xs:sequence>
      </xs:complexType>
   </xs:schema>
</types>
```

Figure 5.1 Data types used in WSDL document of currency converter web service

The XML façade for this complex XML type provides an interface and a class through which the elements (MyCurrency, Name_of_BPEL, Name_of_WS) using Java getter methods. The XML façade also allows modifying the corresponding element data using

setter methods. Thus, the XML façade for the variable in Figure 5.1 (CurrencyInput) consists of an interface (ICurrencyInput) and a class (CurrencyInput) which provides the following methods:

- getMyCurrency() and setMyCurrency()
- getName_of_BPEL() and setName_of_BPEL()
- getName_of_WS() and setName_of_WS()

As a result, variables are defined in the WSDL structure of corresponding web service and they are mapped using XML façades. Thus, we extracted currency converter web service's variables in our PCS as shown in Figure 5.2.

```
public String getCurrency (CurrencyInput myInput)
{
        String currencyType = myInput.getMyCurrency();      // get Currency from BPEL document
        String bpelName = myInput.getName_of_BPEL();        // get Name_of_BPEL from BPEL document
        String wsName = myInput.getName_of_WS();            // get Name_of_WS from BPEL document
            ...
}
```

Figure 5.2 Extracting currency converter web service parameters in PCS

After defining the mapping of variables between WSDL documents and our PCS, we need to define the WSIF binding for currency converter method in our PCS. The WSIF binding of web services are defined in the <*binding*> element of WSDL structure of PCS. The WSIF binding information presents the way to bind to an external web service by specifying web service's encoding information, XML serialized data, supported operations and methods. The WSIF definition of our PCS for currency converter web service is presented on Figure 5.3.

As it can be seen from Figure 5.3, first the type mapping from XML to Java is defined. The input parameter XML type *CurrencyInput* is mapped to the *com.oracle.service.currency.CurrencyInput* Java class. In addition, the WSDL operation *currencyoperation* is mapped to the Java method *getCurrency()* which is located in PCS.

```
<binding name="JavaBinding" type="tns:CachingSchedulerPT">
        <java:binding/>
        <format:typeMapping encoding="Java" style="Java">
         <format:typeMap typeName="cur:CurrencyInput"
                        formatType="com.oracle.service.currency.CurrencyInput"
        </format:typeMapping>
        <operation name="currencyoperation">
                <java:operation methodName="getCurrency"/>
                <input/>
                <output/>
        </operation>
</binding>
```

Figure 5.3 WSIF Binding for currency converter web service on WSDL document of PCS

After defining the Java binding, the service used in PCS is defined. The service provided in the Java class is defined in this section. For currency converter web service example, the Java class of our PCS is defined in the service definition as shown in Figure 5.4. As a result, the currency converter web service will use the *getCurrency()* method defined in *com.oracle.cachingscheduler.CachingSchedulerJava* Java class. (Figure 5.3 and Figure 5.4)

```
<service name="CachingScheduler">
    <port name="JavaPort" binding="tns:JavaBinding">
        <java:address class Name="com.oracle.cachingscheduler.CachingSchedulerJava"/>
    </port>
</service>
```

Figure 5.4 Service definition for PCS in WSDL document of PCS

5.1.2. Working Principle of PCS

In section 5.1.1, the importance and usage of WSIF when building PCS is explained. This section explains the steps that PCS takes when executing with several BPEL processes. The PCS makes decisions based on certain criteria whether to invoke a web service or return its result from cache. The schematic view of PCS is depicted in Figure

5.5. The steps performed in PCS are also explained in the following paragraphs with respect to Figure 5.5.

1. At first step, there are one or more BPEL processes deployed on Oracle BPEL Process Server. Our PCS starts executing when a web service invocation is performed from those BPEL processes. Thus, as it is explained in Chapter 4, our PCS is invoked when a web service invocation is made by BPEL processes.

2. The PCS starts executing by extracting the web service name and its corresponding parameters from the <*invoke*> statement of BPEL processes. It is important to note that, web service name and its parameters are defined in the WSDL structure of each web service as inputs and they are also initialized in BPEL process. After <*invoke*> operation is performed, these values are passed to PCS, so that they can be used in PCS. Then PCS combines web service name and its parameters, and creates an MD5 (Message Digest 5) hash of them. Thus, MD5 hash uniquely identifies the name and parameters of the corresponding web service. MD5 hash is used to check whether the corresponding web service has been called previously with the same parameters or not.

3. Moreover, there is a database which stores information about previously invoked web services. It contains previously invoked web service's name, MD5 hash (web service name and its parameter values), the result returned from the web service and input parameters provided when that web service was invoked. Thus, constructed MD5 is compared with the MD5 in the database to check if the corresponding web service was called previously with the same parameters.

4. If the web service is invoked before with the same parameters, the next operation is decided based on corresponding web service's expiration time. In our architecture, each web service's expiration time is stored in a configuration file (for example, if the value of a currency converter web service is changed on every five minutes, then the expiry time of currency converter web service is stated as 5 minutes). So, if the corresponding web service is previously invoked with the same parameters, the time of its last invocation time is taken from database and compared with current time. If the difference between current time and last invocation time exceeds the expiry time which is taken from configuration

file, that means cache of corresponding web service is expired and the value is a stale value. In this case, web service is invoked with the same parameters and its returned value is updated in database **(Step 6).** Thus, new refreshed value will be in the cache for future use.

5. If the difference of current time and last execution time does not exceed the expiration time of the corresponding web service, it means that the result of the web service does not need to be refreshed by invoking the web service, so the result can be taken from database and returned to corresponding BPEL process. In that way, calling of the web service is avoided every time a BPEL process needs it, since its result exists in the cache.

6. If the corresponding web service was not called with same parameters previously, it is called with specified parameters. In this case, our PCS calls the web service via **SOAP RPC**. However, any method can be used in this part of the study. Thus, the method of web service invocation depends on the programmer. After calling the web service, the returned result is stored into database with its properties (name of the web service, its MD5 hash, result, date obtained the result and its input parameters) for future use.

7. After PCS performs its operation based on the expiry information of corresponding web service, the result of web service is returned to the BPEL process. Thus, BPEL process can use this result in its business operations.

The PCS also stores which web services are called on which BPEL processes in an ordered manner, so that it becomes possible to generate a trace of executed web services and BPEL processes. This trace information can then be used to determine the importance values of web services and BPEL processes, so these importance values can be used to give extra priority to those web services and BPEL processes. However, trace information of previous web service calls are used to make optimizations regarding the disk capacity. Our system uses the tracing information and decides which web service record to remove when the disk capacity exceeds a specified threshold. Since it is not possible to cache forever, we are avoiding keeping unnecessary information on database. Moreover, since

cache size is kept small with necessary data, the time to return cached data will not be long.

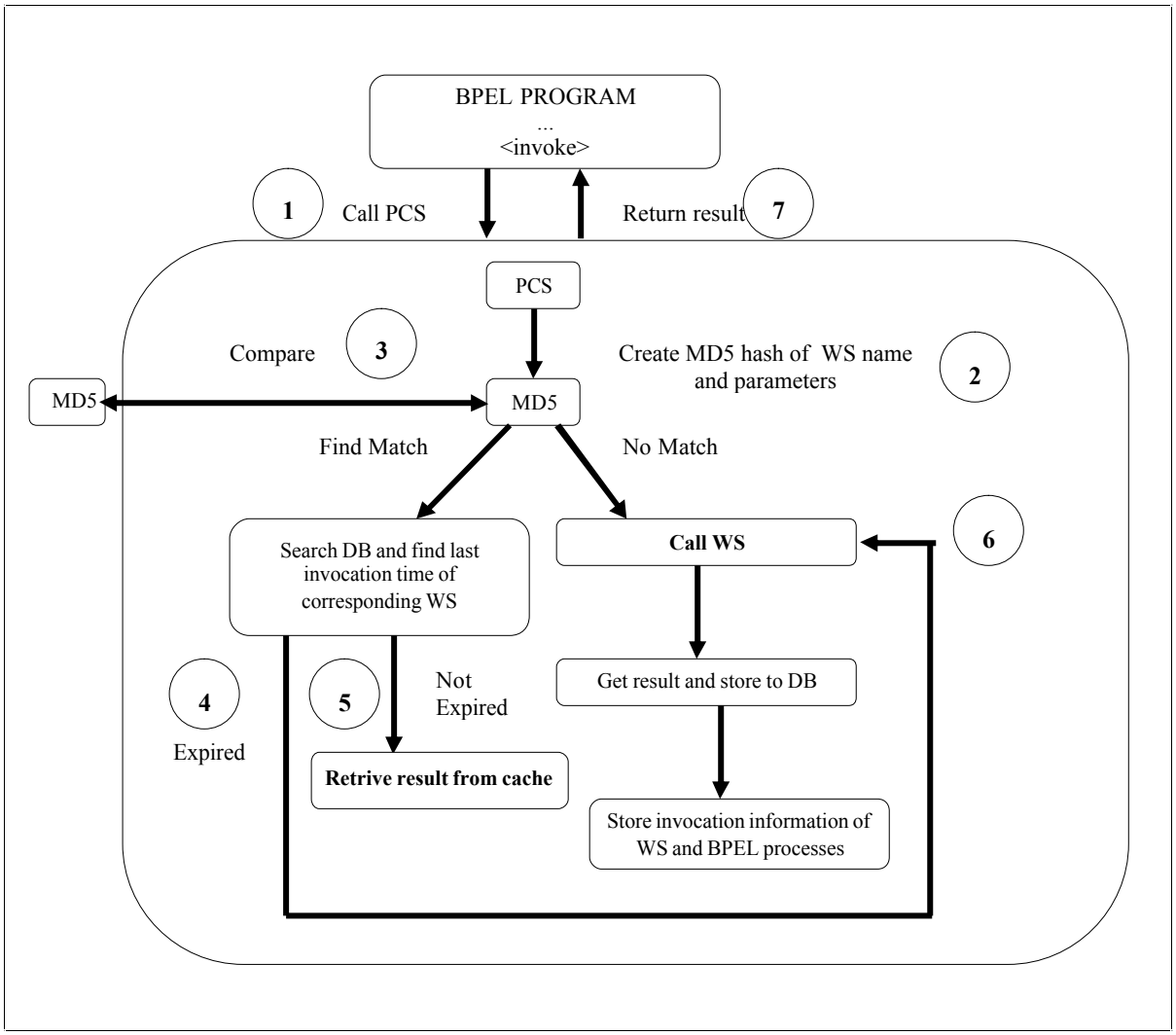

Figure 5.5 Proactive Caching Scheduler (PCS) mechanism

5.2. Daemon Proactive Caching Scheduler (Daemon PCS)

5.2.1. Working Principle of Daemon PCS

In this section, working principle and the idea behind Daemon PCS is explained. The Daemon PCS performs a schedule for all possible web services and provides proactive caching of available web services in a background manner. The schematic view of PCS is

depicted in Figure 5.6. The steps performed in Daemon PCS are also explained in the following paragraphs with respect to Figure 5.6.

1. The Daemon PCS begins execution by taking all web services' addresses and their corresponding expiration and availability information from a configuration file. Expiration information shows the time that corresponding web service's result is changed. Availability information shows the available times that corresponding web service operates. For example, currency exchange web services may not operate at weekends since stock markets don not operate at weekends. Thus, it can be said that currency exchange web services are available during midweek and they are not available at weekends. The Daemon PCS collects this information from a configuration file and performs next steps according the information of the configuration file.

2. After obtaining corresponding web services' expiration and availability information, the Daemon PCS performs a daily schedule of each web service by calculating the times that those web services will be invoked during a day. For example, if the web service that returns the temperature of a city has an expiry time of 30 minutes, this web service is put in a schedule and it is invoked 48 times (every 30 minutes) in a day.

3. Each web service is invoked based on the created schedule by Daemon PCS when its expiration time comes.

4. After getting the results of web service invocation, each web service's result value is stored in database, so in most of the time, database will hold refreshed values for web services. In this way, when many BPEL processes are executed, they can use database to get the corresponding web service result by avoiding invocation of the same web services over and over, which exposes a performance and availability problem. Thus, results from web services can be obtained faster and without adding extreme load to servers.

For the implementation of our Daemon PCS, Java's *Timer* class is used to schedule several recurring web services periodically in a day. Java's *Timer* utility is introduced in Java 2 Standard Edition, Version 1.4.2. The *java.util.Timer* and *java.util.TimerTask* classes are the composition of the *Java timer framework* and they make easy for programmers to

schedule simple tasks. Before this framework was introduced in the Java 2 SDK, Standard Edition, Version 1.3, developers had to write their own scheduler, which required to work with threads and the complexities of the *Object.wait()* method. Java's *Timer* class is a facility for threads to schedule tasks for future execution in a background thread. Tasks may be scheduled for one-time execution or for repeated execution at regular intervals [62] [63]. In other words, Java's Timer class performs the similar functionality of UNIX scheduler (**cron)** facility [63] [64]. However, the scheduling framework of Java's Timer utility is more flexible than cron. As an example, there may be scenarios such that a web service needs to be invoked at 8:00 AM on weekdays and 9:00 AM on weekends. Using *cron*, we need two **crontab** entries (0 8 * * 1,2,3,4,5 and 0 9 * * 6,7) [63]. However, Java's Timer facility provides a more elegant solution since it is easier to control such restrictions in future scheduling automatically.

Scheduling and caching with Daemon PCS is a flexible and performance friendly process since it uses a background thread to schedule web services. Thus, since there is only one thread running as a background process, it does not add much performance overhead into the system.

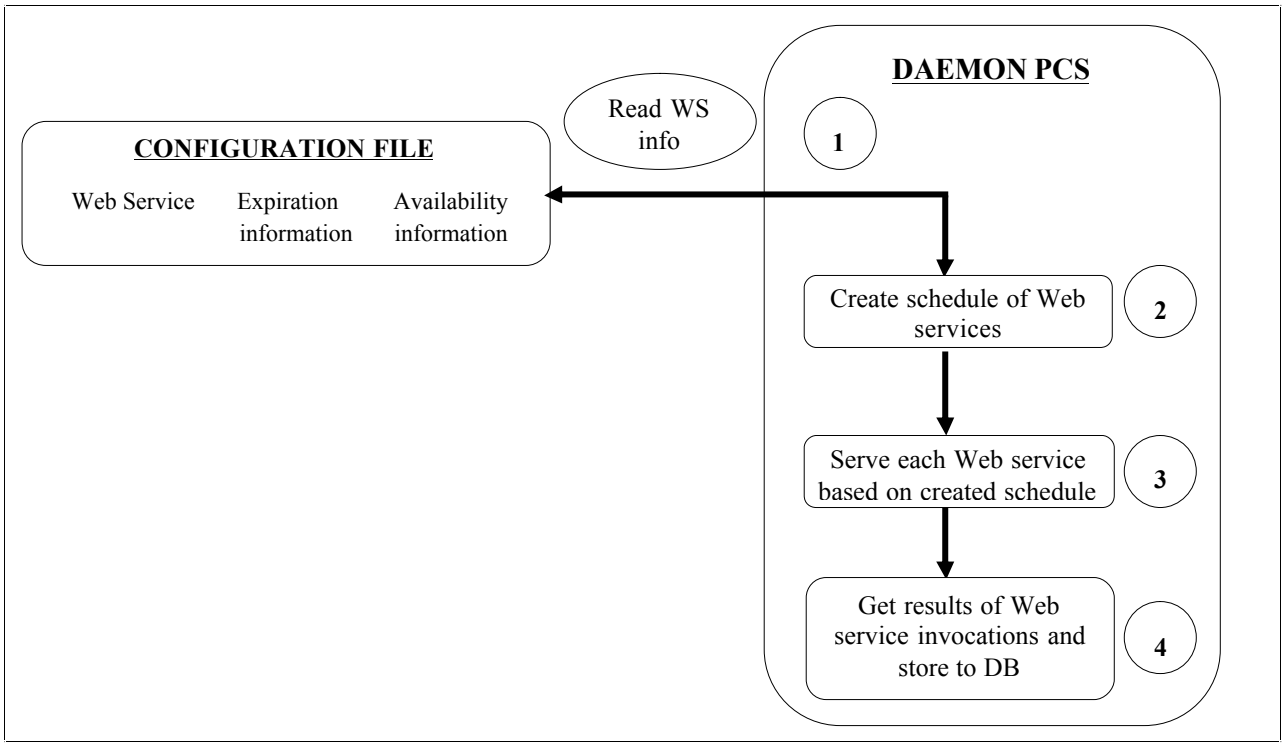

Figure 5.6 Daemon PCS mechanism

6. EXPERIMENTAL TESTS AND RESULTS

In this chapter, experimental work related to our architecture is presented. There are many web service applications that business applications and web sites use in daily life. We tried to select popular web service applications for our test environment. Our selected web service applications can be grouped as two: Applications that have expiration times such as temperature or currency exchange web services; and applications that don not have expiration times (or expiration times are very long), such as Turkish Republic Identity web service and eBay Item web service. At the end of experiments, results gathered with and without our architecture are compared to test the efficiency of our study.

In our test scenarios, we have prepared a BPEL document for each web service and invoked our PCS with the operation name of corresponding web service. Thus, our PCS understands which web service to serve based on the passed operation name from BPEL processes. Furthermore, in order to get accurate and consistent results, we tested each web service 10 times by executing each web service's BPEL process 10 times and computed the average of execution time. We left some time gap between each experimental test in order not to create extreme load in web service side. The test environment is illustrated on Table 6.1. For the software environment, Java 2 Platform SDK 5.0 and Java Runtime Environment 5.0 are used for the construction of our PCS Pre-processor, PCS and Daemon PCS. Furthermore, Oracle BPEL Process Manager 10.1.2 is used for deployment and execution of BPEL processes throughout the study.

Table 6.1 Test environment used in experimentation

Hardware and Network Environment	Software Environment
Toshiba Satellite A30-151 Notebook Pentium IV 3.06Ghz CPU, 1GB RAM Windows XP Professional Version 2002 SP2 256 Kbps. ADSL Connection	Java 2 Platform SDK 5.0 Java Runtime Environment 5.0 Oracle BPEL Process Manager 10.1.2

6.1. US Weather Forecast Web Service

Our first external web service example is US Weather Forecast web service. The US Weather Forecast web service gives the temperature and weather information of cities in the United States. The URL for US Weather Forecast web service is located in http://iwin.nws.noaa.gov/iwin/us/traveler.html web site [65]. The web service gets the name of the city as input parameter and returns name of city, the city's estimated highest/lowest temperature values and its weather forecast (sunny, windy, rainy etc.) In our test environment, our BPEL process gets the name of the city to be searched as a variable and invokes our PCS with the name of the city as the parameter. Our PCS performs invocation of web service based on the expiration time of the web site. As an example, we gave 60 minutes to the expiration time of temperatures of cities. After performing its operation (invoking the web service or getting the result from cache), PCS passes the result to BPEL process. The passed result is stored in a different variable that is declared in the BPEL process. Table 6.2 shows an example of sent input variable and gathered results from US Weather Forecast web service.

Table 6.2 US Weather Forecast Web Service input and output data

INPUT	**OUTPUT**		
Name of City	**Name of City**	**HI/LO**	**Weather**
DENVER	DENVER	69 / 35	SUNNY

Figure 6.1 shows the tests done in US Weather Forecast web service execution times with and without our arcitecture. As it can be observed, the execution with our architecture is 4-5 times faster than the execution without our architecture.

Figure 6.1 Analyses of US Weather Forecast Web Service

6.2. Turkish Republic Central Bank Currency Exchange Web Service

Our second external web service is Turkish Republic Central Bank Currency Exchange web service. Turkish Republic Central Bank web site lists the exchange values for many currencies among the world. The currency exchange information is held in http://www.tcmb.gov.tr/kurlar/today.xml web site. The web service gets the name of the currency as input and returns the exchange of corresponding currency in terms of Turkish Lira. The web service parses the URL http://www.tcmb.gov.tr/kurlar/today.xml and finds the corresponding Turkish Lira exchange for input currency. In our test environment, our BPEL process gets the name of the currency to be searched as input and invokes our PCS with the name of the currency as the parameter. Our PCS performs invocation of web service based on the expiration time of the web site. As an example, we gave 30 minutes to the expiration time of refreshness of currency exchange web site. Table 6.3 shows an example of sent input variable and gathered result from Turkish Republic Central Bank Currency Exchange web service. As it is seen from Table 6.3, it is possible to obtain forex buying, forex selling, banknote buying and banknote selling of currencies.

Table 6.3 Turkish Republic Central Bank Currency Exchange Web Service input and output data

INPUT	OUTPUT			
Currency Type	**Forex Buying**	**Forex Selling**	**Banknote Buying**	**Banknote Selling**
US DOLLAR	1.4501	1.4571	1.4491	1.4593

Figure 6.2 shows the tests done in Turkish Republic Central Bank Currency Exchange web service execution times with and without our arcitecture. As it can be observed, the execution with our architecture is 3-5 times faster than the execution without our architecture.

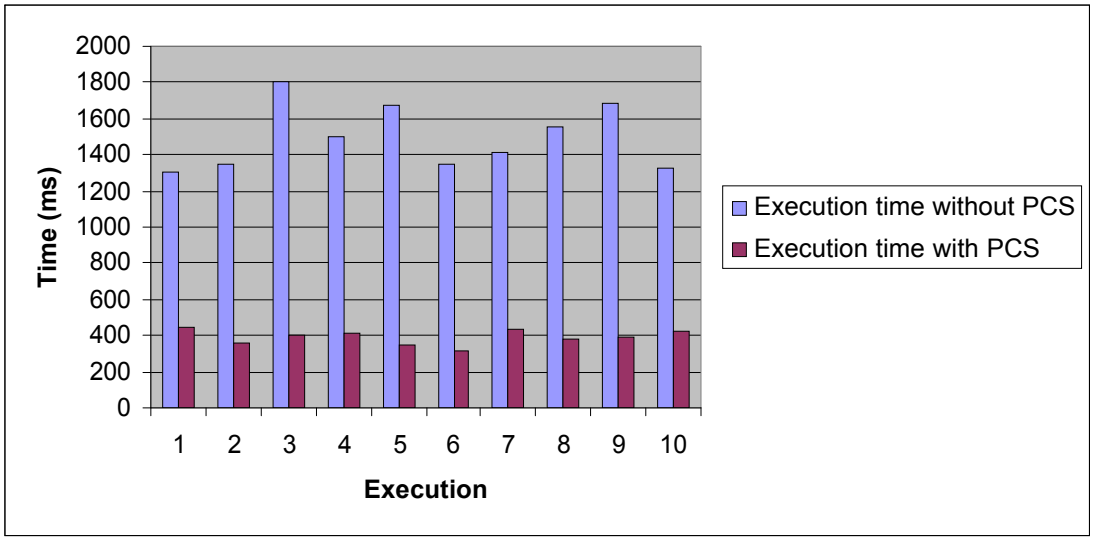

Figure 6.2 Analyses of Turkish Republic Central Bank Currency Exchange Web Service

6.3. Turkish Republic Identity Web Service

Turkish Republic identity web service is our first web service example that does not have an expiration time since the data returned from this web service are static. The web service is located in http://tckimlik.nvi.gov.tr/kpspublic.asmx web site. The web service is implemented in .NET environment, but it is possible to invoke it in any platform. By using this web service, we can get information about a person with his/her Turkish Identity Number, get Turkish Identity Number of a person with his/her personal information, as

well as cities and districts of Turkey. For our experimental test, we tried to get the personal details of a Turkish citizen by using his Turkish Identity Number. Thus, our BPEL process holds the Turkish Identity Number of a person and passes this number to our PCS. Our PCS invokes the web service with passed Turkish Identity Number information as input parameter. The web service returns the name, surname and birth date of corresponding person. Since the information in web service does not change, we do not need our Daemon PCS to proactively refresh cached information. As it is mentioned in previous chapters, our architecture removes less used information from cache. This property of our architecture is useful especially in web services that hold huge amount of data, such as Turkish Republic identity web service. In this way, we are avoided to face the disk capacity problems since it is not possible to hold all information in Turkish Republic Identity web service. Table 6.4 shows an example of sent input variable and gathered results from Turkish Republic Identity web service. We did not put real values of Turkish identity number, name and surname in Table 6.4 in order not to reveal any information about a specific person in this thesis report.

Table 6.4 Turkish Republic Identity Web Service input and output data

INPUT	**OUTPUT**		
Turkish Identity Number	**Name**	**Surname**	**Birth Date**
99999999999	Hasan	Çelik	1982

Figure 6.3 shows the tests done in Turkish Republic Identity web service execution times with and without our arcitecture. As it can be observed, the effectness of caching results in the execution of 3-5 times faster than the execution without our architecture.

Figure 6.3 Analyses of Turkish Republic Identity Web Service

6.4. E-Bay Item Information Web Service

The last external web service in our experimental work is the eBay Item Information web service. As it is known, eBay is one of the most popular web site for shopping in internet. eBay has rich API tools for accessing many of its web services. In our experimental work, the eBay Item Information web service takes a unique item id as input parameter. It searches this item id in its database and returns the name and current price of corresponding item. Thus, our example BPEL process holds the item id information and passes it to our PCS as parameter. Our PCS invokes eBay Item Information web service and passes the item id information as input parameter. In order to make invocation to eBay Item Information web service, eBay SOAP API must be used. Table 6.5 shows the input and output parameters of eBay Item Information web service.

Table 6.5 eBay Item Information Web Service input and output data

INPUT	**OUTPUT**	
Item Id	**Item Name**	**Item Price**
4503257553	Harry Potter and the Half-Blood Prince (Book 6)	17.99 ($)

Figure 6.4 shows the tests done in eBay Item Information web service execution times with and without our arcitecture. As it can be observed, PCS optimizes the execution time ranging from 6 to 10 times.

Figure 6.4 Analyses of eBay Item Information Web Service

Table 6.6 shows the summary of execution times of above 4 external web services with or without our architecture. As it can be seen, execution time is optimized from 4 ranging up to 10 times in different web service using our architecture.

Table 6.6 Summary of complete experimentation

Description of Web Service	**Execution time without PCS (ms)**	**Execution time with PCS (ms)**
US Weather Forecast WS	1600 - 2000	400 - 600
Turkish Central Bank Currency Exchange WS	1300 - 1500	300 - 450
Turkish Republic Identity WS	1200 - 1500	400 - 500
EBAY Item Information WS	3500 - 6000	700 - 1000

7. CONCLUSION

In this research, an architecture is described to solve the performance and availability problems of web services. It is aimed to make the right decision of caching and invoking a web service according to certain specific criterias. In this way, it is aimed to reduce the load of web services in distributed applications. Moreover, it is aimed to prevent the caching to overwhelm the disk size on client size by pruning unimportant web service invocation results.

7.1. Overview

The main parts of web services and usage of these parts in our architecture is stated in the first section of Chapter 2. Then, the importance of web service composition in business applications are emphasized and the chosen tools are described in the second part of Chapter 2. Thus, building blocks of our architecture is explained to reflect the applicability of our approach in web services.

The general overview of our architecture is presented in Chapter 3. At the first part, tools used in our architecture is stated. Then, it is explained how to transform the input elements (BPEL processes) of our architecture to be integrated with our PCS. Then, the core caching architecture is explained including the integration with BPEL processes.

The elaborated version of Chapter 3 is explained in Chapter 4 and Chapter 5. In Chapter 4, transformation of BPEL processe are explained with an example. Then, techniques and logic behind the transformation process is emphasized.

In Chapter 5, the main elements of caching architecture is explained in detail. At the first section, the working principle and influence of WSDL in invoking web services on BPEL processes are explained. Then, the working principle our PCS on BPEL processes is stated. After that, importance and working principle of Daemon PCS is explained in caching process. The possible overheads in the processing of our architecture and their proposed solutions are stated.

In Chapter 6, experimental tests are done using our architecture. Web services that exist in real life scenarios are selected for test environment. Then, experimental tests are done with and withour involvment of our architecture to prove the efficiency of our approach. At the end, the applicability and efficiency of our approach is proven with significant improvements in performance.

Finally, the summary of the research and its contributions are stated in this chapter.

7.2. Future Work

Our PCS stores statistical information about previous invocation of web services. This statistical information is used to find the frequency of invoked web services and prune web service invocation data that has less frequency. For the future work, we aim to extend the aspects of statistical information obtained from web service invocations. As an example, it may be possible to discover some paths regarding previous web service invocations. Using discovered paths, we aim to predict the invocation time of web services in a busines process and apply some pre-fetching of web service invocations before the corresponding web service needs to be invoked. As a result, further optimizations on the concept of web service processing can be done.

APPENDIX A: SOFTWARE CD

Listings of software are given in a CD. The CD contains the files containing the source codes for PCS Pre-processor, Proactive Caching Scheduler, Daemon Proactive Caching Scheduler, database engine and sample BPEL documents used in experimental tests. Other than these, there exists another file named "READ.ME". In the ASCII text file, the following sections appear.

- Files in the CD: In this section, the names of files together with their contents are listed
- Hardware Requirements: In this section, the hardware requirements which are necessary to run the software are noted.
- Software Requirements: In this section, the operating system, the engine and the libraries etc. that are necessary to compile and run the software are listed.

REFERENCES

1. Wohed P., W. M. P. van der Aalst, M. Dumas and A. H. M. ter Hofstede, "Analysis of Web Services Composition Languages: The Case of BPEL4WS", *In Proceedings of the 22nd International Conference on Conceptual Modeling (ER)*, Chicago IL, USA, October 2003, http://is.tm.tue.nl/research/patterns/download/bpel_er.pdf.

2. Booth D. et al., "Web Services Archiecture", *W3C Working Group Note*, February 2004, http://www.w3.org/TR/ws-arch.

3. Graham S. et. al, "Building Web Services with Java: Making Sense of XML, SOAP, WSDL and UDDI", pp 1-20, 2002.

4. Ort E., "Service-Oriented Architecture and Web Services: Concepts, Technologies, and Tools", *Sun Developer Network*, April 2005, http://java.sun.com/developer/technicalArticles/WebServices/soa2/soa2.pdf.

5. Chappell D. and T. Jewell, "Java Web Services", pp. 6-7, March 2002.

6. Microsoft Corporation, *DCOM Technical Overview*, November 1996, http://msdn2.microsoft.com/en-us/library/ms809340.aspx.

7. Raj G. S., "A Detailed Comparison of CORBA, DCOM and Java/RMI", September 1998, http://my.execpc.com/~gopalan/misc/compare.html.

8. Kızıltunç M. K., "Web Hizmetleri", October 2002, http://www.teknoturk.org/docking/yazilar/tt000104-yazi.htm.

9. Jong I., "Web Services/SOAP and CORBA", April 2002, http://www.omg.org/news/whitepapers/CORBA_vs_SOAP1.pdf.

10. Remote Method Invocation Home, http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp.

11. Bray T., J. Paoli, C. M. Sperberg-McQueen, E. Maler and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fourth Edition)", *W3C Recommendation*, August 2006, http://www.w3.org/XML.

12. Berglund A., "Extensible Stylesheet Language (XSL) Version 1.1", *W3C Recommendation*, December 2006, http://www.w3.org/TR/xsl.

13. Java Architecture for XML Binding (JAXB), http://java.sun.com/webservices/jaxb/index.jsp.

14. Java API for XML Processing (JAXP), http://java.sun.com/webservices/jaxp/index.jsp.

15. Java API for XML Registries (JAXR), http://java.sun.com/webservices/jaxr/index.jsp.

16. Java API for XML-Based RPC (JAX-RPC), http://java.sun.com/webservices/jaxrpc/index.jsp.

17. SOAP with Attachments API for Java (SAAJ), http://java.sun.com/webservices/saaj/index.jsp.

18. Gao D., *A Java Implementation of the Simple Object Access Protocol*, M.S. Thesis, The Florida State University, College of Arts and Sciences, December 2001.

19. Gudgin M., "SOAP Version 1.2 Part 1: Messaging Framework", *W3C Recommendation*, June 2003, http://www.w3.org/TR/soap12-part1.

20. Oracle Technology Network, "BPEL Designer Tutorial 1: Developing a Hello World BPEL Process", http://www.oracle.com/technology/products/ias/bpel/pdf/orabpel-Tutorial1-HelloWorldTutorial.pdf.

21. WSDL and UDDI, *W3Schools*, http://www.w3schools.com/wsdl/wsdl_uddi.asp.

22. Dustdar S. and W. Schreiner, "A Survey on Web Services Composition", *International Journal of Web and Grid Services*, Vol. 1, No. 1, 2005.

23. Peltz C., "Web Services Orchestration, A Review of Emerging Technologies, Tools, and Standards", January 2003, http://devresource.hp.com/drc/technical_white_papers/WSOrch/WSOrchestration.pdf.

24. Peltz C., "Web Service Orchestration and Choreography. A look at WSCI and BPEL4WS", July 2003, http://se2c.uni.lu/tiki/se2c-bib_download.php?id=1262.

25. Peltz C., "Web Services Orchestration and Choreography", October 2003, http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1236471

26. Arkin A. et al., "The Web Service Choreography Interface (WSCI) 1.0", *W3C Note*, August 2002, http://www.w3.org/TR/wsci.

27. Patlak Ç., *Web Service Orchestration Standards*, M.S. Thesis, Boğaziçi University, 2003.

28. Andrews T. et al., "Business Process Execution Language for Web Services Version 1.1", May 2003, ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf.

29. Leymann F., D. Roller and S. Thatte, "Goals of the BPEL4WS Specification", August 2003, http://xml.coverpages.org/BPEL4WS-DesignGoals.pdf.

30. Juric M., "BPEL and Java", April 2005, http://www.theserverside.com/articles/article.tss?l=BPELJava.

31. May B. and P. Savur, "Business Process Execution Language, Part 1: An Introduction", October 2006.

32. Lehmann M., “Your First BPEL Project”, September 2004, http://www.oracle.com/technology/oramag/oracle/04-sep/o54web.html.

33. Web Services Invocation Framework, *The Apache Software Foundation*, May 2006, http://ws.apache.org/wsif.

34. Juric M., “Using WSIF for Integration”, October 2005, http://www.oracle.com/technology/pub/articles/bpel_cookbook/juric.html.

35. Introduction to the ActiveBPEL Engine, http://www.activebpel.org/info/intro.html.

36. IBM Software - WebSphere Process Server, http://www.ibm.com/software/integration/wps.

37. IBM AlphaWorks BPWS4J : Overview, August 2002, http://www.alphaworks.ibm.com/tech/bpws4j.

38. Microsoft BizTalk Server: Home, http://www.microsoft.com/biztalk.

39. Web Service Orchestration Software from OpenStorm, http://www.openstorm.com.

40. Oracle BPEL Process Manager Quick Start Guide 10g Release 3, August 2006, http://download-east.oracle.com/otn_hosted_doc/soa/docs/books/b28983.pdf.

41. Oracle BPEL Process Manager, http://www.oracle.com/technology/bpel.

42. Oracle BPEL Process Manager Administrator’s Guide Release 10.1.2, April 2006.

43. Oracle JDeveloper, http://www.oracle.com/technology/products/jdev/index.html

44. Goodman B., “Accelerate Your Web Services with Caching”, December 2002, http://www-128.ibm.com/developerworks/webservices/library/ws-cach1.

45. Myllymaki J. and B. Reinwald, "Database Support for Web Service Caching", *IBM Research Report*, November 2003.

46. Tewari R., M. Dahlin, H. M. Vin and J. S. Kay, "Design Considerations for Distributed Caching on Internet", *International Conference on Distributed Computing Systems (ICDCS)*, pages 273-284, Austin, Texas, June 1999.

47. Terry D. B. and V. Ramasubramanian, "Caching XML Web Services for Mobility", *Microsoft Research*, May 2003.

48. Powell M., "XML Web Service Caching Strategies", *Microsoft Corporation*, April 2002.

49. HTTP/1.1: Caching in HTTP, http://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html.

50. HTTP Caching for Personalized Content, http://www.itamarst.org/writings/dynamiccaching.html.

51. Belloum A. and B. Hertzberger, "Maintaining Web Cache Coherency", *Information Research*, September 2000, http://InformationR.net/ir/6-1/paper91.html.

52. Srinivasan R., C. Liang and K. Ramamritham, "Maintaining Temporal Coherency of Virtual Data Warehouses" *19th IEEE Real-Time Systems Symposium (RTSS'98)*, 1998.

53. Deolasee P. et al. " Adaptive Push-Pull: Disseminating Dynamic Web Data" *In Proceedings of the Tenth International World Wide Web Conference*, pages 265-274, Hong Kong, May 2001.

54. M. N. Çankaya, "ASP.NET'de Sayfayı Önbelleklemek", August 2004, http://www.yazgelistir.com.

55. Pallis G. and A. Vakali, “Insight and Perspectives for Content Delivery Networks”, January 2006.

56. Devillechaise S. and J. M. Menaud, “Caching Web Services: Aspect Orientation To The Rescue”, 2002.

57. Reabow J. and D. Pillay, “Caching XML Web Services to Support Disconnected Operation”, 2004.

58. König D., “Web Services - Business Process Execution Language 2.0”, *Second Annual OASIS Adoption Forum*, October 2005, http://www.oasis-open.org/events/adoption_forum_2005/slides/Koenig.ppt.

59. Juric M., “A Hands-on Introduction to BPEL”, 2003, http://www.oracle.com/technology/pub/articles/matjaz_bpel1.html.

60. Marshall D., “The C Preprocessor”, May 1999, http://www.cs.cf.ac.uk/Dave/C/node14.html.

61. XML DOM Tutorial, W3Schools, http://www.w3schools.com/dom/default.asp.

62. Java’s Timer Class, http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/Timer.html.

63. White T., “Scheduling recurring tasks in Java applications”, November 2003, http://www-128.ibm.com/developerworks/java/library/j-schedule.html.

64. Haefel M., “EJB 2.1:The Timer Service”, October 2002, http://www.theserverside.com/tt/articles/article.tss?l=MonsonHaefel-Column4.

65. Deitel H.M., P. J. Deitel and S. E. Santry, “Advanced Java 2 Platform – How to Program”, 2002.