# BIZBUILDER–AN E-SERVICES FRAMEWORK FOR INTERNET WORKFLOW

By

RAJA KRITHIVASAN

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2001

I dedicate this thesis to my family

ACKNOWLEDGMENTS

I offer my highest gratitude to Dr. Helal for motivating me all through the period of this thesis. It is the enthusiasm, encouragement and guidance he provided every time, that allowed me to pursue and complete this thesis. I am very thankful to Dr. Su, for his valuable suggestions and to Dr. Lam for serving on my committee.

I also thank Arun Swaran and Jie Meng, for their involvement and help in the project. My thanks are due to all members of the Harris Lab, for their suggestions and company.

Finally, to my parents, and all my family members, I am indebted to them for their immeasurable support and affection.

TABLE OF CONTENTS

page

LIST OF TABLES

Table Page

LIST OF FIGURES

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

BIZBUILDER – AN E-SERVICES FRAMEWORK FOR INTERNET WORKFLOW
By

Raja Krithivasan

August 2001

Chairman: Dr. Abdelsalam (Sumi) Helal
Major Department: Computer and Information Science and Engineering

One of the fundamental requirements for solutions to succeed in the business-to-business e-commerce domain is the ability to integrate seamlessly or inter-operate with diverse systems. In other words, the services offered by a business should be easily accessible to any other consumer or business using the Internet infrastructure and standard protocols. Currently used distributed systems solutions do not completely offer a platform for the creation of these E-services. In this thesis, we present BizBuilder – a software solution that offers a framework to create E-services, to convert existing non-Internet adaptive services to E-services and participate in workflows of other business processes, thereby obtaining more business opportunities.

Most businesses have a suitable web-interface for the services they offer, so that a customer can search for a service of their interest and use it. In a similar way, these services can also be used by businesses as a part of their workflow process, provided the services can support certain aspects related to workflow. The primary difference between a business using a service and a consumer using the same is, the consumer typically uses

a web-interface like a Hyper Text Markup Language (HTML) form or an applet to manually fill in the required details. The details are then used to perform the desired service and the customer gets the result of the service almost immediately. The same scenario cannot be applied to businesses, since manual intervention (to fill the details required by the service) in one's business process is time-consuming and prone to errors. In other words, a programmatic model or a framework for accessing these services across the Internet has to be provided. With this framework, businesses can search a desired service, provide the required data and utilize the service, all without manual intervention. The BizBuilder E-services framework provides exactly this facility, to export Java objects as E-services using Simple Object Access Protocol (SOAP) messaging format.

The framework also addresses the issue of workflow in particular and facilitates managing both synchronous and long-running E-services that participate in a workflow. Additionally, BizBuilder provides suitable support to register services to a Universal Description, Discovery and Integration (UDDI) enabled brokering community. Finally, BizBuilder supports the notion of virtual enterprises by providing a service-based infrastructure for new E-services composition from existing ones.

# CHAPTER 1
# INTRODUCTION

The phenomenal growth of business-to-business (B2B) e-commerce has fueled the development of software systems that would integrate themselves seamlessly into the existing B2B space and provide better value-added services. This rise in the growth of B2B e-commerce is a natural outcome of the success of the earlier model, the business-to-consumer (B2C) e-commerce. It was soon realized that the Internet infrastructure could be used by businesses effectively as done by consumers. The focus shifted from business process re-engineering to inter-enterprise process engineering (IPE) and in setting up "virtual corporations." The text "Enterprise E-Commerce" [FIN00] describes IPE "as the competitive weapon for designing and implementing hyper-efficient business processes that are integrated in real-time and jointly owned by suppliers and customers". Businesses could use Internet as their communication medium to get the services they need from other businesses. A supply chain consisting of a manufacturer, distributor and seller can be taken as an example where a business requires the service of another. In this scenario, each of the participating business in the supply-chain can use Internet as the communication medium to obtain their services. In similar lines, we propose a service based B2B model, where a workflow of activities can be created and executed to completion using the Internet as the distributed services infrastructure.

## Distributed Systems and the Internet

The evolution of distributed systems marked a significant change in the way software systems are written, deployed and used. Highly sophisticated systems like CORBA, COM and RMI have helped a large workflow of activities to be distributed physically across a network. Business having realized the potential of these distributed object or system technologies have started to leverage this with the already existing Internet infrastructure. The result, Internet based, distributed systems are being developed and getting popular [LAR96].

## Distributed Systems – An Analysis

Distributed systems like CORBA were developed with main objectives like physical distribution of software components, easier upgrade of software, harness the local resources available and produce more fault-tolerant systems. Their advantages were overwhelming, which led to the creation of varied competing technologies each with their own advantages and disadvantages, though they all were similar in design and concept. For example CORBA supports objects to be created and accessed in a language independent way, i.e., a C++ client object can talk to a Java server object. In addition, complex object management tools like Object Request Brokers (ORBs) were created that would perform functions like object creation, memory related issues, multi-threading and scalability.

Though the ORB based systems were becoming popular, they had an inherent disadvantage when it came to being tied to the Internet. These systems used proprietary protocols for marshalling and un-marshalling of request and response data to perform a remote-procedure call. In other words, there was no one or standard way of message

passing or message structure being followed by these systems. Also the message structure representation was not in text form, meaning it was not human-readable. This scheme had the advantage of being compact in terms of representation and therefore efficient, but difficult to be parsed when it came to interpreting the messages. More problems surfaced when ORBs were deployed on servers running on the Internet. Typically, a web-server hosted by an organization will reside behind a firewall. The firewall in this case acts as a security guard, which inspects the packets or data being sent from the outside domain to the web-server. Most of the firewalls allow http data that are intended for port 80 to be passed and all other data packets intended for some other ports (and therefore some other programs and not the http server) will not be allowed by the firewall. On the contrary, all organizations that have the Internet presence would like to provide their services using these web-servers, similar to the web sites they host. Deploying separate servers to provide business services and implementing security-mechanism were not feasible options. Thus the data representation and communication aspects of these distributed object systems were not readily suitable for the Internet.

## Data Representation and Protocol Standards

Data representation has been an active area of research, mainly being catalyzed by the need to create a common understandable data representation for communication over the Internet and for data-storage and retrieval. The Extensible Markup Language or the XML was created as the result. XML can be seen as a step to solve the problem of ontology [ONT00] that exists when doing business on the Internet. Due to its features and widespread adoption XML is becoming the de-facto language for message representation in e-business.

In a similar way the http and its related protocols have become the first choice for e-commerce communication. The end result of this standardization being, all systems developed for performing e-business over the Internet invariably uses both the standards.

## E-Services and Shallow Workflow

An E-service is any value providing service that is being offered electronically on the Internet. In most cases an E-service is also one that is implemented using standard Internet technologies like HTTP and XML. A simple example of an E-service can be weather update information that is available on the web. A more complex example would be a “Book-Publishing” service, that accepts documents from users, proof-reads them, formats them according to user preferences, prints and binds them and ships them to the user. All these internal processes are transparent and the user may view the whole activity as a single task.

Shallow workflow is the term we use to define the sequence of individual activities that are performed in a selective order to complete a complex task. In the above Book-publishing example the sequence of proofreading-printing-shipping can be viewed as a workflow of individual activities, though the dependencies between the individual activities are not shown here. But it is this concept of shallow workflow (which we will mention as just workflow from here) that leads to the concept of creating *Virtual Enterprises* on the Internet.

## Scope of Thesis

This thesis details BizBuilder – the E-services infrastructure that is created for the Internet workflow project, which is the overall vision. The Internet workflow project can be broadly divided into three major components namely,

- The e-service infrastructure
- Brokering communities and protocols
- The workflow engine

This thesis focuses on the design, architecture and implementation of BizBuilder, the E-services framework. A survey of existing E-services related solutions are also presented. A detailed analysis of how the E-services infrastructure serves as the enabler of the Internet workflow project is depicted, with examples and scenario analysis. Finally the goals accomplished and the opportunities for future work are listed.

# CHAPTER 2
# SURVEY OF RELATED SOLUTIONS

In this chapter, a detailed survey of concepts and technologies that are similar to E-services and workflow are presented. Tiwana and Ramesh [TIW01] and Kuno [KUN00] provide a classification of E-services based on the state-of-the-art and target market. Many architectures and infrastructures are being built recently based on the concept of E-services and dynamic composition of services [MEN00, VAS98b] and some of the prominent ones are surveyed below.

## Commercial E-services Based Products/Solutions

The following are some of the commercial E-services infrastructure or solutions that are currently available and used.

### Microsoft's .NET Framework

Microsoft's .NET (pronounced dot NET) [MSF01] framework is heavily based on the concept of E-services or alternatively called web-services. It is being described as an initiative that relies on the Internet as the software delivery platform or in some sense as the operating system and the applications being viewed as web-services available on the Internet. This framework comes with its own array of supporting tools, a programming language and other data representation standards

Language independence is one of the key elements that need to be taken care of in the arena of E-services. The Component Object Model (COM) was popular but restricted itself to the PC or the Windows platform. This drawback of COM is being removed in the .NET initiative. The result being, a new language C# (pronounced C sharp) [CSA00] is

proposed and is being seen as the language of the .NET initiative. In other words, in order to achieve language independence a common intermediate language (and a common language runtime) was devised and C# code would be compiled to produce the intermediate code named MSIL. Also support is being provided to compile different language source code into this common intermediate representation.

The Common Language Runtime is said to be the execution engine that will manage issues like object loading and execution, memory and security management, conversion from intermediate to native code and so forth.

Apart from being a composite framework that is being envisioned to support a variety of languages, the .NET framework is said to have the following advantages.

- Primary advantage being distribute services across the Internet, as web-services
- Make websites and the Internet “programmable”
- Support for many existing programming languages
- Based on standard Internet standards like XML

The .NET initiative also relies heavily on XML for communication. The Simple Object Access Protocol that is based on XML is also being used in .NET framework along with other tools like BizTalk Server 2000.

HP’s E-Services

HP was one of the first companies to conceptualize and work on the concept of E-services. E-services, according to HP, are “an initiative that envisions a new paradigm for electronic commerce in which a rich array of nimble modular electronic services (E-services) are accessible by virtually anyone and any device.” E-services were targeted more towards the B2B and B2C e-commerce market space.

HP's E-services offer a wide-array of solutions for the e-commerce space such as service discovery, brokering, negotiation, mediation, billing, payment, personalization and management of services. It comes with a suite of products like E-Speak, Jet Send, Chai server etc., where each component performs specific functionalities namely:

- E-Speak [ESP00] addresses issues like service discovery, automated brokering, auditing etc. The issue of dynamic composition of services is also addressed.
- ChaiServer [CHA98] is a web-server targeted for device and non-device environments. Used to expose services on the Internet using Java and URL schemes.
- Jet Send is the communication technology for enabling device local connectivity
- Changengine is a process manager used to automate business processes of an enterprise and also create new process-based solutions

E-services are one of the few concepts to discuss the dynamic composition of composite service, namely eFlow [CAS00a], which is also the area of our research projects. eFlow is a system that specifically addresses the issue of composing new service [CAS00b] using existing services. eFlow offers an environment where new services can be created or assembled on the fly and can dynamically accustom to changes. eFlow also provides tools that allow the user to monitor, analyze and modify a service while in execution.

IBM's Web Services

Web Services development environment [WEB00], according to IBM is "an environment that creates open, platform-neutral web-services for deployment across heterogeneous systems." Web-services are one of the practical systems that allow web-services to be created, searched for and invoked. It is heavily based on XML and java.

This framework also comes with other tools and proposed standards like web services description language (WSDL), universal description, discovery and integration (UDDI) [UDD00] modules for service registration and brokering, simple object access protocol (SOAP) for communication etc.

As with other equivalent concepts, interoperability is concentrated and is achieved by following standard protocols and message formats (like SOAP and HTTP). Web Services includes a service broker and a service requestor in its architecture in order to get a complete framework. One of the significant additions in the Web Service framework is the use of the web services description language [WSD01] that is used to describe certain aspects of a service and utilize it programmatically. The Web Services framework also utilizes the Universal Description, Discovery and Integration (UDDI) standard to describe services. UDDI promises a way to quickly identify services that are relevant and use them in preferred applications.

## Other Related Research

Apart from the above-mentioned major initiatives, there exists numerous other research and commercial products that are similar in idea or concept to E-services and to workflow on the Internet [BOL99, SWA99]. They either directly support an E-service framework or come as additional tools that can be used in creating an E-services infrastructure. Some of them are surveyed below.

### Coyote-Multi Party Framework

The Coyote architecture [DAN98] was one of the earliest research efforts to deal with the concepts of multi-party e-commerce and long-running conversation. Coyote takes into consideration a distributed long running application that spans across multiple organizations. A key concept in Coyote is the multi-party service contract, which

specifies the rules of engagement between the parties, including the inter-business actions, which can be performed, compensation for canceling and so on.

Coyote discusses the idea of using the services provided by different businesses, in an integrated fashion. It illustrates how to build a service flow from a workflow of activities, where service flow is built as a sequence of services linked together, and executed in a Petri-net like manner. Coyote proposes using XML for contract specification, and, code generation based on the content of these service contracts are done to build modules, which enforces these contracts. The service contract includes extensive details like communication properties, role, actions (consisting of method signatures, responsiveness, constraints and sequencing rules) and error handling. The overall system supports features including Contract Registration, Service instance creation and management, Multi-protocol support and management of both synchronous and asynchronous services.

Web Trader

Web Trader [VAS99a, BAN98] is a system, which helps service-client matchmaking dynamic, thus creating a flexible and open marketplace. One of the aims of Web Trader is to create a lightweight trader similar to OMG CORBA but using web mechanisms instead of CORBA. Web Trader includes the concept of service advertisement using XML, which is a complete semantic description of a service. The service advertisement in Web Trader is very service specific, meaning it includes details like name of service, its access point, and the interface from which it is derived. These details are grouped as the three XML elements namely search keywords, metadata and interface. Web Trader takes a different approach in building a “lightweight” solution, specifically; it uses XML extensively in order to represent aspects denoted by IDL in

CORBA. Web Trader also differentiates itself from CORBA traders by adopting a *service repository* based scheme, which acts as a central point for publishing and matchmaking. It argues that this repository-based federation approach is advantageous over the direct-federation approach that OMG adopted. While the direct-based federation approach is simple and safe, it does not necessarily scale well on the Internet. Web Trader also concentrates on issues like QoS during matchmaking and stresses on the "pull" model of service discovery, where agents can roam and find a suitable service matching their needs.

RainMan

RainMan [PAU97] is a distributed workflow system developed in IBM. RainMan is built as a collection of loosely coupled independent services that co-operate with each other. RainMan provides a browser-based workflow specification, participation, management and workflow modification features. This prototype system has been completely developed in Java and uses Java's Remote Method Invocation for transport between distributed components. RainMan provides a Java applet user interface that allows specifying workflows as directed, acyclic graphs. In addition, the system also offers workflow specific lookup services like directory service – to locate performers or service providers in a network, Worklist service – where a worklist is a persistent storage of task request, and performers – where performer is an abstraction for modeling humans or software applications in a workflow. RainMan separates service requesters from service providers using Java interfaces and assumes the implementing entities are heterogeneous. In other words, it does not specifically provide a solution based on the concept of E-services, which are heterogeneous, or take the approach of using an

interoperable messaging structure like XML. But RainMan is one the early systems to focus research in the direction of Internet wide workflow

Wf-XML

Wf-XML [WFX00] is a specification in XML provided by The Workflow Management Coalition, for data transfer of workflow messages. It aims to provide interoperability between workflow systems by being independent of language or protocol implementation. Wf-XML supports simple chained workflows and nested workflows in both synchronous and asynchronous fashion. This specification is abstract and concrete implementations of this specification can be used to achieve interoperability between workflow systems.

Being based on XML, Wf-XML supports the native data types that are supported by XML DTD and promises to enhance this by using XML Schema. The actual message structure is based on the proposed logical resource model [WFX00] consisting of ProcessDefinition, ProcessInstance and Observer. Wf-XML also deals with using the HTTP protocol for binding and has provisions for specifying interoperability contract. Wf-XML thus would be very useful for XML based messaging in workflow systems.

XML-RPC

XML-RPC [XMR00] was one of the early efforts to support remote procedure calling over the Internet. The focus of this specification is not to produce another RPC system, but to create an RPC message representation format, that is easily transportable over the Internet, platform independent and that could be understood by a variety of systems differing by platform or programming language. Naturally, XML was chosen as the choice for message representation. XML-RPC provides a way to code RPC information in XML and use HTTP to actually transport the message and perform remote

procedure calls. Implementations for XML-RPC are available for a wide range of programming languages including C, C++, COM, Java, Python, Tcl, and Perl.

Simple Object Access Protocol

SOAP [SOA00a, SOA00b] is actually an extension of XML-RPC. It was created as an outcome of the standardization effort by W3C to create a lightweight protocol for information exchange in a decentralized, distributed environment. It is based on XML and provides a standard format to encode information required for a method call. In other words, every SOAP message is composed of a method element in XML, which has details including the parameters and values that are needed to perform the method invocation. Apart from providing a simple scheme to encode function call information, it can be easily extended to include details like the location of the service (or method), parameter types and return values. In simpler terms SOAP can be seen as an extensible format that should follow some basic standard XML encoding mechanism so that uniformity in messaging and thereby interoperability is achieved. The BizBuilder framework, detailed in this thesis uses SOAP and more details and illustrations are provided in the following chapters.

# CHAPTER 3
# E-SERVICES AND INTERNET WORKFLOW

This chapter defines the concept of E-services that has been taken up and the solution proposed by this thesis. Also the relation between E-services and Internet Workflow is illustrated.

## What Is an E-Service?

The concept of E-services has become so prevalent that multiple answers exist for the question “what is an E-service?” One interesting definition is by HP, which says

“An E-service is any asset that you make available via the Net to drive new revenue streams or create new efficiencies.”

The above definition lays stress on the business perspective of an E-service. A more technical definition for an E-service could be “ any service or functionality that can be accessed by a business or a consumer programmatically on the Internet, using standard data representation and protocols.” By standards, we stress that any E-service that exists on the Internet should have a data representation that is not proprietary and one that is easily understood. Also the service should be accessible using standard Internet protocols like HTTP.

## E-Services and Internet Workflow

The E-services infrastructure will be used as the fundamental building block for our Internet based workflow system. The following diagram depicts the position of E-services in an Internet workflow scenario.

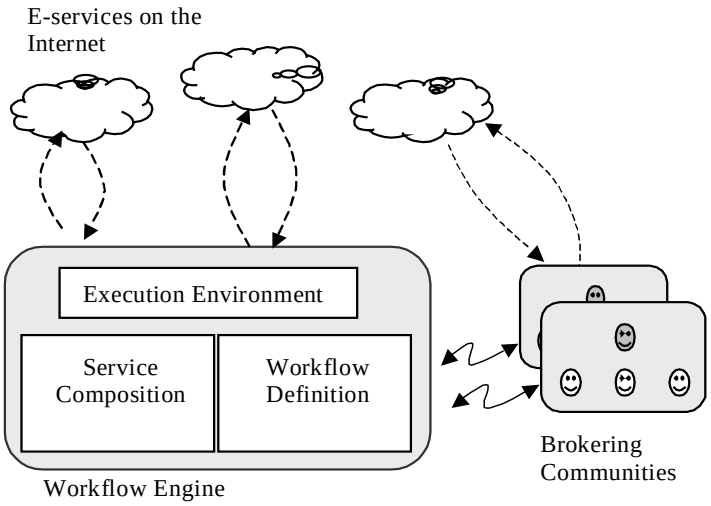


Figure 3.1: E-Services and Internet Workflow Depicted

The E-services infrastructure enables the creation of E-services that are accessible on the Internet and those that can participate in a workflow. It can be seen from Figure 3.1 that the E-services are the individual entities that group together to create a workflow. The other modules represented here include the workflow engine that is responsible for creating, managing and executing workflow tasks. A UDDI based brokering community [HEL01, JAG01] is used as the service repository for matchmaking. The brokering community by itself is sophisticated and provides a framework for publishing, inquiring services and matchmaking. The E-services interact with the brokering community for service advertisement and publishing. On the other hand, the workflow engine uses the brokering community to discover a service of interest. The workflow engine utilizes the individual E-services during the execution of a workflow. This thesis deals only with BizBuilder, the E-services framework that is used to create these E-services.

## Motivating Scenario

The need for an E-services infrastructure can be realized by considering the following practical scenario.

Imagine a person X, who is involved with document proofreading. He might work for a publishing company or may have his own web-presence with suitable software, where he takes job orders from customers and provides the proofreading service. This form of business can be classified as one under B2C e-commerce, where X's proofreading business essentially provides services to consumer through HTML web pages and forms. In order to obtain more business opportunities, X may decide to upgrade his proofreading service, whereby even business like publishing companies, booksellers and reviewers can utilize his services. In other words, X wants his proofreading service to be a part of as many businesses workflows as possible. On the contrary, X may also decide to host a new publishing service of his own, and may decide to use services like formatting, printing, shipping and handling along with his proofreading service and create a new workflow of services in order to provide a value-added service like one-stop Book-Publishing.

Considering the above requirements, X needs a software infrastructure, where he can provide his services as an "E-service" that is accessible to anyone (customer or business), programmatically, without the need to manually fill-in details in web pages. In addition he also needs a software framework, where he can compose a new service, by creating/managing a workflow of existing services on the Internet. This ability to provide an E-service platform, and support the creation of composite services and the construction and management of a workflow are the primary motivating requirements for BizBuilder.

## BizBuilder-The E-Services Infrastructure

BizBuilder, our E-services infrastructure provides the necessary tools and APIs using which any E-service that exists on the Internet can be invoked and utilized. In specific terms, the infrastructure provides a facility to

- Take an existing Java object (or service) implementation and provide an E-service wrapper
- Take an existing Java object implementation and provide an XML interface description of an object
- Provide the facility for advertising a service to a broker
- Provide necessary tools and APIs to invoke an E-service on the Internet
- Provide support for executing a “shallow workflow”

The E-services infrastructure targets a service provider who has a web-presence and business services to be provided (as a legacy system or a non-Internet adaptive distributed system), but the services being not actually available on the Internet (i.e., as E-services). Using this infrastructure the service provider can enable E-services and thereby get more business opportunities. Apart from this primary goal, the service provider can participate in a workflow of another business seamlessly, without having to adapt or build any specific system in order to communicate with the workflow system. Optionally the service provider can also create a workflow of his own that will (re) use services provided by others on the Internet. Though the E-services infrastructure does not directly support the creation and management of this shallow workflow, it provides the necessary support for an E-service to participate in a workflow by providing capabilities to respond to workflow/transaction specific queries. We assume a workflow as a

“shallow workflow,” which can be viewed as a collection of related activities used in conjunction to obtain a new or a value-added service. In programming terms, this shallow workflow can be seen as a sequence of functions (or services) invoked in a specific order, with one using the result of another and the end result is seen as a value-adding service.

The E-service infrastructure also aims at supporting asynchronous long-running services that can be considered to be a part of a workflow. It is quite possible for individual activities in a workflow to take an arbitrary time to complete and therefore the result of that particular activity may not be provided synchronously. In this case asynchronous mechanism like polling or event-notification or push can be used, where the activity after completion can inform the user about the result of the activity. This method of event-triggering results, is possible in traditional distributed systems, but is complex in nature when considered in the e-commerce space. This is because the user or the service requestor has to wait for the results from the activity, this means the requestor serves as the server in a sense, waiting for the event. But the requestor may not be treated as a server anytime in the Internet context, since the requestor can just be a browser client. This asynchrony is possible if the requestor can act like a server waiting for events or triggers. Additionally “callbacks” in http cannot be easily achieved due to the nature of the clients and the http protocol. So the E-services infrastructure partially supports the asynchronous long-running services, as some aspect of this mode is dependent on the nature and ability of the service provider.

# CHAPTER 4
# BIZBUILDER – DESIGN AND ARCHITECTURE

The architecture that involves the main components and the building blocks of our E-services framework, the BizBuilder Framework is outlined below.

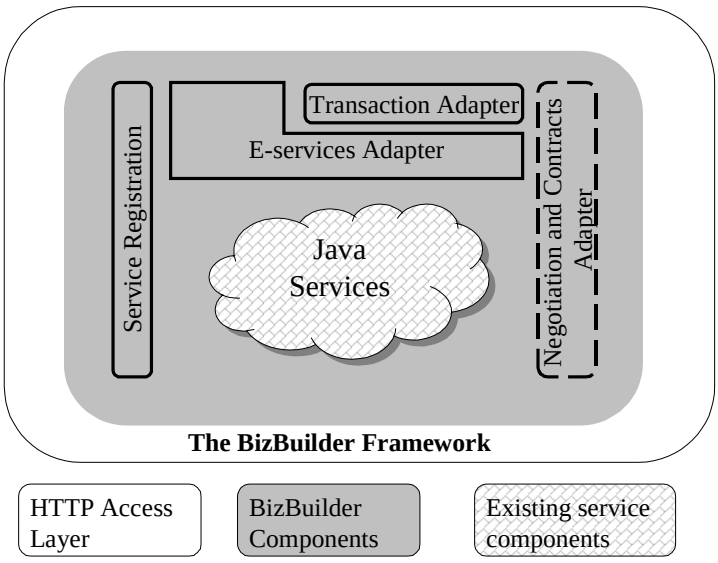


Figure 4.1: BizBuilder Framework

As indicated in the above diagram, the E-service adapter, service registration module, the transaction adapter and the negotiation and contract adapter form the core framework architecture. Each of these modules and their functionality are described below.

## The E-Services Adapter

The E-services adapter is the core component that allows the underlying services (for example services implemented using java objects) to be accessed on the Internet using the standard protocols and format. This module takes an existing service implementation in the form of a java object and creates the required server side framework, so that the java services can be invoked using a XML messaging structure on the HTTP protocol.

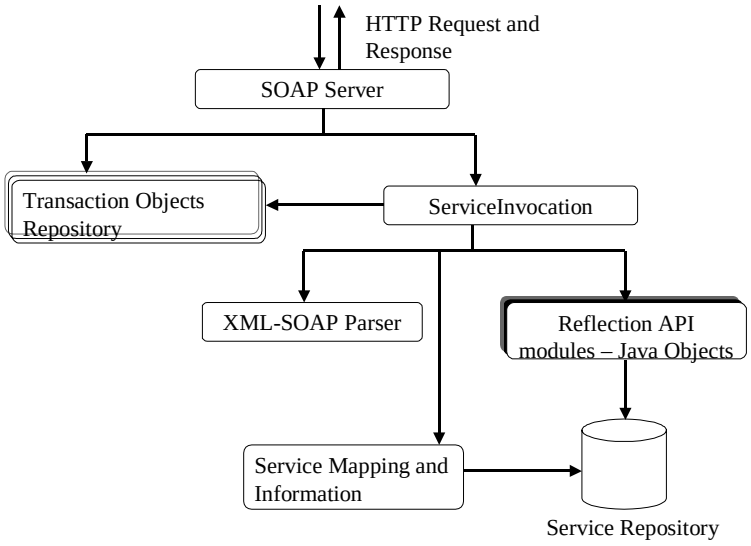


Figure 4.2: Components of the E-Services Adapter

Figure 4.2 depicts the main modules of the E-services adapter and the interactions between these modules. An arrow from component X to component Y indicates that component X uses component Y.

Table 4.1: List of Modules in the E-services Adapter and their Purpose

| Class/Module Name | Description |
|---|---|
| SOAP Server | Servlet based HTTP server that processes SOAP coded requests |
| Transaction Objects Repository | A repository for storing a collection of transaction objects that are used to answer transaction probes |
| ServiceInvocation | The module that performs the operation of method invocation on native objects, using helper classes like XML parser and reflection APIs and takes care of storing objects in case of long-running services |
| XML-SOAP Parser | Utility classes that parses XML-SOAP coded messages and provides services to the Service Invocation module |
| Reflection API modules | Modules built based on the Java reflection API, used to perform the actual method invocation on the java objects |
| Service Mapping and Information | Modules used to provide persistent storage for information regarding services, their associated class files, parameter names and types for service invocation etc. |

As indicated in the above diagram, the E-services adapter by itself is made up of various modules. The important assumption made in this adapter is the presence of the implemented services in the form of java objects. In other words, a service is assumed implemented as a method (or as a collection of methods) in a specific java object and this mapping information, specifying which java object (the class file) implements which method is made available to the adapter to create the required web binding. The service mapping information component indicates this in the above diagram.

Messaging Structure

The SOAP format that is based on XML is used as the communication message structure that will be used by clients to specify the service to be invoked along with the required (parameter) values. SOAP is based on XML and is primarily used to indicate details like method name, parameter name and values to the server, which will parse

these information, do the required service (or method) invocation on the underlying object and sends back the result again in SOAP coded format. The main idea behind using SOAP format for messaging is not just because it is becoming a standard and gaining popularity, but it is based on XML thereby it is easily extensible and also specifically addresses the problem of doing RPC over the Internet using XML.

A sample SOAP message looks like the following

```
<?xml version="1.0" ?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:
 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
 <m:SearchResearchProjects xmlns:m=
     "http://harris.cise.ufl.edu/ SearchResearchProjects ">
     <area> Mobile Computing </area>
     <technology> Bluetooth </technology>
     <year> 2001 </year>
 </m: SearchResearchProjects >
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 4.3: Sample SOAP Message

The above SOAP structure is used to invoke an E-service named SearchResearchProjects. As it can be seen the SOAP structure is composed of a Body inside the Envelope. The first element inside the Body is always the method or the service name. All the children of this method element would typically be the parameters or other necessary information required to perform the method invocation. In this case the parameters that are used (with their values) are area, technology and year. It can also be seen that the type of these parameters are not denoted in the message and assumed known by programs parsing the request. The service provider and the user should know the semantics or the meaning of the parameters associated with the function beforehand. For

example the tag area in the above example refers to the area of research (and not area as associated with real estate available, alternately it could have been made as research_area). SOAP here provides only nametag standardization for this method, guaranteeing that this method can be invoked, as long the parameters names provided in the query are as required.

Apart from the basic data type information like area and year, complex data structures can also be represented in suitable SOAP-XML format. Again, it is the responsibility of the program that uses this message to parse this parameter data and interpret it in the way it is intended.

XML-SOAP Parsing

The XML-SOAP parser module that is indicated in the E-services Adapter architecture diagram does the job of parsing this SOAP-XML structure, identifies and extracts the required information. This module primarily uses an XML parser in order to parse the client request that comes in the form of a SOAP message. It identifies the method to be invoked and also creates a hash-table data structure containing the parameter names and values. Modules like the ServiceInvocation use this parameter value information in order to supply the actual parameter values while performing the method invocation on the object. In addition client programs also reuse the parser modules in order to frame SOAP coded requests and interpret SOAP responses. Suitable data extraction APIs are provided at the client end in order to extract the response data from the SOAP response message.

Service Invocation

The ServiceInvocation module forms the core part of the E-services adapter. This is the module responsible for loading the Java class that implements the service and

performing method invocation. The ServiceInvocation module uses the ServiceMapping module to find the class file that implements the required service. Using this information it loads the required class file into memory. It also uses the mapping information to query the object file and check if the object implements the required method or function. This idea of querying the object to find the available or required method is a very powerful run-time facility that is provided by the Java Reflection API [REF01].

The ServiceInvocation module plays a pivotal role in providing service invocation capabilities. This module heavily uses Java Reflection, a run-time object introspection facility. It ensures that the object being used or the method being invoked as well as the parameters are correct by type and order. In order to perform method invocation during reflection, suitable parameter list have to be created so that they match in type and order. The ServiceInvocation module obtains the parameter information form ServiceMapping modules. The ServiceMapping is a very simple representation of method semantics, similar to IDL representation in CORBA and is maintained for every service (an equivalent of a function). In a similar way mapping information are maintained for service and the object providing the corresponding service. So when a request is received for a particular service, this mapping information is used to load the required object and perform method invocation on it. For simplicity, all service invocation is performed on individual objects and instances of objects are not shared for two or more invocation occurrences of the same service.

Apart form doing the basic and most important function of method invocation, the ServiceInvocation module also handles “long-running services”, which are asynchronous method calls that may take an arbitrary time for completion of execution. An example is a

copy-editing service, part of which can be automatic like spell checking and part of which can involve manual verification of the document. In this case, since manual intervention is involved, the service may take an arbitrary time to execute to completion. In such cases it is not desirable to make the client process wait until the operation is complete; in most cases this is not feasible.

Long Running Services

The service implementers are expected to provide the nature of every service that is being implemented, meaning implementation objects should support queries, which determine whether a service is asynchronous or synchronous in nature. In implementation terms, the service provider objects are required to implement a certain set of interfaces that will be used to perform the above queries. The purpose of these interfaces is to make these services participate in a workflow or issue queries in case of a long-running service. The interfaces also have workflow specific queries that must be implemented if the service wants to participate in a workflow.

The ServiceInvocation module uses these interfaces to query the service object and find the nature of the method call; specifically it determines whether the method is a long-running asynchronous method call or a synchronous method call. In case of a synchronous method call, no complexity is involved for the ServiceInvocation module. The method call is just invoked and the result is obtained and passed to the higher layers for processing. In case of a long-running asynchronous call, the ServiceInvocation module takes a different approach.

The ServiceInvocation module first identifies a long-running process by querying the service object using the transaction interface (that must be implemented by the service provider). When it is found that the service under consideration is a long-running

service the ServiceInvocation module launches a separate thread to invoke this method. At the same time, it creates and assigns this service a unique transaction id and returns this id to the client. In this way it implicitly tells the client that the result of the current execution will not be available immediately. The module manages running the thread and storing the result of the invocation when the service execution is over. The result of this long-running service in this case has to be pulled by the user using the previously sent transaction id as the key, and the result (if available) will be returned.

The ServiceInvocation module uses a transaction object repository, in order to store the objects in execution that provide the asynchronous services. This repository is used to provide response to the transaction probes or queries that the client will request in case of an asynchronous service invocation.

The lifetime of these objects that are used to perform method invocation is an interesting issue that should be managed. In case of synchronous services, these objects are destroyed, as the result is available immediately and it is safely assumed that this objects will not be referenced for anything else apart from the service for which it was created. In case of long-running services, these objects are suitably stored, since transaction probes are expected until the result of the object is obtained. Even if the long-running service under consideration is complete, the object cannot be destroyed until the result of the service invocation is communicated to the client. So these objects are destroyed only after making sure that the results of these long-running services are communicated to the client.

## Transaction Adapter

The ability to make E-services participate in a workflow and allow other service inquirers or users to utilize an E-service and include it as a part of their workflow is one of the main objectives aimed to be achieved by the BizBuilder framework. In order to achieve this, the framework has to facilitate the service inquirers to query about the status of execution of an E-service, so that they can manage and control their workflow. The transaction adapter component is the workflow support interface intended for the service inquirers.

The transaction messages received from the users are different from the ordinary E-service invocation messages. If the E-service adapter receives a transaction message, it delegates the message to the transaction adapter. The transaction adapter takes over and executes the queries, by using the transaction objects repository that was earlier updated by the E-service adapter with suitable transaction objects.

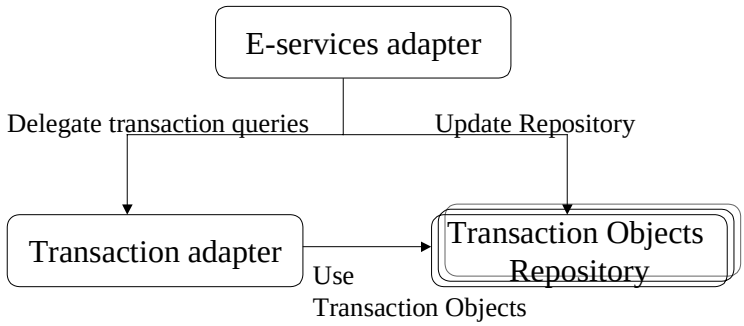


Figure 4.4: E-Services Adapter – Transaction Adapter Interaction

Support for Transaction Probes

An E-service should be able to participate in a workflow in which case it should support transaction specific queries like “how much of the service is done”, “ What is the expected completion time”, “what is the additional cost for early completion by mm/dd/yyyy” etc. These types of queries can be raised by a workflow composer, who may use the service under discussion to perform a small portion or a part of an activity within a workflow. Such queries and their result help the workflow composer to dynamically modify or manage the execution of workflow. This sort of scenario can be applied to a fault-tolerant and a critical workflow process in which a failure in one service or activity of a complex workflow should not affect the whole workflow as such, instead the workflow should be able to dynamically find and utilize an alternate service.

The Transaction adapter provides the necessary functionality to handle these queries, which are implemented in a manner similar to other services implemented by the service object. But in this case, transaction specific queries are identified, and suitable responses in this case have to be created. These are done with the help of sub-modules like the ResponseGenerator and transaction objects repository. The adapter determines the suitable object from the transaction repository and invokes the specific transaction query on the object. It is also important to mention here that all transaction queries have to be ultimately answered by the service provider objects, since the objects are the ones that actually implement the service and know the status of the execution. The BizBuilder provides this messaging infrastructure to query the service objects (that should have already implemented the transaction interfaces) using the transaction probes.

## Web-Server Framework

One of the main requirements of the BizBuilder framework is to provide the ability to access these E-services using standard web interfaces or protocols. In particular, all the E-services should be accessible using HTTP. In order to provide such a facility, the Servlet framework that runs on Apache Tomcat server is chosen. The SOAP server module performs this function, running a server process as a servlet that waits for the SOAP coded XML requests from clients and sends back responses. In a sense, the whole architecture in this case runs under the umbrella of a servlet process. The servlet process waits for individual client requests and each request is run as an individual thread and is managed by the underlying servlet engine. So the SOAP server module provides the interface to the external world in order to access BizBuilder created E-services across the Internet.

## Service Registration

The Service Registration modules provide the interface to the broker components, which serves a central repository for storing details like service description and the corresponding service-provider information. The E-services broker architecture supports the notion of brokering community, i.e. a group of brokers can collaborate and exist as a part of a network. But the broker acts as a middleman that does the job of finding a suitable service for a service requestor. The matchmaking is done based on the service requested, its category, relevance, type and the criteria used to select the service. Based on the results of the matchmaking, suitable information about a candidate E-service is returned to the service inquirer. On the other hand, the broker also interacts with the service provider to allow the latter to register its services. The registration details are the

only information that the broker has about the service provider that is used in the process of matchmaking (i.e. finding a suitable service for the clients). Thus the BizBuilder framework needs the service registration modules to register the E-services with the broker. The sequence of steps that the service provider does in order to register his services with a broker is detailed in the implementation and in the scenario analysis chapters.

Registration Information

The service provider registers to a broker, all relevant information regarding an E-service that may be used for matchmaking or in search criterion. The broker architecture in our Internet workflow system is based on the UDDI – Universal Description Discovery and Integration, an upcoming standard for service publishing and searching in the e-business domain. Since the focus of UDDI is to create a global uniform scheme to describe and find business, it helps our broker architecture to solve problems of standardization. The UDDI architecture is based on a web-services description format named WSDL, which in turn is based on XML to describe the interfaces of a service. Since our architecture also needs a rich service description language for E-services, we chose to use the UDDI architecture and the brokering community is built on top of it.

The service registration information that is sent to the broker follows the WSDL specification. By itself the WSDL format is very comprehensive and defines services in terms of endpoints, their interfaces etc. We use a subset of the WSDL features to define our E-services to the broker. In particular, our E-service description includes details like service name, service provider name and access point, business name (used to group and categorize business) and service-specific information. The service-specific information in this case would include details like service name, the parameters to be supplied, their

types and details about optional and non-optional attributes. In addition, we plan to support constraints specification in these descriptions. Service specific constraints will be included and specified in these service description documents, which will be used by the broker to perform matchmaking based on the constraints. The constraints include value and range constraints for attributes, and inter-attribute constraints.

A sample service description document looks like the following.

```
<?xml version="1.0" ?>
 <definitions name="StockQuoteService”>
        <operation name="getQuote">
                <input message="Symbol" type=“xsd:string”/>
                <output message="Quote" type=“xsd:string”/>
        </operation>

        <binding name="StockQuoteServiceBinding“>
           <operation name = “getQuote”>
             <soapAction="http://www.stocks.com/Services”/>
           </operation>
        </binding>
 </definitions>
```

Figure 4.5: Sample Service Description Document

## Negotiation and Contract Adapter

In order to provide a complete E-services framework, the BizBuilder framework will support for Negotiation and Contracts. This module provides a simple negotiation framework, where the service inquirers can negotiate for a particular E-service to be used. All negotiations will be carried out on a per-service, per-client basis. The Negotiation protocol is a synchronous protocol, in which the negotiating parties use a pre-defined data format to communicate messages. Once an agreement is reached upon the criterion in question (like cost, quantity etc.), a contract is established and the service inquirers are bound to the contract. In other words, any particular service provider who

enters into a contract will be provided a particular service with attributes and quality of service as specified in the contract. Support to modify existing contracts will be provided by these modules and the contracts once created are stored in persistence storage to be used later during actual service invocation.

# CHAPTER 5
# IMPLEMENTATION

This chapter describes the implementation details of the BizBuilder project. The implementation of the various components and the API's and tools provided by the framework are detailed.

## Web Server Framework

E-services by definition, should be accessible on the Internet using standard protocols and representation format. HTTP is the universally accepted protocol that is used in the e-commerce space. Hence, E-services should also be accessible using HTTP as the transport protocol. In order to enable this feature, BizBuilder uses the Servlet-HTTP framework provided by Tomcat-Apache server. Servlet based HTTP access is a natural choice, since servlets support java objects to be used seamlessly within them. Typically, every E-service that is enabled by the BizBuilder framework runs as a servlet process, when invoked using a Uniform Resource Identifier (URI). The Tomcat-Apache server launches a servlet thread for every request to complete its execution.

All messages that are exchanged between the service provider, requestor and broker are in SOAP-XML format. In order to invoke a service, the servlet process receives the SOAP-XML message and gives it to the higher layers for parsing and suitable service invocation. After the service invocation is complete, the servlet process returns the result of the invocation again in SOAP-XML format. The tomcat servlet

framework handles the concurrent execution of servlet threads when requests are received simultaneously.

All XML-SOAP messages are parsed using the oracle XML-DOM [ORA99] parser implementation.

### Service Invocation Using Java Reflection API

As mentioned in the previous chapter, the ServiceInvocation module forms the core component that handles service invocation. The ServiceInvocation module acts as an object management tool, because it is responsible for creating objects in memory, managing them by invoking the required methods on them and destroys them when they are not needed. This is the most fundamental facility that defines BizBuilder as an E-services framework.

All service invocations have to be performed on objects that are created or loaded in memory just in time when needed. This is significantly different from the usual client-server paradigm, where the server process knows about the objects to be loaded at compile time or the objects are pre-loaded. In BizBuilder, objects that are used to provide services can be added to the framework any time. This means the framework can be still be used for new objects without recompilation or modification, because using certain user-supplied information the BizBuilder framework will be able to load newly created objects and perform the required method invocation on them. This facility is possible only due to Java Reflection APIs, which are the run-time object introspection feature.

To perform a service invocation, the framework determines the object on which the service has to be invoked. This service to class name mapping is maintained as file information and it has to be constantly updated for new objects and services. Once the

class to be used is discovered it can loaded using reflection API's. The service to be invoked and the parameter-values that are needed for the service invocation are obtained from the parser modules.

One crucial piece of information that is needed to perform the object invocation is the order of the parameters, since this is a run-time facility. So the parameter type (optionally) and order information is stored as a Service Signature Description SSD file that can be used by the framework. It is important to create such SSD files for new services or modify SSD files if the actual signature of the methods that is implemented changes. A simple SSD description file for a service like ProcessOrder looks like the following.

Table 5.1: Structure of a SSD File

| Parameter Name | Parameter Type |
|---|---|
| UserID | String |
| Order | Order |

The above-depicted structure indicates that the ProcessOrder service has two parameters; the first parameter is UserID whose data type is String, and the second parameter is Order whose data type is Order. In other words the method signature (without return type and exceptions) is indicated by this information. The parameter names indicated here are also used while describing the service and while registering the service with the broker. This information is stored using the ServiceMapping modules described in earlier chapters. The SSD file provides the semantics of the method, namely, it says how to treat a string (as an UserID in the above case), how to treat the second parameter and so on. The BizBuilder framework provides the necessary user interface for

the service provider to take an existing object implementation (as a class file in java) and create the required SSD and other information required for using the new services.

The ServiceInvocation module uses these information to frame the exact method call and performs the invocation using reflection API's provided by Java. This technique of method invocation is not completely safe and secure, since no compile time error checking is possible and any errors would result in run-time exceptions. The ServiceInvocation module suitably handles any run-time errors that can occur, e.g., the user-provided value is not of the correct data type or format.

Support for User-Defined Objects

While performing the actual E-service invocation, user-defined objects can be used apart from the basic data-types supported by java. The BizBuilder API converts this user-defined object into suitable XML representation that can be transported to the service provider. On the other end, the BizBuilder API provides support to construct this java object, from the XML representation and uses it to perform the actual method invocation on the java object. Though this support is provided only for user-defined objects in java, this will be extremely useful in scenarios where user-defined objects are extensively used to represent composite information (like Order, User-Info etc). It is also worth noting that the API will support any user-defined object, and XML representation will be provided only for the public attributes of the user-defined objects. Also the XML conversion is provided only for attributes of basic data types and not for Java data structures like Hashtable, Array etc. In other words, the user-defined objects in this case represent a collection of simple data types. So an order object with attributes {orderno =O1055, itemno = A4500, cost=500} will be converted and represented in XML as in the following diagram.

```
<Order>
 <orderno> O1055 </orderno>
 <itemno> A4500 </ itemno >
 <cost> 500 </cost>
</Order>
```

Figure 5.1: XML Representation for a Sample Order Object

Long-Running Service and Transaction Queries

The BizBuilder framework supports long-running services and transaction queries by providing a transaction interface in java, which every service provider has to implement, in order to support long-running services and participate in workflows. The transaction interface includes methods like the following

- ***GetStatus*** provides the current status of execution of the task or service; typical return values include status like expected completion in 2 hours.
- ***IsComplete*** queries whether the service has completed executing or returns the percentage of task completed; typical return values are 80% completed.
- ***Query*** is a general purpose query message intended for the task (e.g., can the task be finished in 1 hour). These types of messages can be used to interact with a service during execution to perform activities like re-negotiation.
- ***Tell*** is a one-way notification message to the activity or the task.
- ***Commit*** is used to commit the execution of the current task or service.
- ***Abort*** stops the execution of the current task or service.
- ***GetResult*** provides the result of the current execution of the service or activity.

- ***IsSynchronous*** provides information on whether a service is synchronous or long running.

All of the above mentioned queries depend on the implementation and the current execution status of the service or the task. Only the actual service provider or the object that implements the service can supply all these details or the result of the queries. Hence, all the above-mentioned queries are provided as a Java interface and the service provider is required to implement these interfaces in order to support or participate in workflows.

The BizBuilder framework finds the nature of service that is being invoked, i.e. whether it is a synchronous or a long-running query. In the case of a synchronous query, the result will be returned and the transaction adapter need not perform any function in this scenario. In case of a long-running service, the transaction adapter comes into play. It launches a separate thread of execution for the current service. Apart from this it also stores the object and generates a transaction id, unique to this particular service. This id serves as the identifying key, using which the client can send transaction probes to the transaction adapter. The transaction adapter uses this transaction id in identifying the actual object on which the method is invoked, using the map maintained in the transaction repository.

Support for E-Service Provider

The BizBuilder framework provides a user-interface using which the service provider can use utilize java objects and create an E-service. The process of creating this E-service is automated and is guided by the BizBuilder user-interface. A sample creation is detailed in the next chapter in scenario analysis. The E-service creation process involves creating a set of files (like service-mapping, SSD files) to be used by the

framework for service invocation. All these required files are created and populated by the BizBuilder UI and the service provider need not manually perform any other tasks. Hence, the support for the service provider is given in the form of an easy to use User-Interface to manage the E-service creation process. Apart from this tool, the service provider has to run a web-server, to provide access to the E-services and register this URI (through which the services can be accessed) with the broker.

APIs for an E-Service User

BizBuilder provides a set of APIs in java for a user of an E-service. The user in this case can be a workflow engine or a simple user of the service. The important APIs provided by the framework and their purpose is listed below.

Table 5.2: APIs for an E-service User

| Method Name | Purpose |
|---|---|
| CreateXMLMessage (String servicename) | Used to create an XML message for a particular service, the servicename is given as the parameter |
| AddParam (String paramname, Object inputobject) | Used to add a parameter (or value) in the service request. The paramname and any java object are given as parameters to this method |
| SetServiceURI(String URI) | Used to set the URI of the service provider |
| Call(String xmlrequest) | Used to invoke the actual E-service, giving the XML request message, return the result as an XML message |
| GetResult(String xmlresponse) | Used to get the result as a java hashtable with name-value mapping, takes the XML E-service response (obtained from the above Call method) as a parameter |

The above-mentioned set of Java APIs (along with other utility functions) can be used by a service requestor to access an E-service, provided the user knows the format of the XML request message and the service URI. These APIs use an XML-DOM parser internally. A sample user-interface for an E-service client is provided along with the BizBuilder framework, that demonstrates the execution of an E-service. But a workflow engine can effectively utilize these APIs programmatically, to dynamically invoke an E-service from any service provider.

## Service Registration and Broker Interactions

The Broker component of the Internet Workflow system is based on UDDI and is organized as a hierarchical community to support various business domains and adaptation. The community is organized into layers consisting of brokers and super brokers forming a tree like structure. The BizBuilder framework by itself does not provide any support for broker interactions. But the service provider can register his services using the interfaces provided by the broker [JAG01]. The brokering community provides suitable web-interfaces for all interactions with the service provider.

In a typical scenario, the service provider finds a broker of his interest and registers himself with the broker. The service provider then browses the various service templates and provisions that are available with the broker, that might relate or best describe his service. The service provider uses the business name or classification maintained by the broker, as the key to search the required templates. After finding a suitable service template of his interest, the service provider actually creates an instance of the template and registers his service with the broker. The service template is different from the instance of it that is used in registration, in a sense that the instance contains the

actual description about the service that is being provided and the access points for the service (URI) etc.

It is also possible that the service provider may not be able to find a suitable service template that best describes his service. In this case the service provider can take two approaches. If the service that is being provided is significantly different from the existing ones or is completely new, the service provider can suggest a new template to be included and published by the broker. This suggestion for the new template however will be ratified by the broker and may or may not be accepted. On the other hand, if the service provider can choose to modify his service definition or interfaces, in order to exactly comply with a template and then follow the template and register his service based on the template of his interest. The protocol of message passing that the service provider follows to register his service to a broker is depicted below.

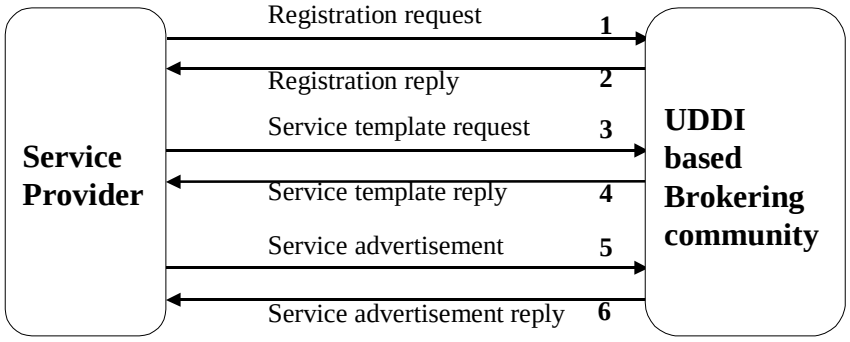


Figure 5.2: Protocols of Communication Between Service Provider and the Brokering Community

Overall Scenario of Operations

The BizBuilder E-services framework is a part of the broader vision, the Internet workflow project. The various components that participate in a workflow scenario are the service providers (using the BizBuilder framework), the service inquirers and the broker module. The workflow engine and the processing modules can be located in all or any or even none of the service provider and enquirers. The typical sequence of operations that occurs in identifying and invoking a service is provided depicted by the following diagram.

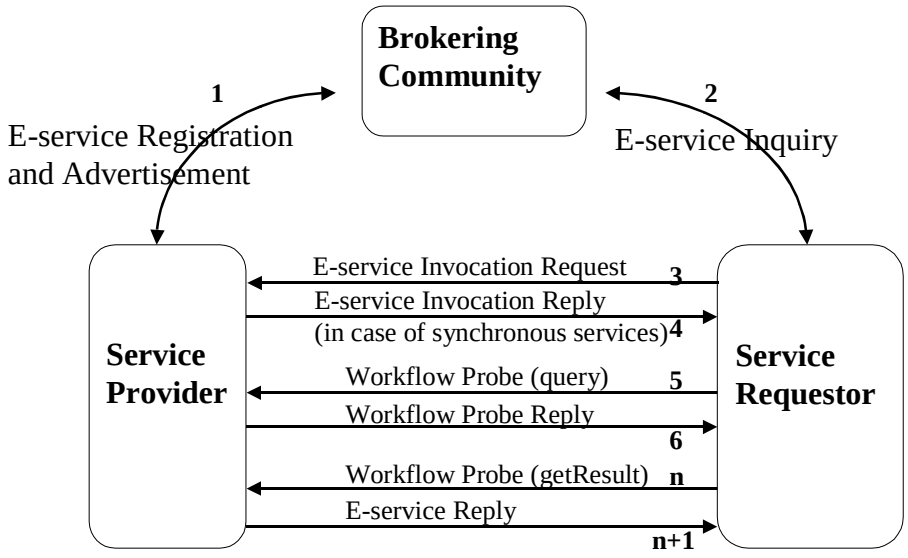


Figure 5.3: Overall Sequence of Operations in the BizBuilder Framework

# CHAPTER 6
# SCENARIO ANALYSIS

In this chapter, we take up a sample scenario and analyze how the BizBuilder software can be used in hosting E-services and how these E-services can be made to participate in a workflow and reply to workflow related queries.

## Proofreading – A Sample E-Service

### Tasks in Proofreading

Proofreading of a technical document is a practical application that can be thought of as a service provided by an organization or a small business or an individual on the Internet. The activities done during proofreading includes tasks like spell checking a document, grammar and thesaurus checking. Beyond these basic facilities, proofreading can involve more complex tasks. A proofreading of a technical book is completely different from proofreading of a document. For example, proofreading a technical book may involve actually verifying the contents, verifying the validity and correctness of a diagram etc. Again, this verification of the contents of the document is domain specific, meaning that a chemist is needed to proofread a proposition of a new chemical component; a computer scientist is needed to proofread or comment on a software architecture proposal and so forth.

The proofreading example illustrated above demonstrates more than one specific task that is involved. Proofreading, by itself, is a composition of various simple sub-tasks like spell checking, content-verification etc. Also part of this sub-task can be automated

using software components and rest of it needs manual attention. Spell checking software components are available, whereas content validation and verification needs to be performed by a human. In other words the proofreading example is an apt candidate for an E-service, that can provided as a value-added services targeted to document creators, publishers etc.

Using BizBuilder to Create a Proofreading E-Service

For this scenario, we will assume an individual X, who does proofreading of documents, of a particular domain, say biotechnology systems. We will also assume that X has some software components developed in-house or purchased from a third-party developer. X can use the BizBuilder software to create proofreading E-service, that would be available on the Internet. On the contrary, this proofreading service can also be provided on the web, using HTML pages and web-servers, similar to services provided by yahoo's web site. This is typically a business-to-consumer scenario, where customers browsing the web, looking for this kind of service are targeted. If a customer wants to use this service, he or she is needed to fill in the required details like, personal information, credit information and uploads the documents to be proof-read. Typically the proofread document may be emailed back after a couple of days.

BizBuilder on the other hand, by providing this proofreading service as an E-service, brings into picture the business-to-business scenario. Through suitable advertising facilities provided by BizBuilder, the proofreading E-service can be registered with a brokering site that serves as the hub or the repository for searching available services. Once the proofreading E-service is discovered, more details about the nature of service and facilities provided can be obtained from the broker. Now the proofreading E-service can be invoked or utilized programmatically since the semantics

of the service would have been well understood. All the above mentioned steps of discovering a service, invoking and utilizing them can be automated, meaning a business like Book-publishing service providers can use this proofreading service as a part of their business workflow.

Creating the Proofreading E-Service

The sequence of steps to be followed by X to create the Proofreading E-service is mentioned below.

1. X selects the existing software components (assumed to be in the form of Java objects in class files) that will be used for performing the service.
2. X describes the service, providing details like parameters expected in order to perform the service, results returned and access points (URI) etc. in XML.
3. X provides other required information to BizBuilder for E-service invocation.

A sample illustration of the sequence of steps is provided in figure 6.1.

Supporting Transaction Queries and Long-Running Processes

In order to support transaction queries and participate in workflows, the service provider has to implement the transaction interface provided (in Java). In particular it is the responsibility of the service provider to implement every transaction query according the status of the execution of the service and provide the corresponding result. A sample demonstration of the service request, transaction probe and result is shown in figure 6.2 and figure 6.3.

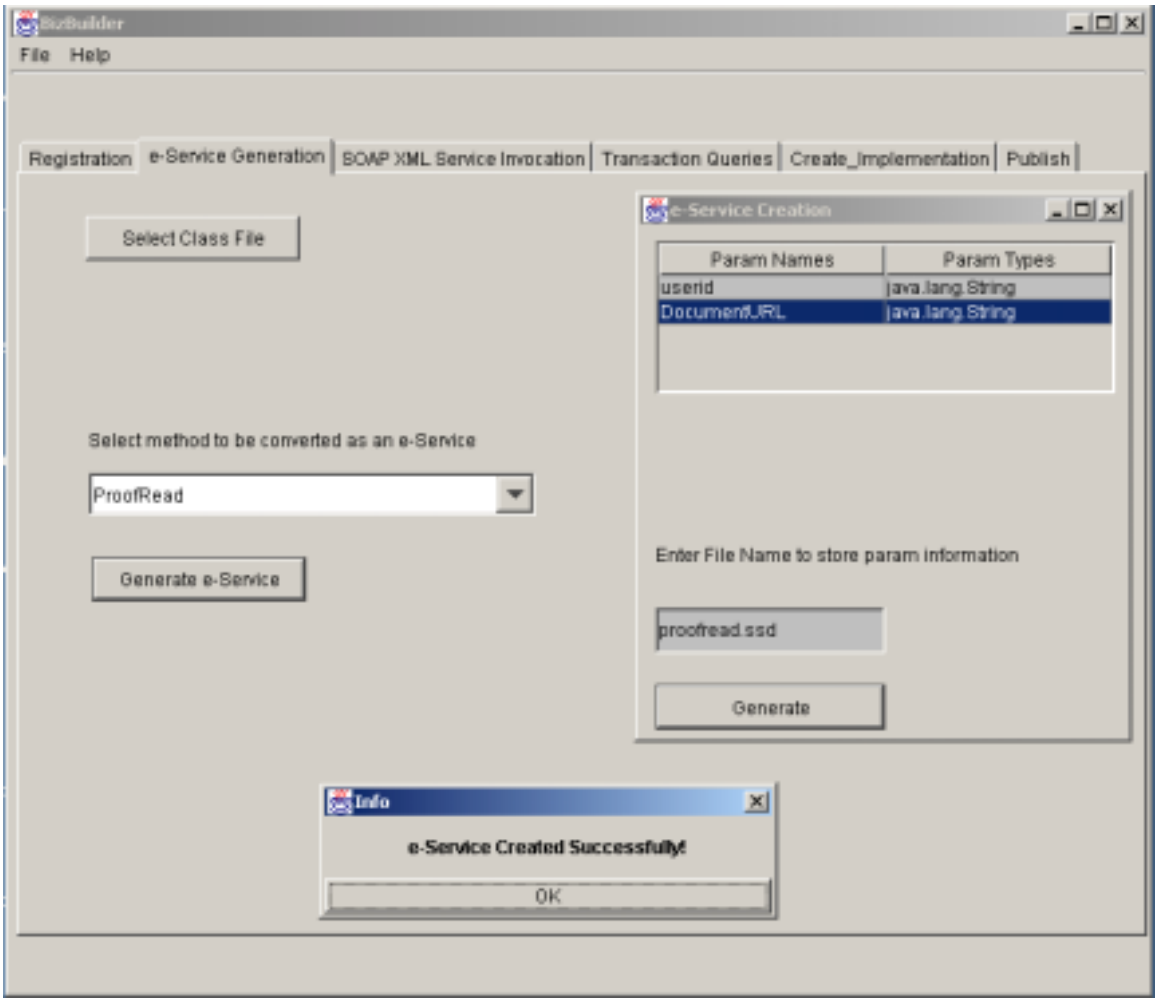


Figure 6.1: Screen-Shot of E-Service Creation Process Using BizBuilder Tool

Registering the E-Service

The BizBuilder framework provides the registration support to register the service provider with a broker. Additionally the service provider can browse the various categories and templates maintained by the broker and can publish the E-service to one of the existing categories. The sequence of steps is shown below

1. The service provider registers himself with a broker.

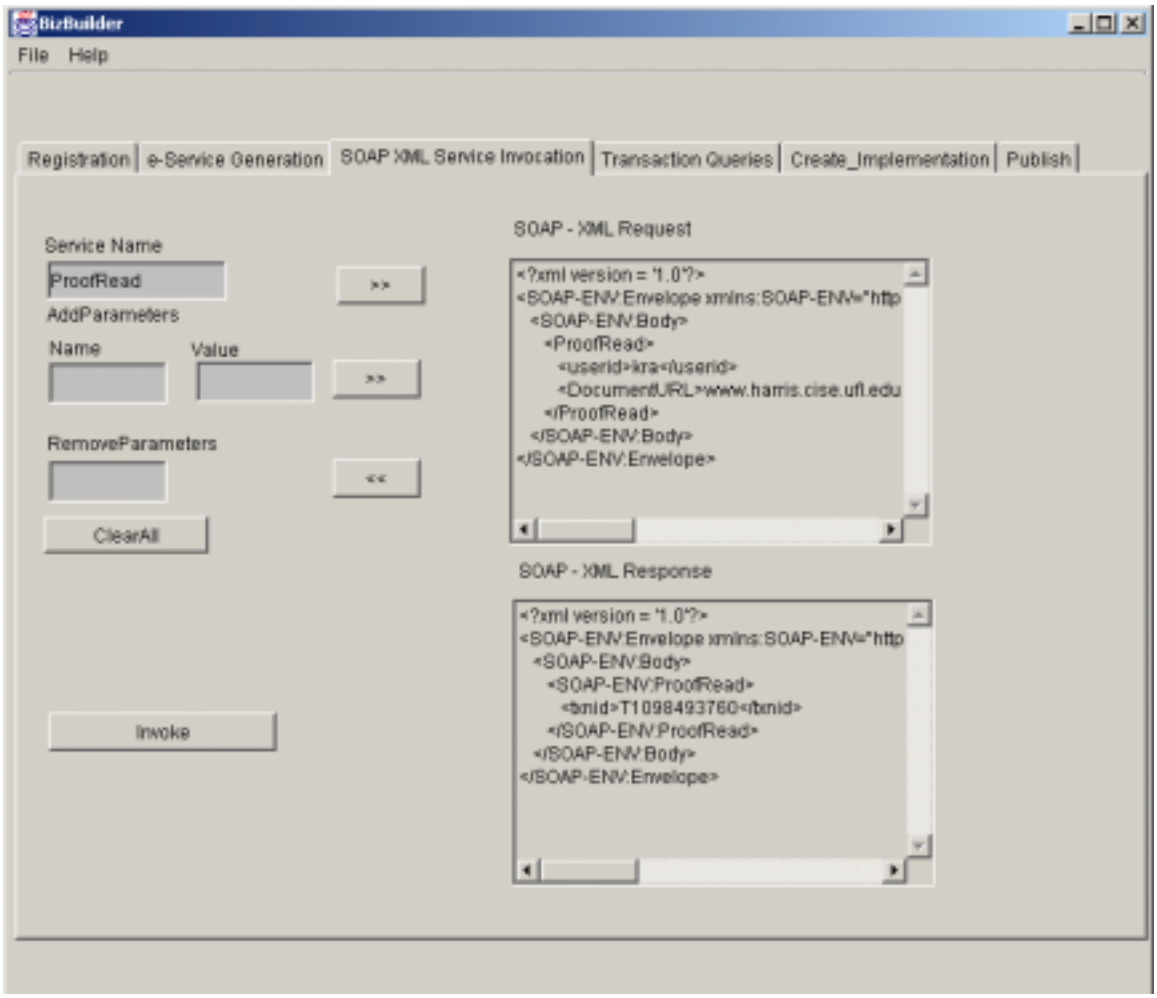


Figure 6.2: Screen-Shot of E-Service Invocation Using BizBuilder Tool

2. The service provider browses the set of available service descriptions grouped as templates with a broker. (These templates are classified based on the business or the type of service provided.)
3. The service provider chooses a template of interest.
4. The service provider creates a description of the service he is providing, as a WSDL document. The service description is intended for the template he has chosen and he registers this service description that contains the details about the service, access points, etc with the broker.

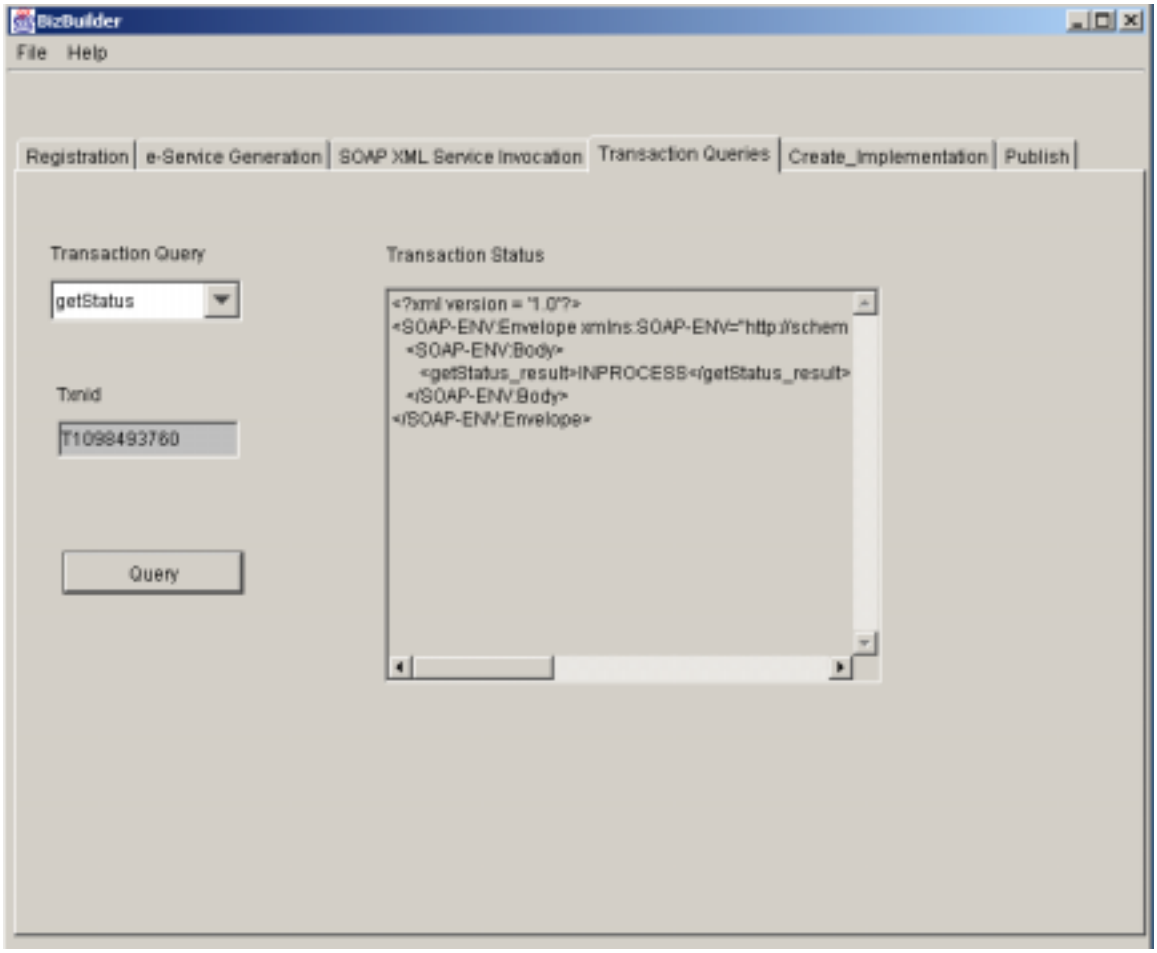


Figure 6.3: Screen-Shot of Transaction Query Invocation Using BizBuilder Tool

5. Alternatively, instead of step 3, the service provider may not be able to find a template that matches his intended service description; in this case he can choose to modify his service to match any of the available templates or create a new template and submit it to broker for approval. In case this template is approved, he can continue to register his service under this newly created template.

Book-Publishing – The Composite E-Service

The previous section described the creation of an E-service using the BizBuilder system. As stated earlier, using this framework, a composite service can be created by combining existing E-services and executing them in a pre-defined order as a workflow of activities. A sample scenario is illustrated below.

Book-publishing is a composite service, which can be thought of as a combination of individual simple services like proofreading, formatting and binding, shipping and handling. So a new value-added service like Book-publishing can be provided, which will serve as a one-stop point for documents or book printing services. In other words, this is similar to a scenario of creating a virtual enterprise by using already available third-party service, but all of them combined and utilized in a strategic manner to provide a value-added service.

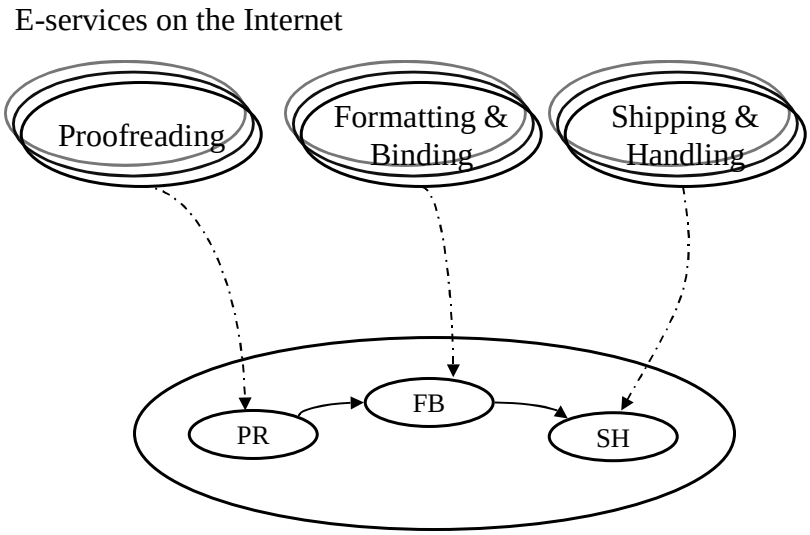


Figure 6.4: Composite E-Service Creation Example

The BizBuilder E-services framework supports this notion, although it does not provide the workflow creation and management tools. The workflow creation/management tools are covered in a separate project work, which is also a part of the overall goal, the Internet workflow system. Figure 6.4 depicts the scenario of creating a composite service out of existing services.

It can be seen that multiple E-services of the same kind are available, for example, there are multiple instances of Proofreading service, and the composite service can use any one of the available Proofreading services, based on quality of service attributes provided by the individual service.

# CHAPTER 7
# CONCLUSION AND FUTURE WORK

This chapter is a brief overview of the goals accomplished by this thesis. It also suggests the possible future work that can be carried on in the area of E-services and Internet workflow.

## Goals Accomplished

This thesis explained the need and importance of E-services in the business-to-business e-commerce space and in the area of Internet workflow. It also detailed how BizBuilder would enable seamless integration of services on the Internet. This thesis also presented a survey of the related research and state of the art in the domain of E-services and Internet Workflow and these various implementations were compared and contrasted. The problem definition of this thesis was stated and a motivating scenario was presented to demonstrate the need of a framework like BizBuilder, how this would support the integration of diverse systems on the Internet and how it will create more business opportunities, by enabling the participation of E-services in a business process or a workflow.

The BizBuilder architecture proposed by this thesis was elaborated. A detailed description of the various components of the architecture and their functionality was explained. The architecture also detailed how an existing service (in the form of a Java class file) can be converted to an E-service, explaining the role of XML/SOAP and HTTP and how they enable the creation of E-services. It also detailed the various transaction

interfaces that are supported by BizBuilder and the user-requirements needed for enabling the participation of these services in a workflow. Detailed explanation on how long-running services are handled was also provided. The architecture also covered the aspects of service registration with a UDDI enabled brokering community and the facilities provided for the same. The implementation chapter detailed the system that was developed and the software used for building the BizBuilder framework. Finally, a scenario example was presented to demonstrate the practical purpose and utility of the framework.

## Future Work

The BizBuilder framework and the topic of Internet Workflow provide ground for further research and development work, some of which are detailed below

- The BizBuilder framework provides support only for Java objects; this support can be extended to include legacy systems and other popular object implementations like COM, CORBA, and Java Beans etc.
- XML schema can be used extensively along with SOAP, to enforce more control and validation for SOAP messaging
- Service description could also use XML schema to provide richer descriptions that would include user-defined or business objects apart from the data types that are supported by WSDL
- On the lines of the above-mentioned point, support to represent business objects in SOAP for transporting them and using them in E-services would be a desirable feature

- Service descriptions can be extended to include service constraints that may be relevant to users and used in match-making by the broker
- Transactional support provided for long-running services can be extended. Some of the transactional services could be protocols like prepare_to_commit, commit, abort etc, that are not currently supported in the BizBuilder framework
- In addition to providing transaction support, BizBuilder can be extended by providing support for service negotiation, licensing and usage tracking for the services provided
- A visually interactive object management tool can be provided to the service provider using which one can monitor the execution of the tasks and can interact with transaction messages if manual response is needed

LIST OF REFERENCES


[BAN98] T. Bannon, “A short description of Web Trader,” http://www.objs.com/OSA/WebTrader-Desc.html, 1998

[BOL99] G. A. Bolcer, G. Kaiser, “SWAP: Leveraging the Web to Manage Workflow,” IEEE Internet Computing, Vol.3, No.1, January/February 1999

[CAS00a] F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, M. Shan, “eFlow: a Platform for Developing and Managing Composite e-Services,” HP Labs, Palo Alto, 2000

[CAS00b] F. Casati, S. Ilnicki, L. Jin, M. Shan, “An Open, Flexible, and Configurable System for E-Service Composition,” HP Labs, Palo Alto, 2000

[CHA98] “HP ChaiServer: An Overview,” Hewlett-Packard, Palo Alto, 1998

[CSA00] “C# Introduction and Overview,” Microsoft Corporation, http://msdn.microsoft.com/vstudio/nextgen/technology/csharpintro.asp, 2000

[DAN98] A. Dan, D. Dias, T. Nguyen, M. Sachs, H. Shaikh, R. King, S. Duri, “The Coyote Project: Framework for Multi-party E-Commerce,” Proceeding of the 7th Delos Workshop on Electronic Commerce, Greece, 1998

[ESP00] “Enabling New Business Opportunities, A Primer on e-Speak,” Hewlett-Packard, Palo Alto, 2000

[FIN00] P. Fingar, H. Kumar, T. Sharma, “Enterprise E-Commerce,” Meghan-Kiffer Press, ISBN 0-929652-11-8, 2000

[HEL01] A. Helal, M. Wang, A. Jagatheesan, R. Krithivasan, “Brokering based Self Organizing e-Service Communities,” Proceedings of the Fifth International Symposium on Autonomous Decentralized Systems with an emphasis on Electronic Commerce, Dallas, 2001

[JAG01] A. Jagatheesan, A. Helal, “Architecture and Protocols for Sangam Brokering Communities,” Internal Report accessible from www.harris.cise.ufl.edu/projects/e-services.htm, 2001

[KUN00] H. Kuno, “Surveying the E-Services Technical Landscape,” HP Labs, Palo Alto, 2000

[LAR96] D. Larner, “Migrating the Web toward Distributed Objects,” Xerox Corporation, Palo Alto, 1996

[MEN00] D. Mennie, B. Pagurek, “An Architecture to Support Dynamic Composition of Service Components,” Carleton University, http://www.ipd.hk-r.se/bosch/WCOP2000/submissions/mennie.pdf, 2000

[MEN01] J. Meng, S. Su, H. Lam, A. Helal, “Achieving Dynamic Inter-organizational Workflow Management by Integrating Business Processes, Events, and Rules,” Internal Report accessible from www.harris.cise.ufl.edu/projects/e-services.htm, 2001

[MSF01] “An Introduction to Microsoft .NET,” Microsoft Corporation, http://www.microsoft.com/net/intro.asp, 2001

[ONT00] “The Need for Shared Ontology,” www.ontology.org, http://www.ontology.org/main/page1.html, 2000

[ORA99] “Oracle XML Parser Documentation,” Oracle Corporation, http://www.oradoc.com/ora816/server.816/a76935/oracle_x.htm, 1999

[PAU97] S. Paul, E. Park, J. Chaar, “RainMan: A Workflow System for the Internet,” IBM T.J. Watson Research Center, New York, 1997

[REF01] “Java Reflection Documentation,” Sun Microsystems, http://java.sun.com/products/jdk/1.1/docs/guide/reflection/, 2001

[SOA00a] “Simple Object Access Protocol,” The World Wide Web Consortium, http://www.w3.org/TR/SOAP/, 2000

[SOA00b] K. Scribner, M. C. Stiver, “Understanding SOAP,” Sams Publishing, ISBN 0-672-31922-5, 2000

[SWA99] “Simple Workflow Access Protocol,” SWAP Working Group, http://www.ics.uci.edu/~ietfswap/, 1999

[TIW01] A. Tiwana, B. Ramesh, “e-Services: Problems, Opportunities, and Digital Platforms,” Proceedings of the 34th Hawaii International Conference on System Sciences, Hawaii, 2001

[UDD00] “UDDI Technical White Paper,” www.uddi.org, http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf, 2000

[VAS99a] V. Vasudevan, T. Bannon, “WebTrader: Discovery and Programmed Access to Web-Based Services,” Poster Paper at Eighth International World Wide Web Conference, Ontario, Canada, 1999

[VAS98b] V. Vasudevan, “Augmenting OMG traders to handle service composition,” Object Services and Consulting Inc., Baltimore, MD, 1998

[WEB00] “Web-Services Architecture Overview,” IBM, http://www-106.ibm.com/developerworks/library/w-ovr/, 2000

[WFX00] “Workflow Management Coalition Workflow Standard – Interoperability Wf-XML Binding,” The Workflow Management Coalition, http://www.aiim.org/wfmc/standards/docs/Wf-XML-1.0.pdf, 2000

[WSD01] “Web Services Description Language,” The World Wide Web Consortium, http://www.w3.org/TR/wsdl, 2001

[XMR00] “XML-RPC Specification,” XMLRPC.com, http://www.xmlrpc.com/spec, 2000

## BIOGRAPHICAL SKETCH

Raja Krithivasan was born on March 28, 1977 in Thiruvarur, Tamil Nadu, India. He attended Bharath Institute of Science and Technology, Chennai and graduated with a Bachelor's Degree in Computer Science and Engineering from, University of Madras in 1998. After working for a year, he decided to pursue his Master's degree in Computer Science and joined the University of Florida, Gainesville, in August 1999. Upon completing his master's degree, Raja will join Verizon in Dallas, Texas. A sports fanatic, he follows almost every game with extreme enthusiasm in cricket, volleyball and American football.