BizBuilder–An E-services Framework Targeted for Internet Workflow

Raja Krithivasan and Abdelsalam (Sumi) Helal

Computer and Information Science and Engineering Department
University of Florida, Gainesville, FL32611
Phone: (352) 392-6833
{rkrithiv, helal}@cise.ufl.edu
http://www.harris.cise.ufl.edu/projects/e-services.htm

Abstract. 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 by any consumer or business using the Internet infrastructure and standard protocols. A framework is needed by which new E-services can be created, or existing non-Internet services can be converted to E-services. Also, since these E-services are good candidates to be part of business workflows, a facility is needed to make them support and participate in such workflows. This paper describes BizBuilder, our E-services framework used to create these E-services. BizBuilder addresses the issue of transactions in particular and facilitates managing both synchronous and long-running E-services that can participate in workflows. Additionally, BizBuilder provides suitable support to register E-services to a Universal Description, Discovery and Integration (UDDI) enabled brokering community.

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 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) [10]. Businesses could use Internet as their communication medium to set-up business deals and utilize the services they need from other businesses, just in time when needed. A supply chain consisting of a manufacturer, distributor and seller can be taken as an example where a business requires the service of another.

1.1 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 consumers 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 inherently 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, using which he can compose a new service, by creating/managing a workflow of existing services on the Internet. This ability to provide an E-services platform, support the creation of composite services, and the construction and management of a workflow of services are the primary motivating requirements for BizBuilder.

1.2 E-services and Workflow

As depicted in Figure 1, the E-services framework will enable the creation of E-services that are accessible on the Internet and those that can participate in workflow. The other modules represented here include the workflow engine [11] that is responsible for creating, managing and executing workflow tasks, brokering communities that are used as the service repository for matchmaking. The brokering community [6] by itself is sophisticated and provides a mechanism for publishing, inquiring and matchmaking of E-services. In this paper, we concentrate only on BizBuilder – the E-services framework.

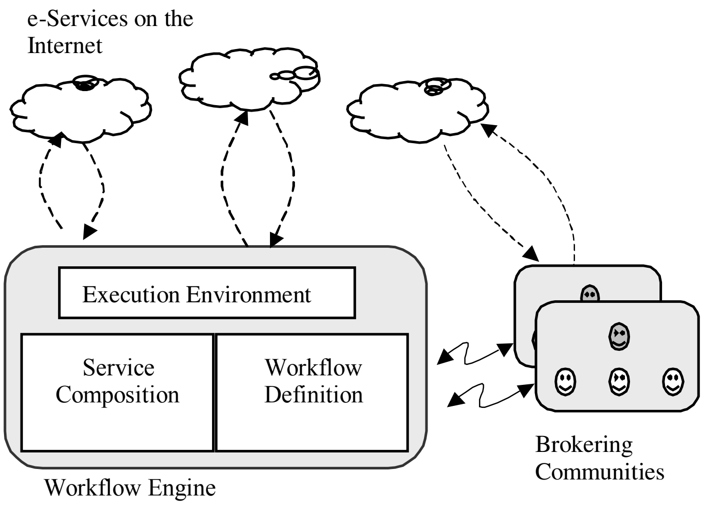


Fig. 1. E-services and Internet Workflow

1.3 Related Work

Distributed systems like CORBA offer an efficient way for service discovery and invocation, where the services themselves are physically distributed. However these systems are tightly coupled systems, meaning they need a homogenous framework to be used. Hence they do not lend themselves to being readily usable on the Internet. Architectures like WebTrader [4] and Coyote [7] have taken a different approach by using XML for representing service interfaces or contracts. On the other hand, workflow on the Internet has been dealt in systems like RainMan [5]. Dynamic composition of service components has also been discussed in [8]. Though all of these architectures are closely related to the idea of using services as the participating entities in a workflow, they do not deal with the concept of E-services, in which the focus is on providing services that are accessible using the standard data representation formats and protocols. In other words, these architectures propose a solution based on technologies like JavaBeans or RMI that are not readily inter-operable when considered in the diverse e-commerce space. Various architectures have been currently proposed based on the concept of E-services [1], [2], [3]. We have extended the concept of E-services to be used in conjunction with Internet based Workflow systems.

2 The BizBuilder E-services framework

The concept of E-services has become so prevalent that multiple definitions for the term E-service exist. Our framework assumes an E-service as "any service or functionality that can be accessed by a business or a consumer programmatically on the Internet, using standard representation and protocols". By mentioning about the representation and protocols for the Internet, we stress that any E-service should have a representation scheme that is not proprietary and one that is easily understood. Also the service should be accessible using standard Internet protocols like HTTP.

BizBuilder provides the necessary tools and framework using which an E-service can be created and utilized. In specific terms, the framework provides the facility to:

- Take an existing object (or service) implementation and provide an E-service wrapper,
- Provide a XML service description of an E-service (to represent an E-service),
- Provide the facility for advertising an E-service to a UDDI [9] enabled broker,
- Provide necessary tools and APIs to invoke an E-service on the Internet,
- Provide support for executing a "shallow workflow".

The BizBuilder framework targets a service provider who has a web-presence and business services to be provided, but the services not actually being available on the Internet (i.e., as E-services). Using this infrastructure the service provider can enable E-services, thereby making it available to both businesses and consumers. 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 existing services on the Internet. Though the E-services infrastructure does not directly support the creation and management of this workflow, it provides necessary support for an E-service to participate in a workflow by providing capabilities to respond to transaction-specific queries. We assume a workflow as a "shallow workflow", which can be viewed as a collection of related activities executed concurrently. In programming parlance, 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-added service (like the book-publishing example).

The E-service infrastructure also supports 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 span across organizational boundaries, and take an arbitrary amount of time to execute to completion. Therefore the result of such activities may not be provided synchronously. In this case asynchronous mechanism like event-notification or push, is possible in tightly coupled systems, but is complex in nature when considered in the e-commerce space. This is because the user or the service requestor may wait for the results (asynchronously, without blocking other activities), and hence acts like a server. But the requestors may not be treated as a server anytime in the Internet context, since the requestor can just be a browser client. Additionally, "callbacks" in HTTP cannot be easily achieved due to the nature of the clients and the HTTP protocol.

3 BizBuilder Architecture

This section details the overall BizBuilder architecture and the important components. Figure 2 depicts the architecture and shows the significant modules that constitute the framework.

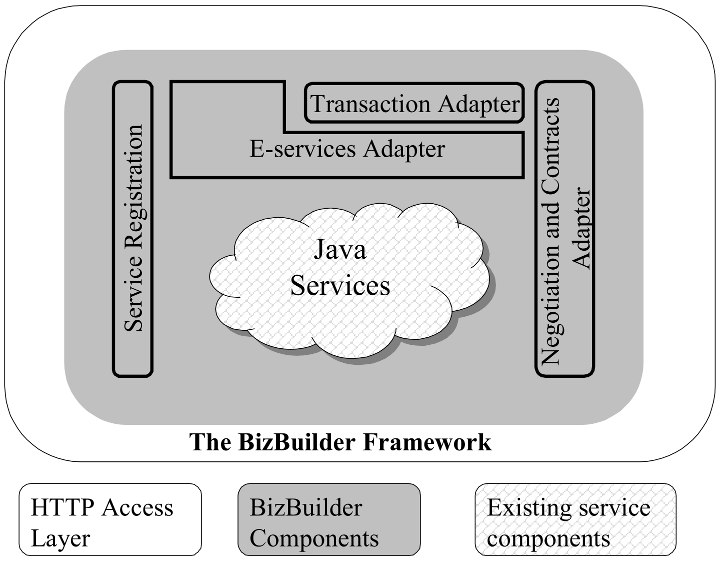


Fig. 2. Components of BizBuilder Framework

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.

3.1 Messaging Structure

One of the important requirements for E-services is to provide a messaging mechanism for invocation that is not proprietary, but easy to adopt. This was one of the drawbacks of traditional distributed systems, which required participants to have a homogenous structure as that of the service provider. The Simple Object Access protocol – SOAP, which specifically addresses the issue of RPC over XML, is used in BizBuilder for E-service invocation. Therefore, every E-service request and response is based on the SOAP format. Thus the framework has suitable support to read SOAP coded message, execute the actual services and reply results coded in SOAP format.

3.2 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.

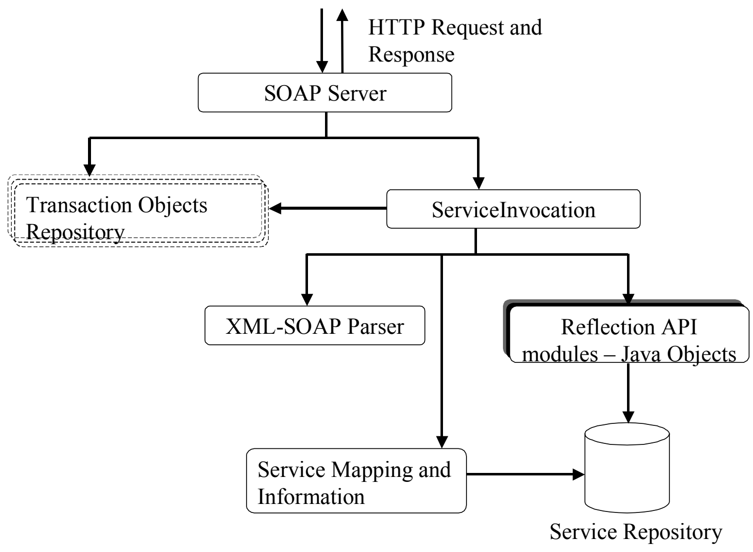

HTTP Request and Response
SOAP Server
Transaction Objects Repository
ServiceInvocation
XML-SOAP Parser
Reflection API modules – Java Objects
Service Mapping and Information
Service Repository

Fig. 3. Depicts the main modules of the E-services adapter and the interactions between them. An arrow from component X to component Y indicates that component X uses component Y

Table 1. List of Modules in the E-services Adapter and their Purpose

Module Name	Description
SOAP Server	Servlet based HTTP server that processes SOAP coded requests
Transaction Objects Repository	A repository for storing a collection of service objects that are used to answer transaction probes. This repository does not strictly belong to the E-services adapter but is also accessed by the Transaction adapter
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 ServiceInvocation 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.

Service Invocation in E-services Adapter. The ServiceInvocation module forms the core part of the E-services adapter. This is the module responsible for loading the objects that implement the service and performing method invocation. Service Invocation is done by extensively using Java Reflection, a run-time object introspection facility.

Apart form doing the basic function of method invocation, the module also takes care of "long-running services". In other words the long-running services are basically asynchronous method calls, which may take an arbitrary time for completion of execution. An example may be a proofreading service, part of which can be automatic like spell checking and part of which can involve manually verification of the document. In such cases it is not desirable to make the client process wait until the operation is complete, in most cases it is not feasible. The ServiceInvocation module handles this appropriately.

3.3 Long Running Services

In order to handle long-running services the E-service adapter executes the asynchronous call as a separate thread. The ServiceInvocation module first gets the information that this is 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 result of this long-running service in this case has to be queried by the user using the previously sent transaction id as the key. The Transaction adapter detailed below, processes these transaction queries and the result (if available) will be returned.

The lifetime of these objects that are used to perform method invocation is an interesting issue that should be handled. In case of synchronous services, these objects are destroyed, as the result is available immediately and it is assumed that these objects will not be referenced for anything else apart from the service for which they were created. In case of long-running services, these objects are suitably stored, since transaction probes can be expected at any point during execution. Even if the long-running service under consideration is complete, the objects 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.

3.4 Transaction Adapter

The ability to make E-services participate in a workflow is one of the main objectives of BizBuilder. In order to achieve this, the framework has to provide a way for 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 transaction support interface intended for the service inquirers.

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 an activity is done?” or “ What is the expected completion time?” etc. These types of queries can be raised by a workflow engine, which may use the service under discussion to perform an activity or a part of an activity within a workflow. Such queries and their result will help the workflow engine 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.

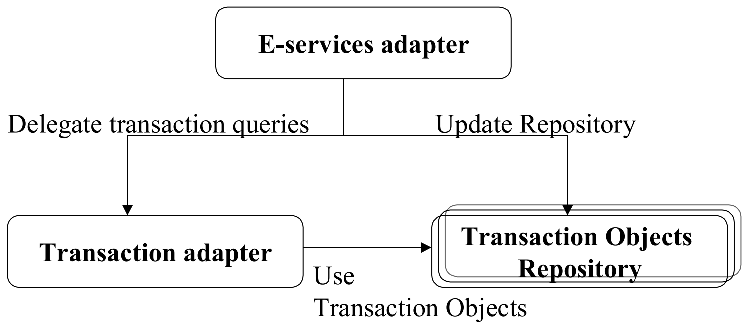


Fig. 4. E-services Adapter – Transaction Adapter Interaction

The Transaction adapter (in association with the E-services adapter) provides the necessary functionalities to handle these transaction probes or queries. The Transaction adapter uniquely identifies transaction queries, executes these queries on the corresponding objects in the transaction objects repository and generates suitable responses. It is also important to mention here that all transaction queries have to be ultimately answered by the service objects, since the objects are the ones that actually implement the service and know the status of the execution. BizBuilder provides this messaging infrastructure to query the service objects using the transaction probes.

3.5 Service Registration

The Service Registration modules provide the interface to the broker components, which serves as a central repository for storing details about the nature of services and the corresponding service-provider 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 is based on the UDDI – Universal Description Discovery and Integration, a standard for service publishing and searching in the e-business domain. The UDDI architecture uses web-services description format named WSDL – Web Services Description Language, which in turn is based on XML to describe the interfaces of a service. Since our architecture needs a rich service description language for E-services, we chose to use the UDDI framework and the brokering community is built on top of it.

3.6 Negotiation and Contract Adapter

In order to provide a complete E-services framework, BizBuilder also includes support for Negotiation and Contracts. This module provides a simple negotiation framework, where the service inquirers can negotiate a particular E-service for usage. 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. 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.

4 Implementation

The core components of the BizBuilder framework have been implemented, which allows servlet-based E-services to be accessed using HTTP and XML. Suitable graphical interface tools are provided, using which, a service provider can create E-services from existing service implementation (in the form of java objects). The service provider can also register these E-services to an UDDI enabled broker. Support is provided to browse the existing service templates with a broker, and register an implementation of the service under a service category of relevance.

4.1 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.

4.2 Service Invocation Using Java Reflection API

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 that parse the requests.

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 changes. A simple SSD description file for a service like ProcessOrder looks like the following.

Table 2. Structure of a SSD file

Parameter Name	Parameter Type
UserID	String

MyOrder	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 MyOrder 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 the same that are used while describing the service and registering them with the broker. The SSD file provides the semantics of the method, namely, it says how to treat the 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.

4.3 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). The XML conversion is provided only for attributes of basic data types in Java and not for Java library 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 follows.

XML representation of a Sample Order Object

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

4.4 Long-Running Services and Transaction Probes

The BizBuilder framework supports long-running services and transaction probes or 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 objects that implement the service can respond to these 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.

4.5 Support for E-service Providers

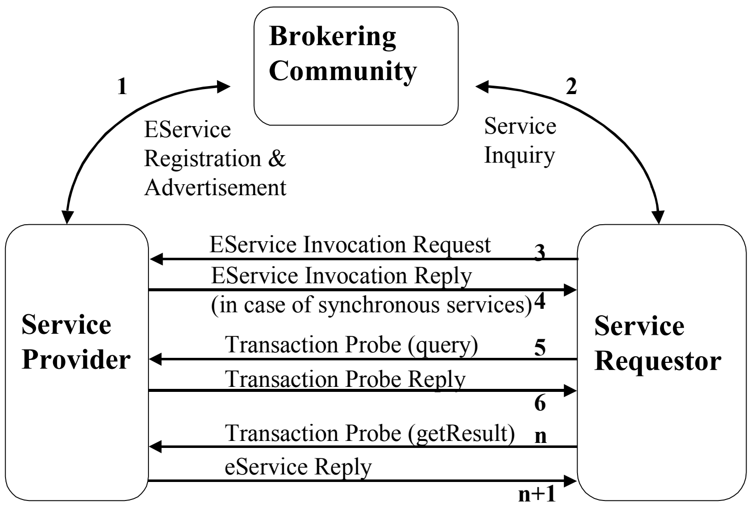
The BizBuilder framework provides the tools and user-interface, using which the service provider can utilize existing java objects and create E-services. The process of creating the E-services is automated and is guided by the BizBuilder user-interface. The E-service creation process involves creating a set of files (like service-mapping, SSD files) to be used by the framework for service invocation. Apart from the BizBuilder tool, the service provider has to run a web-server (like Tomcat-Apache), to provide access to the E-services and register this URI (through which the services can be accessed) with the broker. A set of Java APIs is provided in order to programmatically utilize these E-services individually or within a workflow.

4.6 Brokering Interactions

The framework allows the service provider to register the E-services with a UDDI enabled brokering community. The service provider typically registers his business and identifies a business category and service template within which an E-service can be published. Once the template is identified, the E-service can be registered within the selected template. The templates maintained by the brokering community serves as a categorization mechanism for services. The framework provides suitable user-interface to facilitate the above process.

4.7 Overall Scenario of Operations

The various components that participate in a workflow scenario are the service providers (using the BizBuilder framework), the service inquirers and the service brokers. 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 occur in identifying and invoking a service is depicted in the following diagram.



Brokering Community
1
EService Registration & Advertisement
2
Service Inquiry
Service Provider
Service Requestor
EService Invocation Request
3
EService Invocation Reply (in case of synchronous services)
4
Transaction Probe (query)
5
Transaction Probe Reply
6
Transaction Probe (getResult)
n
eService Reply
n+1

Fig. 5. Overall Sequence of Operations in BizBuilder Framework

5 Conclusions and Future Work

The need for businesses to interoperate seamlessly has been well understood and E-services is a simple and powerful concept that can enable businesses achieve this goal. In this paper, we proposed the BizBuilder architecture, which enables the creation of these E-services that are based on XML and HTTP. The framework enables creating these E-services from existing non-Internet adaptive systems. These E-services are represented and utilized using SOAP, which is based on XML and hence are interoperable. The framework supports the E-services to explicitly participate in inter-organizational workflows seamlessly. Support for advertising the E-services to a UDDI enabled broker is also provided in the architecture.

The BizBuilder framework currently supports only Java objects; this support can be extended to include legacy systems and other object implementations like CORBA, COM etc. Also service description can be extended to include service constraints to enable better matchmaking of services. Licensing and usage tracking of services is another useful feature that could be supported.

References

1. Kuno, H., "Surveying the E-Services Technical Landscape", HP Labs, 2000
2. Casati, F., Ilnicki, S., Jin, L., Krishnamoorthy, V., Shan, M., "eFlow: a Platform for Developing and Managing Composite e-Services", HP Labs, 2000
3. Web Services, IBM, http://www-106.ibm.com/developerworks/library/w-ovr/
4. Vasudevan, V., Bannon, T., "WebTrader: Discovery and Programmed Access to Web Based Services, Object Services and Consulting Inc., 1998
5. Paul, S., Park, E., Chaar, J., "RainMan: A Workflow System for the Internet", IBM T.J. Watson Research Center, 1997
6. Helal, A., Wang, M., Jagatheesan, A., Krithivasan, R., "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
7. Dan, A., Dias, D., Nguyen, T., Sachs, M., Shaikh, H., King, R., Duri, S., "The Coyote Project: Framework for Multi-party E-Commerce", Proceeding of the 7th Delos Workshop on Electronic Commerce, Greece, 1998
8. Mennie, D., Pagurek, B., "An architecture to support Dynamic Composition of Service Components"
9. Jagatheesan, A., Helal, A., "Architecture and protocols for Sangam Communities", Internal Report accessible from http://www.harris.cise.ufl.edu/projects/e-services.htm
10. Fingar, P., Kumar, H., Sharma, T., "Enterprise E-Commerce", Meghan-Kiffer Press, ISBN 0-929652-11-8, 2000
11. Meng, J., Su, S., Lam, H., Helal, A., "Achieving Dynamic Inter-organizational Workflow Management by Integrating Business Processes, Events, and Rules," Internal Report accessible from http://www.harris.cise.ufl.edu/projects/e-services.htm, 2001