Migrating and Specifying Services for Web Integration

Ying Zou, Kostas Kontogiannis

Dept. of Electrical & Computer Engineering
University of Waterloo
Waterloo, ON, N2L 3G1, Canada

{yzou, kostas}@uwaterloo.ca

Abstract. With the explosive growth of the Internet, businesses of all sizes aim on applying e-business solutions to their IT infrastructures, migrating their legacy business processes into Web-based environments, and establishing their own on-line services. To facilitate process and service integration, a complete and information rich service description language, is essential for server processes to be specified and for client processes to be able to locate services that are available in Web-enabled remote servers.

Within the context of emerging technologies, such as XML, the Internet, and Network-Centric Computing, we propose an architecture that allows for Web-based integration of distributed components and services. The architecture is based on component wrapping, a service description language that allows for the specification of services, and on techniques that support service registration and dynamic service localization.

1 Introduction

Tremendous changes are taking place in the business world today due to the frequent introduction of new technologies. As these technologies become the mainstream, the focus of e-commerce activities is shifting from customer-to-business transactions, to an e-business to business (B2B) model [8], which integrates business services and business process models, across corporate Intranets or the Internet. Towards this objective, multi-tier architectures, networking, and distributed object technologies have made possible for organizations to deploy complex software applications over the Internet.

Modern software systems must conform to requirements, such as flexibility, adaptability, time to market, and ability to withstand continued business process reengineering. Driven by these requirements, the migration and integration of legacy systems towards new platforms and operating environments provide an effective strategy for organizations to maintain their competitive edge [1]. In this context, many consulting firms such as the Gartner Group are predicting that organizations that integrate new development with the existing legacy systems will have a higher success rate, at optimal cost, in the implementation of client/server applications.

In this paper we present an architecture that allows for the migration and integration of existing stand-alone services into distributed environments.

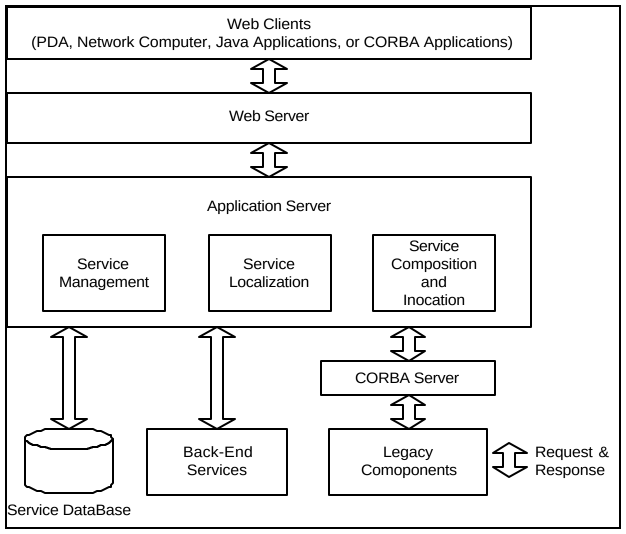


Fig. 1. Overall Architecture

In the core of the system lies a service description language that provides standard, enriched, and well-understood information about the interfaces and the functionality of the offered services. A service registration tool allows for services to be easily registered with the environment. Finally, a search engine can effectively locate services according to specific search criteria, allowing thus for service location transparency.

This paper is organized as follows. Section 2 introduces the architecture for Web-based Integration. The issues that arise in Web-based service integration are addressed in section 3. The issues pertaining to the migration of software components as legacy services are presented in section 4. Section 5 discusses the major components of a service description language. The service registration and localization modules are presented in section 6 and 7 respectively. An application scenario is illustrated in section 8. Finally, section 9 provides a summary and the conclusion of the paper.

2 Architecture for Web-based Service Integration

The Web-based service integration architecture focuses on the use of Web as an open infrastructure where e-business related services and tasks can be defined, composed, and enacted in a fully customizable way.

As e-commerce services can be scattered virtually everywhere on the Web, we need an architecture that allows for the separation of business application logic from the client-side presentation logic. The three-tier architecture is instrumental for the deployment of the distributed objects on a Web-enabled environment.

We present an open, multi-tier infrastructure, where a service can publish itself, and easily be integrated with other legacy components and services. The proposed architecture is depicted in Fig. 1. The first layer (top) consists of a wide range of Web clients, including Web browsers for handheld and embedded systems, or Java/Pure and CORBA based applications running on fully loaded desktops. The second layer relates to services provided by the Web server and application server. The Web server captures the requests from Web clients and directs the requests to the Application server. The application server has been widely adopted as the runtime environment of choice for integrating heterogeneous applications. The core part of the architecture is the underlying services that are added to the application server, including Service Management, Service Localization, and Service Composition and Invocation.

The Service Management module maintains a database of the descriptions of the available services. It enables the deployed services to dynamically register their information in a repository. The Service Management module provides a repository for the client processes to use in order to locate available services and compose them for the completion of elaborate business tasks. A service description language provides a customizable way to represent distributed services with enriched information.

The Service Localization module is responsible for selecting the required services among many available ones, according to the criteria set by the client process. The service localization enables the clients to search the service by functionality, signatures, performance, and customizability.

Finally, the Service Composition and Invocation module provides a framework and a scripting language for dynamically enacting and composing remote services. This module serves as an integrator that allows back-end services and legacy systems to be composed seamlessly.

In order to enable the integration of legacy applications in a Web-based environment, we adopt the CORBA standard. The standard allows for legacy applications to be encapsulated in remote objects using wrapper classes and behave as distributed components. This wrapping technology allows clients in virtually any software or hardware platform to invoke remote legacy components in their native operating environments. The legacy application is resident on the CORBA server, which acts as the gateway for the integration. Such a framework provides system support for the invocation and integration of legacy back-end services from any clients.

3 Web-based Service Integration Mechanism

The generic three-tier architecture can be further separated into four layers [11]: *a)* presentation layer (Web client); *b)* content layer (Web server); *c)* application layer (application server); and *d)* back-end data and services layer, as shown in Fig. 2. The Web server is responsible to accept HTTP requests from Web clients, and deliver them to the application server, while the application server is in charge of locating the services and returning responses back to the Web server. On the other hand, a thin client in the presentation layer has little or no application logic.

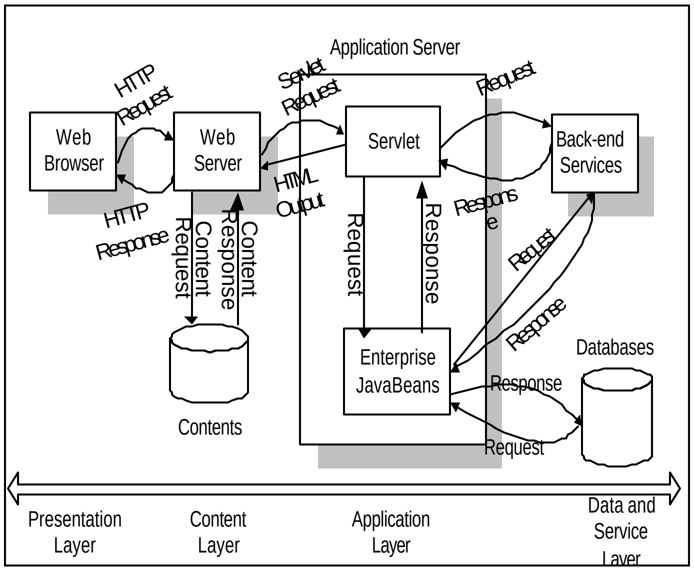


Fig. 2. Control Flows in Three-Tire Architecture

The Web server can maintain a content repository, or a file system, where the information-based resources are stored and serve as static HTML pages. Upon receiving a request from the client, the Web server retrieves the requested document from the content repository and sends it to the client. In this case, the client entirely relies on the Web server. Programming languages, such as Java, and scripting languages, like CGI, can be used to access the database.

To provide the dynamic information generated by software services, the Web server needs to constantly interact with the application server. A servlet provides the dynamic HTML content to clients. When the Web server receives the request for a servlet, it re-directs the client's request along with the parameters to the application server, which loads the servlet and runs it. Servlets not only have all the features of Java like automatic memory management, advanced networking, multithreading, and so forth but also, allow for enterprise-wide connectivity in the form of JNI (Java Native Interface), JDBC, EJBs, RMI, and CORBA. Servlets can make calls to back-end services, other servlets, or to the Enterprise JavaBeans [12].

Once the servlets are deployed on an application server, they can be accessed from any other Web server. This can be achieved provided that the client's request contains the URL of the servlet with the correct name, type, parameters, and initial values. The combination of EJBs and Servlets, CORBA objects and Servlets, and RMI objects and Servlets, can be used to invoke back-end services accessed by the Web clients via HTTP connections. However, CORBA and Enterprise Java Beans are not a panacea for all problems that may arise when integrating services in a distributed environment, but they provide the building blocks for distributing applications over a diverse range of platforms and operating environments.

4 Example Service Migration to a CORBA Environment

In order to integrate existing systems that encapsulate valuable business logic, the first step is to re-engineer these systems so that they can be used in a distributed environment. In the approach discussed in this paper, we utilize reverse engineering and design recovery techniques to identify specific components that encapsulate valuable business logic for a specific application. These techniques have been investigated as part of another project with IBM and are presented in [13, 14]. Once specific legacy components are identified through the use of program analysis, their behavior must be specified in terms of well-defined interfaces. In order to integrate the identified components to a heavily heterogeneous Web-enabled distributed environment, we must define an appropriate middleware. The CORBA specification provides a suitable infrastructure for integration, due to its platform, language, and vendor independence.

The component interface hides the implementation details inside the component and allows only signatures of services to be published to its clients. Moreover, it defines a set of properties and behaviors that represent a component's API. Properties are represented in terms of attributes, which can be accessed by accessors and mutators. Similarly, method parameters and return types can be represented by IDL interfaces.

In a related software migration case study we have used reverse engineering techniques in order to analyze and migrate the AVL GNU tree libraries from C procedural code to a new C++ object oriented implementation [13, 14]. The new migrant object oriented AVL tree library can be considered as a component, consisting of several classes. In this section we present how such a collection of C++ classes from the GNU AVL tree library cab be migrated in a CORBA environment.

In a nutshell, the interface for the new AVL tree component consists of several stub interfaces that correspond to wrapper classes. To migrate the standalone identified components into a distributed computing environment, the object wrapping approach can be adopted. The wrappers implement message passing between the calling and the called objects, and redirect method invocations to the actual component services. The concrete process to accomplish wrapping is implemented in terms of three major steps.

The first step focuses on the specification of components in CORBA IDL as shown in Fig. 3.

The second step deals with the CORBA IDL compiler to translate the given IDL specification into a language specific (e.g. C++), client-side stub classes and server-side skeleton classes. Client stub classes and server skeleton classes are generated automatically from the corresponding IDL specification. The client stub classes are proxies that allow a request invocation to be made via a normal local method call. Server-side skeleton classes allow a request invocation received by the server to be dispatched to the appropriate server-side object. The operations registered in the interface become pure virtual functions in the skeleton class.

The third step focuses on wrapper classes that are generated and implemented as CORBA objects, directly inheriting from the skeleton classes. The wrapper classes encapsulate the standalone C++ object by reference, and incarnate the virtual functions by redirecting them to the encapsulated C++ class methods. The new

```
module AVL{

interface corba_ubi_btRoot;
interface corba_ubi_btNode;
interface corba_SampleRec;

typedef char corba_ubi_trBool;

interface corba_SampleRec{
        void putName(in string val);
        string getName();
        void putNode(in corba_ubi_btNode val);
        corba_ubi_btNode getNode();
        long getDataCount();
        void putDataCount(in long aval);
};

interface corba_ubi_btNode {
        void putBalance(in char val);
        char getBalance();
        long Validate();
        //......
};

interface corba_ubi_btRoot{
        corba_ubi_trBool ubi_avlInsert(
                 in corba_ubi_btNode  NewNode,
                in corba_SampleRec  ItemPtr,
                in corba_ubi_btNode OldNode );
        // ......
};
};
```

Fig. 3. AVL Component Interface Definition

functionality of the legacy object can be added in the wrapper class as long as the method name is registered in the interface.

For example, the SampleRec class is one of the classes identified within the AVL tree component. The wrapper_SampleRec inherits from the skeleton class sk_AVL::_sk_corba_SampleRec, which is generated from the CORBA IDL to C++ compiler. The wrapper class, wrapper_SampleRec, encapsulates a reference of SampleRec class as shown in Fig. 4.

When a client invokes a method through CORBA, it passes the CORBA data type parameters. The wrapper classes need to translate the CORBA specific data types from the client calls to the data types used by encapsulated C++ classes. Fig. 5 illustrates the transformation from the CORBA specific type such as corba_SampleRec_ptr to the SampleRec used in the C++ function. In the same way, the wrapper classes convert the returned values from the C++ class to the CORBA specific data type.

```
class wrapper_SampleRec : public _sk_AVL::_sk_corba_SampleRec
{
private:
        SampleRec& _ref;
        char *_obj_name;
public:
        wrapper_SampleRec(
                SampleRec& _t,
                const char *object_name = NULL):
                _ref(_t),
                _sk_AVL::_sk_corba_SampleRec(object_name);
        SampleRec* transIDLToObj(
                AVL::corba_SampleRec_ptr obj);
        void putNode(
                AVL::corba_ubi_btNode_ptr val);
        AVL::corba_ubi_btNode_ptr getNode();
        ~wrapper_SampleRec(){
                delete &_ref;
                free (_obj_name);};
        //......
};
```

Fig. 4. An Example Wrapper Class

```
SampleRec* wrapper_SampleRec::transIDLToObj(
        AVL::corba_SampleRec_ptr obj)
{
if (CORBA::is_nil(obj)) return NULL;

// set up the data members of _ref object
 _ref.putName(obj->getName());
 _ref.putDataCount(obj->getDataCount());

//translate the ubi_btNode to corba_ubi_btNode_ptr by wrapper
//class NodeWrap
ubi_btNode *NodeImp = new ubi_btNode();
wrapper_ubi_btNode NodeWrap(*NodeImp, _obj_name);

//translate corba_ubi_btNode_ptr type returned from
//obj->getNode() to ubi_btNode * by transIDLToObj() in
//wrapper object NodeWrap.
_ref.putNode(NodeWrap.transIDLToObj(obj->getNode()));
 return &_ref;
}
```

Fig. 5. Example for Object Type Translation

Since IDL does not support overloading and polymorphism, each method and data field within the interface should have a unique identifier, in order to disambiguate references to programming entities that correspond to different languages. For example, C++ supports overloading, but C does not. If the polymorphism and overloading methods occur in one class, it is necessary to rename these methods by adding the prefix or suffix to the original name when they are registered in the interface, avoiding changing the identified objects. This “naming” technique allows unique naming conventions throughout the system, without violating code style stand-

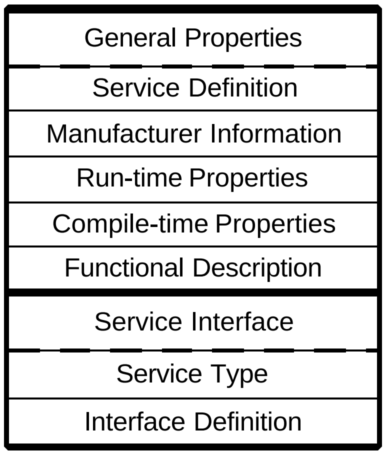


Fig. 6. Key Element of Service Specification

ards. The wrapper classes are responsible to direct the renamed overloaded and polymorphic methods to the corresponding client code.

If the polymorphism and overloading methods occur in the inheritance relationship, we can take advantage of C++ upcast feature, only register the sub-class in the component interface, and upcast the sub-class to its super class when the polymorphic or overloading methods in super class are invoked.

5 Service Description Language

In this section, we present the prototype of a service description language that provides a standard format to represent, register, and store information related to back-end services. To facilitate the integration of back-end services, a meta level description language is essential to effectively locate registered services. The meta-level description for the software services can be published at the same time as the distributed objects are deployed onto the application servers, or some time later when the enterprise would like to make their software services available.

5.1 Structure of Service Description Language

Generally, a service can be represented in a multi-faceted way, by specifying, for example, vendor, run time specifications, compile time requirements, method signatures, as well as, pre- and post-conditions. Each of these aspects is denoted as a Service Description Fact. Different Description Facts specify different properties of the services.

In a nutshell, the specification of the software services is divided into two layers namely *General* properties and *Service Interface* properties, as illustrated in Fig. 6.

```
<?xml verstion="1.0"?>
<SDL>
<GeneralInfo>
 <ServiceDef>
    <ServiceName> <!-- Specify service ID and name -- >
    </ServiceName>
    <ServiceCatalog><!--Specify service category-->
    </ServiceCatalog>
    <URL > <!-- Specify service URL linke -->
    </URL>
    <VersionNumber /> <!--Specify version number-->
 </ServiceDef>
 <Manufacturer> <!-- Specify vendor information-->
 </Manufacturer>
 <RunTimeEnv> <!-- Specify run-time environments-->
     <OSs>
              <OS name="" version=""/>
     </OSs>
 </RunTimeEnv>
 <CompileTimeEnv> <!-- Specify compile time environments-->
 </CompileTimeEnv>
 <Funcationality>
              <!-- Specify abstract and detailed information -->
              <!-- about service funcationality-->
  </Functionality>
</GeneralInfo>
<ServiceInterface>
  <Types>
  <!--Lists the Types of components inside the service interface. -->
  </Types>
  <ServletML>
  <!--Lists the servlet interface -->
      <Parameters>
              <Parameter>
                  <Name /><Type /><Value />
              </Parameter>
     </Parameters>
  </ServletML>
  <EJBML> <!--Specify the EJBs interface -->
  </EJBML>
  <CORBAML> <!--Specify the CORBA interface -->
  </CORBAML>
</ServiceInterface>
</SDL>
```

Fig. 7. Overall Structure of Service Description Language

Each layer contains specific information at different levels of abstraction. The structure of a service description document is illustrated in Fig. 7.

To enable a service binder to locate the requested correct service with high precision and recall levels, the General properties should contain facts that relate to such aspects as general service definition, manufacture information, run-time and compile-time properties, signatures, version numbers, implementation language, and functional descriptions. For example, for a CORBA wrapped service object, it is important to specify the ORB agent address, which is responsible for invoking the requested CORBA object by the name and URL address of the object.

For the purpose of Web-based service integration, it is important to disclose the interface of the distributed components to client processes. Similarly, the Service

```
<?xml version="1.0"?>
<!ELEMENT newTags (newTag)+>
<!ELEMENT newTag (startingPoint, tagDef)>
<!ELEMENT startingPoint (#PCDATA)>
<!ELEMENT tagDef
     (tagName, attList*, containedTags*)>
<!ELEMENT tagName (#PCDATA, tagContent*)>
<!ELEMENT attrList (attr)+>
<!ELEMENT tagContent (#PCDATA)>
<!ELEMENT containedTags (tagDef+, group)>
<!ELEMENT group (group* | tagName*)>
<!ATTLIST group groupName CDATA #REQUIRED>
<!ATTLIST group groupType (SEQ|OR) #IMPLIED >
<!ATTLIST group groupOccurs
             (once|optonal|requried) #IMPLIED>
<!ATTLIST tagDef occurs
             (once|optional|required) #REQUIRED>
<!ATTLIST attr attrName CDATA #REQUIRED>
<!ATTLIST attr attrType CDATA #REQUIRED>
<!ATTLIST attr attrValue CDATA #IMPLIED>
```

Fig. 8. DTD for Adding New Fact and Content

Interface layer specifies the APIs of the registered components. As stated earlier, there are different technologies to make the back-end services available to remote clents. These technologies include servlets, EJBs, and CORBA. Each type of back-end services is registered by its own specific interface description.

For servlets, the inputs are embedded in HTML forms, which contain the HTML types of inputs, the names of parameters and the allowable values. For the EJBs, the back-end services can be composed of several beans (session beans, or entity beans) in one jar file. Each bean has its own home interface and remote interface. When a service is implemented by the CORBA standard, it may include several CORBA IDL interfaces as encapsulated in the CORBA IDL "module" name scope. For the interface within CORBA and EJBs components, it is necessary to declare the available methods, parameters and the types of method parameters and return values. To reduce the complexity in definition of service description language, we inherit the interface from EJBs and CORBA IDL by inserting them under the <EJBML> tag and <CORBAML> tag respectively.

5.2 Extensibility of the Service Description Language

The extensibility of the service description language is crucial for representing services in distributed Web-enabled environments. For example, new service categories can be added or existing service descriptions can be extended.

The DTD for the Service Description Fact specifies its syntax and allows for such Service Descriptions to be proven syntactically valid. The DTD for the Service Description language is illustrated in Fig. 8. New Service Description facts can be added by using the *newTag* element contained in the *newTags* element.

```
<?xml version="1.0"?>
<!DOCTYPE newTags SYSTEM "patSpec.dtd">
<newTags>
  <newTag>
    <startingPoint>SDL.GeneralInfo</startingPoint>
      <tagDef occurs="once">
        <tagName>RunTimeEnv</tagName>
        <containedTags>
            <tagDef occurs="required">
              <tagName>OSs</tagName>
              <containedTags>
                <tagDef occurs="required">
                  <tagName> OS </tagName>
                  <attList>
                    <att attrName="name" attrType="CDATA" />
                   <att attrName="version" attrType="CDATA" />
                  </attList>
                </tagDef>
            </containedTags>
          </tagDef>
          </containedTags>
      </tagDef>
   </newTag>
</newTags>
```

Fig. 9. Run Time Environment Fact Definition

With the fact specification DTD, the addition of new facts is uniquely identified and inserted in a way that maintains the syntactic validity of the description. In Fig. 9, for example, the addition of the Run-time environment fact is illustrated. In this example, the new fact is inserted under the *GeneralInfo* element with the tag name of *RunTimeEnv*. *RunTimeEnv* element can occur once under *GeneralInfo* element. It contains an *OS* element, which specifies the operating systems to run the service. The *OS* element may occur one or more times, and can have attributes that denote its name and version number.

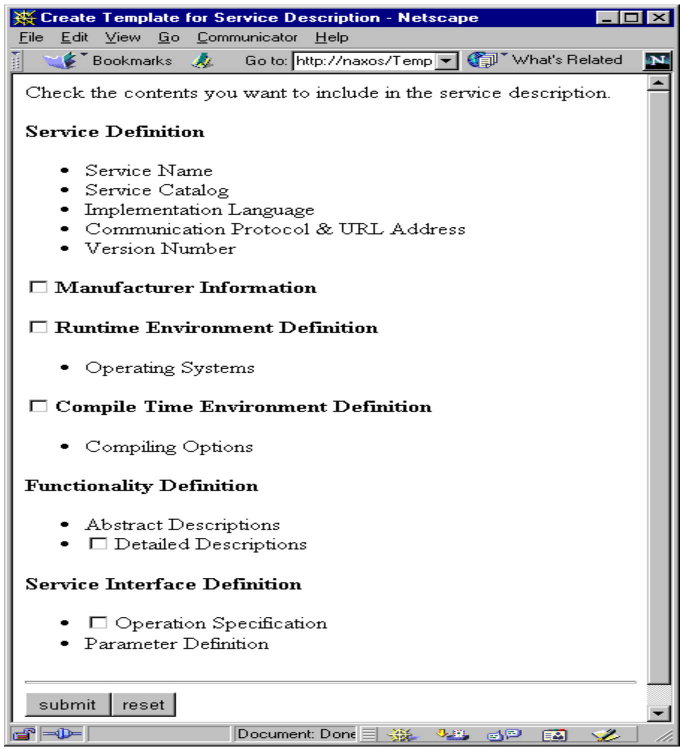
Meanwhile, new content can be easily introduced into the existing service description under the tag <ServiceCatalog> category (Fig. 7).

5.3 Structure of Database

Service Descriptions and fact specifications (DTDs) require a database for the persistent storage of the XML encoded component and service interface description.

To keep the database management simple and achieve flexibility in the service description, we use one table to map the service ID and the external XML filename for each service description. Each fact consisting of a service description is stored in a separate table. The primary key of these tables is the service ID generated during registration. In the same way, another table is created to store the file name of the DTD for each fact.

The Service Management module (shown in the Fig. 1) is responsible to maintain the service database. It can insert a new service description, delete, and modify the existing one. For this task, we utilize the IBM DB2 XML extender to map XML


Create Template for Service Description - Netscape
File Edit View Go Communicator Help
Bookmarks Go to: http://naxos/Temp What's Related
submit reset
Document: Done

Check the contents you want to include in the service description.

Service Definition

- Service Name
- Service Catalog
- Implementation Language
- Communication Protocol & URL Address
- Version Number

☐ **Manufacturer Information**

☐ **Runtime Environment Definition**

- Operating Systems

☐ **Compile Time Environment Definition**

- Compiling Options

Functionality Definition

- Abstract Descriptions
- ☐ Detailed Descriptions

Service Interface Definition

- ☐ Operation Specification
- Parameter Definition

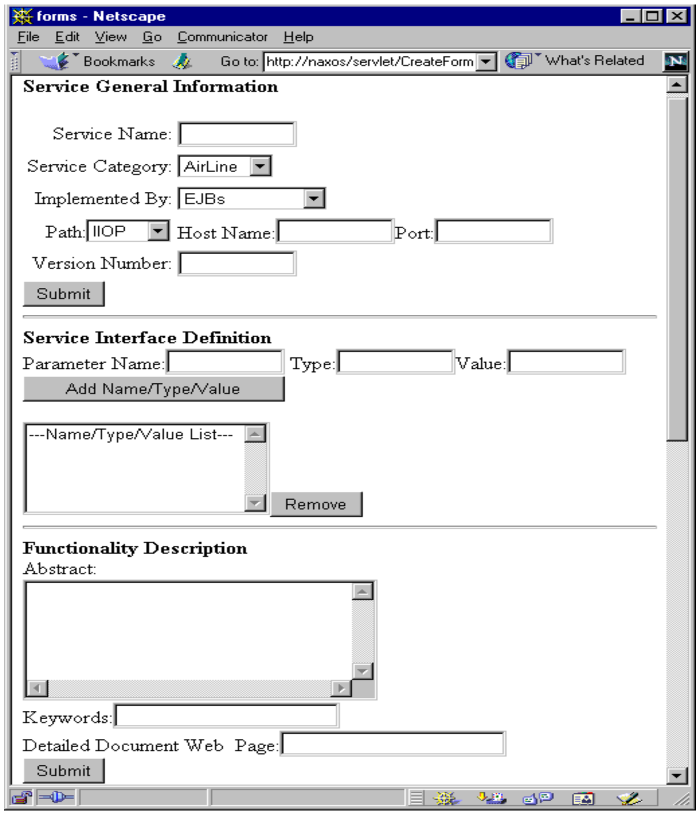
Fig. 10. Service Specification Fact List

Service Descriptions to DB2 tables. In general, the service manager retrieves from the database table the description filename, and then extracts the whole XML document by using traditional SQL queries. When a service is registered, the service manager can check for duplicate definitions, generate the service ID, and insert the description into the database. The Service Management Module is implemented by Enterprise Java Bean, which provides the support for transactions.

6 Service Registration

For the service registration, we have designed a Web based interface to serve as a service registration authoring tool, which allows for the user to specify the service description by filling in forms in a Web interface, as shown in Fig. 10. Then the service description is generated automatically from the information provided.

As mentioned earlier, in order to provide maximum customizability, the service description language is separated into independent facts. Moreover, the environment allows for new facts to be added at anytime. This interface allows the user to select the required facts by filling predefined forms. Some facts are indispensable, such as Service Definition. After submission, the Web Interface will create an HTML form as


forms - Netscape
File Edit View Go Communicator Help
Bookmarks
Go to: http://naxos/servlet/CreateForm
What's Related

Service General Information

Service Name:

Service Category: AirLine

Implemented By: EJBs

Path: IIOP Host Name: Port:

Version Number:

Submit

Service Interface Definition

Parameter Name: Type: Value:

Add Name/Type/Value

---Name/Type/Value List---

Remove

Functionality Description

Abstract:

Keywords:

Detailed Document Web Page:

Submit

Fig. 11. Service Description Interface

shown in Fig. 11, where the user can add more information about the newly registered service.

For example, once the legacy components are wrapped as distributed objects, as discussed in section 4, their information can be described through the Web interface and registered into the service repository.

7 Service Localization

A prototype service localization mechanism allows for distributed software services to be located much like a search engine locates content (data) in Web pages.

The system can provide two ways for clients to submit search queries, via the Web HTML Interface, or by an XML formatted document that may be part of a client's request for a service.

The design of the query language aims to allow users to provide as many features as possible in order to specify the services being sought. The service description facts

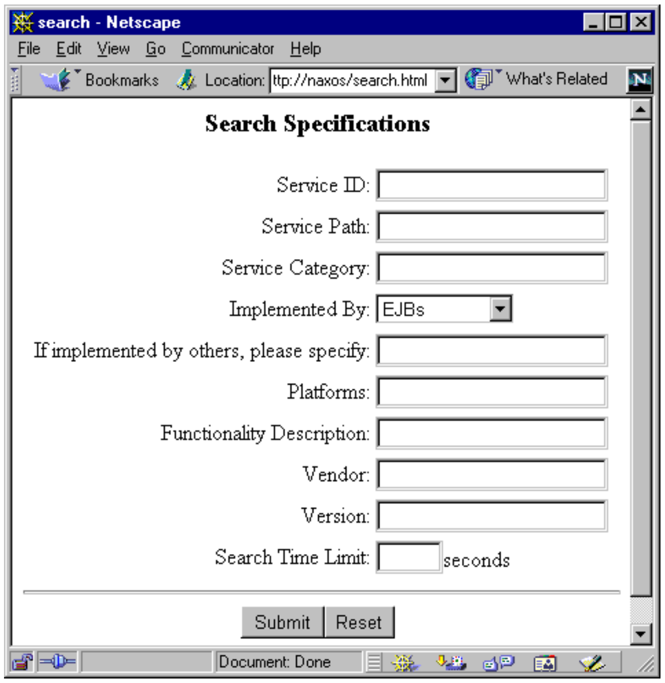
```
<?xml version="1.0" ?>
<!ELEMENT searchSpec (ID*, location*, category*, implBy*, platforms*,
funcDsc*, vendor*, version*, timeLimit*)>
<!ELEMENT ID (#PCDATA)>
<!ELEMENT location (#PCDATA)>
<!ELEMENT category ((keyword, (AND | OR)*)+ | (NOT, keyword)*)>
<!ELEMENT implBy ((keyword, (AND | OR)*)+ | (NOT, keyword)*)>
<!ELEMENT platforms ((keyword, (AND | OR)*)+ | (NOT, keyword)*)>
<!ELEMENT funcDsc ((keyword, (AND | OR)*)+ | (NOT, keyword)*)>
<!ELEMENT vendor (#PCDATA)>
<!ELEMENT version (#PCDATA)>
<!ELEMENT timeLimit (#PCDATA)>
<!ELEMENT keyword (#PCDATA)>
<!ELEMENT AND (keyword | AND | OR | NOT)*>
<!ELEMENT OR (keyword | AND | OR | NOT)*>
<!ELEMENT NOT (keyword | AND | OR | NOT)*>
```

Fig. 12. DTD for Service Localization Query

as shown in Fig. 7, allow for the user to search for services according to specific criteria such as service categories, functionality, implementation techniques, and operating platforms. The grammar for the query language is defined in terms of a DTD as shown in Fig. 12. The root element for this DTD is the *searchSpec* element, which can include zero or more children, such as service ID, service location, service category, etc. The query doesn't need to include every element under the *searchSpec* element. Some elements, such as *category* are composed of the several *keyword* elements, *AND*, *OR* and *NOT* elements. The *keyword* element represents the keyword for the search criteria. The keywords can be conjuncted by *AND*, *OR*, and *NOT* elements. For example, *AND* elements can have children as *keyword* elements, *AND* elements, *OR* elements, and *NOT* elements. When new Description Facts are added into the service description language, the DTD of the query will be edited correspondingly in order to reflect the changes in the new service descriptions.

The Web interface of the service search engine is currently designed for the HTTP users, as shown in Fig. 13. After submission, the search criteria composed in the XML format are sent to the Service Localization module as illustrated in Fig. 1.

By extracting the requirements from the query specification, the service localization module looks up the service database. If multiple results meet the search criteria, the available services can be listed and ranked according to their registered features, such as performance, or estimated response time from the server. Since service description facts are encoded in XML, the search locator has been implemented using the XML DOM (Document Object Model) API and incorporating search logistics, such as exact search, sub-string search, precedence search, and stemming.


search - Netscape
File Edit View Go Communicator Help
Bookmarks
Location: ttp://naxos/search.html
What's Related
Submit
Reset
Document: Done

Search Specifications

Service ID:	
Service Path:	
Service Category:	
Implemented By:	EJBs
If implemented by others, please specify:	
Platforms:	
Functionality Description:	
Vendor:	
Version:	
Search Time Limit:	seconds

Fig. 13. Web Interface for Search Query

8 Application Scenario

As an example, consider an application scenario for the Web-based service integration architecture presented in this paper, whereby a global infrastructure enables distributed components that have been developed independently or migrated from legacy systems, to be integrated with each other and to facilitate complex business tasks.

In this way, distributed components located virtually everywhere in the world, can be combined on as required basis, forming thus collaborative systems. This integration can happen dynamically by allowing general service properties, functionality and, signatures of components that are specified in the service description language to be registered in the service database. Client processes can search for available distributed components in a same manner as a search engine would be used to locate information resources on the Internet. After meeting the search requirements, the client process can invoke the identified services without necessarily downloading all the components to the local client machine. On-going work, focuses on the invocation of services that is based on scripts encoded in XML and is enacted using the Event-Condition-Action (ECA) paradigm [15]. The overall proposed architecture is under development in collaboration with IBM Canada, Center for Advanced Studies, and is illustrated in Fig. 14. The core of the system is

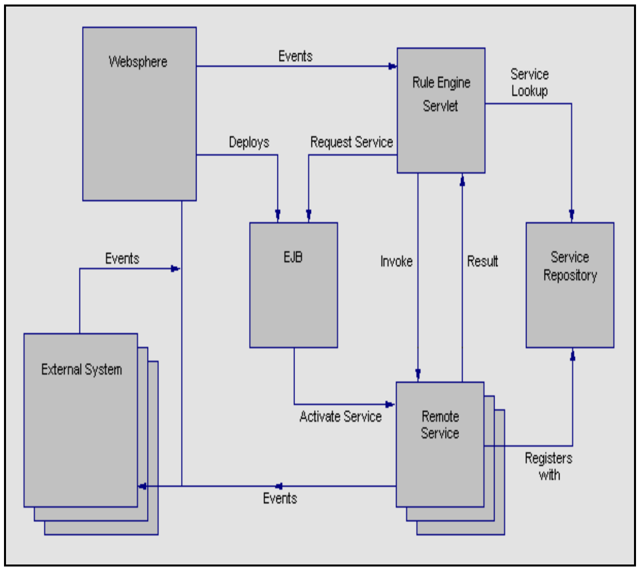


Fig. 14. Overall Service Integration Architecture

the Rule Engine Servlet, which accepts triggering events from the Web server. Once the premises (events and conditions) of specific ECA rules are satisfied, the requested service (action) by the rule is localized and invoked. Upon completion, services (actions) produce new events that may trigger new ECA rules. Deadlocks and loops are detected by building a rule dependency graph for a given script [16].

In its current form, the Internet and the Web provide reliable connectivity between client and server processes by using pre-defined, hard-coded transaction scenarios. These pre-defined transaction scenarios are currently implemented in terms of hard-coded URL links, CGI scripts and, Java applets. In this context, we address the issue of customizing the integration of services between client and server processes in distributed e-commerce and e-business environments. Services that are represented as remote components are encapsulated in wrapper objects. Such software components are obtained either as modules of legacy systems that encode mission critical business logic or, as modules of new applications developed with specific functional requirements in mind. In either case, these software components implement specific tasks that can be thought of, as building blocks of more complex interaction scenarios between client and server processes.

The customization of the transaction and integration logic required by various processes to complete complex tasks, opens new opportunities in Web-enabled e-Commerce and e-Business environments. In this sense, business partners can customize their business transaction models to fit specific needs or, specific contract requirements. This customization is transparent to third parties and, provides means to complete business transactions accurately and on-time. Organizations can enter the e-business arena by building and deploying extensible and customizable services over the Internet using software components that are readily available as services over the

Internet. Moreover, virtual agencies and portals that provide a wide range of services can be formed by integrating existing functionality and content over the Web. For example, a virtual travel agency can be formed, by composing in a customized manner, services that are readily available in various travel related Internet Web sites. Client processes may post requests to the virtual agency. The agency can enact its transaction logic (scripts) in order to integrate and compose data and services from a wide spectrum of sites. In this scenario, data about pricing, availability and, travel related special offers, can be fetched by various sites, processed by the agency and presented to the client in a customized and competitive for the agency way.

The prototype system under development at the Center for Advanced Studies is focusing on building virtual malls where different virtual stores (agencies) provide goods and services and, compete for the pricing, and the range of services offered.

9 Conclusion

In this paper, we present the issues of migrating monolithic services into distributed environments, and propose a service description language to specify back-end services. With the aid of a service description language, service registration and a service localization mechanism, component integration can be realized and service location transparency can be achieved.

In this context, we are especially interested in Web-based platforms because the Web is becoming the common denominator for accessing and presenting information over the Internet, Intranets and, Extranets. Moreover, the Web provides the deployment platform for many new enabling technologies such as CORBA, RMI and, EJBs.

As a result, this Web-based service integration infrastructure allows for the reuse of the existing software components, shortens the time to architect new applications, and eases the enterprise integration of business operations.

10 Acknowledgements

The authors would like to thank Bill O'Farrel and Steven Perelgut of IBM CAS and Evan Mamas and Richard Gregory of the University of Waterloo, for their valuable suggestions and insights.

References

1. Umar, Amjad, "Application (Re)Engineering: Building Web-Based Applications and Dealing with Legacies", Prentice Hall PTR, 1997.
2. RamPrabhu, Robert Abarbanel, "Enterprise Computing: The Java Factor", Computer, P115, June 1997 IEEE.

3. Walter Brenner, Rüdiger Zarnekow, and Harmut Wittig, “Intelligent Software Agents: Foundations and Applications”, Springer-Verlag Berlin Heidelberg 1998.
4. Alan R. Williamson, “Java Servlets By Example”, Manning Publications Co., 1999.
5. Victor Lesser, et al. “Resource-Bounded Searches in an Information Marketplace”, IEEE Internet Computing, March/April 2000.
6. Tuomas Sandholm and Qianbo Huai, “Nomad: Mobile Agent System for an Internet-Based Auction House”, IEEE Internet Computing, March/April 2000.
7. Ying Zou, Kostas Kontogiannis, “Localizing and Using Services in Web-Enabled Environments”, 2nd International Workshop for Web Site Evolution, Switzerland, 2000.
8. “Business-to-Business e-Commerce with Open Buying on the Internet”, http://www.ibm.com/iac/papers/obi-paper/intro.html.
9. “Gaining Competitvie Advantage in the Supply Chain: IBM Solution for Business Integration”, http://www-4.ibm.com/software/info/ti/issues/scm.html.
10. Ronald Bourret, “XML and Databases”, http://www.informatik.tu-darmstadt.de/DVS1/staff/bourret/xml/XMLAndDatabases.html.
11. Paul Dreyfus, “The Second Wave: Netscape on Usability in the Services-Based Internet”, IEEE Internet Computing, March/April 1998.
12. Joqquin Picon, et al, “Enterprise JavaBeans Development Using VisualAge for Java”, http://www.redbooks.ibm.com.
13. Prashant Patil, Ying Zou, Kostas Kontogiannis and John Mylopoulos, “Migration of Procedural Systems to Network-Centric Platforms”, CASCON’99, Toronto, 1999.
14. Kostas Kontogiannis, Prashant Patil, “Evidence Driven Object Identification in Procedural Code”, STEP’99, Pittsburgh, Pennsylvania, 1999.
15. Richard Gregory, Kostas Kontogiannis, “Requirements for a Distributed Tool Integration System”, http://www.swen.uwaterloo.ca/~rwgregor.
16. George Koulouris et.al “Distributed Systems: Concepts and Design”, Addison-Wesley, Second Edition, 1996.