UDDIe: An Extended Registry for Web Services

Ali ShaikhAli, Omer F. Rana, Rashid Al-Ali, David W. Walker,
Department of Computer Science
Cardiff University, UK
{Rashid.A.ShaikhAli,O.F.Rana,David.W.Walker}@cs.cardiff.ac.uk

Abstract

The Universal Description, Discovery and Integration (UDDI) is a specification for distributed Web-based information registries for Web Services. UDDI allows HTTP-enabled business services to be published, and subsequently searched, based on their interface. UDDI consists of three components: “white pages” to hold basic contact information and identifiers for a company, “yellow pages” to enable companies to be listed based on their industry categories (using standard taxonomies), and “green pages” to record interface details of how a Web service is to be invoked. UDDI is however limited in scope – allowing white, yellow or green pages to be searched based on a few attributes, and does not provide an automatic mechanism for updating the registry as services (and service providers) change. We implement UDDIe – an extension to UDDI, which supports the notion of “blue pages”, to record user defined properties associated with a service – and to enable discovery of services based on these. UDDIe enables a registry to be more dynamic, by allowing services to hold a lease – a time period describing how long a service description should remain in the registry. UDDIe can co-exist with existing UDDI – and has been implemented as an open-source software.

1 Introduction and Motivation

Providing a Service interface to component based systems has become an important recent research and development challenge. A “service” may be defined as a behaviour that is provided by a software component, for use by another, based only on an interface. The interface acts as a contract between services, enabling the discovery, advertising, delegation and composition of services [3]. A number of tools have emerged recently which enable automatic creation of Web Service interfaces (WSDL and SOAP based) from existing Java based implementations, such as SOAP-switch [1] and Java2WSDL in the Web Service Toolkit [4]. Such tools are essential to bootstrap interest in Web Services, by allowing existing applications to be made available as Web Services. An alternative approach, adopted in the Web Services Invocation Framework [5], is to provide multi-protocol bindings – enabling automatic invocation of the relevant binding based on the content carried in the request. Hence, bindings are providing for COM objects, for HTTP requests etc.

Once an interface has been defined, it is important to place this into a registry service – so that other applications can automatically discover and make use of the service. In the context of Web Services, the discovery mechanism is generally provided by the Universal Description, Discovery and Integration (UDDI) registry – which may be organised along the same ways as Domain Name Servers. Hence, services may register with one or more registry services (using the same identifier) – and may be discovered by search distributed over one or more registries. Current UDDI implementations are however limited in scope – in that they only allow search to be carried out on limited attributes of a services – namely on serviceName (which may be selected by the service provider), keyReference (which must be unique for a service), or based on a categoryBag (which lists all the business categories within which a service is listed) [9]. Furthermore, public UDDI registries may contain a lot of listings for businesses that no longer exist, or sites that are no longer active [7]. The interaction between UDDI registries is also an importance concern – and there is no main consensus at the present time of who should own the root UDDI registries.

We extend UDDI as “UDDIe” to address some of these restrictions, and provide three main extensions: (1) support for “leasing” – to enable services to register with UDDI for a limited time period (to overcome the problem of missing or inconsistent links), (2) support for search on other attributes of a service – achieved by extending the businessService class in UDDI with propertyBag, and (3) extending the find method to enable queries to UDDI to be based on numeric and logical (AND/OR) ranges. The extensions allow UDDIe to co-exist with existing UDDI implementations – and enable a query to be issued to both simultaneously. We describe these extensions and their use in subsequent sections.
2 UDDI Extensions and Implementation

UDDIe uses an XML schema which extends that used in standard UDDI [9], by using the same specification and standards for the registry data structures and Application Programming Interface (API) specification for inquiring and publishing services. Extensions in UDDIe are based on four types of information: business information; service information, binding information; and information about specifications for services. A service may be discovered by sending requests based on service information. The extensions provided in UDDIe consist of the following:

- **Service Leasing:** Service providers may want to make their service available for limited time periods (for security reasons, for instance) – or the service may change often. UDDIe supports “Finite” and “Infinite” leases – where a finite lease can be immediate, or based on a future lease. When using finite leases, service providers must define the exact period for which the service should be made available for discovery in the registry. The lease period is restricted by the maximum allowable lease period defined by the UDDIe administrator. Depending on the type of application domain for which the UDDIe registry is to be used, the value of the maximum allowable lease may change. This parameter is left to the UDDIe administrator to set. For example, if a service provider is interested in publishing a service in UDDIe for two hours, but the maximum granted lease is one hour, publication of the service will be rejected by the registry. A “future lease” allows a service provider to make the lease period start at a future time – the service will only be discoverable once this lease has been activated. Alternatively, service providers may want to publish their services for an infinite period of time. Such leases are allowed in UDDIe, but only if the ratio of finite/infinite lease services is within a threshold (a parameter set by the UDDIe administrator).

- **Replication:** The UDDI Business Registry (UBR) is conceptually a single system built from a group of nodes that have their data synchronized through replication. A series of operator nodes each host a copy of the content, thereby replicating content among one another. Content may be added to the UBR at a single node, and that node operator becomes the content master. Any subsequent updates or deletes of the data must occur at the operator node where the data was inserted. UDDIe can be used as a private operator node that is not part of the UBR. Private nodes do not have data synchronized with the UBR, so the information contained within is distinct. The availability of private nodes is significant if an organisation considers sharing their service content a security problem. This is useful in instances where a company does not want to expose certain service offerings and business processes to others – for instance, suppliers set up to handle large contracts may not be able to handle individual customers.

In UDDIe a `businessService` [9] structure represents a logical service – and is the logical child of a `businessEntity` – the provider of the service. Service properties are contained in the `propertyBag` entities – such as the Quality of Service (QoS) that a service can provide, or the methods available within a service that can be called by other services (an important feature missing in current UDDI implementations). Figure 1 illustrates the attributes associated with a `property` – and consists of a `propertyName`, `propertyType` and `propertyValue`. Some of these are user defined attributes – such as `propertyType` – and can be `number`, `string`, `method` etc. Range based checks, for instance, are only allowed if the `propertyType` is a number.

The API for interacting with the registry system extends three classes within existing UDDI implementations. The extensions provided in the API include:

- **saveService:** This set of APIs is mainly used for publishing service details. This has been extended from the original UDDI system to introduce dynamic metadata for services. Such metadata could be used to represent attributes such as cost of access, performance characteristics, or usage index associated with a service, along with information related to how a service is to be accessed, and what parameters the service will return. The `saveService` call utilises the `propertyBag` mechanism provided in UDDIe.

- **findService:** This set of APIs is mainly used for inquiry purposes. In particular we extend this set of API from the original UDDI to include queries based on various information associated with services, such as Service Property and Service leasing.
getServiceDetails: This set of APIs is mainly used for requesting more detailed information about services, such as BusinessKey information etc. We extend this set of APIs to include Service Property information as well.

renewLease: This newly defined set of APIs is used by both the operator/UDDI administrator to control leasing information, and by the service provider (SP) to renew and set leasing information. The leasing concept works as follows: every service is associated with some lease information, either for limited duration or for an infinite time period. The maximum number of services with an infinite lease is controlled by the operator to efficiently maintain the registry. In the case of limited duration, the SP provides a start-from date and an expiration date for the lease period. The operator has control on setting up the default leasing period. Moreover, if a lease expires the SP could always renew the expired lease, provided that the request falls within the allowed number of times to renew a particular lease, which is controlled by the operator/UDDI administrator. When the lease period expires, the service becomes invalid and clients cannot make use of the expired services. It is important to continually renew a lease or request an infinite lease – and an event manager is used to alert all connected users if the lease of the service is about to expire. Figure 2 shows attributes in the lease element. renewLease is therefore invoked by the service provider – i.e. the entity responsible for advertising the service in the registry. Lease renewal does not apply to an infinite lease.

startLeaseManager: This newly defined set of APIs is used to monitor lease constraints, by generating processes to monitor and update the lease period. A service with an expired lease is removed from the registry. The operator has control over how often to run these processes.

In addition to these sets of APIs, we introduce support for a Qualifier-based search – to find services based on some property along with a qualifier value, such as EQUALTo, LESSThan and GREATERThan. Support for logical operations is introduced to enable querying for properties with logical AND/OR operators. We believe these extensions to the UDDI registry and query mechanisms would add a great search flexibility, making UDDI a more powerful search engine. The ability for UDDIe to co-exist with standard UDDI version is also an important aspect of this work – as we do not break compatibility with existing UDDI deployments. UDDIe may be downloaded\(^1\), as an open-source software. It extends the public domain implementation of UDDI available at uddi.org.

3 Uses

UDDIe can be utilised in any applications where providers register their service for a limited duration, or where users search for services based on range-based/logical criteria. This may include business providers who want to register their services with attributes which are numeric – such as service cost, or service quantity, or when specifying performance or Quality attributes associated with a service. Service registration for limited durations is particularly useful when the environment within which such a service is used is dynamic and may change often. Here, an infinite lease for a service may not be suitable – or a service provider may predict a future time when their service may be usefully deployable and opt for a future lease.

To use UDDIe, a user must define service properties, as illustrated in code fragment 1. A UDDIe proxy is established to enable a user to publish and interrogate the registry. A service called Math is then registered – and properties associated with the service specified. These properties are then registered into a propertyBag.

```
UDDIeProxy proxy = new UDDIeProxy();
proxy.setInquiryURL("http://localhost:8080/uddie/inquiry");
proxy.setPublishURL("http://localhost:8080/uddie/publish");

//Get Authorization by sending a // username and password
AuthToken token = proxy.get_authToken("ali", "ali");

//Define service name and add them to a vector
//The maximum allowed names is 5
Name name = new Name("Maths");
Vector names = new Vector();
names.add(name);

// Define Service properties
```

\(^1\)http://www.cs.cf.ac.uk/user/A.Shaikhali/uddie/
We then define a FindQualifier that may be used to search for the registered service – and issue a query:

```java
// Define Find Qualifier for
// property exact match (Logical AND)
FindQualifier findQualifier = new FindQualifier("exactPropertyMatch");
FindQualifier findQualifier2 = new FindQualifier("exactNameMatch");
FindQualifier qualifiers = new FindQualifier();
qualifiersVector.add(findQualifier);
qualifiersVector.add(findQualifier2);
qualifiers.setFindQualifierVector(qualifiersVector);
```

Once a query has been issued, details of the services which match our requests are printed using a ‘for’ loop, and returned in the ServicesInfos object, as illustrated in code segment 3. The first loop (using index ‘i’) returns a list of services, and the second (using index ‘j’) returns properties associated with each service.

```java
for (int i=0;i<services.size();i++)
{ ServiceInfo service = (ServiceInfo)services.get(i);
System.out.println("Service
returned Name: "+
service.getName().getText());
System.out.println("Service
returned Key: "+ service.getServiceKey());
eServiceDetail serviceDetail =
proxy.eServiceDetail(
service.getServiceKey());
Vector serviceVector =
serviceDetail.getBusinessServiceVector();
BusinessService returnedService =
(BusinessService)serviceVector.firstElement();
```

Figure 3 illustrates how a client can use the UDDI registry. A request is received from an external client using HTTP, and parsed by the Servlet. A client must distinguish whether the request is a UDDI request or a UDDIe request when publishing information about a service. However, when searching for a service, both UDDI and UDDIe requests are treated identically – as UDDIe extends the find service in UDDI with additional attributes. If these extensions are not implemented (as happens in existing UDDI registries) the default version of find service will be invoked – however, if service properties are also specified in the call, then the updated version in UDDIe will be used. Java classes to implement UDDIe extensions are stored as a jar file, and can be invoked by the parser when the extensions are encountered. Publishing service information (as illustrated in code fragments 1 and 2) will require the user to define properties associated with a service, and add these to the propertyBag.
3.1 Quality of Service Management

We currently use UDDIe to support Quality of Service (QoSM) management in the context of Grid Computing. A service in this case represents either a scientific code or a mathematical routine, and each service has a description provided in a WSDL document. A service also has other attributes, such as bandwidth, CPU and memory requirements encoded in the service interface. Some of these parameters, such as bandwidth, packet jitter and loss are measured by a “Bandwidth Broker” (a network monitoring program that is used to derive these low level metrics) [8]. These attributes allow a service user to choose between services based on their QoS attributes, rather than just their functional properties. The system is implemented in the context of our G-QoSM framework [2], whereby clients/applications send their requests for services with QoS properties to our QoS broker (code fragment 4 gives an example of the WSDL document used). The broker processes the request and submits the service request portion to UDDIe. The broker is not part of the UDDIe registry, but utilises it to record services with specific QoS attributes. The UDDIe registry replies with a list of services that support this particular query. Finally, the broker applies a selection algorithm to select the most appropriate service with respect to the client/application request and sends the result to the client/application. The selection algorithm that the broker uses is based on the Weighted Average (WA) concept. In this algorithm, we introduce the notion of a QoS importance level, whereby the client/application is requested to associate a level of importance, such as High, Medium or Low, with every QoS attribute. Based on this QoS importance level and the value of a QoS attribute, the algorithm computes the WA for every returned service and select the service with the highest WA.

```xml
<wsdl:definitions
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/
   targetNamespace="http://MyService-Interface">
   <wsdl:message
      name="printNameResponse">
     ....
   </wsdl:message>
   ....
   <service_cost> 5 </service_cost>
   <network_bandwidth> 256K </network_bandwidth>
   <memory> 48MB </memory>
   ....
</wsdl:definitions>
```

Code Fragment 4

4 Conclusion

The implementation of a registry for Web services, which extends UDDI, is outlined. It is particularly useful when storing service properties with range based attributes – and also enables search for services based on these properties. UDDIe and WSDL provide an important mechanism for specifying and deploying Web Services – especially when extending a WSDL document with additional attributes such as service quality and performance data. Using UDDIe, it is also possible to publish information about method calls (when wrapping object based implementations) available within a service (as its properties) and to subsequently search based on these method signatures. The extension via propertybag therefore allows a better way to make existing object based systems available as services – especially when used with tools such as SOAPswitch [1] and Java2WSDL [4].

References