Implementations of a Service-Oriented Architecture on Top of Jini, JXTA and OGSI

Nathalie Furmento, Jeffrey Hau, William Lee,
Steven Newhouse, and John Darlington

London e-Science Centre, Imperial College London, London SW7 2AZ, UK
lesc-staff@doc.ic.ac.uk

Abstract. This paper presents the design of an implementation-independent, Service-Oriented Architecture (SOA), which is the main basis of the ICENI Grid middleware. Three implementations of this architecture have been provided on top of Jini, JXTA and the Open Grid Services Infrastructure (OGSI). The main goal of this paper is to discuss these different implementations and provide an analysis of their advantages and disadvantages.

Keywords: Service-Oriented Architecture, Grid Middleware, Jini, JXTA, OGSI

1 Introduction

Service-oriented architectures are widely used in the Grid Community. These architectures provide the ability to register, discover, and use services, where the architecture is dynamic in nature. From all the initiatives to define standards for the Grid, a consensus seems to emerge towards the utilisation of such an architecture, as we can for example see with the OGSI initiative of the Global Grid Forum [4].

The ICENI Grid Middleware [5] is based on a service-oriented architecture (SOA) as well as on an augmented component programming model [6]. The goal of this paper is to show how the SOA has been designed to be implementation-independent. This gives us an open model where different low-level libraries can be plugged in.

The following of the paper is organised as follows. § 2 shows in details the design of the Service-Oriented Architecture, the different implementations are explained in § 3. A discussion on the current implementations is presented in § 4, before concluding in § 5.

2 Design of the ICENI's SOA

A typical Service-Oriented Architecture is presented in Figure 1. One can see three fundamental aspects of such an architecture:

1. *Advertising.* The Service Provider makes the service available to the Service Broker.

M. Dikaiakos (Ed.): AxGrids 2004, LNCS 3165, pp. 90–99, 2004.

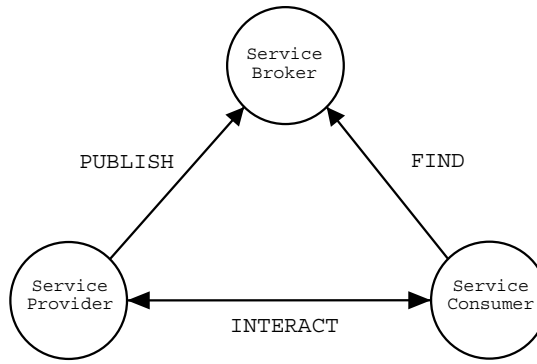


Fig. 1. The Service-Oriented Architecture

2. *Discovery.* The Service Consumer finds a specific Service using the Service Broker.
3. *Interaction.* The Service Consumer and the Service Provider interact.

In the context of ICENI, a Service Broker is represented by a public Computational Community or *Virtual Organisation*, where authorised end-users – the Service Consumers – can connect by using their X.509 certificates to query and access services. Once a service is advertised and discovered, any interaction with it is controlled by the service level agreement (SLA) that is going to define the entities that are allowed or denied access to the service, as well as the interval time the access is allowed or denied.

The lifetime of an ICENI service can be described by the three following steps: creation, advertising and discovery. Each of these steps is represented in ICENI by a set of interfaces. We are now going to explain these different steps and demonstrate them through a basic Counter Service.

2.1 Creation

A service is defined by its interface, i.e. the list of methods it provides. For example, the interface and the implementation of a counter service providing basic functionalities to add and subtract a value can be defined as shown in § A.

It is important at this level to notice there is no information on how the service is going to be implemented. We will see in the following sections how this abstract ICENI service is going to be implemented by using for example the Jini library.

The instantiation of the service is done through a call to the `IceniServiceMetaFactory`, which first instantiates a `IceniServiceFactory` for the used implementation, and asks this factory to return a new instance of the service. At that point, all the necessary classes to implement the abstract ICENI service are automatically generated. Calling for example the following line of code results in the creation of an ICENI service of the type `Counter`.

```
IceniService xServ = IceniServiceMetaFactory.newInstance("Counter");
```

2.2 Advertising

Once created, a service can be advertised on a specific domain or virtual organisation through the `IceniServiceAdvertizingManager` service. The service is advertised with a SLA that defines the access policy that will be used to enforce interaction with the service. The same service can be advertised in different organisations with different SLA's. This gives a flexible mechanism to control how different organisations may access the service, by allowing advertising the service capabilities as required.

The advertising of a service is done through a XML document that defines the SLA's of the service for all the virtual organisations where the service is to be made available. Appendix A shows a SLA XML document that gives access from Monday to Friday noon to all the persons connecting from the virtual organisation *public1* and belonging to the organisation *eScience*.

2.3 Discovery

By connecting to a virtual organisation, a service consumer can query a service and interact with it once discovered. ICENI provides different types of query such as interface matching that allow to listen to all services of a specific interface, or service data matching that allow to query services based on the value of their service data elements. The different steps to discover services are shown in Figure 2.

1. Instantiate a discovery manager.

```
IceniServiceDiscoveryManager xDiscovery =
                    IceniServiceDiscoveryManagerFactory.newInstance();
xDiscovery.setLocation("<address of virtual organisation>");
```

2. Instantiate a discovery query. Here we use the instantiation mechanism based on the interface of the service to listen to.

```
IceniServiceDiscoveryQuery xQueryCounter =
                    IceniServiceDiscoveryQueryFactory.newInstance(CounterService.class);
```

3. Register a listener. For every new service matching the query, the `servicePublished()` method will be called with the service as a parameter. Similarly, the `serviceUnpublished()` method will be called for each service disappearing from the virtual organisation.

```
xDiscovery.registerListener(xQueryCounter, new IceniServiceDiscoveryListener() {
   public void servicePublished(IceniService pService) {
     // code to execute when a new service is available
   } // end servicePublished
   public void serviceUnpublished(IceniServiceId pServiceId) {
     // code to execute when a service is no longer available
   } // end serviceUnpublished
});
```

Fig. 2. Discovery and Interaction with ICENI Services

2.4 Invocation

Any interaction with the service is controlled by an external entity, it first authenticates the service consumer through its X.509 certificate and authorises it against the policy of the service it wishes to access.

2.5 Other Requirements

On top of defining an interface, ICENI services also define a set of service data elements. A service data element is defined through a name and a value, the value being either a simple string or a well-formed XML document. These elements also define a liability time interval by specifying from when to when the service data is expected to be valid. Service Data elements are for example used by the JXTA implementation to perform discovery (See § 3.2), and are similar to the service data notion of OGSI (See § 3.3).

One of the main concerns in grid middleware is that security should be present at any level of the infrastructure. We need to provide basic security for remote calls such as mutual authentication, authorisation and integrity. We also need to know that the code downloaded across a network can be trusted. The SOA of ICENI provides an authentication and authorisation model which allows to check the access to its services, but this model needs to be extended into a full security model in order to be used in any production Grid. Applications such as health care applications dealing with patient records require strong security and encryption mechanisms.

3 Implementation of the ICENI's SOA

This section reviews the three different implementations of the ICENI's SOA by showing for each of them how the different aspects of the SOA have been implemented, as well as its advantages and disadvantages.

3.1 Implementation Using Jini

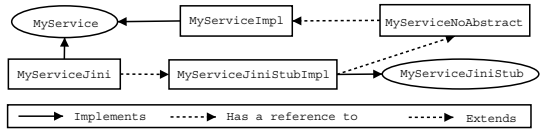
Jini network technology [11] is an open architecture that enables developers to build adaptive networks that are scalable, evolvable and flexible as typically required in dynamic computing environments. The first version of the ICENI Grid Middleware was directly implemented on top of the Jini API [7].

When using Jini, the following classes are automatically generated for a service named `MyService`.

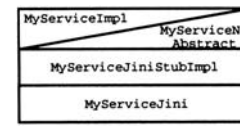
- **MyServiceJiniNoAbstract.java** extends the implementation of the service `MyService` to provide an implementation for all the basic ICENI/Jini mechanisms.
- **MyServiceJiniStub.java** is the main Jini interface extending the interface `java.rmi.Remote`. It acts as a proxy for `MyService` service, and defines exactly the same methods.

- **MyServiceJiniStubImpl.java** is the implementation of the interface `MyServiceJiniStub`. It uses a reference to `MyServiceJiniNoAbstract` to redirect all the method calls on the service.
- **MyServiceJini.java** implements the interface `MyService` by using a reference to `MyServiceJiniStub` toredirect an ICENI service's method call as a Jini service's method call.

Figure 3(a) shows an interaction diagram of these different classes and interfaces.



0.25 a) Class and interface diagram



b) Instantiation diagram

Fig. 3. Jini Implementation of an ICENI Service

Creation. This step creates an object of the class `MyServiceJini` and initialises it with the corresponding stub, i.e. an instance of the class `MyServiceJiniStubImpl`. We obtain an object as shown in Figure 3b).

Advertising. The object `MyServiceJiniStubImpl` – hold by the ICENI service created in the previous step – extends indirectly the interface `java.rmi.Remote`, it can therefore be made available in a Jini lookup service.

Discovery. The object returned from the Jini lookup service is a `MyServiceJiniStubImpl`. It is going to be wrapped in an instance of the class `MyServiceJini` before being returned to the listener. We obtain here a similar object to the one obtained when creating the service.

Invocation. Any method call is done on an instance of the class `MyServiceJini` and is finally redirected on an instance of the class `MyServiceImpl` as one can see in Figure 3.

Advantages/Disadvantages. The functionalities provided by the SOA of ICENI and the Jini library are basically the same. It was therefore very easy to implement the SOA on top of Jini without tying up ICENI to Jini and get an implementation-independent SOA. Moreover, as shown in [8], the Jini implementation is very scalable, these experiments are testing the performance of Jini when increasing the number of Jini services, they demonstrate a good result in the performance when discovering and accessing the Jini services. The potential problems when using Jini lie in security and in the connection of services across firewalls.

3.2 Implementation Using JXTA

Project JXTA [14] provides a set of XML based protocols for establishing a virtual network overlay on top of current existing Internet and non-IP based networks. This standard set of common protocols defines the minimum network semantics for peers to join and form JXTA peergroups – a virtual network. Project JXTA enables application programmers to design network topology to best match their requirement. This ease of dynamically creating and transforming overlay network topology allows the deployment of virtual organisation.

The fundamental concept of the ICENI JXTA implementation is the ICENI peergroup. The ICENI peergroup provides a virtual ICENI space that all ICENI JXTA services join. The peergroup contains the core ICENI services – `IceniServiceDiscoveryManager` and `IceniServiceAdvertizingManager`. These two services allow any services in the ICENI group to advertise their presence or to discover other services using ICENI ServiceData embodied in JXTA advertisements. Figure 4 presents an overview on how ICENI services behave when implemented on top of JXTA.

Creation. The creation of an ICENI service is just a matter of opening two separate JXTA pipes. Pipes are the standard communication channels in JXTA, they allow peers to receive and send messages. One of the two required pipes is a listening pipe that will listen for the control message broadcast to the whole ICENI peergroup. The other is the service's private ServicePipe. ServicePipes provides the communication channel for invocation messages. Depending on service functionality and requirement, these pipes could have varied properties such as encryption, single/dual-direction, propagation, streaming, ...

Advertising. Once joined the ICENI peergroup, a service can advertise its presence by publishing its `ServiceData` elements. This is a two step process: (1) Create a new `IceniServiceAdvertisement`. This is a custom advertisement that contains the ICENI service identifier `IceniServiceID` and the service data `ServiceData`. The Service Id can be automatically generated during advertisement creation and `ServiceData` will be converted into XML format and embedded into the advertisement; (2) Publish the advertisement by using the `IceniServiceAdvertizingManager` service from the ICENI peergroup.

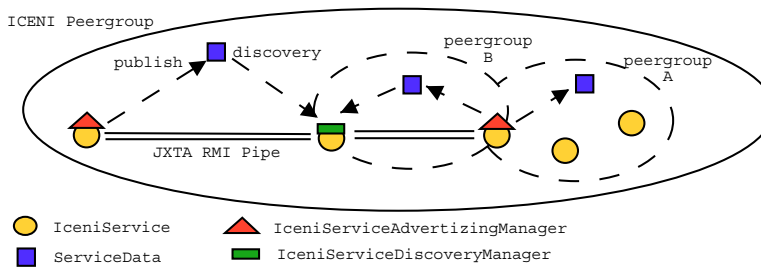


Fig. 4. JXTA Implementation of the SOA

Discovery. Peers in the ICENI peergroup can discover available services by using the `IceniServiceDiscoveryManager` service. Search can be conducted using service ID or service data elements.

Invocation. Invocation behaviour of ICENI JXTA services depends on the specific protocol each service is running. There are currently some projects working on providing different service architecture over JXTA such as JXTA-rmi [15] and JXTA-soap [16]. These projects wrap the original invocation messages (such as SOAP) into JXTA pipe messages and transport them through JXTA pipes to enable peers to invocate services using well-known service invocation API.

Advantages/Disadvantages. JXTA provides an architecture that gives middleware programmers the flexibility and ease of creating virtual organisations. It also provides an easy interface for publishing and discovering data in a peer to peer manner. Different invocation architectures can be overlayed over JXTA pipes. And finally, it is based on lightweight, firewall-proof, interchangeable network protocols. The potential problems with using JXTA as an architecture for building Grid middleware lies in security and performance. JXTA's P2P nature makes it harder to secure than traditional platforms. Also it will be difficult for the requirements of high performance grid application to be met by JXTA's current XML based messaging protocols.

3.3 Implementation Using OGSI

The Open Grid Services Infrastructure is an effort to build on the wide adoption of web services as an inter-operable foundation for distributed computing. The Grid Services Specification [17] describes a set of core port types using WSDL that are essential for the Grid setting. In ICENI, important notions of an `IceniService` are mapped to the relevant constructs in the `GridService` port type, such as meta-data as service data, and lease as termination time. Our implementation is based on the Globus Toolkit 3.0 [1] core distribution. It is the Java reference implementation of the Grid Services Specification. It allows Java objects to be deployed as OGSI services. The hosting environment acts as a SOAP processing engine that can be executed as an embedded HTTP server or operate as a Java Servlet inside a servlet engine. We have enhanced the implementation with an Application Programming Interface (API) for runtime dynamic deployment of service without the use of deployment descriptor. It serves as the kernel for the ICENI OGSI implementation.

Creation. To transparently transform an `IceniService` object into an OGSI-compliant service, the runtime system reflectively interrogate the class information of the service object and generate adapted classes that can be deployed through the deployment API. Adaptation is performed using the ASM byte-code generation library [3]. The adapted class is loaded from the byte stream into the running virtual machine using a specialised `ClassLoader`. The adapted object represents a service object that conforms to the requirement of GT3, such as an extension to the `GridServiceBase` interface. The adapted class acts solely as the delegate hosted by GT3 and directs invocation to the service object.

Advertising and Discovery. OGSI currently does not mandate a particular form of advertising and discovery mechanisms. We have chosen to use an instance of the `ServiceGroup` port type as a representation of a community. A `ServiceGroup` service is set up at a well-known location. When an `IceniService` is created, the Grid Service Handle of the OGSI-service representing this service object is published to the known `ServiceGroup`. Future implementations can experiment with using UDDI directory for long-lived services, such as Factory or Virtual Organisation Registry. For transient services, the Globus Toolkit 3.0 Index Service [2] can cater for the dynamic of temporal validity of service and its meta-data. Also, it provides a rich query mechanism for locating service instances based on their service data and port types.

Invocation. When a client locates an `IceniService` from the `IceniServiceDiscoveryManager`, the OGSI implementation returns a Java Reflection Proxy implementing the interfaces expected by the client. The proxy traps all invocations on the object. The invocation handler uses the JAX-RPC [13] API to marshal the parameters into SOAP message parts based on the WSDL description of the service.

Advantages/Disadvantages. The OGSI-compliant implementation allows ICENI services and clients to communicate through an open transport and messaging layers instead of the proprietary RMI protocol used by Jini. Also, non-ICENI clients can interact with ICENI services as if they are OGSI-compliant services. The extensible nature of OGSI permits different transport and messaging protocols to be interchanged. Our current implementation uses the web service security standards for encrypting message as well as ensuring authenticity of the caller. One disadvantage of the current invocation model is that ICENI clients can only transparently invoke OGSI services that originate from an ICENI Service. This is due to the fact that the Java interface of the OGSI service is pre-established before the conversation. For ICENI client to invoke an external OGSI service, stubs need to be generated at compile-time, or the Dynamic Invocation Interface of the JAX-RPC API could be used instead. Other disadvantages are GT3 is resource hungry, we would need a lightweight OGSI implementation to provide a valid ICENI/OGSI implementation. Moreover, the XML to Java marshaling is expensive, not automatic for complex types, and as for JXTA, XML based messaging protocols cannot meet the requirements of high performance grid application.

4 Discussion

The three implementations we have presented all provide the basic functionalities needed by the ICENI Service-Oriented Architecture at different levels of implementation difficulty. The JINI implementation offers good performances, the two other implementations being based on XML messaging protocols are not as promising, but offer better security models. Working on these three implementations proved to be very beneficial as it showed us that a valid and robust

SOA can only be obtained by good performances and a powerful and extensible security model.

We believe that these concerns can be dealt with by using Jini 2.0 [12]. This new version of the Jini Network Technology provides a comprehensive security model which one of the main goals is to support pluggable invocation layer behaviour and pluggable transport provider. We could therefore use OGSI instead of RMI as a remote communication layer, and benefit of the encryption and authentication features of the web service security standard.

To allow our three implementations to inter-operate and hence be able of getting a virtual organisation composed for example of ICENI/Jini services and ICENI/JXTA services, we have developed a OGSA Gateway that allows ICENI services to be exposed as Grid Services [5]. This allows us for example the following configuration: use Jini inside a local organisation, and use JXTA to cross boundaries between networks potentially configured with firewalls.

In order to improve search capability of ICENI services, an adaptation framework is being developed. The ICENI Service Adaptation Framework [9] builds on top of ICENI middleware to provide ways of annotating services using Resource Description Framework (RDF) and The Web Ontology Language (OWL). Semantic annotated services enable users to search through capability rather than static interface definitions. Once user's requirement is semantically matched with a semantic service, an adaptation proxy conforming to user's interface requirement is automatically generated. The adaptation proxy provides a implementation and architecture independent way for both the client and the server to invoke the required functionality.

5 Conclusion

We have shown in this paper the design of a Service-Oriented Architecture for a Grid Middleware that is implementation-independent. This Service-Oriented Architecture has been successfully implemented on top of Jini. We are currently prototyping the JXTA and the OGSI implementations of the SOA.

These three implementations all provide a useful subset of the functionalities of a Grid Middleware, we are now planning to work on a new implementation which will provide a full security model by using characteristics of our existing implementations.

The ICENI Grid Middleware has been used to develop high level grid services such as scheduler services [18] or visualisation services [10].

References

1. The Globus Toolkit 3.0. `http://www-unix.globus.org/toolkit/download.html`
2. GT3 Index Service Overview. `http://www.globus.org/ogsa/releases/final/docs/infosvcs/indexsvc_overview.html`
3. E. Bruneton *et al.* ASM: A Code Manipulation Tool to Implement Adaptable Systems. In *Adaptable and Extensible Component Systems*, France, Nov. 2002.

4. Global Grid Forum. http://www.gridforum.org/
5. N. Furmento, W. Lee, A. Mayer, S. Newhouse, and J. Darlington. ICENI: An Open Grid Service Architecture Implemented with Jini. In *SuperComputing 2002*, USA, Nov. 2002.
6. N. Furmento, A. Mayer, S. McGough, S. Newhouse, T. Field, and J. Darlington. ICENI: Optimisation of Component Applications within a Grid Environment. *Parallel Computing*, 28(12):1753–1772, 2002.
7. N. Furmento, S. Newhouse, and J. Darlington. Building Computational Communities from Federated Resources. In *7th International Euro-Par Conference*, volume 2150 of *LNCS*, pages 855–863, UK, Aug. 2001.
8. N. Furmento *et al.* Performance of ICENI/Jini Service Oriented Architecture. Technical report, ICPC, 2002. http://www.lesc.ic.ac.uk/iceni/reports.jsp
9. J. Hau, W. Lee, and Steven Newhouse. Autonomic Service Adaptation using Ontological Annotation. In *4th International Workshop on Grid Computing, Grid 2003*, USA, Nov. 2003.
10. G. Kong, J. Stanton, S. Newhouse, and J. Darlington. Collaborative Visualisation over the Access Grid using the ICENI Grid Middleware. In *UK e-Science All Hands Meeting*, pages 393–396, UK, Sep. 2003. ISBN 1-904425-11-9.
11. Jini Network Technology. http://www.sun.com/software/jini/
12. Jini Network Technology, v2.0. http://developer.java.sun.com/developer/products/jini/arch2_0.html
13. Sun Microsystems. Java API for XML-Based RPC 1.1 Specification. http://java.sun.com/xml/jaxrpc/index.html
14. Project JXTA. http://www.jxta.org/
15. Project JXTA-rmi. http://jxta-rmi.jxta.org/servlets/ProjectHome
16. Project JXTA-soap. http://soap.jxta.org/servlets/ProjectHome
17. S. Tuecke *et al.* Open Grid Service Infrastructure (OGSI) v.1.0 Specification, Feb. 2003.
18. L. Young, S. McGough, S. Newhouse, and J. Darlington. Scheduling Architecture and Algorithms within the ICENI Grid Middleware. In *UK e-Science All Hands Meeting*, pages 5–12, UK, Sep. 2003. ISBN 1-904425-11-9.

A The Counter Service Example

– **Interface for a Counter Service**

```
public interface CounterService extends ResourceService {
    public int addValue(int pValue) throws IceniServiceException;
    public int subtractValue(int pValue) throws IceniServiceException;
} // end interface CounterService
```

– **Implementation for a Counter Service**

```
public abstract class Counter extends ResourceImpl implements CounterService {
    protected int _counter = 0;
    public int addValue(int pValue) throws IceniServiceException {
      _counter += pValue; return _counter; }
    public int subtractValue(int pValue) throws IceniServiceException {
      return addValue(-pValue); }
} // end class Counter
```

– **Service Level Agreement for a Counter Service**

```
<publicDomain name="public1">
 <policy:accessPolicy>
   <policy:allow startDay="monday" stopDay="friday" stopHour="12" stopMn="00">
    <policy:entity type="organisation" name="eScience"/>
   </policy:allow>
 </policy:accessPolicy>
</publicDomain>
```