Discovery and control of Jini services within an OSGi framework

Confidential, Draft
RFC-0031
16 Pages

Abstract

The Jini network architecture is to become widely used. For OSGi-compliant gateways to be ready to meet future customer needs in that direction, there is a need for a standardized API, which enables OSGi services to interact with Jini networks using Jini services or being able to act as such.
This RFC presents an API that allows discovery and control of Jini services within an OSGi framework, and export OSGi Services as Jini services.

0 Document Information

0.1 Table of Contents

0.2 Status

This document specifies an API for discovery and control of Jini services within an OSGi framework for the Open Services Gateway Initiative, and welcomes discussion and suggestions for improvements. Distribution of this document is unlimited within OSGi.

0.3 Acknowledgement

I want to thank Iliana Kostova and the technical writer team in ProSyst that helped me to write this RFC.

0.4 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in [1].

`Source code is shown in this typeface.`

0.5 Revision History

The last named individual in this history is currently responsible for this document.

Revision	Date	Comments
Initial	Sep 05 2001	First draft based on RFP 24 Jini Driver API. Pavlin Dobrev, ProSyst Software AG p.dobrev@prosyst.com Krasimira Velikova, ProSyst Software AG k_velikova@prosyst.bg
Second	Nov 29 2001	Updates from October OSGi member meeting in San Francisco. Pavlin Dobrev, ProSyst Software AG p.dobrev@prosyst.com
Third	Dec 21 2001	Updates from DEG conference call at 20th of December 2001. Pavlin Dobrev, ProSyst Software AG p.dobrev@prosyst.com
Fourth	Feb 20 2002	Updates from January OSGi member meeting in Las Vegas. Updates from DEG conference call at 14th of February 2001. Pavlin Dobrev, Svetozar Dimov, ProSyst Software AG p.dobrev@prosyst.com, s_dimov@prosyst.bg

Fifth	May 17 2002	Updates according to joint CPEG/DEG Meeting April 16, 2002, Fairfax, VA. • Minor editing changes • Added “jini” prefix to LUS_EXPORT_GROUPS, CM_LUS_EXPORT_GROUPS and CM_LUS_IMPORT_GROUPS • Removed CM_PID from interface • Added, “The Jini Driver is defined as a Service Factory. For every bundle a different set of Service Templates is maintained. Jini Driver registers all Jini Services that match at least one Service Template.” • Added, “The kind of services that can be exported to the jini network is defined in point [3.1.2.1].” • Added point “3.2.1 Java RMI package” that explains RMI requirement.” • Added point “6 Implementation issues” that discuss dynamic importing and Java Serialization.” Pavlin Dobrev, ProSyst Software AG p.dobrev@prosyst.com

1 Introduction

The goal of this RFC is to specify an API that can be used to develop OSGi bundles that interoperate with Jini services in a Jini network.

The Jini network technology provides simple mechanisms that enable devices to form impromptu communities, which can be assembled without any planning, installation or human intervention. Each device provides services that other devices in the community may use. These devices offer their own interfaces, ensuring reliability and compatibility.

The Jini technology uses a Jini *lookup service* (LUS), where devices and services register themselves. When a device is attached to a network, it uses an add-in protocol, called *discovery and join*. The device first locates the lookup service (discovery), and then uploads an object that implements all of its service interfaces (join).

To leverage the Jini architecture from within an OSGi framework, this RFC defining an API that can:

- Enable access to Jini services from within an OSGi framework; and
- Provide OSGi services to members of a Jini community.

This API acts as a bridge between a Jini network (community) and an OSGi framework. Using this API, OSGi services from the framework can be exported easily to the Jini network, and Jini services from the Jini network can be imported into the OSGi framework. This results in two possible transformations: Jini-to-OSGi and OSGi-to-Jini. The *Jini Driver* module in the OSGi framework is responsible for these transformations.

2 Motivation and Rationale

The Jini network architecture is to become widely used. For OSGi-compliant gateways to be ready to meet future customer needs in that direction there is a need for a standardized API, which enables OSGi services to interact with Jini networks using Jini services or being able to act as such.

3 Technical Discussion

3.1 Jini Driver Bundle

The Jini Driver module is an OSGi bundle that operates according to the driver-like model of the OSGi Device Access Service and deals with Jini lookup services and OSGi services. The Jini driver discovers particular types of Jini services in the Jini lookup services and registers them as services in the OSGi framework. Similarly, the Jini Driver transforms to OSGi services any Jini lookup services that are discovered in the network, along with the Jini services that they currently hold (*importing* services). Conversely, the Jini Driver can discover services that are registered in the OSGi framework as compliant with the Jini technology and place them in the discovered Jini lookup services (*exporting* services).

By using the Jini Driver bundle, developers do not need to call the standard discovery and join mechanisms, because the bundle does this for them. From a developer's point of view, there is no difference between Jini services and conventional OSGi services; any operation defined in OSGi can be performed on these services.

The Jini Driver bundle provides two transformations: Jini-to-OSGi, where discovered Jini services are converted to OSGi services, and OSGi-to-Jini, where OSGi services are registered as Jini services in the discovered Jini lookup services. These two transformations are detailed next.

3.1.1 Jini-to-OSGi Transformation

The Jini-to-OSGi transformation enables applications in an OSGi framework to interact with Jini services quickly and easily by using the API defined by the OSGi framework specification. Thus, OSGi bundles need not include extra components to use Jini services.

On starting the Jini Driver Bundle registers as a listener for events fired when Jini Lookup Services are discovered using the standard Jini discovery utilities. The CM_LUS_IMPORT_GROUPS configuration property of the Jini Driver Bundle defines groups of LUS that the driver is interested in.

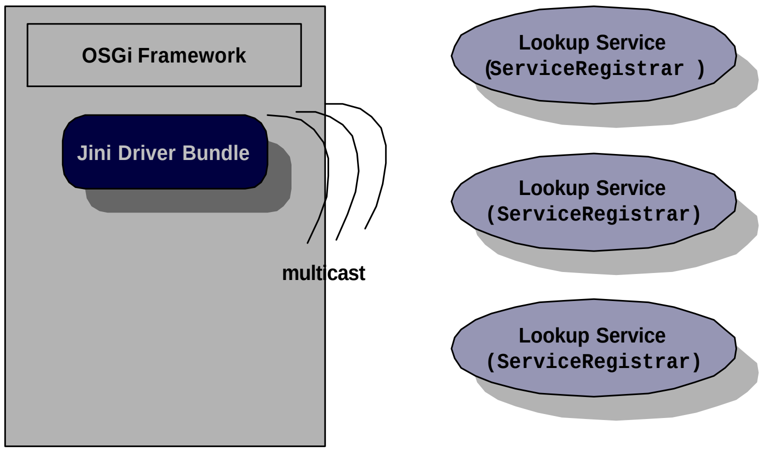
OSGi Framework
Jini Driver Bundle
multicast
Lookup Service (ServiceRegistrar)
Lookup Service (ServiceRegistrar)
Lookup Service (ServiceRegistrar)

Figure 1: The Jini Driver multicasts discovery message about Jini Lookup Services.

When a Jini Lookup Service is discovered, the Jini Driver Bundle retrieves its proxy object and registers it as a `net.jini.core.lookup.ServiceRegistrar` service in the OSGi framework. Next, the Jini Driver finds that Jini service registered in the Jini Lookup Services that matches a given template and registers it in the OSGi framework. This template is set by the method `setServiceTemplates` of the Jini Driver. This registration is possible only if the Jini service packages are exported in the framework. Jini Driver performs dynamic import of statically exported packages.

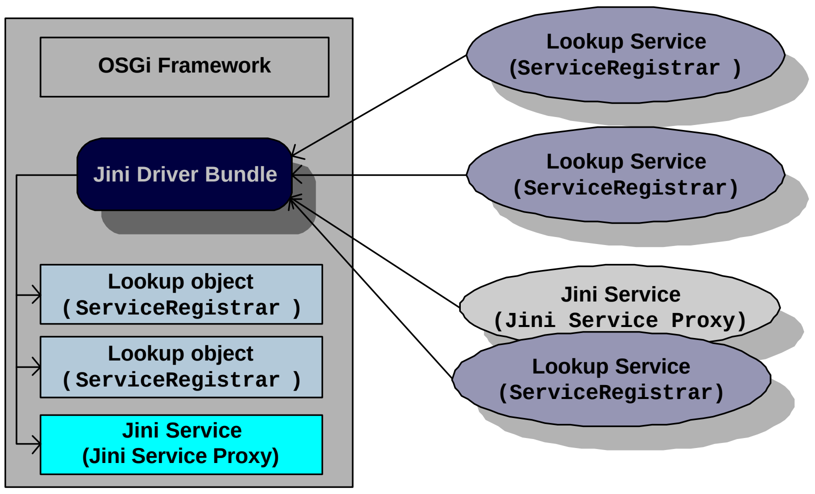
OSGi Framework
Jini Driver Bundle
Lookup object (ServiceRegistrar)
Lookup object (ServiceRegistrar)
Jini Service (Jini Service Proxy)
Lookup Service (ServiceRegistrar)
Lookup Service (ServiceRegistrar)
Jini Service (Jini Service Proxy)
Lookup Service (ServiceRegistrar)

Figure 2: Registering the objects of found Jini Lookup Services as services and importing Jini services.

3.1.2 OSGi-to-Jini Transformation

The OSGi-to-Jini transformation in the Jini Driver offers convenience to services running in the OSGi framework: these services need not register Jini services; instead, the Jini Driver can do so. This significantly reduces the complexity in definition and registration of Jini services. The kind of services that can be exported to the jini network is defined in point [3.1.2.1].

OSGi defines a service property that specifies a service as Jini-compliant. When a service is registered in the framework, the Jini Driver inspects this property. If it is a Jini-compliant service, the driver registers it in the discovered Jini lookup services. This property has a value that defines a possibility for a service to be registered into the Jini network. The Jini Driver Bundle is responsible for managing the life cycle of the Jini service in compliance with its state in the framework. In particular, when the service is registered in the framework, the Jini Driver must automatically register it in the Jini Lookup Services. When the service bundle is stopped or uninstalled, the Jini Driver removes it from the Jini Lookup Services.

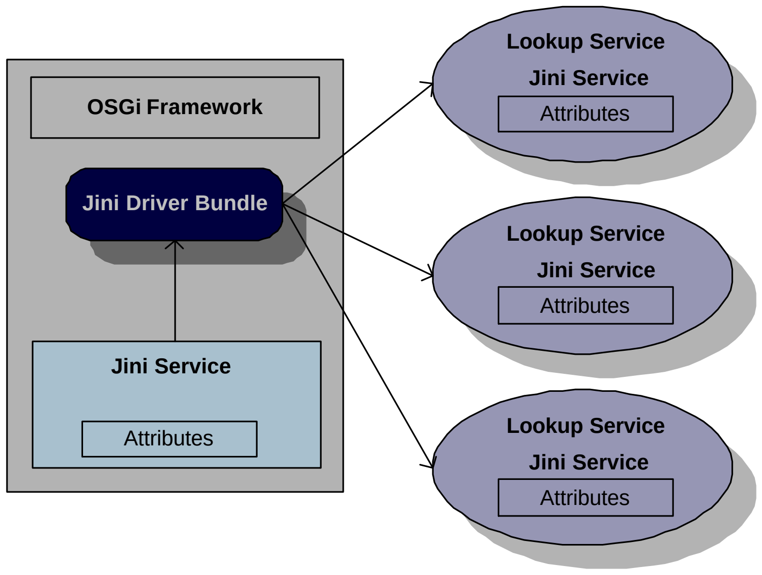

Figure 3: Automatic registration of OSGi services as a Jini services.

3.1.2.1 Jini-Export requirements for OSGi services

A Jini service is a conventional OSGi service having additional requirements. These requirements come from the Jini specification. The same service object of OSGi service is used as a service object of the Jini service that will be registered in LUS. This object will be serialized by LUS proxy and sent to the LUS implementation for storage. Later when a client requests the service, this is the object that will be given. To avoid confusion between the original service and the object that is sent to the Jini Lookup Service, I shall use the term *exported service object*. There are several things to note about the exported service object:

- The object must be serializable. Some objects, such as Swing's `JtextArea,` are currently not serializable and therefore cannot be used.
- The object is created in the service's VM. However, when it runs it will do so in the client's VM. It may need to be a proxy for the actual service that already exists in the OSGi framework.

3.1.3 Components

3.1.3.1 Jini Driver Service

The Jini Driver service represents the device driver functionality of the Jini Driver Bundle. The interface of this service is `org.osgi.services.jini.JiniDriver`. Using this service allows you to specify the Jini Lookup Services to register services in, as well as which Jini Lookup Service and Jini Services will be registered in the framework. The criterion to filter Jini Lookup Services is based on their group names. The criterion to filter other Jini Services is based on their service templates.

The Jini Driver is defined as a Service Factory. For every bundle a different set of Service Templates is maintained. Jini Driver registers all Jini Services that match at least one Service Template.

3.1.3.2 Service Registrar

As it was described above, when a Jini Lookup Service is discovered in the network, the Jini Driver registers it as a Service Registrar in the framework. The interface of the service is `net.jini.core.lookup. ServiceRegistrar`. You can perform any of the familiar Jini operations on these registrars, retrieving the available Jini services and so on.

The Service Registrar for a Jini Lookup Service owns the EXPORT, SERVICE_ID and ENTRY properties with values the attributes defined in the Jini network.

3.1.3.3 Jini Services

When a Jini service is imported into the framework, it is register as an OSGi service with the following set of properties:

- *DEVICE_CATEGORY* which has value a string array with the *"jini"* element.
- *SERVICE_ID* that contains the Jini service ID that is used in the LUS.
- *ENTRIES* which has value `net.jini.core.entry.Entry[]` holding the *attribute entries* that is stored in Jini Lookup Service.

The values of the properties are the same as an attributes of the Jini Service which is defined in the Jini network.

A Jini service that represent exported service object as defined in 3.1.2.1 should be registered as an OSGi service with the following properties:

- *DEVICE_CATEGORY* which would have value a string array with the *"jini"* element.
- *EXPORT* that marks that this OSGi service must be exported by the Jini Driver as a Jini service to the Jini network.
- *LUS_EXPORT_GROUPS* which should contain a string array of the LUS groups, which are of interest to the OSGi service.
- *SERVICE_ID* that should contain the Jini service ID to use in the LUS.
- *ENTRIES* which has value `net.jini.core.entry.Entry[]` holding the *attribute entries* to store in found Jini Lookup Services.

3.2 Package Dependencies

3.2.1 Java RMI package

The dependence on Java RMI is determined by the particular Jini implementation. If the Jini implementation is not RMI based the only requirement is to have some part of RMI because these classes and interfaces are used by the Jini Interfaces defined in [4].

Note that the Sun Jini Reference Implementation requires full RMI Support and a Java 2 compatible VM. Here is a minimal list of RMI that is required for non-RMI based Jini implementations:

- `java.rmi.MarshalException`

- `java.rmi.MarshalledObject`
- `java.rmi.NoSuchObjectExceptionRemote`
- `java.rmi.RemoteException`
- `java.rmi.UnmarshalException`

3.2.2 Jini packages

The Jini specifications define many Java APIs that are needed to implement a Jini Driver into the OSGi framework. The bundle programmer may define a Jini Library bundle providing the main Jini-related packages to requesting bundles. This bundle may export:

- `net.jini.core.entry` - contains the components of the Jini Entry Specification.
- `net.jini.core.lookup` - contains the components of the Jini Lookup Service Specification.
- `net.jini.core.event` - contains the ingredients of the Jini Distributed Event Specification.
- `net.jini.core.lease` - contains the API of the Jini Distributed Leasing Specification.
- `net.jini.core.discovery` - contains the unicast discovery part of the Jini Discovery and Join Specification.
- `net.jini.admin` - contains the interface that an administrable Jini service should implement.
- `net.jini.discovery` - contains the Jini Discovery Utilities Specification.
- `net.jini.lookup` - contains utilities for managing the communication with Jini Lookup Services.
- `net.jini.lookup.entry` - contains an utility to implement in order to specify a modification of an attribute entry

3.3 Example Usage

The following example demonstrates how to interact with the Jini services found in the framework.

```
import net.jini.core.lookup.*;
import net.jini.core.entry.*;

import org.osgi.framework.*;

import java.rmi.RemoteException;

class Example {

  interface SearchedInterface {
    void doesnothing();
  }

  public class SearchedEntry implements Entry {
    public String value;
    public SearchedEntry() {
      value = "";
    }

    public SearchedEntry(String value) {
      this.value = value;
    }
  }

// This method demonstrates how Jini Service can be found in LUS
// registered in framework

  Object SearchInJini(ServiceRegistrar lus) {
    Class[] classes = {SearchedInterface.class};
    Entry[] entries = {new SearchedEntry("Some info")};

    ServiceTemplate template = new ServiceTemplate(null, classes, entries);

    try {
      return lus.lookup(template);
    } catch (RemoteException ex) {
    }

    return null;
  }

// This method demonstrates how the Jini service can be found in
// framework using an LDAP search filter

  ServiceReference SearchInFramework(BundleContext bc) {

    String filter = "(objectClass=" + SearchedInterface.class.getName() + ")";
    Entry entry = new SearchedEntry("Some info");

    try {
      ServiceReference[] ref = bc.getServiceReferences(null, filter);

      if (ref == null) {
        return null;
      }

      for (int i = 0; i < ref.length; i++) {
        if (entry.equals(ref[i].getProperty("ENTRY"))) {
          return ref[i];
        }
      }
    } catch (InvalidSyntaxException ex) {
    }

    return null;
  }
}
```

4 API Specification

4.1 org.osgi.service.jini

Interface JiniDriver

public interface **JiniDriver**

A basic interface for a Jini Driver.

This Driver acts as a bridge between a Jini network (community) and an OSGi framework. Using it OSGi services from the framework can be exported to the Jini network, and Jini services from the Jini network can be imported into the OSGi framework. This results in two possible transformations: Jini-to-OSGi and OSGi-to-Jini. The Jini Driver is responsible for these transformations.

An OSGi service is a Jini service if it is registered in the framework with the specified properties.

In OSGi-to-Jini transofmation the driver registers OSGi services as a Jini services in the discovered LUS.

In Jini-to-OSGi transformation it registers with the framework all discovered LUS and services in the LUS matching the given template.

The Jini Driver can be configured through a set of properties to export/import Jini Services.

The properties `DEVICE_CATEGORY`, `EXPORT`, `LUS_EXPORT_GROUPS`, `SERVICE_ID`, `SERVICE_TYPES` and `ENTRIES` are service register properties for particular Jini Service (imported or exported).

The properties `CM_LUS_IMPORT_GROUPS` and `CM_LUS_EXPORT_GROUPS` are for configuration of the Jini Driver. These properties are kept in the Configuration Management Service defined by OSGi.

Field Summary	
`static java.lang.String`	**`CM_LUS_EXPORT_GROUPS`** Optional property, which should be a string array, containing the LUS groups that the driver is interested in, when exporting framework services to the Jini network.
`static java.lang.String`	**`CM_LUS_IMPORT_GROUPS`** Optional property, which should be a string array, containing the groups of LUS, that the driver is interested in, when importing Jini services.
`static java.lang.String`	**`DEVICE_CATEGORY`** Constant for the value of the service property `DEVICE_CATEGORY` used by all Jini services.
`static java.lang.String`	**`ENTRIES`** Optional property, which should be an `Entry` array, holding the attributes set of the framework service that represents Jini proxy in the registration with a

All Page Within This Box

	LUS.
static java.lang.String	**EXPORT** The `Export` service property is a hint that marks an OSGi service to be picked up and exported by the Jini Driver in the Jini network.
static java.lang.String	**LUS_EXPORT_GROUPS** Optional property, which should contain a string array of the LUS groups that are of interest to the OSGi service.
static java.lang.String	**SERVICE_ID** Optional property, which should contain a string representation of the Jini service ID.

Method Summary

net.jini.core.lookup.ServiceTemplate[]	**getServiceTemplates**() Gets the current set of templates that is used for searching and registering services registered in discovered LUS.
void	**setServiceTemplates**(net.jini.core.lookup.ServiceTemplate[] template) The Jini Driver is defined as a Service Factory.

Field Detail

4.1.1 DEVICE_CATEGORY

`public static final java.lang.String` **`DEVICE_CATEGORY`**

Constant for the value of the service property `DEVICE_CATEGORY` used by all Jini services.

Value: `jini`

See Also:
`org.osgi.service.device.Constants.DEVICE_CATEGORY`

4.1.2 EXPORT

`public static final java.lang.String` **`EXPORT`**

The `Export` service property is a hint that marks an OSGi service to be picked up and exported by the Jini Driver in the Jini network. Imported services do not have this property set.

The property has no value. The name of the property is `jini.export`;

4.1.3 LUS_EXPORT_GROUPS

`public static final java.lang.String` **`LUS_EXPORT_GROUPS`**

Optional property, which should contain a string array of the LUS groups that are of interest to the OSGi service. This overrides the property `CM_LUS_EXPORT_GROUPS` of the Jini Driver. If the value of this property is not defined, `CM_LUS_EXPORT_GROUPS` will be used. The name of the property is `jini.lus.export.groups`.

4.1.4 SERVICE_ID

`public static final java.lang.String` **`SERVICE_ID`**

Optional property, which should contain a string representation of the Jini service ID. It is used by the Jini Driver when exporting framework service. The driver automatically fills the values of this property when importing the Jini service. The name of the property is `jini.service.id`.

4.1.5 ENTRIES

`public static final java.lang.String` **`ENTRIES`**

Optional property, which should be an `Entry` array, holding the attributes set of the framework service that represents Jini proxy in the registration with a LUS. The name of the property is `jini.entries`.

See Also:
`Entry`

4.1.6 CM_LUS_IMPORT_GROUPS

`public static final java.lang.String` **`CM_LUS_IMPORT_GROUPS`**

Optional property, which should be a string array, containing the groups of LUS, that the driver is interested in, when importing Jini services. The driver discovers only the LUS members of at least one of these groups. It discovers all if the property is null or the property is not defined, and does not perform discovery if the length of the array is 0. If LUS are discovered, which after changing the value of this property are not members of the groups, all registered services from them are unregistered. The name of the property is `jini.lus.import.groups`.

4.1.7 CM_LUS_EXPORT_GROUPS

`public static final java.lang.String` **`CM_LUS_EXPORT_GROUPS`**

Optional property, which should be a string array, containing the LUS groups that the driver is interested in, when exporting framework services to the Jini network. The driver discovers only the LUS, which are members of at least one of these groups. It discovers all if the property is null or the property is not defined, and does not perform discovery if the length of the array is 0. If Jini Lookup Services are discovered, which after changing the value of this property are not members of the groups, the registration of all Jini services from the framework, which are registered with them, is cancelled. The name of the property is `jini.lus.export.groups`.

Method Detail

4.1.8 setServiceTemplates

`public void` **`setServiceTemplates`**`(net.jini.core.lookup.ServiceTemplate[] template)`

The Jini Driver is defined as a Service Factory. For every bundle a different set of Service Templates is maintained. This method sets a new set of ServiceTemplates (`net.jini.core.lookup.ServiceTemplates`) that are used for searching and registering services in the discovered LUS. A service registered in a LUS will be registered in framework if it matches at least one of the templates in one of the sets.

The `ServiceTemplate(null, null, null)` matches all services.

Parameters:
`template` - template to be added.

4.1.9 getServiceTemplates

`public net.jini.core.lookup.ServiceTemplate[] `**`getServiceTemplates`**`()`

Gets the current set of templates that is used for searching and registering services registered in discovered LUS. A service, registered in a LUS, will be registered in the framework if it matches at least one of the templates in this set

Returns:
an array containing templates or null if the set of templates is empty.

5 Security Considerations

Security in inherited from Jini.

6 Implementation issues

6.1 Dynamic import of statically exported packages

Because Jini dynamically imports code into the OSGi framework there is a requirement Framework to support dynamic import of statically exported packages.

6.2 Java Serialization

Note that the serialization protocol for java 1 and java 2 virtual machines is different. Jini uses java serialization as a protocol for exchanging java objects. They are two possible solutions that go round the problem. First is all JVM to be from the same class (Java 1 or Java 2 compatible). Second is implementation of Java serialization protocol of Java 2 under Java 1 compatible virtual machines.

All Page Within This Box

7 Document Support

7.1 References

[1]. Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.

[2]. http://www.jini.org

[3]. RFP 24 Jini Driver API

[4]. Jini Specifications, http://www.sun.com/jini/specs/

[5]. ProSyst mBedded Server documentation, http://www.mBeddedServer.com

7.2 Author's Address

Name	Pavlin Dobrev
Company	ProSyst Software AG
Address	Gustav-Heinemann-Ufer 54, 50968 Cologne, Germany
Voice	+359 2 963 10 04
e-mail	p.dobrev@prosyst.com

Name	Krasimira Velikova
Company	ProSyst Software AG
Address	Gustav-Heinemann-Ufer 54, 50968 Cologne, Germany
Voice	+359 2 963 10 04
e-mail	k_velikova@prosyst.bg

7.3 Acronyms and Abbreviations

OSGi	Open Service Gateway Initiative
LUS	Jini Lookup Service

7.4 End of Document