Resource and service registration and lookup in cooperating embedded systems

Magnus Persson
magnus.persson@md.kth.se
Department of Machine Design
Royal Institute of Technology – KTH, Sweden

Abstract

This paper investigates if service registration and lookup could become a usable component in future middlewares for cooperating embedded systems, e.g. wireless sensor networks and automotive networks. As a main part of this investigation, three different traditional middleware architectures have been investigated, focusing on how service registration and lookup is implemented in them. They are all targeted at a classical distributed computing environment – Jini, OSGi and UPnP. The possibilities for reuse of the architectural design patterns in middlewares for cooperating embedded systems is a keystone of the comparison. The main difference found between them was the degree of distribution – UPnP is totally distributed and Jini to some extent, while the OSGi specification does not take distributed applications into consideration at all.

1. Introduction

In traditional distributed computer systems, service register and lookup services (like name services) serve a very vital role. The examples are several - the most obvious example is the DNS (domain name service), but other examples are abundant, like the ARP (address resolution protocol) within TCP/IP, lookup services within P2P networks etc. Sometimes service lookup is very closely related to routing.

In the typical cooperating embedded systems, service registration and lookup has not played such a crucial role. Many traditional networked embedded systems (e.g. within industrial automation and traditional automotive networks) have been designed statically – i.e. no changes to the network after deployment have been foreseen. This means all services have been located at a specific node at design time, and service lookups have not been necessary as the address of the service is fixed during the whole lifetime of the system.

In wireless sensor networks, the situation has been the opposite. Wireless sensor networks typically are designed after the principle that each individual node in itself is unimportant. As service lookups typically return a specific network address, they aren’t compliant to this design assumption. Instead of using service lookups, most research within the wireless sensor networks area have instead focused on alternative approaches, like geographical routing.

There are two reasons why it is interesting to investigate architectural approaches for service registration and lookup also within cooperating embedded systems. The first is that there is a trend towards more dynamic behaviours in networked embedded systems and middleware-based systems. The assumption that the system structure is known at design time no longer holds true. [1] As more and more cooperating embedded systems will be composed in ways not foreseen at design time, with expectations of self-organization, service registration and lookup services become vital to the systems.

One specific application area where the benefits of this approach are becoming more obvious is within automotive embedded software, where a number of future application scenarios aren’t supported by the current software architectures, e.g. ad-hoc networking with mobile devices (cell phones, PDAs, music players etc). The requirements on configurational flexibility of these scenarios aren’t covered by current state-of-the-art vehicle electronic systems. This is exemplified by the DySCAS project [2]. DySCAS strives to develop a middleware for dynamically reconfigurable automotive systems to meet these new requirements. [3]

Secondly, there might be some applicability of the concept of lookup services to wireless sensor networks. Traditionally, most wireless sensor networks have been designed with all nodes identical. It is quite possible to envision heterogeneous wireless sensor networks, with more than one type of nodes. A lookup service could in such systems be used to locate more powerful or specially equipped nodes.

This recent development poses several interesting research questions relating to the architectural design of middleware. How do resources and services register themselves in the different middlewares? How do the naming/lookup services work? Are they applicable to cooperating embedded systems, or specifically, wireless sensor networks? Are there specific architectural issues to think about when designing a name/lookup service for a cooperating embedded system?

2. Jini

Jini [4] is a Java-based middleware for distributed applications, with the overall goal to make networked applications more flexible and easily administered. This is done by using several parts together: a set of components,
a specific programming model, and finally, services that can be made part of the system.

Jini also has a hierarchical structure. Several Jini groups can be joined hierarchically to form what is called a Jini federation, to make scaling to bigger networks than workgroup-sized ones possible.

2.1. Proxy objects

In Jini, the concept of a proxy object or simply proxy is central. A proxy is a local object that acts as a stand-in for a remote object. It presents the same interface as the local code, but issues related to distribution, like network-related functions, parameter and return value transmission are hidden from the user.

In Jini, unlike many other middleware architectures, the proxy object is not a part of the middleware itself, or created statically during design time, but its code is dynamically transmitted over the network at runtime and then dynamically added as part of the code of the connecting client application.

Using a proxy object included in the application at runtime means that the actual network protocol between the service and the client is abstracted away, and it becomes possible to upgrade the service network protocol without having to rewrite parts of the client application.

This is a vital difference between Jini and other distributed systems, and it makes exchanging or upgrading parts of the system a lot easier – one of the main design goals of the Jini middleware. [5]

2.2. Discovery, join and lookup

The actual discovery and lookup mechanisms in Jini are fairly simple, as shown in figure 1. A service that wants to join a Jini federation sends out a multicast packet over the LAN. When this packet is received by a lookup server (several can be used separately, e.g. because of reliability), a discovery response is sent back to the joining service. This response contains a proxy object to the lookup server, which thereafter is used to connect to the lookup server.

This proxy object can be used to register an offered service with the lookup server (to join) by placing a proxy object for the service itself in the lookup service(s).

Similarly, when a lookup is made, the response contains a proxy object which is used to connect to the service. This proxy object is used by the client to establish client-server communications directly with the server. [6]

3. OSGi

OSGi [7, 8] is a specification of an open, common, architecture for service based architectures. It specifies a standardized component oriented computing environment for networked services (what is also called service oriented architecture, SOA). The target devices are consumer electronics, PCs, cars, cellphones etc.

OSGi plays a complementary role to other technologies, including Jini and UPnP. While these concentrate on device interoperability, OSGi has a focus on delivery, deployment and remote administration of services for devices in a distributed network. Directly supporting distributed applications is not the main focus of OSGi. [9]

3.1 OSGi framework and component model

Within OSGi, applications are distributed in bundles, implemented by standard .jar files augmented with a special description file.

OSGi has implemented specific support for lifecycle management of the applications in the bundles, see figure 2. This provides the possibility to start/stop, update and uninstall a bundle. Resolving a bundle means to prepare starting of it by making sure that all the classes that the bundle is dependant on already has been loaded. In case any are missing, they are automatically loaded before the bundle is started.

Additionally, the OSGi specification includes some additional issues not covered here – e.g. a number of standard services and a security model.

![Figure 2. OSGi bundle life cycle states.](image)

3.2 Service registration in OSGi

OSGi has a simple service registration mechanism. A bundle publishes a service by registering it with the framework service registry at any moment during the STARTING, ACTIVE or STOPPING life cycle states. A service object registered with the framework is exposed to other bundles installed in the OSGi environment. This is done by a simple call to the registerService() Java function. The function returns a ServiceRegistration object, which can be used to change or remove the service registration at a later instance.

Except registering and searching for services, the service registry also provides a possibility to receive
notifications when services register or unregister themselves. [7]

4. UPnP

UPnP [10] is a set of networking protocols aimed at “transparent networking”. The protocols build on standardized protocols like TCP/IP, HTTP, UDP and XML. Basic abstractions that exist within UPnP are devices, services and control points. Within UPnP, no assumption is made on programming environment or programming language – the standard exclusively deals with interoperability between several devices, which could be implemented in any manner.

4.2 Brief UPnP Vocabulary

Some specific terms, as illustrated in figure 3, are used within the UPnP standard. A controlled device (or simply device) functions in the role of a server on the network and provides some kind of service. The computer using the service is called a control point.

![Figure 3. A UPnP control point invoking an action on a controlled device.](image)

4.3 Discovery and description

UPnP uses a markedly different approach to service lookup compared to the two other middlewares described in this paper. Instead of relying on one central point, the mechanisms are totally distributed in a network of peer nodes.

The UPnP specification is split in several phases as illustrated in figure 4, corresponding to the different steps of the process of starting and using a UPnP device.

0. Addressing
1. Discovery
2. Description
3. Control
4. Eventing
5. Presentation

![Figure 4. UPnP phases](image)

In UPnP, addressing is done by standard mechanisms like DHCP, and in the case that isn’t available, Auto-IP [11] is used instead.

After addressing, which is the 0th step of UPnP and basically a repetition of other standards, comes the first unique step of UPnP, discovery. When a device is added to the network, it advertises its services by a broadcast discovery message. Similarly, when a control point is added to a network, the UPnP discovery protocol lets the control point search for devices of interest on the network.

The discovery messages only contain minimal information about the device: type, identifier and a pointer to more detailed information (in the form of a URL to an XML description file).

If possible, a device should also send discovery messages when disconnected from a network, to advertise that it is no longer available.

After a device has been discovered, interested control points can send a description request by downloading the XML description files from the URL supplied in the UPnP discovery message. The UPnP description consists of two logical parts – a device description file containing a device description and one or several service description files containing service descriptions describing the capabilities of the device.

A device description file includes manufacturing information as model name and number, serial number, manufacturer name, URLs to specific web sites etc. For each service in the device, the device description lists the service type, name, and URLs for a XML service description file, URLs for control and eventing, and a presentation page for the device as a whole.

A control point which has received the description files can use the services that the UPnP device supplies. There are three forms of usage; control, eventing and presentation.

The control phase is used when the control point requests the device to invoke actions. This may change the state of the device.

Eventing allows a control point to monitor state changes in devices. Control points interested in state changes in a specific device subscribe to a service provided by the device. The device’s service will notify all registered control points upon state variable changes.

Finally, a device can optionally supply a URL for an administrative HTML-based presentation interface to allow for administration and monitoring with.

5. Discussion

The investigated middleware architectures differ in several ways. The most interesting architectural differences, degree of distribution and the support for code mobility within OSGi, are discussed below.

5.1 Centralization vs. distribution

The approaches taken by on the one hand OSGi and Jini, on the other hand UPnP, is markedly different
with respect to centralization. While OSGi has a totally
centralized approach and Jini has a mainly centralized
approach with its hierarchical architecture, both with a
centralized register for all services, UPnP has chosen a
totally distributed approach, where no information about
running services is centralized at all.

The first of these approaches has several
advantages: it is simple, and as long as the service register
is reachable, it can be guaranteed that a desired service will
be found. A (big) disadvantage of the centralized approach
is however scalability with network size.

In the distributed model of UPnP, it’s the other
way around. Even though the reliability of its discovery
protocol isn’t dependant on the availability of a central
register, it can’t be guaranteed that an available service
always will be found. Marked disadvantages are that it’s
not quite as simple, as many messages are necessary on the
network due to limited message size, and distribution of
name and lookup data. The distributed architecture of
UPnP is however an advantage in scalability.

5.2 Code mobility

One interesting approach taken in the Jini
middleware is that the code for connecting to the service is
not assumed to be known, neither at design time nor first
deployment. Transmitting the code for connection to the
service dynamically is an interesting concept. It is however
not possible to implement in today’s cooperating
embedded systems. Most of them totally lack the ability to
execute code dynamically. Moreover, the Jini framework,
thanks to its Java-based implementation, assumes a very
homogeneous platform, which simply normally is not
available in embedded networks.

6. Related work

Within the RUNES project, which strives to develop an
adaptive middleware for heterogeneous systems built
around wireless sensor networks, related work and
development is being done; including investigation on the
role of name services and discovery protocols within
middleware for wireless sensor networks. [12]

7. Conclusions and future work

Service registration and lookup have so far not
been important in architectures of cooperating embedded
systems. This is mostly because traditional embedded
systems have been statically designed and relatively simple
– the addressing was already finalized during design time,
and hence no such features were necessary. As embedded
systems become more and more dynamic, it has become an
issue and middleware for embedded applications will need
to provide such services.

The question on exactly how this is done best is
still open – this paper has presented a couple of solutions
to the problem from the traditional middleware domain.
The main difference between them has been found to be
the amount of distribution assumed in the architecture.
While OSGi doesn’t take distribution into consideration at
all, Jini has provisions for hierarchical structures. Finally,
UPnP is a totally distributed approach consisting solely of
peer nodes. A future research question is to practically or
through simulation find out in what contexts each of these
fit best – when scalability issues become a problem with the
centralized approaches, both in terms of system size
and of how rapidly the system changes.

In wireless sensor networks, the situation is
different. In the traditional sensor network no node is more
important than the others – and because of that, there has
been no need to define services that are able to locate
specific nodes. Also, as the need for communication with
individual nodes is quite limited, some sensor networks
have chosen to use geographic routing or other special
routing schemes instead.

It would be interesting to practically investigate
the implications on wireless sensor network architectures it
would have to make them heterogeneous, and use a simple
discovery mechanism, similar to the UPnP one, to locate
more powerful nodes. Even if today’s sensor networks
generally are homogeneous, one could easily envision
heterogeneous networks in the future, and the potential
benefits. Some nodes could have larger batteries, more
powerful radios or specialized sensing equipment.
Diversifying and making the nodes more specialized could
mean advantages for the sensor network as a whole.

References

Ekelin, Dejiu Chen, Martin Törngren, Gerrit de Boer,
Isabell Jahnich, Simon Burton, Ola Redell,
Alexander Weber, Vasco Vollmer: A Future
Dynamically Reconfigurable Automotive Software
System, at Elektronik im Kraftfahrzeug, Dresden,
Germany, June 27-28 2006
[4] Sun Microsystems, Jini™ Architectural Overview,
January, 1999
[5] Sun Microsystems, Jini Specification, version 2.1,
available (accessed on 28 May 2007) from
http://java.sun.com/products/jini/2_1index.html
centric Computing, Communications of the ACM,
IEEE, Vol.42, Iss.7, pages 76-82, July 1999
[7] OSGi Alliance, OSGi Service Platform Specification,
Release 4, Version 4.01, July 2006
[8] OSGi Alliance, About the OSGi Service Platform –
Technical Whitepaper, Revision 4.1, 11 November
2005
[9] Richard S Hall and Humberto Cervantes, Challenges
in Building Service-Oriented Applications for OSGi,
Communications Magazine, IEEE, Vol.42, Iss.5,
pages 144-149, May 2004
[10] UPnP Forum, UPnP Device Architecture 1.0,
Document Version 1.0.1, 20 July 2006
[11] RFC 3927, Dynamic Configuration of IPv4 Link-
Local Addresses, http://www.ietf.org/rfc/rfc3927.txt