Blank page

UPnP Services and Jini Clients

Jan Newmarch
Monash University

Home middleware

- One of the next battlefields for systems will be the home
- Possible middleware candidates include
  - UPnP
  - Jini
  - HAVi
  - many others
- Home networks will require
  - Zero configuration
  - Lightweight middleware

Jini

- Relatively heavyweight: the Jini libraries are over 1M in code size
- A pain in the b*tt to set up
- Won’t work easily in a badly setup or "incomplete" environment
- Poor industry support
- No Jini devices exist

UPnP

- Fairly lightweight, uses existing protocols: SOAP, HTTP, TCP, UDP, Multicast
- Easy to set up
- Will work in minimally configured environments
- Strongly supported by a Microsoft backed industry group
- Several UPnP devices exist

Technical comparison

	Jini	**UPnP**
Service adverts	via a Lookup Service (LUS)	Direct multicast
Service discovery	via an LUS	Direct multicast
Discovery protocol	fixed	fixed
Service invocation protocol	Unspecified (JRMP, Jeri, IIOP, etc)	SOAP
Object references	Java proxy objects	URLs or XML documents
Mobility	LUS proxy (to client and service) Service proxy (to client) Method call arguments (to service) Method call result (to client) Listener registration (to service and LUS) Unknown class definitions downloaded from an HTTP server	None
Language	Java only	Agnostic

Object mobility

- Requires object introspection (for marshalling)
- Requires specialised class loader (e.g `RMIClassLoader`)
- Requires security management (e.g. `RMISecurityManager`)
- Requires HTTP server
- Not available in e.g. KVM

Method level object mobility

- UPnP avoids object mobility by using primitive data types: int, boolean, string, etc
- Jini can equally use these types
- If Jini services only use primitive data types, then there is no method-level object mobility - just like UPnP
- This requires no change to any of Jini - just how the interfaces are specified
- Paranthetic note: UPnP should be in trouble over A/V structures - it avoids them by hiding XML documents in primitive strings

LUS proxy mobility

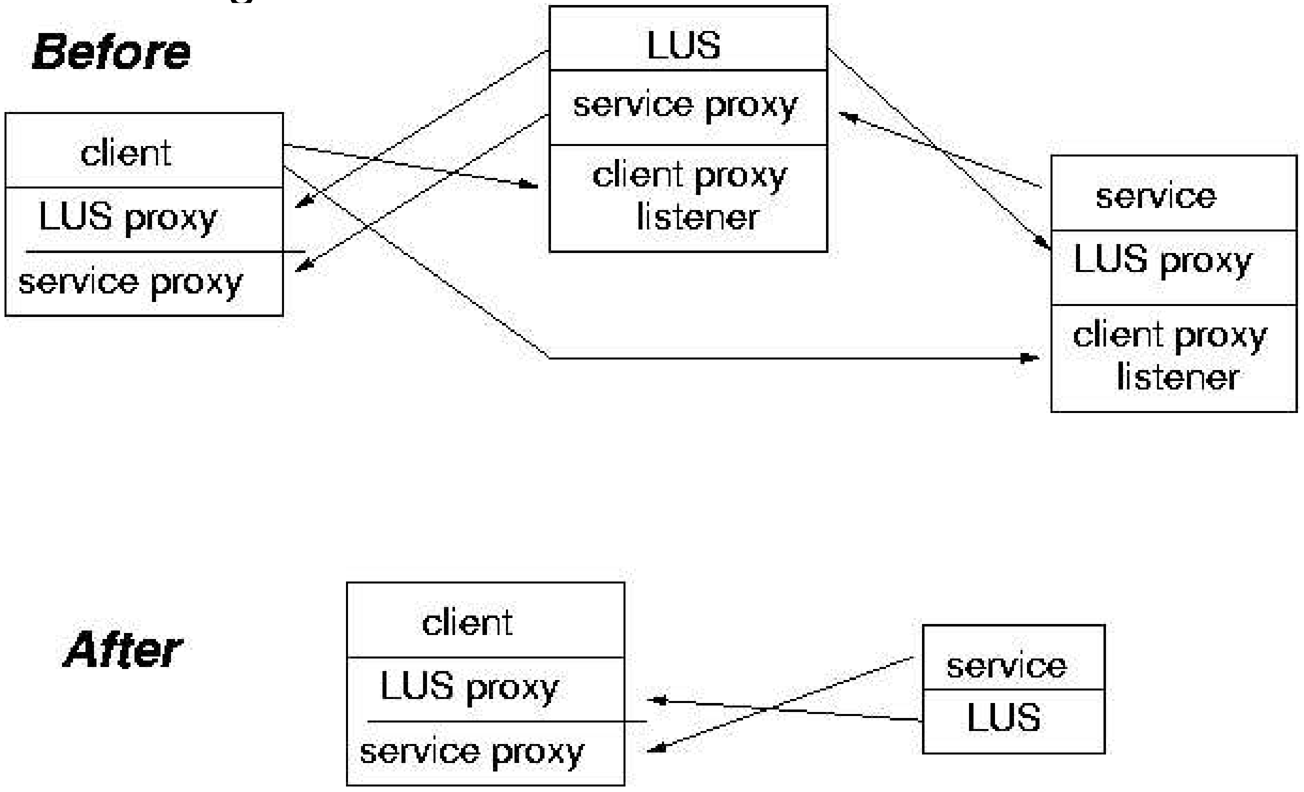
- A home service is likely to be on a small footprint device: temperature sensor, infrared sensor, washing machine, fridge, etc
- A home service may not be able to support object mobility *to* the service
- Embed an LUS within the service, so mobility of an LUS proxy *to* the service is avoided

Listeners

- Event listeners could be registered with the LUS proxy...
- Listeners could be registered with the service...
- ...just don’t move them to the service/LUS

Object mobility changes

The change is shown as



Removing the HTTP server

- To load an unknown class, the JVM must be able to find the class definition
- For new mobile objects, the class definitions are usually stored on an HTTP server
- The Java `Proxy` class reduces this: a proxy supporting an interface can be generated on the fly
- The generated proxy uses an `InvocationHandler` to deal with method calls
- Jini and RMI have standard invocation handlers known to the client so no mobile code is needed for protocols such as JRMP, Jeri, IIOP, etc
- Additional classes unknown to the client may need to be downloaded - not a problem for UPnP data types
- Anyway, a UPnP device contains an HTTP server

Situation so far: ZeroConf

- Jini can function at the level of UPnP data types trivially
- Combining the LUS with the service can remove all mobile code to the service
- Use of `Proxy` and a standard invocation handler avoids use of an HTTP server
- *Result:* Zero Configuration of the service!

Lessons from JMatos

- JMatos is a Jini LUS, not Open Source
- The Sun LUS proxy communicates back to the LUS itself for any queries
- The JMatos LUS is self-contained and does not need to talk to its original server - a "complete" proxy
- This reduces network traffic, speeds up responses, etc

Lessons for Jini/UPnP

- It can be more convenient to send a "complete" LUS proxy
- Network traffic can be reduced
- There is no need to define a "proxy to source" protocol
- But it isn’t necessary...
- What follows can be done more easily using a complete proxy, but can be done anyway

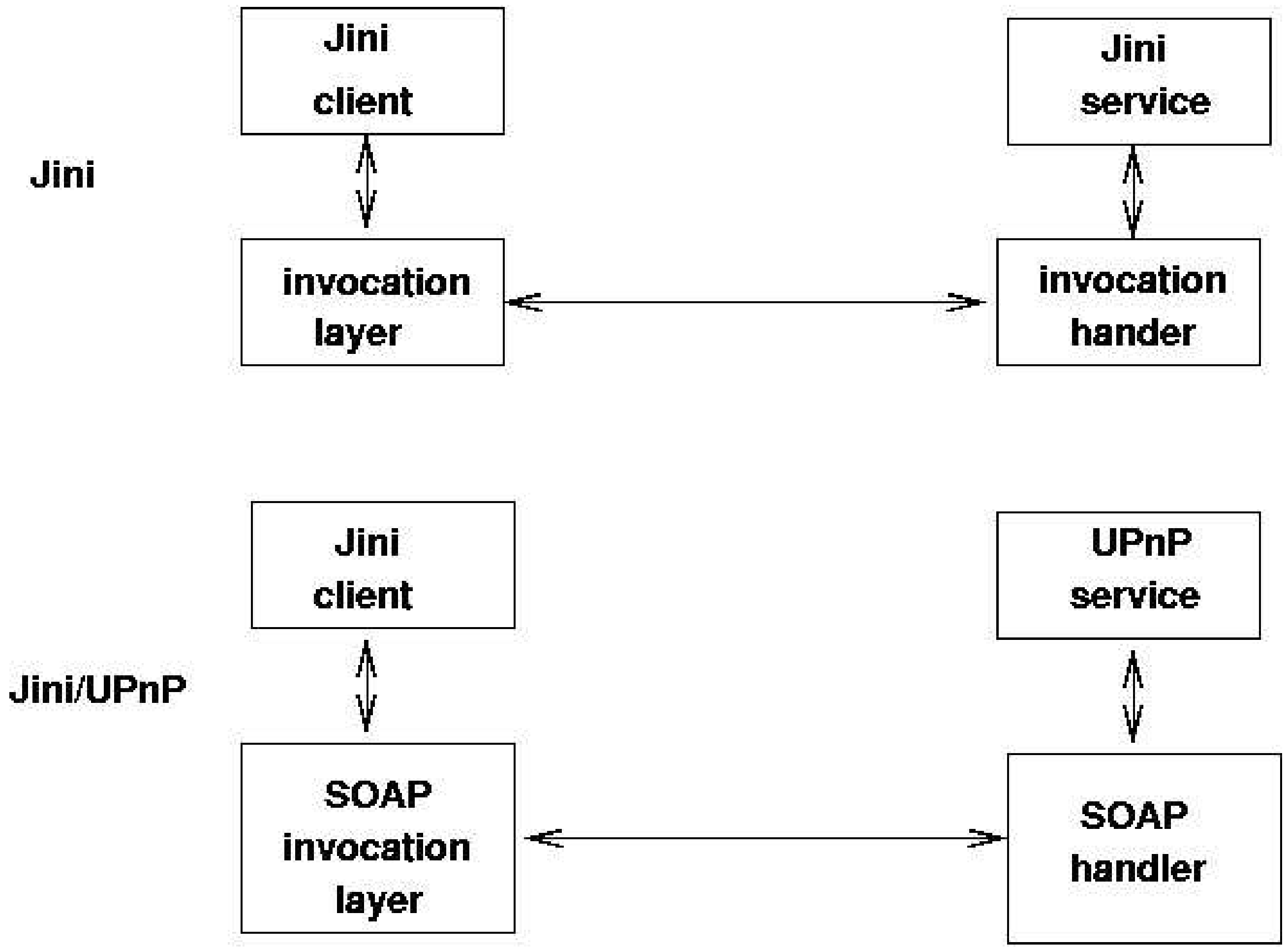
Piggybacking service definitions off UPnP

- The UPnP consortium has defined a number of standard home services
- Services are defined in an XML file
- This can be parsed to define a Jini interface to talk to the service
- This is similar to mapping CORBA IDL to Java and mapping Web Service WSDL descriptions to Java
- It is simple for UPnP since the data types are primitive, and you only need to look at `in` or `out` parameters
- Define "holder" classes for `out` parameters such as `IntHolder`

Making UPnP services available as Jini services

- A Java-based UPnP service can advertise itself as a Jini service too!
- By combining a Jini LUS into a UPnP service it can advertise using both protocols
- The Jini dynamic proxy can use a SOAP invocation handler to talk directly from a Jini client to a SOAP service
- Alternatively, a specialised LUS can listen for UPnP announcements and deliver them as Jini services

SOAP Invocation Handler



What’s in it for Jini?

- It brings a lower-capability middleware into the Jini world
- It allows Jini to piggyback on the UPnP consortium efforts
- Additionally, clients can still talk to *non*-UPnP services - for example a *Jini* client can talk to a *Jini* hardware clock using information from a *Jini* software calendar!

Jini without Java

- An LUS must deliver a `MarshalledObject` to a request
- The *receiver* must understand what to do with a Java object
- The *producer* doesn't: it could just read this object out of a file and squirt it to the receiver
- Even a very stupid service could send a marshalled LUS containing a service proxy to an enquirer
- A smart preparer could write a marshalled object into a file for a dumb service
- Jini *on* a Nutshell :-)

Current status

- Prototype system running
- Generation of Jini interfaces from UPnP XML description done
- Examples written and working
- Code cleanups and completion underway
- Makes use of CyberGarage UPnP for Java - remove this dependency?

Jan Newmarch (http://jan.netcomp.monash.edu.au)

jan.newmarch@infotech.monash.edu.au