Managing Dynamic Distributed Jini Systems

Michael Fahrmair, Chris Salzmann, Maurice Schoenmakers

Institute for Software & Systems Engineering
School of Informatics
Munich University of Technology - Germany

http://www4.in.tum.de/~[fahrmair|salzmann|schoenma]

[fahrmair | salzmann | schoenma]@in.tum.de

Introduction

Dynamic Distributed Systems
Are distributed systems that are able to change their structure (set of components & wiring) autonomously during runtime.

Examples for DDS are: Salutation, UPnP and Jini

Problem:
If the wiring of the components is done autonomously errors and internal processes are much harder to trace.

Goal:
Design a tool that graphically describes the internal mechanisms in a DDS on the fly.
Outline

I Platform: Jini

II Tool Carpat
   • Principle: Reflective Meta Level
   • The meta model
   • Carpat Beans
   • GUI and features

III Future Work & Outlook

Jini in a Nutshell

• developed by Sun Microsystems
• based on Java and partly on RMI
• proposes interfaces to program dynamic distributed systems
• idea: dynamic pool of cooperating services

Services
   – are described by attributes and interfaces
   – are accessible with a mobile service proxy
   – Join: services announce their presence at discovered lookup services

Lookup Services
   – are catalogs of available services
   – contain for each service descriptions and service proxies

Clients
   – Lookup: search services in discovered lookup services with templates
   – retrieve service proxies to use a services
   – Services can also be clients
**Principle: A Reflective Meta Level**

**Observation:** Runtime structure ➔ architectural structure  
**Management:** Runtime structure ◀ architectural structure

Architectural structure: 
**Components and Connectors**

Invisible structure at runtime: 
**Jini / Salutation Services**

---

**Carp@**

A tool to create an architectural overview to observe and manage Jini services and clients

**Observation**
- Clients & Services, Locations
- Channels, Messages exchanged between components
- Provided and required interfaces

**Administration and management**
- change service attributes, check memory resources, start & stop components
- configuration of channels and locations

Carp@ is itself a set of Jini services and clients
The Meta Model

- Abstraction of service components
- Independent of middleware
- Content created by reflection

```
Component 0..* 1 Location

Service 0..*

Out Port

<table>
<thead>
<tr>
<th>Source</th>
<th>0..* Channel</th>
<th>0..*</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In Port

Interface In Port

Event In Port

Service Out Port

Event Out Port

Look up Service

```

Carp@ Beans

- Not all information was available by standard interfaces
- Selected Solution:
  - in each client or service a single Carp@ - Bean is introduced
- Carp@ - Bean is a special Jini Service
  - analyzes the service with standard reflection as far as possible
  - provides additional information (for example the location)
  - is notified by client or service about changes
  - is found by report service with normal Jini techniques
  - is requested for meta-information by the report service
  - propagates changes to the report service as events
**Carp@ Beans**

<table>
<thead>
<tr>
<th>Location</th>
<th>Java</th>
<th>Jini</th>
<th>Carp@</th>
</tr>
</thead>
<tbody>
<tr>
<td>Components</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Interfaces</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>- Provided Ports</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>- Required Ports</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Events</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>- Provided Ports</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>- Required Ports</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Channels</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Message tracing</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Memory Usage</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
</tbody>
</table>

- Introduced API calls
- Also clients became Jini services
- By normal reflection, like Jini does
- Tracing received references

**Carp@: GUI**

- Navigation Tree with different lists
- Structure View • Clients & Services • Locations
- Message Tracing
- Attribute Editor
- Start & Stop services
Status Quo & Future Work

Development of DDS needs new description techniques, tools and methodologies.

Improvement of Carp@:

• Create additional views on the meta-model (Message Sequence Charts, Deployment Diagrams,...)

• "On the fly" byte code instrumentation: Insert Carp@-Beans in predefined service and clients automatically or assisted (using JOIE bytecode modifier).

• Map model to Salutation and UPnP

Informations & Download: http://www4.in.tum.de/~carpat