A framework for managing dynamic service-oriented component architectures

Walter Rudametkin\(^1,2\), Lionel Touseau\(^1\), Didier Donsez\(^1\), Francois Exertier\(^2\)

1 LIG - ADELE
Grenoble University
{firstname}.{name}@imag.fr

2 BULL SAS
Echirolles, France
{firstname}.{name}@bull.net
Building complex software systems

- Large systems
  - Millions of lines of code (e.g., Eclipse ~33 mloc)

- Entangled dependencies
  - Hundreds of different modules that must co-exist

- Run-time adaptation
  - We want the software to change/update @ runtime
  - Requires managing the architecture
We propose a framework that eases the development of architecture managers

- Provides basic services
- Provides abstractions of the architecture
- Helps make pertinent decisions on changes
- Calculates the cost of reconfigurations
- Can be used to create specialized managers
  - E.g., Minimize footprint, adapt to new requirements, high availability, user context, healing...
How things work
(in our world)
Service Oriented Computing

- Service Registry
  - Lookup
  - Publish
  - Bind

- Service Client
- Service Provider
Dynamic Service Oriented Computing

Service Registry

Service Client
Look up
Notify
Bind
Service Provider
Publish
Un-publish
Service-Oriented Components

Required Service

POJO
(Business code)

Component Membrane

Provided Service
Service-Oriented Components

Client → Bind → Provider

Lookup → Service Registry → Notify

Publish
Dependencies: modules and components

Required Services

Component

Provided Services

Component

Component
Dependencies: modules and components

Module

Component

Component

Required Services

Provided Services

Required implementation code

Provided implementation code
Dependencies: modules and components

Required Services

Component

Component

Module

Provided Services

Required Resources

Required implementation code

Provided Resources

Provided implementation code

Provided implementation code
Service-Oriented Component abstraction levels

<table>
<thead>
<tr>
<th>Run time view</th>
<th>Service-Oriented Component Model</th>
<th>Object Oriented Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Service Component instances</td>
<td>Object instances</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Design time view</th>
<th>Service Component Types</th>
<th>Class definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deployment view</th>
<th>Deployment unit repository</th>
<th>Execution framework</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODULE 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODULE 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODULE 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODULE 4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What do we do with all of this?
Our approach

- **model@run.time**
  - Architecture model based on dependencies (Graph)

- **Management framework**
  - Exports the application model
  - Calculates the cost of a reconfiguration
    - Based on dependency information
  - Proposes management services
    - Repository access, Remote services, Resource management, Application monitoring, ...
Why analyze dependencies?

- Primary constraint for reconfigurations
  - Required for installing, instantiating, executing software
- Affect component lifecycle
- Complicate uninstallation
  - E.g.: We want to reduce footprint and not break the application
- Missing dependencies can break the application
  - Halt components, cause state-loss, unavailability, ...
- For Centralized Applications
  - (i.e., single memory space)
Impact of a dynamic reconfiguration

- We use the dependency graph to calculate impact
  - Modules stopped (state-loss)
  - Components stopped (possible state-loss)
  - Modules installed and/or restarted
  - Components installed and/or restarted
  - Bindings and re-bindings

- Rollback and recovery not considered
Example: Domino effect

- All components running
Example: Domino effect

- One component stops
Example: Domino effect

- All components are affected and stopped
Example: Domino effect

- The component becomes available again
Example: Domino effect

- All components run again
Our prototype
Framework overview

Architecture Manager Framework

Problem Specific Component

Generic Architecture Services

- Resource Manager
- Repository Manager
- Resolver
- Distant Service Manager
- RunTime Manager
Framework overview: big picture
Implementation details

- Based on the OSGi Service Platform
- Makes extensive use of other projects
  - Apache Felix
  - Apache iPOJO
  - OW2 Chameleon - ROSE
  - OW2 JonAS
  - Eclipse P2
  - SIGAR (SpringSource)
  - ...
Final remarks
Conclusions

- Framework for handling and understanding dynamism in service oriented component platforms
  - Run time impact of dynamic reconfigurations

- We provide the basic mechanisms for manipulating an application's architecture.

- More “intelligent” features can be implemented on top

- The project will be open-sourced on the OW2 JOnAS project in the (near) future.
Define reactive properties (application reflexes)
- Because some actions are not controlled by the application or the manager (e.g., devices, remote services)

Use a Reference or Abstract architecture
- To enforce and/or validate the architecture
- To provide autonomic architecture adaptation and evolution (e.g., based on QoS)
Questions?
### OSGi Dependency classification (related to impact)

<table>
<thead>
<tr>
<th>Static</th>
<th>Package, Stale references, Dynamic-import, Fragments, Extension points, Handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic</td>
<td>Services, Publish-subscribe pattern, Whiteboard pattern</td>
</tr>
<tr>
<td>Either</td>
<td>Resources, Extender pattern, Configuration</td>
</tr>
</tbody>
</table>
Dependencies: design-pattern

Publish Subscribe pattern

Publisher  Event Admin  Subscriber

Common datatype
Dependencies: design-pattern

Whiteboard pattern

Shell Service

Command 1

Command 2

Command 3

Inverted dependencies
Dependencies: design-pattern

Extender pattern

Component Container

Component 1
- Meta-data

Component 2
- Meta-data
Dependencies: modules and components
Conséquences (4)

- Arrêts en cascade dans le cas d’une composition
  - Effet domino