Development and Execution of Adaptive Component-based Applications

Dipl.-Inf. Andreas Rasche
Operating Systems & Middleware Group
Prof. Dr. Andreas Polze
Hasso-Plattner-Institute
University of Potsdam
Outline

- Adaptive applications using alternative application configurations
- Dynamic reconfiguration in component-platforms (Java/.NET)
  - Reaching a reconfigurable state
  - Dynamic update and state transfer
- AOP tools for generating (re-)configuration specific logic
- Case study: adaptive control applications in a remote lab
- Conclusions
Adaptive Component-based Applications

- Varying resources and context parameters demand adaptation
- Requirement: keep application properties (app.-level QoS) in user-desired range
- Components are units of deployment that can be composed by a third party
- Same interfaces can be implemented by multiple components having different properties
- Different combinations of components (configuration) can fulfill functional requirements of an application
- Applications can be composed for different usage situations
- Solution: Selection and activation of appropriate configuration for given environmental properties allow for adaptation
- Challenge: Integrate dynamic reconfiguration in component platforms
Capsules – Components at Runtime

- A capsule logically groups a set of objects
- Each object has a type
- Each type is defined in an component
- Each component has a version

- root objects
- capsule objects
- primitive types (string, int, byte)
- internal references
- external references

A capsule logically groups a set of objects. Each object has a type, and each type is defined in a component. Each component has a version.
The Adapt.Net Configuration Infrastructure

Configuration Manager

IConfigure

Monitoring

Parameter

Adaptation Engine

XML-Adaptation Profile

XML-Configuration Description
Adaptation through dynamic reconfiguration

- Application configuration:
  - Set of parameterized capsules
  - Set of connectors among capsules
  - Mapping to computers in a distributed system

- Dynamic reconfiguration includes:
  - Addition/removal of capsules
  - Changing capsule parameter
  - Migration(new location)/ updating (new logic) capsules
  - Changing connections between capsules

- Reconfiguration actions must remain consistency
  - No method execution during updates
  - No execution at capsules with unconnected sink capsules
Reaching a reconfigurable state

- A capsule is reconfigurable if there is **no on-going method execution of capsules’ objects on any threads’ stack**!

- A reconfigurable state can be reached by:
  - Blocking new method calls from threads and other capsules
  - Waiting for all ongoing method calls to complete

- Acyclic graphs: connections can be blocked orderly

- Cyclic graphs: single threads must be blocked

- Reader-Writer-Locks for synchronization
  - Read-Lock is acquired for each normal method call
  - Write-Lock is acquired by the update logic
  - Usage of recursive locks for recursive calls
Reconfiguration of Distributed Applications

- RW-Locks in .NET- and Java-platform do not work distributed

- **Problem**: When blocking a thread it must not have on-going method calls on involved capsules

- **Solution**: logical thread-IDs and counters
  - Counter per capsule with on-going methods per thread
  - Update counter when entering/leaving a capsule via a root-object
  - During blocking: threads with no on-going method on involved capsules (counter in all capsule is zero) can be blocked
Reconfiguration of Distributed Applications

- RW-Locks in .NET- and Java-platform do not work distributed

**Problem:** When blocking a thread it must not have on-going method calls on involved capsules

**Solution:** logical thread-IDs and counters
- Counter per capsule with on-going methods per thread
- Update counter when entering/leaving a capsule via a root-object
- During blocking: threads with no on-going method on involved capsules (counter in all capsule is zero) can be blocked
Application-specific synchronization

- In case of synchronization among application threads the algorithm must be extended.
- All capsules on a path between involved capsules (the block-set) are added to the block-set.

1. ID 1
2. lock(R)
3. continue with R locked
4. context switch
5. ID 2
6. try to lock R
7. continue with R locked
8. ID 2
AOP tools and (re-)configuration specific logic

- Synchronization logic for dynamic reconfiguration
  - Management of capsules’ counters
  - Blocking of threads

- Implementation of component’s configuration interface
  - Set-up of communication connections
  - Parameterization
  - Initiation of blocking for dynamic reconfiguration
  - State transfer for migration and dynamic updates

- New programming model for marking connection endpoints and parameters

```java
public class Filter{
    [Parameter]
    int compression;
    [Connection]
    IStream sink;
}```
Dynamic Updates

- Complex reconfiguration operation
  - activation of new code (and data layout)
- Capsules have to be updated dynamically to:
  - Activate more appropriate algorithms at runtime
  - Integrate bug-fixed versions (remove security vulnerabilities)
  - Change graphical representation of adapted architecture
- Classes cannot be exchanged directly (in Java/.NET)
  - New versions of objects must be created
  - State must be transferred from old to new version
- Algorithm for reaching reconfigurable state used to apply update atomically
Traversing the Object Graph

- Start from all root objects
- For each field of all objects traverse all references
- In case of an update:
  - Create an instance of the new version
  - Copy the state by transferring all fields from the old to the new instance
  - For reference fields: traverse target first an install potential new version afterwards
- Usage of Reflection (GetFields, Set-/GetValue)

<table>
<thead>
<tr>
<th>MyObject V1.0</th>
<th>MyObject V2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nr: 42</td>
<td>Nr: 42</td>
</tr>
<tr>
<td>Weight: 65</td>
<td>Weight: 0</td>
</tr>
</tbody>
</table>

Object temp = oldObj.GetValue(„Weight“);
newObj.SetValue(„Weight“, temp);
Traversing the Object Graph II

- Cycle recognition (visited nodes)
- Creation of new types (no constructor execution)
- Dynamic assembly loading (shadow copies)
- Arrays (update type and content)
- Delegates (update target and method)
- Generics (update bound types)
- Type and assembly objects
- Activation/deactivation/update of aspects
- State transformation for changed data layout
Case Study: Adaptive Control Applications in the Distributed Control Lab

.web-browser

Mobile Access

Visual Studio Integration

Foucault’s Pendulum

Industrial Control

“Higher Striker” real-time control

experiment manager

result management

job/experiment management
Fault Tolerance and Security with dynamic reconfiguration

- **Problem:** malicious code submitted via the Internet
- **Solution:** execute an adaptive control application
  - Verified safety controller
- **Observed parameters**
  - Pendulums amplitude
  - Duration of job execution
  - State of user capsule (abnormal termination)
Conclusions

- Configurations can be composed/developed independently
  - Non-functional app.-properties can be tested for aimed situation
  - New configurations can be added (by a separate planner/...)

- Algorithm for dynamic reconfiguration of distributed multithreaded applications with cyclic dependencies
  - Low overhead for normal method execution

- Dynamic updates for activating alternative algorithms/ hot-fixes
  - Without manipulation of the virtual machine

- AOP capable of generating (re-)configuration specific logic

- Adaptive applications can be used for protecting experiment hardware in a remote laboratory environment
http://www.dcl.hpi.uni-potsdam.de
Further Reading

- **ReDAC - Dynamic Reconfiguration of distributed component-based applications with cyclic dependencies** Rasche, Andreas; Polze, Andreas: Submitted to 11th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 5-7 Mai 2008, Orlando, Florida


- **Heterogeneous Adaptive Component-Based Applications with Adaptive.Net** Andreas Rasche, Marco Puhlmann and Andreas Polze in Proceedings of International Symposium on Object-oriented Real-time distributed Computing (ISORC), Seattle, Washington, USA, May 2005