Dynamic Adaptation

CBSS Course Presentation
Instructor: Dr. Ladan Tahvildari
Provided by: Mazeiar Salehie
PhD Student
E&CE Department
University of Waterloo
Spring 2005

References

• Main references
  – M. Sadjadi, P. McKinley and B. Cheng, **Transparent Shaping of Existing Software to Support Pervasive and Autonomic Computing**, In proc. of DEAS workshop, USA, 2005

• Further reading
Outline

- Introduction
  - Definition
  - General Approaches
- Enabling Technologies
- Compositional Adaptation
  - Where
  - How
  - When
- Sample Tools
  - TRAP
  - ACT
- Research Challenges
- Conclusion

Introduction

- **Definition**: “A software application is adaptable if it can change its behavior dynamically (at run time) in response to transient changes in its execution environment or to permanent changes in its requirements.”

- Two major drivers:
  - Pervasive/Ubiquitous Computing: changing environments
  - Autonomic Computing: self-managing systems
**General Approaches**

- Two general approaches have been used to realize dynamic adaptation:
  - Parameter Adaptation
  - Compositional Adaptation

**Parameter Adaptation**

- Parameter adaptation involves the modification of variables that determine program behavior.

- A weakness of parameter adaptation is that it cannot adopt algorithms or components left unimplemented during the original design and construction of an application.
Compositional Adaptation

- Compositional adaptation results in the exchange of algorithmic or structural parts of the system with ones that improve a program’s fit to its current environment.

- In comparison to parameter adaptation, compositional adaptation enables an application to adopt new algorithms/components for addressing concerns unforeseen during original design and construction.

Outline

- Introduction
  - Definition
  - General Approaches
- Enabling Technologies
- Compositional Adaptation
  - Where
  - How
  - When
- Sample Tools
  - TRAP
  - ACT
- Research Challenges
- Conclusion
Enabling Technologies

Component-Based Design

Compositional Adaptation

Component-Based Design

(a) Static recomposition.
(b) Dynamic recomposition.
Enabling Technologies

- Separation of Concerns
- Component-based Design

Compositional Adaptation

Separation of Concerns

- Business Logic
- Aspects
- Development Time
- Aspect Weaver
- Compile Time
- Woven Code
- Run Time
Enabling Technologies

- Separation of Concerns
- Computational Reflection
- Component-based Design
- Compositional Adaptation

Behavioral Reflection

Enabling Technologies

- Separation of Concerns
- Computational Reflection
- Component-based Design

Middleware


<table>
<thead>
<tr>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain-Specific Middleware Services</td>
</tr>
<tr>
<td>Common Middleware Services</td>
</tr>
<tr>
<td>Distribution Middleware</td>
</tr>
<tr>
<td>Host-Infrastructure Middleware</td>
</tr>
<tr>
<td>Operating Systems and Protocols</td>
</tr>
<tr>
<td>Hardware Devices</td>
</tr>
</tbody>
</table>
Outline

• Introduction
  – Definition
  – General Approaches
• Enabling Technologies
• Compositional Adaptation
  – Where
  – How
  – When
• Sample Tools
  – TRAP
  – ACT
• Research Challenges
• Conclusion

Where to Compose

A typical client/server application.
Where to Compose (2)

<table>
<thead>
<tr>
<th>Project</th>
<th>Institution/Organization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language-based projects</td>
<td></td>
</tr>
<tr>
<td>AspectJ</td>
<td>Xerox Palo Alto Research Center</td>
</tr>
<tr>
<td>Composition Filters</td>
<td>University of Twente, The Netherlands</td>
</tr>
<tr>
<td>Program Control Language (PCL)</td>
<td>University of Illinois</td>
</tr>
<tr>
<td>Open Java</td>
<td>IBM Research</td>
</tr>
<tr>
<td>R-Java</td>
<td>University Federal de Sao Carlos, Brazil</td>
</tr>
<tr>
<td>Kava</td>
<td>University of Newcastle, UK</td>
</tr>
<tr>
<td>Adaptive Java</td>
<td>Michigan State University</td>
</tr>
<tr>
<td>Transparent Reflective Aspect Programming in Java (TRAP/J)</td>
<td>Michigan State University</td>
</tr>
<tr>
<td>Middleware-based projects</td>
<td></td>
</tr>
<tr>
<td>Domain-specific services layer:</td>
<td>Boeing</td>
</tr>
<tr>
<td>Boeing Bold Stroke (BBB)</td>
<td></td>
</tr>
<tr>
<td>Generic services layer:</td>
<td></td>
</tr>
<tr>
<td>CorbaServices</td>
<td>Object Management Group</td>
</tr>
<tr>
<td>BBN Technologies</td>
<td></td>
</tr>
<tr>
<td>Adaptive Core Template (ACT)</td>
<td>Michigan State University</td>
</tr>
<tr>
<td>Interoperable Replication Logic (IRL)</td>
<td>University of Roma, Italy</td>
</tr>
<tr>
<td>Distribution layer:</td>
<td></td>
</tr>
<tr>
<td>.NET remoting</td>
<td>Microsoft</td>
</tr>
<tr>
<td>Open ORB and Open COM</td>
<td>Lancaster University, UK</td>
</tr>
<tr>
<td>The ACE ORB (TAO) and Component Integrated ACE ORB (CIAO)</td>
<td>Distributed Object Computing Group</td>
</tr>
<tr>
<td>Dynamic TAO and University Interop Core (UIC)</td>
<td>University of Illinois</td>
</tr>
<tr>
<td>Orbix, OrbixE, and ORBacus</td>
<td>Iona Technologies</td>
</tr>
<tr>
<td>Squirrel</td>
<td>University of Kaiserslautern, Germany</td>
</tr>
<tr>
<td>AspectJ</td>
<td>Friedrich-Alexander University, Germany</td>
</tr>
<tr>
<td>Most infrastructure layer:</td>
<td></td>
</tr>
<tr>
<td>Java virtual machines (JVM)</td>
<td>Sun Microsystems</td>
</tr>
<tr>
<td>Common Language Runtime (CLR)</td>
<td>Microsoft</td>
</tr>
<tr>
<td>SpecVJ</td>
<td>Trinity College, Dublin</td>
</tr>
<tr>
<td>Pravec</td>
<td>Boston Federal Institute of Technology</td>
</tr>
<tr>
<td>Adaptive Communication Environment (ACE)</td>
<td>Distributed Object Computing Group</td>
</tr>
<tr>
<td>Ensemble</td>
<td>Cornell University</td>
</tr>
<tr>
<td>Cross-layer projects</td>
<td></td>
</tr>
<tr>
<td>Distributed Extensible Open Systems (DEOS)</td>
<td>Georgia Institute of Technology</td>
</tr>
<tr>
<td>Grace</td>
<td>University of Illinois</td>
</tr>
</tbody>
</table>

How to Compose

<table>
<thead>
<tr>
<th>Technique</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function pointers</td>
<td>Application execution path is dynamically redirected through modification of function pointers.</td>
<td>VTables in COM, delegates and events in .NET, callback functions in Corba</td>
</tr>
<tr>
<td>Wrappers</td>
<td>Objects are subclassed or encapsulated by other objects (wrappers), enabling the wrapper to control method execution.</td>
<td>ACE, R-Java, PCL, QaO, TRAP/J</td>
</tr>
<tr>
<td>Proxies</td>
<td>Surrogates (proxies) are used in place of objects, enabling the surrogate to redirect method calls to different object implementations.</td>
<td>ACT, AspectJ</td>
</tr>
<tr>
<td>Strategy pattern</td>
<td>Each algorithm implementation is encapsulated, enabling transparent replacement of one implementation with another.</td>
<td>Dynamic TAO and UIC</td>
</tr>
<tr>
<td>Virtual component pattern</td>
<td>Component placeholders (virtual components) are inserted into the object graph and replaced as needed during program execution.</td>
<td>ACE and TAO</td>
</tr>
<tr>
<td>Metaobject protocol</td>
<td>Mechanisms supporting intercession and introspection enable modification of program behavior.</td>
<td>Open Java, Kava, TRAP/J, Open ORB, Open COM, Iguana/J</td>
</tr>
<tr>
<td>Aspect weaving</td>
<td>Code fragments (aspects) that implement a crosscutting concern are woven into an application dynamically.</td>
<td>AspectJ, Composition Filters, TRAP/J, AspectJ, Iguana/J, Prose</td>
</tr>
<tr>
<td>Middleware Interception</td>
<td>Method calls and responses passing through a middleware layer are intercepted and redirected.</td>
<td>ACT, IRL, Prose</td>
</tr>
<tr>
<td>Integrated middleware</td>
<td>An application makes explicit calls to adaptive services provided by a middleware layer.</td>
<td>Adaptive Java, Orbix, OrbixE, ORBacus, BBS, CIAO, Iguana/J, Ensemble</td>
</tr>
</tbody>
</table>
### How to Compose (2)

- The role of composer
- Transparency: “Is an application aware of the infrastructure needed for re-composition?”
  - Transparent shaping: “supports the design and development of adaptable programs from existing programs without the need to modify the existing programs’ source code.”

### When to Compose

- **Tunable** software: prohibits modification of code for the business logic, only supports fine-tuning of cross-cutting concerns.
Dynamic Adaptation Phases

– Producing an adapt-ready program – a program whose behavior can be adapted at run-time by insertion or removal of adaptive code (interceptor or hook) at sensitive JoinPoints

– Run-time adaptation in which the composer switches the hook at sensitive joinPoints from original to alternative hooks.

Outline

• Introduction
  – Definition
  – General Approaches

• Enabling Technologies

• Compositional Adaptation
  – Where
  – How
  – When

• Sample Tools
  – TRAP
  – ACT

• Research Challenges

• Conclusion
Sample Tools (1): Transparent Reflective Aspect Programming (TRAP)

- Existing program
- Selecting classes
- Generating hooks
- Weaving hooks

![Diagram of the TRAP/J process]

Data Flow: A File A Process TRAP/J Boundary

ORIGINAL APPLICATION (.class files)

METALEVEL

CLASS

BASELEVEL

CLASS

ASPECT

WRAPLEVEL

CLASS

INTERCEPTING ASPECTS

METALEVEL

CLASS

INTERCEPTING ASPECT GENERATOR

REFLECTIVE CLASS GENERATOR

GENERATED ADAPT-READY APPLICATION (.class files)

Case study: Wireless Audio Streaming

FEC stands for "Forward Error Correction"
Sample Tools (2): Adaptive CORBA Template (ACT)

- ACT uses CORBA portable interceptor, which can be incorporated into a CORBA program at startup time using a command-line parameter. Later at run-time, these hooks can be used to insert adaptive code into the adapt-ready program.

---

Transparent Shaping Contributions & Artifacts

---
### Relationships with other Contributions

<table>
<thead>
<tr>
<th>Adaptable Applications, Existing/Non-Adaptable Applications</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRAPU, Composition Filters, RMI, ASCAD, ...</td>
<td>Middleware</td>
</tr>
<tr>
<td>BBS, ...</td>
<td>Domain-Specific Services</td>
</tr>
<tr>
<td>QuO, OGS, ...</td>
<td>Common Services</td>
</tr>
<tr>
<td>ACTI, RIL, TAO-LB, ...</td>
<td>Distribution Services</td>
</tr>
<tr>
<td>Java RMI, TAO-Dynamic, TAO-CR, JacORB, Squirrel, OpenCorba, OpenORB, Electra, ...</td>
<td>Host-Infrastructure Services</td>
</tr>
<tr>
<td>IguanaJ, PROSE, Guaraná, ...</td>
<td>System Platform</td>
</tr>
<tr>
<td>Ólafos, OGS, ...</td>
<td>Windows OS, Linux OS, Sun Solaris OS, Mac OS</td>
</tr>
</tbody>
</table>

Transparent shaping boundary

Hooks to incorporate adaptive code dynamically

---

### Research Challenges

- **Assurance**
  - Techniques are needed to ensure that the system still executes in an acceptable or safe manner during/after the adaptation process
  - Certify all components for correctness with respect to their specification

- **Security**
  - Hiding adaptation management from would-be intruders and prevent them from impeding or corrupting the adaptation process.

- **Interoperability**
  - Tools and methods to integrate the operation of adaptive components across the layers of a single system, among multiple systems, and between different adaptive frameworks.

- **Decision making**
  - Deciding how, when and where to adapt the system (by composer)
Conclusion

• Dynamic Adaptation (according to Aksit et al.) is a special case of self-configuring
  – The focus is on realizing the idea of system families and variations at run-time

• Compositional Adaptation (Transparent Shaping)
  – Separation of Concerns
  – Component-Based Design
  – Computational Reflection
  – middleware

• There is no reference to dynamic architecture and ADLs

• There is no details on “how” and by “which” mechanism a “composer” can decide about the dynamic composition
  – Using rule-based, policy-based, machine learning approaches or control theory similar to autonomic computing
Support Slides

TRAP Layered Run-time Model
**TRAP/J**

Trap/J run-time support  
A simplified run-time class graph.

**TRAP & ACT**

Rule-Based Interceptor  
Dynamic interceptor  
Generic interceptor  
Stub

Generic Interceptor

Image Client  
Image Server

**ASA Sender**  
**ASA Receiver**

Java Socket  
Class

MetaSocket  
close
InvokeMethod

Pipeline

Filter
Delegate  
Meta Object  
Wrapper Object

Program object  
Flow of service request  
Hook

Thursday, July 21, 2005  
Mazeiar Salehie, E&CE Dept., UWaterloo