Grid Development Tools for Eclipse

Thomas Friese  <th.friese@siemens.com>
Corporate Technology, Information and Communications

Matthew Smith  <matthew@informatik.uni-marburg.de>
Bernd Freisleben  <freisleb@informatik.uni-marburg.de>

Distributed Systems Group
Dept. of Mathematics and Computer Science
University of Marburg, Germany
Outline

- Motivation
- Goals
- Service Generator
  - Structure / Rough Idea
  - Developer Perspective
  - Demo
  - Implementation
- Process Editor
- Grid Management Functionality
- Remote Debugging
- Conclusions
Motivation

IDEs widely adopted by programmers…

…Developing Globus Services still means (the coding approach):
- Define the Interface in WSDL
- Define classes, WSDD, JNDI, namespace2package.mappings
- Run python script (that calls ant) to build the service
- Deploy service

Coding is insufficient for large and complex projects:
- Debugging is a pain, artifacts are related yet not synchronized
- Software Architecture/Modeling, Model Driven Development increasingly applied to large applications

Distributed application development needs more:
- management capability, distributed debugging
Using GT4, the developer has to create all this code manually for a service that simply accumulates a value.

This does not contain a client, registration or discovery for the service, deployment descriptors, WSDL files...
GDT Goals

**Design time**
- Service generation
  - Roundtrip engineering based on core model representation
  - Separation of concerns for application/middleware experts
- Collaborative workflow editing
  - Multiple experts multiple sites
  - Application composition from basic services

**Run time**
- Grid management
  - Deployment & undeployment of Grid services
  - Monitoring & visualization of Grid environment
- Grid-wide debugging
  - Component oriented debugging (single container)
  - Environment oriented debugging (distributed traces)
GDT Service Generator

PIM

UML2 Model

Annotated Application Logic

Upper Layer

“Grid” Upper Layer Platform Specific Model

PSM

Target Platform Specific Model (e.g. GT4)

Lower Layer

Code

Platform Binding Code

Application Logic Carrying Code
public class SwABenchmark
{
    @GridAttribute private long rt = 0;
    @GridMethod public void testA(Ref a)
    {
        ...
        rt = System.nanoTime()-st;
    }
}
Globus Implementation

- Annotated class (AC) holds application logic
- Code handled by GDT
  - Synchronized to internal model → change in any element of the implementation (annotated class, WSDL...) reflected throughout entire implementation
- Input for Globus tools
- Stubs created by Globus tools

- Annotated class is part of service
- Multiple services per project supported
- GAR packaging just one click
Service Implementation

- **Service Styles**
  - Factory
    (Instances Created by Factory Service)
  - Simple
    (Instances Created by Service)
- **Resource Style**
  - MAGE
    (Property Access through Reflection to AC)
  - Globus Style
    (User Managed Access to Property in Resource)
- **Access to attributes using standard WSRF interface**

GDT Managed Code

Annotated Class (Business Logic)

Globus Code
### Annotations

<table>
<thead>
<tr>
<th>Annotation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@GridService</td>
<td>Tag class as service main class</td>
</tr>
<tr>
<td>name</td>
<td>Service Name (e.g. “Math”)</td>
</tr>
<tr>
<td>namespace</td>
<td>Target Namespace (e.g. <a href="http://fb12.de/ns/m">http://fb12.de/ns/m</a>)</td>
</tr>
<tr>
<td>targetPackage</td>
<td>Target Package (de.fb12.grid.math)</td>
</tr>
<tr>
<td>serviceStyle</td>
<td>SSTYLE_{FACTORY, SIMPLE}</td>
</tr>
<tr>
<td>resourceStyle</td>
<td>RSTYLE_{MAGE, GLOBUS}</td>
</tr>
<tr>
<td>@GridMethod</td>
<td>Tag method as GS-method</td>
</tr>
<tr>
<td>@GridAttribute</td>
<td>Tag attribute as resource property</td>
</tr>
<tr>
<td>@GridOperationProvider</td>
<td>Specify operation providers</td>
</tr>
</tbody>
</table>
Headless builds

- GDT is usable in non-GUI modes
  - Headless, command line operations
  - Custom ant target to include in application build scripts

```xml
<target name="import.math">
  <gdt.generator project="TestService"
    globus="$\{env.GLOBUS_LOCATION\}"
    annotated="path/to/Math.java" service="Math">
    <preimport dir="Math_extra">
      <include name="src/**/*\.*.java"/>
    </preimport>
    <preimport dir="Client" destdir="src"
      intoservice="false">
      <include name="**/\.*.java"/>
    </preimport>
  </gdt.generator>
</target>
```
Demo

Demonstrates the service generator in action.
Implementation

- Eclipse integrated
  - Delta visitors
  - Custom Eclipse builder
  - Wizards
- Extensive use of model driven approach
  - GDT code generated from meta-models using EMF
  - Target system code generated from internal model representation
- Emitters
  - Java Emitter Templates (JET) + Enhanced JMerge
  - Custom emitters (re-using AXIS/Globus tools for WSDL)
- Interpreters alter model based on information from
  - Eclipse JDT Java model representation
  - Annotation Processor (using eclipse JDT APT)
Binding Model

1. Receive Delta Set
2. Changes → Model
3. Run Emitters
4. Merge with Code
Performance

- Interactive Operation
  - Eclipse stays responsive during development
  - Runtime of the integrated builder *
    - 1 Attribute, 1 Method : 800ms
    - 25 Attributes, 25 Methods : 1,3s
  - Runtime of the stubs generation / packaging
    - Mainly dependent on the GSBT derivative Ant scripts >15sec
    - Separate from integrated automatic builds
- Headless / Ant based operation
  - Overhead imposed by Eclipse integration: 5-10sec to load platform
  - Overall cost (per service) +
    - First run: 45-55sec
    - Subsequent runs: 30-40sec

* Pentium M 1,7GHz, 1GB Ram, 60GB 7200rpm HD
+ P4HT 3.0GHz, 2GB Ram, 160GB 7200rpm SATA-HD, ~20 nightly integration builds/tests
Collaborative Grid Process Editor

- Graphical process editor
- Hide Complexity from Application Experts
- Reveal Details to Grid Experts

Collaborative editing (ECF)
- Same-Time
- Different-Place

Extensible Architecture
- Additional constructs
- Additional execution platforms
Integrated Management Functionality

Service / Node Discovery & Visualisation
- Different discovery mechanisms (GT4 MDS, MAGE P2P, Unicore/GS)
- Different node relations (discovery network, actual interaction)

Integrate Management Functionality
- Service (Un-)Deployment
- Security configuration

Debug support
- Remote service/container debugging
- Source attachment version management
- Process/Value traces accross entire Grid
@GridMethod public void startCounter()
{
    Thread t = new Thread(new Runnable()
    {
        int i = 0;
        public void run()
        {
            while (true)
            {
                try
                {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {}
                System.out.println("Counting: "+i);
                i++;
            }
        }
    });
    t.start();
    return 0;
} …
public static void main(String[])
    String instanceURI = "http://127.
    try {
        CDLCounterClient client = new CDLCounterClient(instanceURI, instan
        client.startCounter();
    } catch (Exception e) {
        e.printStackTrace();
    }

1. Run the Client

2. Counter thread started
Service (Remote-)Debugging (2)

1. Set breakpoint

2. Attach to container
Eclipse debugger displays container execution state

Counter thread of sample service stopped at breakpoint
Conclusions

- We have:
  - Service generator (released in GDT, currently: 1.0.9)
  - Round-Trip support: Annotated Class & WSDL
  - Initial remote component debugging support
  - Deployment, management, process creation (MAGE)
- Outlook
  - Better Array & Complex Type support
  - More Target System Models
  - Deployment, management support for GT4
  - Remote debugging: component & environment
- Resources
  - Project Info: http://mage.uni-marburg.de/gdt
  - Mailing-Lists
  - Moved CVS → SVN, anon SVN access will be available soon
  - Issue tracking: http://mage.uni-marburg.de/trac/gdt