Grid programming with components: an advanced component platform for an effective invisible grid

GCM component programming with ADL:
A Methodology using Grid IDE

Artie Basukoski,
J. Thiyagalingam
V. S. Getov

University of Westminster, London, U.K.
V.S.Getov@westminster.ac.uk
Outline

- Background
- Motivation
- GCM Reference Implementation
- Grid IDE – Strategy
- Grid IDE – Different Views
- Using GIDE – Illustrated with an example
- Legacy Code Wrapping
- Further Work and Conclusions
Background: Building Grid Applications

- Proprietary middleware (Globus 1.0, Legion, Unicore, …)
  - Resources exposed through an API
  - Non interoperable!
- Object-based middleware
  - Resources exposed through distributed objects (Java, CORBA, etc.)
  - Some interoperability issues with the communication protocols (CORBA IIOP)
  - Not anymore at the top of the hype!
- Service-based middleware
  - Resources exposed through services
  - Strong support from the Industry
  - At the top of the hype!
  - Need some extensions (stateful Web services)
Motivation

- “Grid Everywhere” and Pervasive Computing strategies demand truly dynamic software infrastructures.
- Developing Grid Applications with GCM-specific programming model requires an intuitive form of assistance.
- Under GCM model, applications are considered as compositions of components.
- An integrated development environment to facilitate development/composition, deployment and monitoring is essential.
- GridCOMP Grid IDE (GIDE) extends the capabilities of Eclipse to support GCM-based development.
One of the Main Research Challenges for Future Grids

To develop the software design and development methodology of a generic component-based Grid platform for both applications and tools/systems to have a single, seamless, “invisible” Grid software services infrastructure.

Possible Solution:

Grid Component Model (GCM)
GCM Reference Implementation

- 1 - Primitive Component Programming
- 2 - Legacy Code Wrapping, Interoperability
- 3 - Composition and Composites, Deployment
- 4 – Autonomic features
- 5 – IDE for GCM (Composition GUI, etc.)
Main issues: composition and dynamic properties – deployment, monitoring and steering
Strategy: Eclipse Framework for GIDE

- Simplify complexity through graphical composition/tools.
- But, allowing ONLY graphical composition can be inflexible and inefficient.
- Support for 3 levels of Development.
  - Graphical Composition.
  - Based on GCM and Proactive.
  - Java.
- Seamless integration with Eclipse.
  - Widely supported. Many existing plugins (IC2D).
- “Let’s not restrict developers.”
Grid IDE Core Block Diagram
Development Environment Design

- Composition Perspective
  - Graphical but also allow code editing.

- Deployment Perspective.
  - Drag/drop to scheduler.
  - Launch/Stop through right click actions.

- Resource Monitoring Perspective.
  - Host View
  - Resource List View.

- Component Monitoring/Steering Perspective.
  - Graphical display of Component status.
  - Relocation via drag/drop
  - IC2D already does graphical monitoring of hosts, JVMs and Active Objects.
GIDE – An Insight into composition

- GIDE builds on GMF for providing graphical front-end
- The IDE includes
  - Built-in ADL parser
  - Verifier
  - Diagram-generator
  - Semantic-Generator
  - ADL-exporter
- ADL files are verified, parsed and then appropriate internal representations of compositions (semantic representation) and diagrams are generated.
- GIDE delegates the user-interactions to these internal representations
Domain model

```xml
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema targetNamespace="http://perun.hscs.wmin.ac.uk/GridCOMP/gidecomposition"
xmlns:gidecomposition="http://perun.hscs.wmin.ac.uk/GridCOMP/gidecomposition"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  <xs:element name="Component" type="gidecomposition:Component" />
  <xs:complexType name="Interface">
    <xs:attribute name="Name" type="xsd:string" use="required" />
    <xs:attribute name="Id" type="xsd:unsignedShort" use="required" />
    <xs:attribute name="Type" type="xsd:string" use="required" />
    <xs:attribute name="Cardinality" type="gidecomposition:CardinalityType" use="required" />
    <xs:attribute name="CardinalityIn" type="xsd:unsignedByte" use="required" />
    <xs:attribute name="CardinalityOut" type="xsd:unsignedByte" use="required" />
  </xs:complexType>
  <xs:complexType name="Component">
    <xs:sequence>
      <xs:element name="Interfaces" type="gidecomposition:Interface"
        maxOccurs="unbounded" />
      <xs:element ref="gidecomposition:Component" maxOccurs="unbounded" />
      <xs:element name="Connections" type="gidecomposition:ConnectionType"
        maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="Name" type="xsd:string" use="required" />
    <xs:attribute name="Id" type="xsd:unsignedShort" use="required" />
    <xs:attribute name="LastModified" type="xsd:string" use="required" />
  </xs:complexType>
</xs:schema>
```
GIDE – An Insight in to composition …
Data Centre Environment

- Separate RCP application.
- Clear and fixed functional views.
  - Deploy
  - Resource Monitor
  - Component Monitor and Steer
- Restrict personalisation.
  - Data Centres have high rates of turnover.
  - High demand means generally low expertise.
- “Lets protect operators from the details.”
Composition – An example

- This example builds on Use-Case 5
- The use-case include three components – two primitives and one composite
- The composition is expressed as an ADL file
- The tutorial will illustrate the general usage and then how we could import compositions from ADL files
- This is based on current version – whose features are evolving
User Interface of the Composition Editor

Grid programming with components: an advanced COMPONENT platform for an effective invisible grid
Import and Editing Views
• Create a new Java Project to start with.
This should bring-in the necessary files to the project
Now create an “Other Project” – GIDE-Composition
Grid programming with components: an advanced platform for an effective invisible grid.
Name the Composition Project
- This should create two files (semantic and diagram)
- Should bring the canvas for composition
- Drag and drop component from toolbox
- Change properties
Create a connection between them by using the link tool
● Zoom in, if necessary
Once created, select and arrange them for better layout
The auto-arrangement is done by GMF-backend
To Import ADL Files, select Import from the file Menu and select the file for import.
- Select the project space to import into
The file will be imported along with the semantic/diagram files
Diagram will be rendered automatically
- The underlying ADL file can be edited
- Composite components can also be imported by selecting the composite ADL file.
Resource Monitoring

- Enables an operator to dynamically view the underlying resources prior to deployment
- Implemented using a highly scalable, high-performance and platform independent library
- Permits remote monitoring
Approaches for componentising legacy code

- The current version of GIDE has no direct support for legacy code-wrapping
- Support for such componentisation and code-wrapping is essential to support legacy applications
  - Current focus is on GENIE – Grid Enabled Integrated Earth System Modelling: componentised and Grid Enabled
  - Alternative approach is to hand-tune the code to comply with the underlying model – JEM3D
- Future version of the GIDE may support GCM-compliant code wrapping
Some Future Goals

- Utilise the Eclipse GMF approach for providing highly interactive development IDE
- Support and different interface types
- Improved support for ADL files
- Implement and Integrate component monitoring
- Optional: model verification, improved context support for managing ADL files
Conclusions

- Implemented an initial version of the prototype of GIDE to support GCM
- Adopted GMF approach and re-engineered the overall design
- Integrated platform supports better resource monitoring
- Further work is needed to align with production usage
Questions