A Component Framework for HPC Applications

John Darlington, Steven Newhouse, Nathalie Furmento, Anthony Mayer, Stephen McGough

Grid Middleware Group
London e-Science Centre
Imperial College, London
icpc-sw@doc.ic.ac.uk
Outline

• Components for Grid Computing
• An Extensible Component Framework
• Separation of Abstraction & Implementation
• Static & Dynamic information
• Example: Linear Solver
• Further developments
Motivation & Background

- Grid developments typically low-level services
- Lack of penetration into user base
- High barrier to utilisation
- Goal: from explicit “hands on” grid use to automatic, transparent use
Requirements of Grid Computing

• Mobile
  – Dynamic, unreliable resource context
    – “Complex Resources”

• High Performance
  – Raison d’être of grid computing
    – “Complex Applications”

• Accessible
  – e-science portals, etc
    – “Transparent Mapping”
Information!

• User:
  – Who, what, when, where

• Resources:
  – Availability, character, policy

• Application:
  – Composition, behaviour, performance
Software Abstraction: Components

- Separation between implementation and abstraction
- Expressability suitable for end-user programming
- Encapsulation of meta-data to enable dynamic optimisation
High Performance Software Components

- Express an application as a composition of components (high-level abstract data types)
- Separation of implementation & abstraction allows very late binding of low-level libraries
- Use component meta-information to inform all stages of application construction and execution
- Minimise execution time while maximising throughput over the available resources
Separating Meta-Information from Implementation

Application Construction

Component Repository

Optimisation, verification etc.

Scheduling

Code

Code

Code
Augmented Component Lifecycle

1. Compose Application
2. Cross Component Optimisation
3. Dynamic Implementation Selection
4. Deployment

Requires Resource Information
Conventional Software Lifecycle

Source → Compile → Executable → Link → Executable

Library → Library → Library

Component Software Lifecycle

Source → Compile

Interface 1 → A

Interface 2 → B

Compile → Compose

Implementations

Composite Application
An Augmented Component Lifecycle

Implementations

Compose

Composite Application Description

Deploy

Run-Time Representation

Dynamic Implementation Selection
Very late binding

- Mobility of the high-level representation
- Performance of low-level implementation
- Dynamic Implementation Selection
- Cross-component optimisation
User Roles : Levels of Abstraction

• End-User
  5. Customise component instances
  6. Assemble composite application

• Scientist
  1. Design new component interfaces
  2. Specify component meta-information

• Developer
  3. Develop new implementations
  4. Specify implementation meta-data
An Extensible Component Framework

Repository

Optimising Tools

Application Construction Tools

Information!

Grid Environment
Framework: Repository

- Component Abstractions
  - Interface
  - Other meta-data (behaviour)
- Implementation Codes
- Implementation Meta-Data
  - Resource Requirements
  - Performance Characteristics
Framework: Application Construction

- Connection – oriented programming
  - Customisable component instances
  - Connections between inports & outports

- Extensibility: Visual or textual programming
Visual Application Construction Tool
Static Application Information

- Component XML (CXML):
  - Application as network of components: Application Description Document
  - Component interfaces
  - Implementation Performance Characteristics

- Stored in Repository
- Produced by Application Construction
CXML as an Intermediate Language

Annotated Java wrapper class around a native library

User

Components

Component Network

Grid Resources

CXML Repository

Developer

Component + implementation

Optimiser

Java wrapper class native library

Implementations

Execution Plan
Dynamic Information

• Run-Time Representation
  – Network of Java Proxy Objects
  – JPO corresponding to each component instance
    – Platform Independent
• Native Implementations
  – Platform Dependent
  – Wrapped & controlled by the JPOs
• Deployed on Grid
Dynamic Implementation Selection

Java Proxy Objects

Implementations

Grid Environment
Run Time Representation Architecture

- Port Definition Object
  - Outports trap method calls & data access calls
  - Inports make calls from other JPOs (pull model)
- Data Definition Object
  - Implementation independant data provides mobility
- Implementation Definition Object
  - Implementation Performance Characteristics
  - Binding Code
- Execution Object
  - Wrapping for native implementations
Java Proxy Object Architecture

PDO → DDO → IDO → EO

Native Code

Lightweight Implementation
Performance Guided Implementation Selection

• Selection Policy
  - From Component Framework

• Resource Information
  - From Grid Infrastructure

• Implementation Performance Models
  - From Run-Time Representation

• Application Composition
  - From Run-Time Representation
Composite Performance Modelling

- Models derived empirically
- Composed according to component behaviour – encapsulated in CXML.
Example: Linear Solver

- Linear Equation Source
  - Unsymmetric Matrix
  - DoF
  - Matrix
  - Vector
- Linear Equation Solver
  - LU
  - BiCG
- Display Vector
  - Vector
Linear Solver: Repository CXML

<repository>
  <component package="icpc.denseLinearAlgebra.real"
             name="LinearEquationSourceRowsColumnsUnsymmetric" version="1">
    <propertyDefinition type="external" name="degrees of freedom" value="1000"/>
    <port behaviour="OUT" objectPackage="icpc.matrix.real"
           objectName="MatrixRowsColumnsUnsymmetric" portName="matrix"/>
    <port behaviour="OUT" objectPackage="icpc.vector.real"
           objectName="Vector" portName="vector"/>
    <implementation language="java" platform="java" url="file:..">
      <action portName="matrix">
        <binding method="getMatrix"> ... </binding>
        <classPerformanceModel type="initial" url="http:" />
      </action>
      <implementation language="C" platform="Linux" url="file:.."> ... </implementation>
    </implementation>
  </component>
  <object package="icpc.matrix.real" name="MatrixRowsColumnsUnsymmetric"
           version="1">
    ... <method name="getMatrix" type="action">
      <argument mode="out" typeName="MatrixRowsColumnsUnsymmetric"
                 typePackage="icpc.matrix.real" />
    </method>
  </object>
</repository>
<application>
  <network>
    <instance componentName="LinearEquationSourceRowsColumnsUnsymmetric"
      componentPackage="icpc.denseLinearAlgebra.real" id="1">
      <property name="degrees of freedom" value="100"/>
    </instance>
    <instance componentName="LinearSolverRowsColumnsUnsymmetric"
      componentPackage="icpc.denseLinearAlgebra.real" id="2"/>
    <instance componentName="DisplayVector"
      componentPackage="icpc.vector.real" id="3"/>
    <dataflow sinkComponent="2" sinkPort="matrix"
      sourceComponent="1" sourcePort="matrix"/>
    <dataflow sinkComponent="2" sinkPort="vector"
      sourceComponent="1" sourcePort="vector"/>
    <dataflow sinkComponent="3" sinkPort="vector"
      sourceComponent="2" sourcePort="solution"/>
  </network>
</application>
Solution Time $v$ Matrix Size

- Network
- BCG
- LU
System Overhead vs Matrix Size
Open Extensible Technologies

• XML used to define:
  – application, component, implementation meta-data
  – the computational, storage and software resources
  – the resource usage policy
• Java used to construct the framework, run-time representation and interfaces to the assembled components
• Framework part of complete Grid services package – utilises Jini & Java
Further Work via Extensible Framework

• Multiple Cost Models:
  – Data transfer costs

• Parallel Implementations

• Distributed Applications

• Further use of run-time information:
  – Verification / optimisation at application level, eg numerical stability of FEM code
Summary

- Components for Grid computation
- Separation of abstraction & implementation
- Extensible architecture based on CXML
- Implementation selection through run-time representation
Acknowledgements

• London e-Science Centre, Grid Middleware Group:
  – John Darlington
  – Steven Newhouse
  – Anthony Mayer
  – Nathalie Furmento
  – Stephen McGough
  – Tony Field

• Funding: EPSRC GR/N13371

• Further Information:
  – http://www-icpc.doc.ic.ac.uk/components
  – email:icpc-sw@doc.ic.ac.ac.uk