On Abstractions of Software Component Models

Christian Perez
Christian.Perez@inria.fr
PARIS project-team, INRIA Rennes
France

Abstractions for Distributed Systems Workshop
Las Palmas, Gran Canaria, Spain, August the 25th 2008
Acknowledgement

Contributors to this presentation
- Thierry Priol (PARIS project-team leader)
- Andre Ribes (PhD, 2004)
- Sebastien Lacour (PhD, 2005)
- Hinde Bouziane (PhD, 2008)
- Julien Bigot (PhD student)
Content of the Talk

- Introduction
- Increasing the level of abstraction of component models
  - Master worker paradigm
  - Data sharing paradigm
  - MxN
- Automatic deployment of multi-technology applications
- Conclusion
(Scientific) Applications on Resources?

- Structural Mechanics
- Optics
- Thermal
- Dynamics

- Homogeneous cluster
- SAN
- LAN
- WAN
- Supercomputer
- Visualization

DPA Workshop, August 25th 2008
Applications on Resources?

Programming Environment (Virtualization)

- Structural Mechanics
- Optics
- Thermal
- Dynamics

- SAN
- Homogeneous cluster
- WAN
- Visualization
- Supercomputer
- LAN
Resources and Programming Models

- Resources are utilized through a programming model
- Sequential resources, sequential programming model
  - From assembly languages to imperative/functional/OO
- Parallel resources, parallel programming model
  - From message passing/thread to data parallel/tasks/mixed languages
1st Evolution: Code Reuse

- **Motivation**
  - Do not reinvent the wheel every day

- **Applications can be made of different building blocks**
  - Should support several programming models

- **Possible solutions**
  - Libraries
    - Sequential code are made of several languages
  - Interface Definition Languages

- **Need of a common model to act as a gateway**
  - Sequential application: libc standard (ELF)
2nd Evolution: the Grid concept

- Motivation
  - On demand computing
- Applications do not know their execution environment
  - Should abstract from resources
- Tracks
  - Virtual machines
  - Just In Time compilation
- Need to control the compilation from the programming model to the resource model (OS model?)
Software component

- Technology that advocates for composition
  - Fabrication stage & reuse stage

- A component is a black box that interacts by its ports
  - Port ~ access point
    - Name
    - Description and protocol
    - Abstract notion, without implementation

- Consequence:
  - Separation of the interface and the implementation
    - Type of a component = set of its ports

- A component may have several implementations
  - Different languages/OS/architectures
  - Different algorithms
Assembly of Software Components

- Description of an assembly
  - Dynamically (API)
  - Statically

- Architecture Description Language (ADL)
  - Describes
    - Component instances
    - Port connection
  - Available in many CMs

Diagram:
- Composite
  - C1
  - C2
  - C3
  - C4
  - C5

DPA Workshop, August 25th 2008
Which Usage for Components?

- Two kinds of role
  - Components as an execution model entity
    - Usual role
  - Components as a programming model entity
    - Does it make sense?
      - Components are already a first class citizen of ADL
      - If applications will be made of components, there is a clear need to have high level component assembly language
        - Is still an ADL?
Why Increasing the Level of Abstraction of Component Models?

- Improve productivity
  - Many recurring composition patterns
- Reduce complexity
  - Embed expertise in “simple” concept
- Hide implementation details
  - Resource dependent composition

- Simplify application development while keeping high performance
Increasing the Level of Abstraction of Component Models

- At port level
  - Data shared port
  - Collective communication
- At component level
  - SPMD paradigm (MxN communication)
- At assembly level
  - Master-worker paradigm
  - (Skeletons)
- Assembly kind
  - Spatial and temporal composition (Thursday talk, 11h30)
Increasing the Level of Abstraction of Component Models at Assembly Level

Master-Worker Paradigm
Generalization of the problem

- Simultaneous independent computations (~ForAll loop)
- Dedicated API/environments
  - BOINC, XTremWEB,
  - DIET, NetSolve, Nimrod/G, ...
Limits with existing component models

- Resources infrastructure dependence
  - SMP machine vs Cluster

- Request delivery implementation at the burden of the master
  - No transparency
  - Complex

- Request delivery policy outside
  - Resources infrastructure dependence!
  - For grids, complex request delivery implementations
  - No transparency
Master-Worker oriented Component Collection

Programmer/designer view

- Master
- Worker
- Binding
- Exposed provided port

Framework implementation view

- Resources infrastructure
- #workers
- Pattern selection
- Set of request transport mechanism patterns
  1. Random
  2. Round-Robin
  3. NetSolve
  4. Diet
Examples of Patterns

Simple component based pattern

Hierarchical scheduling pattern

DIET pattern
Some Experimental Results on Grid5000

- Scheduling policies
  - Round-robin (RR)
  - Load balancing (LB)
  - Request sequencing

- MW Patterns
  - « Proxy » component
  - DIET

---

### Table

<table>
<thead>
<tr>
<th></th>
<th>4096 heterogeneous req. (15s-20s)</th>
<th>40 heterogeneous req. (20s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time (s)</td>
<td>Eff. (%)</td>
</tr>
<tr>
<td>1024 wks</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C-RR</td>
<td>80.11</td>
<td>99.87</td>
</tr>
<tr>
<td>C-LB</td>
<td>80.12</td>
<td>99.85</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th></th>
<th>512 reqs of 20s + 1 req of 100s</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time (s)</td>
</tr>
<tr>
<td>128 wks</td>
<td></td>
</tr>
<tr>
<td>C-LB</td>
<td>180.01</td>
</tr>
<tr>
<td>C-DIET</td>
<td>100.82</td>
</tr>
</tbody>
</table>

---

### Table

<table>
<thead>
<tr>
<th>#workers</th>
<th>Bordeaux</th>
<th>Lyon</th>
<th>Nancy</th>
<th>Orsay</th>
<th>Rennes</th>
<th>Sophia</th>
<th>Toulouse</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>78</td>
<td>124</td>
<td>86</td>
<td>192</td>
<td>256</td>
<td>192</td>
<td>96</td>
</tr>
<tr>
<td>512</td>
<td>39</td>
<td>62</td>
<td>43</td>
<td>96</td>
<td>128</td>
<td>96</td>
<td>48</td>
</tr>
<tr>
<td>128</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>22</td>
<td>22</td>
<td>-</td>
</tr>
</tbody>
</table>

---

*Fig. 18. Worker distribution for experiments on Grid5000.*
Increasing the Level of Abstraction of Component Models at Port Level

Data Shared Port
Problem overview

- Multiple concurrent accesses to a piece of data
- Data localization and management
  - Intra-process: local shared memory
  - Intra-machine: shared memory segment
  - Intra-cluster: distributed shared memory (DSM)
  - Intra-grid: grid data sharing service (JuxMem)
Limits with data sharing in component models

- **Ports**: active communication operation
  - Data must be part of a message

- **Centralized approach**
  - Bottleneck for the performance
  - Single point of failure

- **Distributed approach**
  - Explicit management of data replication/migration by components
  - Functional code mixed with data management code
Data Shared Port Model

- Principle: rely on a data sharing management service

- Transparent data localization and management
  - Local shared memory, DSM, JuxMem, ...

```cpp
class data_ref {
public:
    float* get_pointer();
    long get_size();
    void acquire();
    void acquire_read();
    void release();
};
```

```cpp
component D {
    shares array<float> u;
};
```

```cpp
component A {
    accesses array<float> v;
};
```
Increasing the Level of Abstraction of Component Models at Component Level

Parallel Components
SPMD Components

What the application designer should see…

… and how it must be implemented!

Component A

Component B

Component A

Component B
Components for code coupling: SPMD paradigm in GridCCM

- SPMD component
  - Parallelism is a non-functional property of a component
    - It is an implementation issue
  - Collection of sequential components
    - SPMD execution model
  - Support of distributed arguments
    - API for data redistribution
    - API for communication scheduling w.r.t. network properties
  - Support of parallel exceptions

![Diagram](image_url)

- Application
  - Communication Library (MPI)
- GridCCM runtime
  - Application view management
    - Data distribution description
  - Communication management
    - Comm. Matrix computation
    - Comm. Matrix scheduling
    - Communication execution
- Redistribution Library 1
- Scheduling Library
- Communication Library
- CORBA stub/skeleton
- Object Request Broker

![Graph](image_url)

- Component A
  - Configuration
- Component B
  - Configuration

![Graph](image_url)

- Aggregated Bandwidth in MB/s
- Component configuration
- Number of processors per parallel component
- Without scheduling
- With scheduling

DPA Workshop, August 25th 2008
Just In Time ADL Compilation?

- All “advanced” component construct implementations are based on some transformations to a “low/execution” level component model
- Compilation enables to adapt abstraction to the actual resources
  - Transparent adaptation if the compilation is on the fly

```
Advanced Component Model

Compilation

“Simple” Component Model

Programming Model

Execution Model
```
Prototype Architecture Overview

- Current supported concepts
  - Spatial & temporal composition of hierarchical components
  - Master-worker paradigm
    - Implemented as a special case of a self transforming composite
- 3 back-ends
  - Simulation: does not actually instance and run components
  - JAVA: only support JAVA components
  - CORBA Component Model
    - Utilized ADAGE to actually deploy components
- Internals
  - Abstract representation of the assembly

Program  Parser/Lexer (ANTLR)  Abstract Representation  Centralized Engine  Instances
Automatic Deployment of Muti-Technology Applications (on Grids)

Christian Perez
Example of Application

[Diagram showing a network of nodes and arrows representing interactions between master, log, worker, CCM/DIET, LA, Manager, Provider, and DIET/CCM.]
Manual Deployment on Grids

- Discover available grid resources
- Select grid resources for execution
  - OS, architecture compatibility
- Map the application onto the selected resources
  - MPI processes
  - Components
- Upload and install executables, stage input files in
- Launch processes on remote computers
- Set configuration parameters of the application
  - Components' attributes
  - Network topology information

Too complex!
Automatic Deployment

- Hide application complexity
- Hide grid complexity

Automatic
- Resource discovery
- Execution node selection
- File installation
- Process launch
- Application configuration

stop reading your e-mails!
ADAGE: Automatic Deployment of Applications in a Grid Environment

- Deploy a same application on any kind of resources
  - from clusters to grids
- Support multi-middleware applications
  - MPI+CORBA+JXTA+DIET+GFARM...
- GADe: Generic Application Description
  - Processes, groups of processes, code to load, connections, recursive trees
- Planners as plugin
  - Round robin, random
  - Advanced recursive tree heuristics
- Support for dynamic applications
- Some successes on Grid5000
  - 29,000 JXTA peers on ~400 nodes
  - 4003 components on 974 processors on 7 sites

http://adage.gforge.inria.fr
Conclusion

- Software components seems promising to be a *programming* entity
  - Improve productivity, reduce complexity & hide implementation details
- Abstractions applied to component models
  - NxM component
  - Master-Worker paradigm
    - wth dynamic adaptation (Dynaco)
  - Data sharing between component
  - Hierarchical distributed data
  - Collective communication between components
  - Spatial-temporal composition
    - merged with skeleton in cooperation with UNIPI, CoreGRID REP
- Challenge: mapping applications onto resources
  - Strategies to select the right transformation?
    - W.r.t to which criteria? QoS?
  - A same application for a large variety of resources?