Formal Models for Concurrent Reconfiguration of Component Assemblies

Vincent Lanore
December 10, 2015

Based on work done with Christian Pérez

Discrete Structures Day
Subset of Avalon working on programming models

- for cloud
- for computing grids
- for High-Performance Computing (HPC)...

People

- Christian Pérez
- Hélène Coullon
- Jérôme Richard
- Pedro Silva
- myself
Programming model:

- Idea
- **Formal specification**
  - formal syntax
  - formal semantics
- **Properties?**
- Implementation
- Evaluation on use cases
  - performance
  - code metrics
Thesis Approach

Programming model:

- Idea
- **Formal specification**
  - formal syntax
  - formal semantics
- Properties?
- Implementation
- Evaluation on use cases
  - performance
  - code metrics

Benefits

- sturdier approach
- formal results
- connections with formal software engineering communities
1. Context
   - Component Models
   - Reconfiguration

2. The DirectMOD Component Model

3. Other Problems

4. Conclusion and Perspectives
Plan

1. Context
   - Component Models
   - Reconfiguration

2. The DIRECTMOD Component Model

3. Other Problems

4. Conclusion and Perspectives
How to ease reuse by third parties?

code
Component-Based Programming

How to ease reuse by third parties?

entry point

code

function pointer
How to ease reuse by third parties?

code
How to ease reuse by third parties?

Component-Based Programming
How to ease reuse by third parties?

→ a software component
How to ease reuse by third parties?

→ a component assembly
Component Models

Ports/component definitions

- type
- type
- type
- thing

Benefits
- reuse
- separation of concerns
- structure-level view

HPC Component Models
- Examples: CCA, L2C...
Component Models

Ports/component definitions

+ assembly model

Benefits

• reuse
• separation of concerns
• structure-level view

Examples: CCA, L2C...

type

type

type

...
Component Models

Ports/component definitions

Benefits

- reuse
- separation of concerns
- structure-level view

+ assembly model
Component Models

Port/component definitions

Benefits
- reuse
- separation of concerns
- structure-level view

+ assembly model

HPC Component Models
- Examples: CCA, L2C...
### Reconfigurable applications

- Application structure changes at runtime

**Example:**
- Adaptive Mesh Refinement (AMR)
  - Fixed resolution cells
  - One process per cell
  - Complex data structure
  - Dynamic process pool
  - Dynamic data structure

⇒ Complex for programmers
Reconfigurable applications

<table>
<thead>
<tr>
<th>Reconfigurable applications</th>
<th>application structure changes at runtime</th>
</tr>
</thead>
</table>

**Example:** Adaptive Mesh Refinement (AMR)
Reconfigurable HPC Applications

Reconfigurable applications
application structure changes at runtime

Example: Adaptive Mesh Refinement (AMR)

Discretization
Reconfigurable HPC Applications

Reconfigurable applications
application structure changes at runtime

**Example:** Adaptive Mesh Refinement (AMR)

- 1 cell = 1 process
  + 1 mesh of data

- fixed resolution cells
- one process per cell
Reconfigurable applications

application structure changes at runtime

Example: Adaptive Mesh Refinement (AMR)

Discretization

- fixed resolution cells
- one process per cell
Reconfigurable HPC Applications

Reconfigurable applications
application structure changes at runtime

**Example:** Adaptive Mesh Refinement (AMR)

Variable resolution

- fixed resolution cells
- one process per cell
- complex data structure
Reconfigurable HPC Applications

Reconfigurable applications
application structure changes at runtime

**Example:** Adaptive Mesh Refinement (AMR)

**Variable resolution**

- fixed resolution cells
- one process per cell
- complex data structure
Reconfigurable applications
application structure changes at runtime

Example: Adaptive Mesh Refinement (AMR)

Dynamic re-meshing

- fixed resolution cells
- one process per cell
- complex data structure
- dynamic process pool
- dynamic data structure
Reconfigurable applications
application structure changes at runtime

Example: Adaptive Mesh Refinement (AMR)

Dynamic re-meshing

- fixed resolution cells
- one process per cell
- complex data structure
- dynamic process pool
- dynamic data structure

⇒ complex for programmers
One possible way to write AMR:

**Pro**
- structure-level reconfiguration

**Challenges**
- application model
- performance
1 Context
   - Component Models
   - Reconfiguration

2 The DIRECTMOD Component Model

3 Other Problems

4 Conclusion and Perspectives
Programming Model Roles

Component programmers writes

Assembler (end user) writes

reusable

application-specific
Programming Model Roles

Component programmers

Transformation programmers

Assembler (end user)

Reusable

Application-specific
Programming Model Roles

Component programmers write Compute

Transformation programmers write NOP

Locking/synchronization specialist writes in1 out0 adapt0

Assembler (end user) writes Compute0

Domain A

Domain B

Domain C

reusable

application-specific
The DirectMOD Component Model

Base component model

- non-reconfigurable
- similar to L2C/CCA
- call-stack-based operational semantics (see manuscript)
- resource model (see manuscript)

DirectMOD: a full reconfigurable model

- additional concepts to
  - specify locking scope
  - specify reconfiguration
- extended syntax + reconfiguration semantics
Basic Assembly

Elements
- components
- ports

Relations
- point-to-point references
- owner
New element: domains

- manages a subassembly
- unit of locking
- reconfigure its contents
- user-defined scope
New element: domains

- manages a subassembly
- unit of locking
- reconfigure its contents
- user-defined scope
Transformations

- origin
- transformation
- state
- topology
- destination

Compute

Insert
Transformations

- Compute
- Monitor

`Insert` edge from Compute to Compute.

`replace` edge from Monitor to Compute.
Adapters

- special kind of port
- links transformation to its target
Full DirectMOD Assembly

- Specify starting assembly
- Assembly representation during runtime
specify starting assembly
during runtime
• specify starting assembly
• assembly representation during runtime
The set of DirectMOD ports on nameset $N$ is defined by:

\[
\text{Ports}_{dmod}(N) = \{\text{USE}(\text{name}, \text{ref}) \mid (\text{name}, \text{ref}) \in N^2\} \\
\cup \{\text{PROVIDE}(\text{name}) \mid \text{name} \in N\} \\
\cup \{\text{ADAPT}(\text{name}, \text{transfo}) \mid (\text{name}, \text{transfo}) \in N^2\}
\]

where:

- provide and use ports are defined as for the preliminary model;
- the $\text{name} \in N$ in the $\text{ADAPT}$ operator is the name of the adapter;
- $\text{transfo} \in N$ is the transformation reference of the adapter;
- $\alpha$ is the target assembly.
Example Definition: Transformations

Let $\mathcal{A}$ be a set of assemblies and $\mathcal{N}$ a set of names. A transformation $\tau$ is of the form:

$$\tau = (name, \alpha, \omega, s, t)$$  \hspace{1cm} (2)

where:

- $name \in \mathcal{N}$ is the name of the transformation;
- $\alpha \in \mathcal{A}$ is the origin of the transformation;
- $\omega \in \mathcal{A}$ is the destination of the transformation;
- $s: \text{Support}(\alpha) \rightarrow \text{Support}(\omega) \cup \{\bot\}$ is the state mapping;
- $t: \text{Support}(\alpha) \rightarrow \text{Support}(\omega) \cup \{\bot\}$ is the topology mapping.
The DirectMOD Component Model

The DirectMOD model

- specialized graph structure
  - components, ports, resources
  - concurrent semantics
- transformations

Challenges and perspectives

- proofs with transformations
- locking / deadlock detection
Plan

1. Context
   - Component Models
   - Reconfiguration

2. The DIRECTMOD Component Model

3. Other Problems

4. Conclusion and Perspectives
SpecMOD Principle

Very simple assembly model
- components (squares)
- endpoints (circles)
- edges

Rich type system
- component types
- endpoint types
- specialization relations

Assembly

Master

\( n \) workers

Worker

Worker1

Worker2

Type system

\( n \) workers

2 workers

3 workers

2 workers
SpecMOD Principle

Very simple assembly model

- components (squares)
- endpoints (circles)
- edges

Rich type system

- component types
- endpoint types
- specialization relations

Assembly

Worker

Worker1

Worker2

Type system

Worker

n workers

2 workers

3 workers

2 workers

2 workers

Master

2 workers

2 workers

n workers

2 workers

3 workers
SpecMOD Principle

Very simple assembly model
- components (squares)
- endpoints (circles)
- edges

Rich type system
- component types
- endpoint types
- specialization relations

Assembly

Worker

Type system

Worker

Worker1

Worker2

Worker

n workers

2 workers

3 workers

2 workers

Worker

Worker
SpecMOD Principle

Very simple assembly model
- components (squares)
- endpoints (circles)
- edges

Rich type system
- component types
- endpoint types
- specialization relations
SpecMOD: a general specialization calculus

- type system definition
- assembly definition
- specialization operations as rewriting rules
- parametric type systems encoding
Encoding Example
Work with Chardet Maverick
Components implemented by component assemblies called composite components

Structure
- bigraphs?

Challenges
- transformation expression
Components implemented by component assemblies
called composite components

**Structure**
- bigraphs?

**Challenges**
- transformation expression
Components implemented by component assemblies called composite components

Structure
- bigraphs?

Challenges
- transformation expression
Plan

1. Context
   - Component Models
   - Reconfiguration

2. The DIRECTMOD Component Model

3. Other Problems

4. Conclusion and Perspectives
Conclusion

Discrete structures themes:
- specialized graph-like structures
  - multi-sorted graphs
  - list edges
  - hierarchy
- graph transformations
- graph specialization through rewriting

Problems and perspectives
- proofs with transformations
- locking / deadlock detection
- encoding / decoding complex structures