Formal Models for Concurrent Reconfiguration of Component Assemblies

Vincent Lanore

December 10, 2015

Based on work done with Christian Pérez

Discrete Structures Day

Avalon Programming Model People

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

Thesis Approach

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

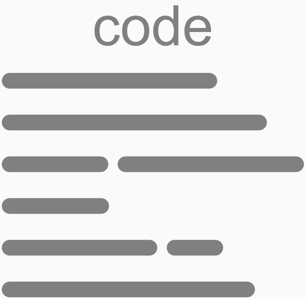
Plan

Plan

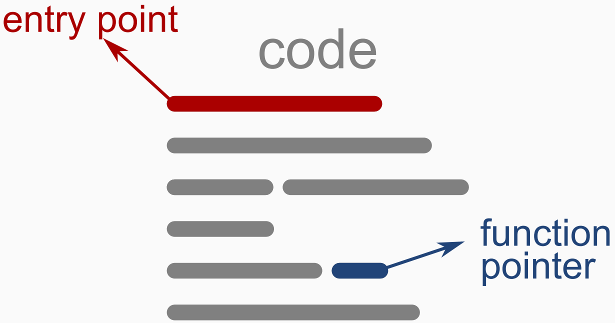
1. Context
   - Component Models
   - Reconfiguration
2. The DirectMOD Component Model
3. Other Problems
4. Conclusion and Perspectives

Component-Based Programming

How to ease reuse by third parties?

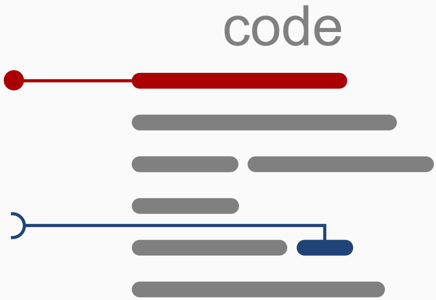


How to ease reuse by third parties?



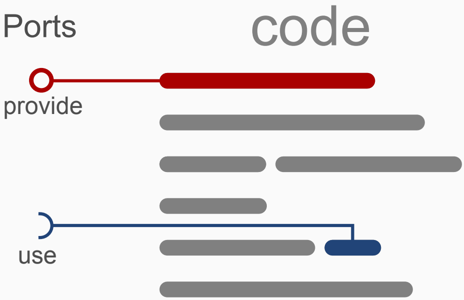
Component-Based Programming

How to ease reuse by third parties?



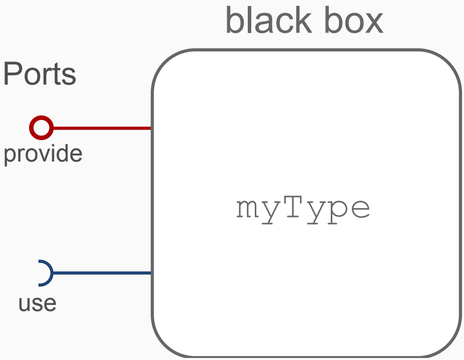
Component-Based Programming

How to ease reuse by third parties?



Component-Based Programming

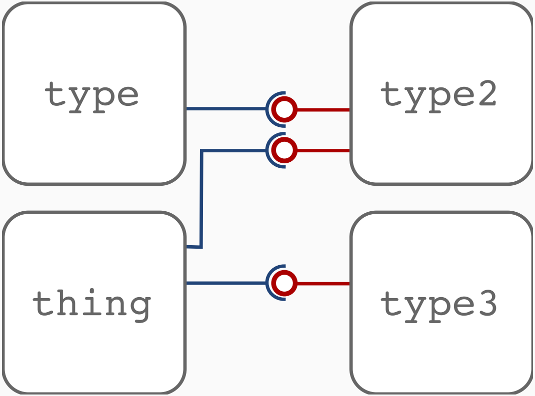
How to ease reuse by third parties?



$\rightarrow$ a software component

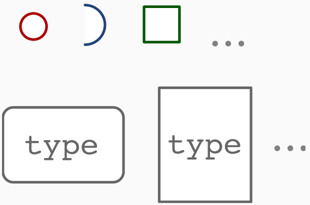
Component-Based Programming

How to ease reuse by third parties?



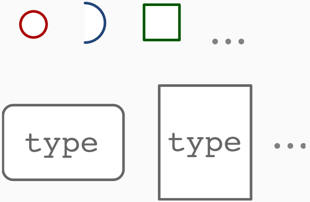
→ a component assembly

Ports/component definitions

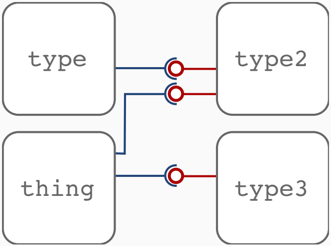


Component Models

Ports/component definitions

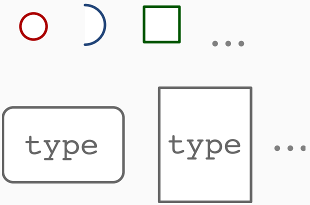


+ assembly model

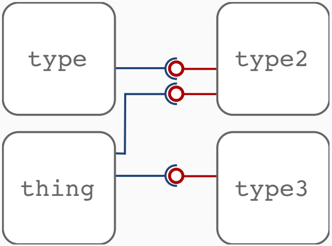


Component Models

Ports/component definitions



+ assembly model

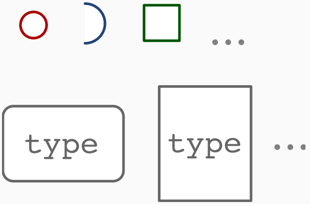


Benefits

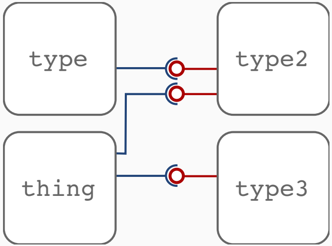
- reuse
- separation of concerns
- structure-level view

Component Models

Ports/component definitions



+ assembly model



Benefits

- reuse
- separation of concerns
- structure-level view

HPC Component Models

- Examples: CCA, L2C...

Reconfigurable HPC Applications

Reconfigurable applications
application structure changes at runtime

Reconfigurable HPC Applications

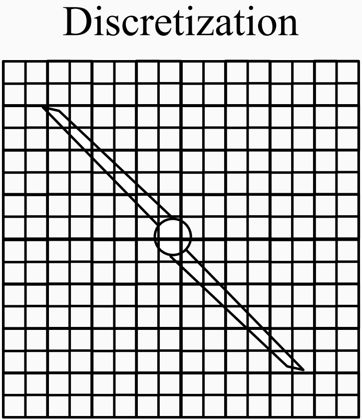
Reconfigurable applications
application structure changes at runtime

Example: Adaptive Mesh Refinement (AMR)

Reconfigurable applications

application structure changes at runtime

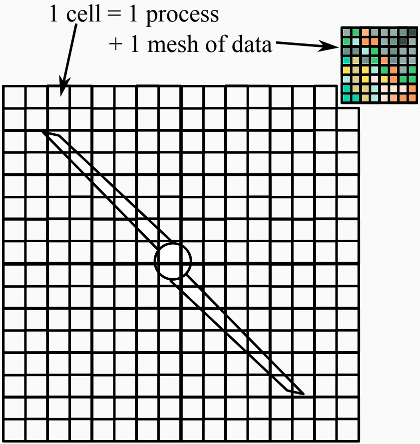
Example: Adaptive Mesh Refinement (AMR)



Reconfigurable applications

application structure changes at runtime

Example: Adaptive Mesh Refinement (AMR)



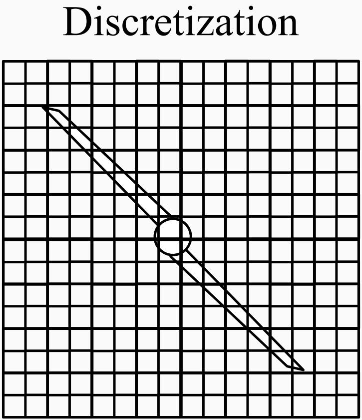
- fixed resolution cells
- one process per cell

Reconfigurable HPC Applications

Reconfigurable applications

application structure changes at runtime

Example: Adaptive Mesh Refinement (AMR)

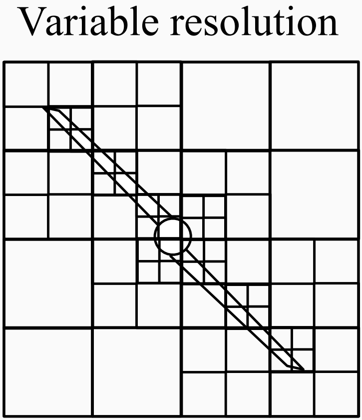


- fixed resolution cells
- one process per cell

Reconfigurable applications

application structure changes at runtime

Example: Adaptive Mesh Refinement (AMR)

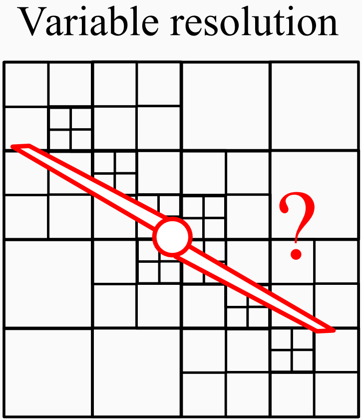


- fixed resolution cells
- one process per cell
- complex data structure

Reconfigurable applications

application structure changes at runtime

Example: Adaptive Mesh Refinement (AMR)

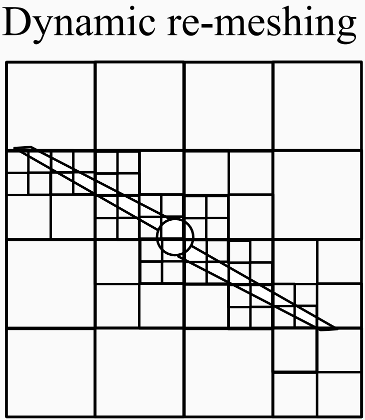


- fixed resolution cells
- one process per cell
- complex data structure

Reconfigurable applications

application structure changes at runtime

Example: Adaptive Mesh Refinement (AMR)

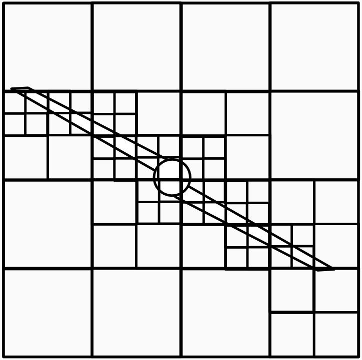


- fixed resolution cells
- one process per cell
- complex data structure
- dynamic process pool
- dynamic data structure

Reconfigurable applications
application structure changes at runtime

Example: Adaptive Mesh Refinement (AMR)

Dynamic re-meshing

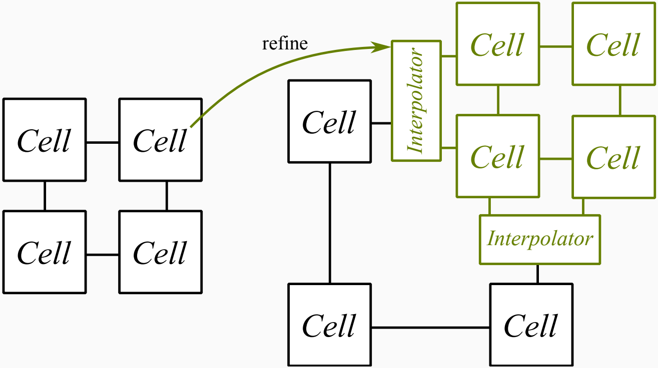


- fixed resolution cells
- one process per cell
- complex data structure
- dynamic process pool
- dynamic data structure

⇒**complex for programmers**

Structure-level Reconfiguration

One possible way to write AMR:



Pro

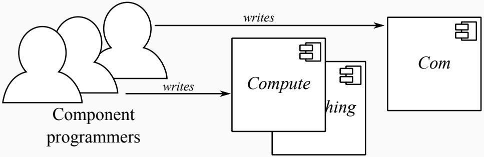
- structure-level reconfiguration

Challenges

- application model
- performance

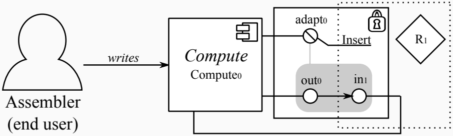
Plan

Programming Model Roles

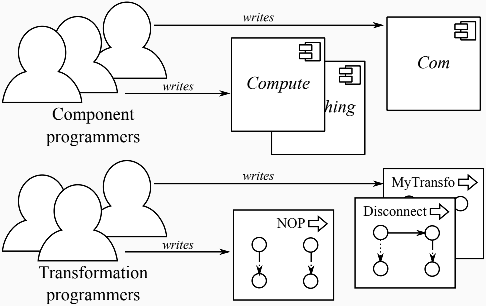


reusable

application-specific

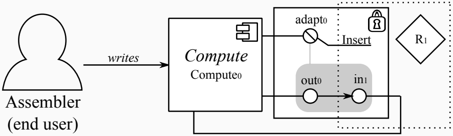


Programming Model Roles

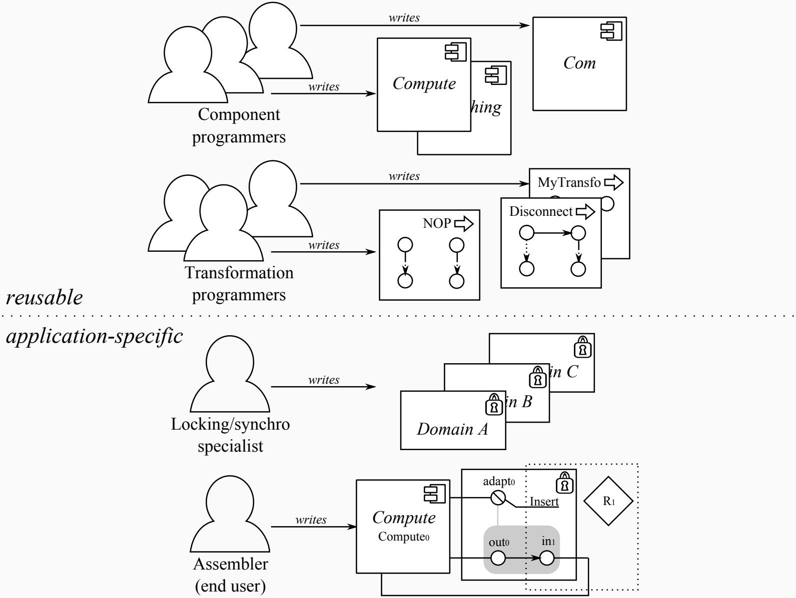


reusable

application-specific



Programming Model Roles



The DirectMOD Component Model

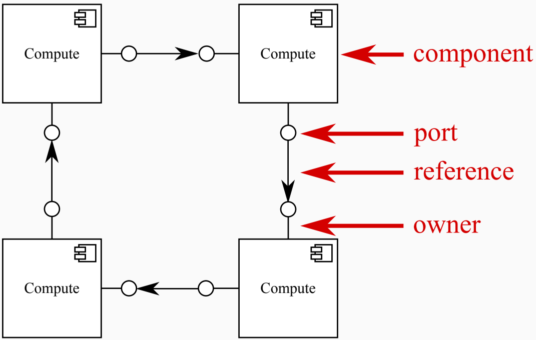
Base component model

- 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



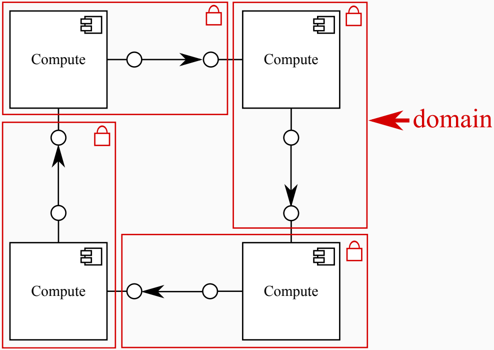
Elements

- components
- ports

Relations

- point-to-point references
- owner

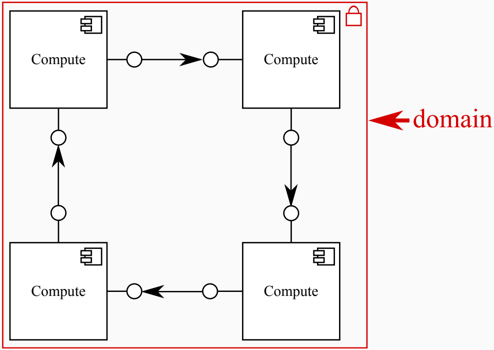
Domains



New element: domains

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

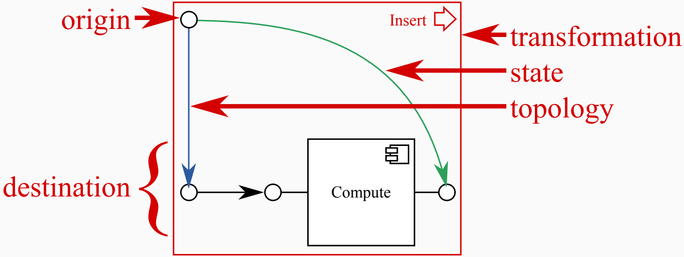
Domains



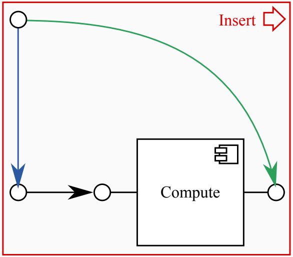
New element: domains

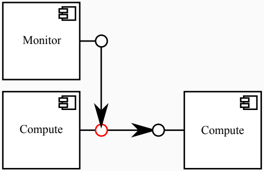
- manages a subassembly
- unit of locking
- reconfigure its contents
- user-defined scope

Transformations

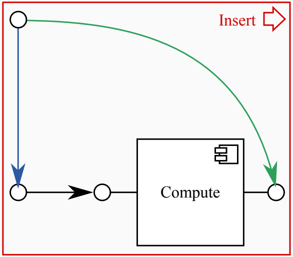


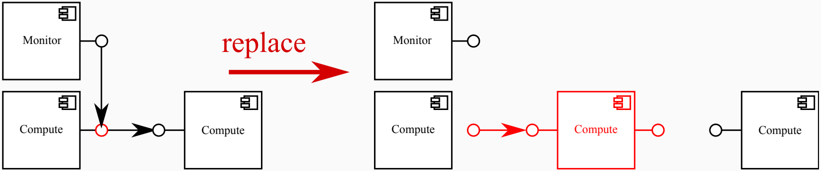
Transformations



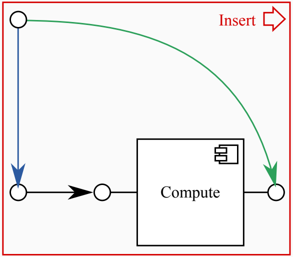


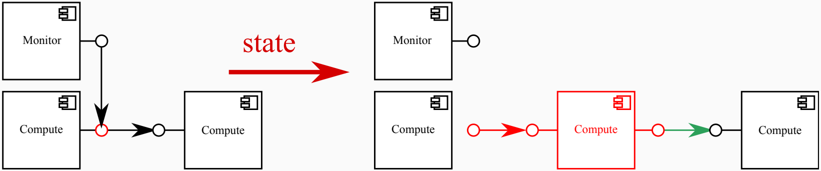
Transformations



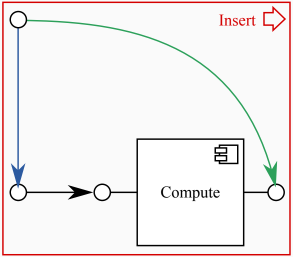


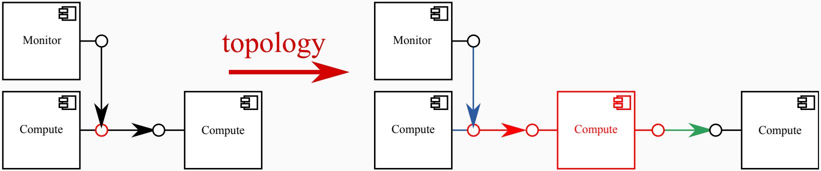
Transformations



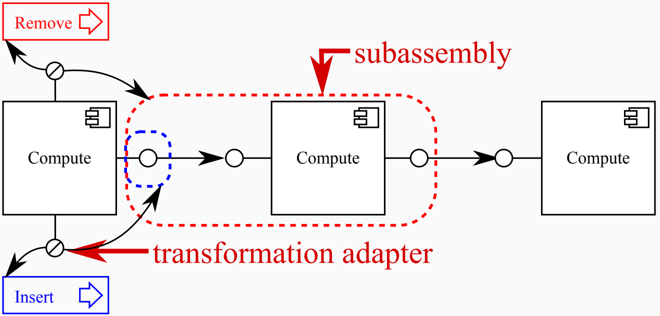


Transformations



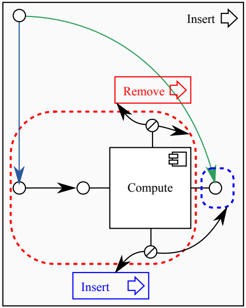


Adapters

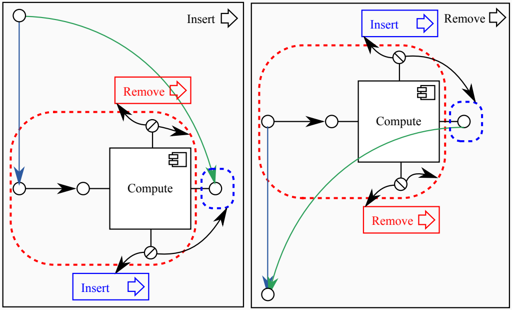


- special kind of port
- links transformation to its target

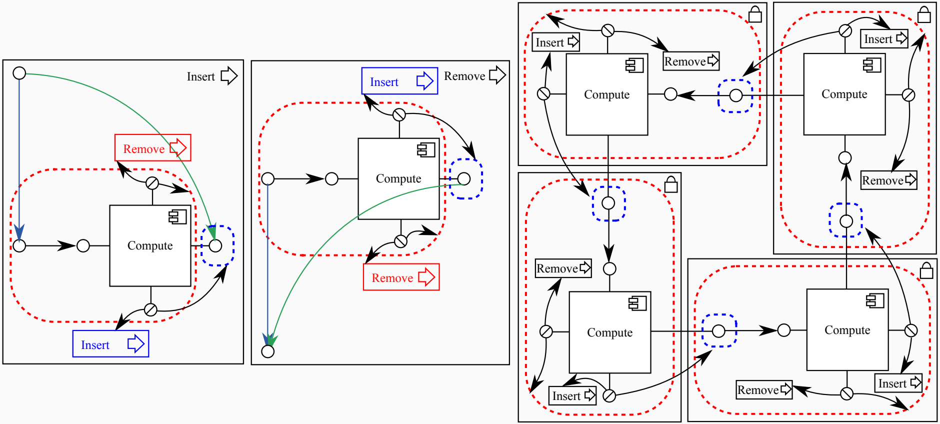
Full DirectMOD Assembly



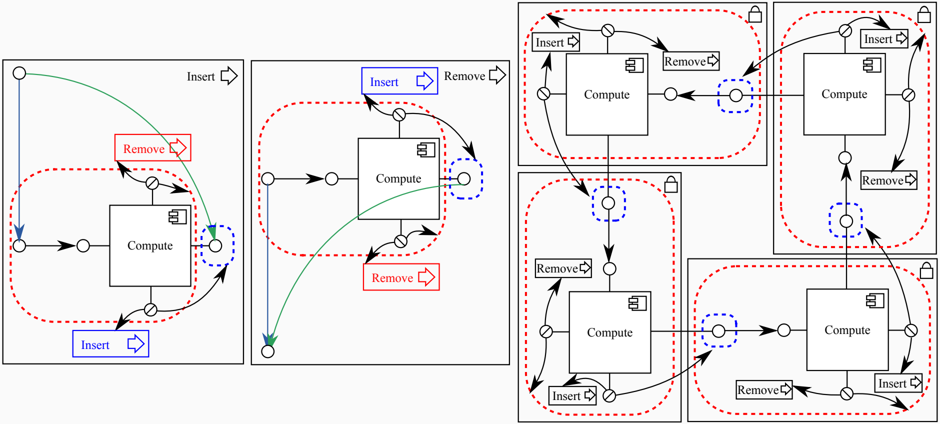
Full DirectMOD Assembly



Full DirectMOD Assembly



Full DirectMOD Assembly



- specify starting assembly
- assembly representation during runtime

Definition Example: Ports

The set of DirectMOD ports on nameset $\mathcal{N}$ is defined by:

$$
\begin{aligned}
&\mathit{Ports}_{dmod}(\mathcal{N}) \\
&\quad = \{\mathit{USE}(\mathit{name}, \mathit{ref}) \mid (\mathit{name}, \mathit{ref}) \in \mathcal{N}^2\} \\
&\qquad \cup \{\mathit{PROVIDE}(\mathit{name}) \mid \mathit{name} \in \mathcal{N}\} \\
&\qquad \cup \{\mathit{ADAPT}(\mathit{name}, \mathit{transfo}) \mid (\mathit{name}, \mathit{transfo}) \in \mathcal{N}^2\}
\end{aligned} \tag{1}
$$

where :

- provide and use ports are defined as for the preliminary model;
- the $\mathit{name} \in \mathcal{N}$ in the ADAPT operator is the name of the adapter;
- $\mathit{transfo} \in \mathcal{N}$ is the transformation reference of the adapter;
- α is the target assembly.

Example Definition: Transformations

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

$$\tau = (\mathit{name}, \alpha, \omega, s, t) \tag{2}$$

where:

- $\mathit{name} \in \mathcal{N}$ 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;
- $s : \mathit{Support}(\alpha) \rightarrow \mathit{Support}(\omega) \cup \{\bot\}$ is the *state mapping*;
- $t : \mathit{Support}(\alpha) \rightarrow \mathit{Support}(\omega) \cup \{\bot\}$ 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

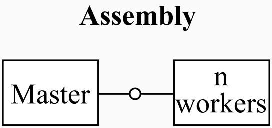
SpecMOD Principle

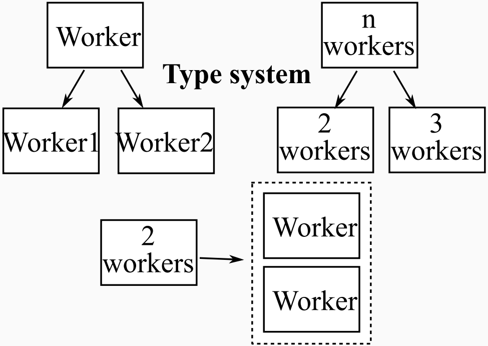
Very simple assembly model

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

Rich type system

- component types
- endpoint types
- specialization relations





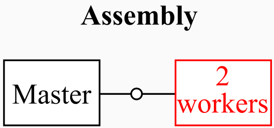
SpecMOD Principle

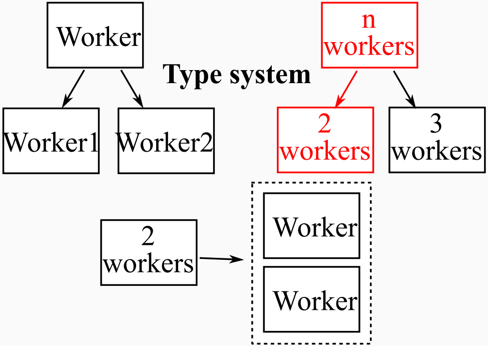
Very simple assembly model

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

Rich type system

- component types
- endpoint types
- specialization relations





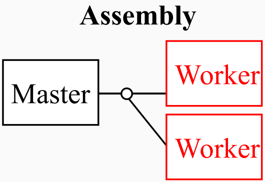
SpecMOD Principle

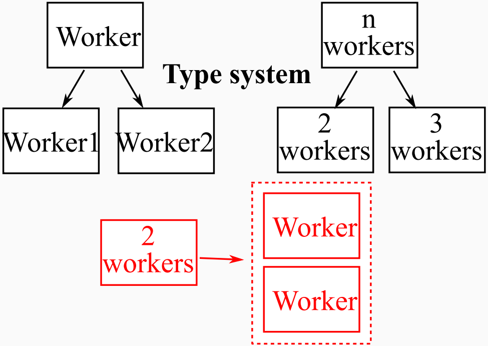
Very simple assembly model

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

Rich type system

- component types
- endpoint types
- specialization relations





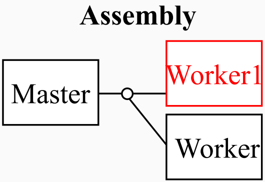
SpecMOD Principle

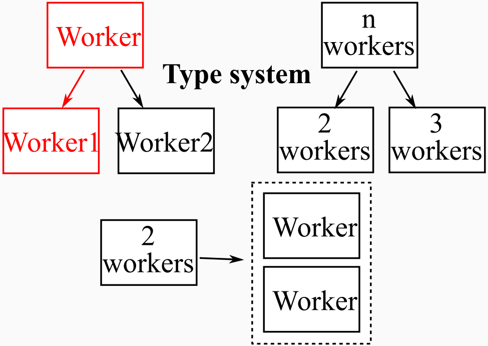
Very simple assembly model

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

Rich type system

- component types
- endpoint types
- specialization relations



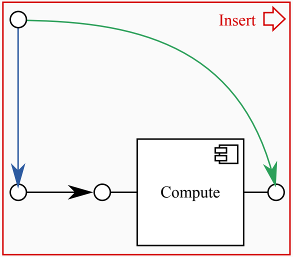


Formal Work

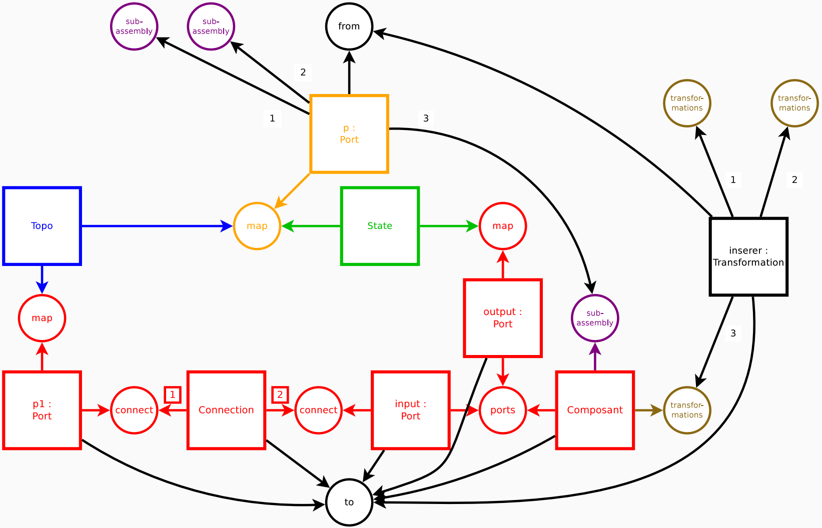
SpecMOD: a general specialization calculus

- type system definition
- assembly definition
- specialization operations as rewriting rules
- parametric type systems encoding

Encoding Example



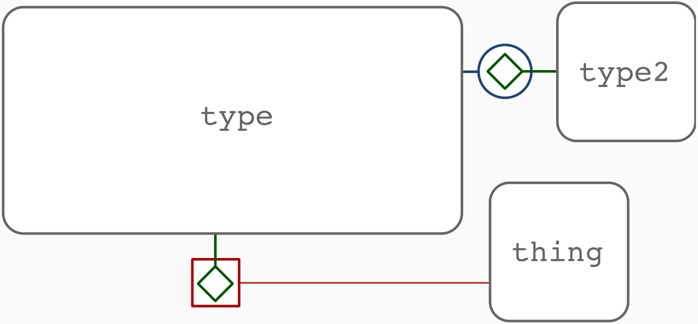
Encoding Example



Work with Chardet Maverick

Hierarchy

Components implemented by component assemblies called composite components



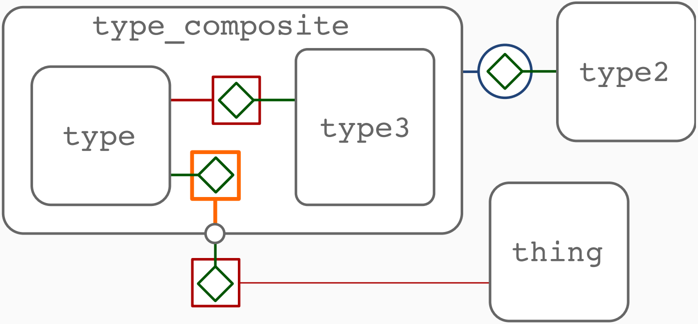
Structure

- bigraphs?

Challenges

- transformation expression

Components implemented by component assemblies called composite components



Structure

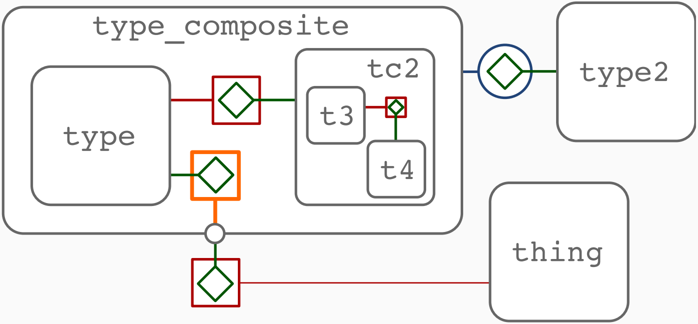
- bigraphs?

Challenges

- transformation expression

Hierarchy

Components implemented by component assemblies
called composite components



Structure

- bigraphs?

Challenges

- transformation expression

Plan

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