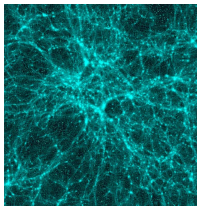
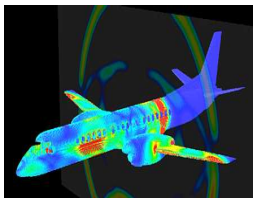
Model, Connector and Deployment

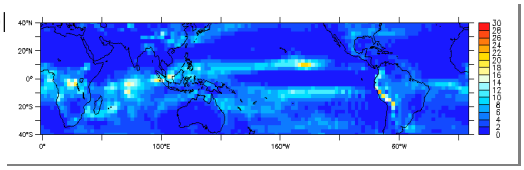
Christian Perez
LIP/INRIA
2010-2011

Content

- Models
- MDE for extending CM
  - Principle
  - Genericity
- Connectors
  - “Classical” connector
  - “Open” connectio
- Deployment
- Conclusion

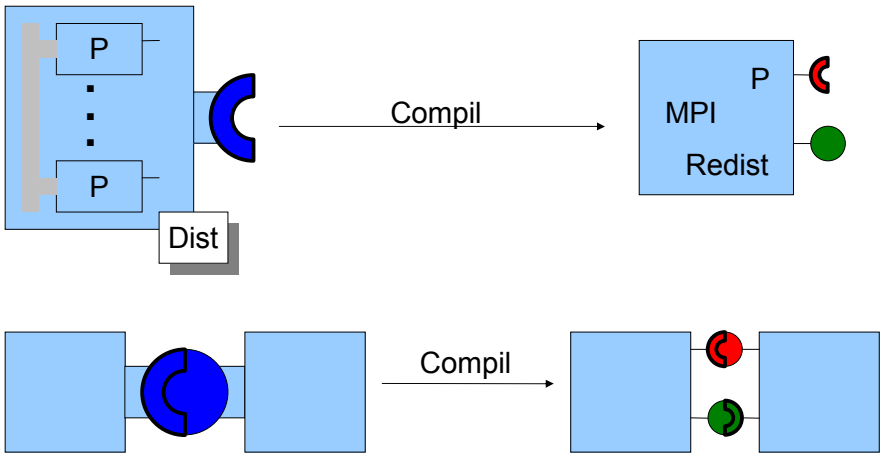






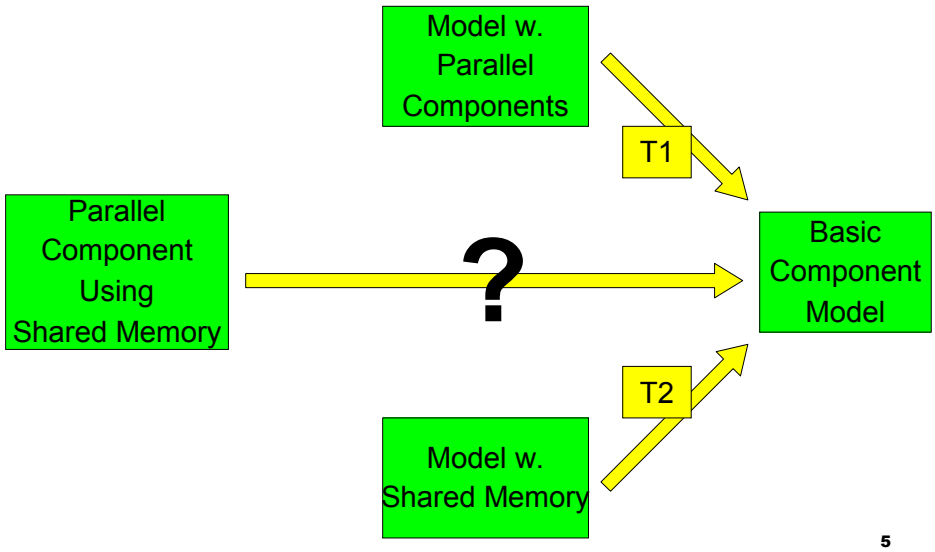
Using Model Transformation To extend Component Models

Approaches for the Extension of Component Models



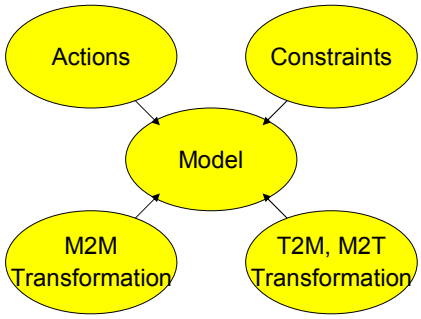
4

Interaction between transformations



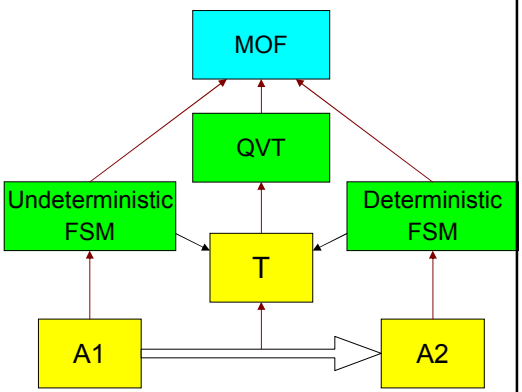
5

Proposed approach: Model based transformations



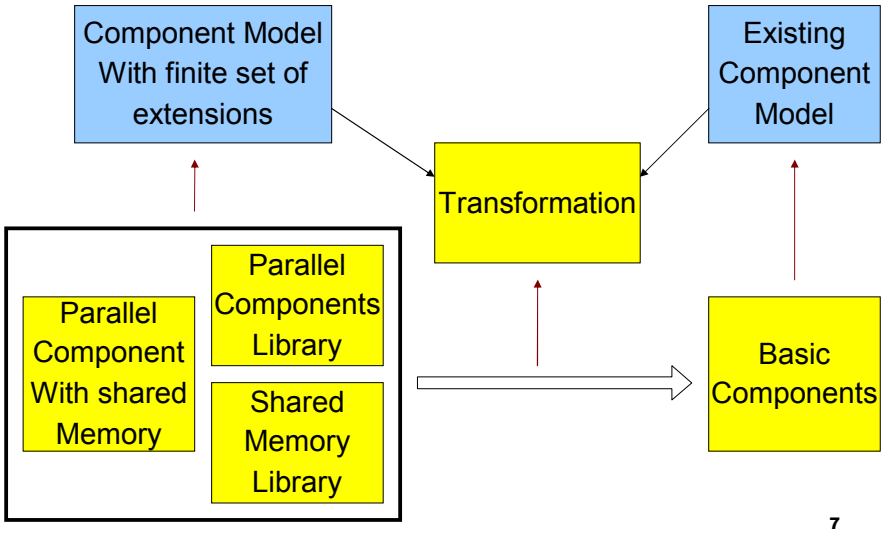
Meta-modeling Ecosystem

Model to Model Transformation



6

Proposed Approach for Component Model Extension



7

Extension classification

- Component implementations
  - Parallel components
  - Workflow / Dataflow
  - Skeletons
  - ...

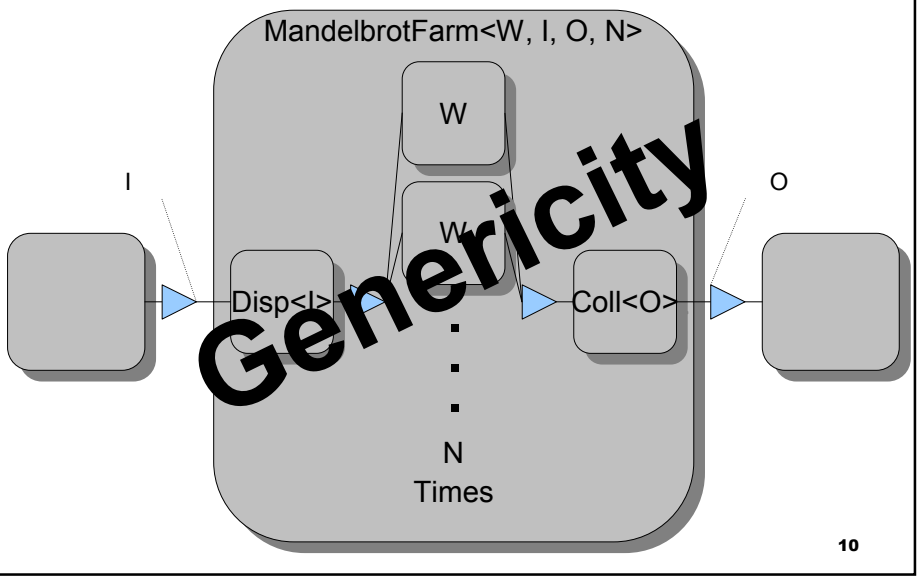
  → Partially solved by Genericity
- Interactions between components
  - Event & messages
  - MxN method calls
  - Shared memory
  - Collective communications
  - ...

  → Connectors

8

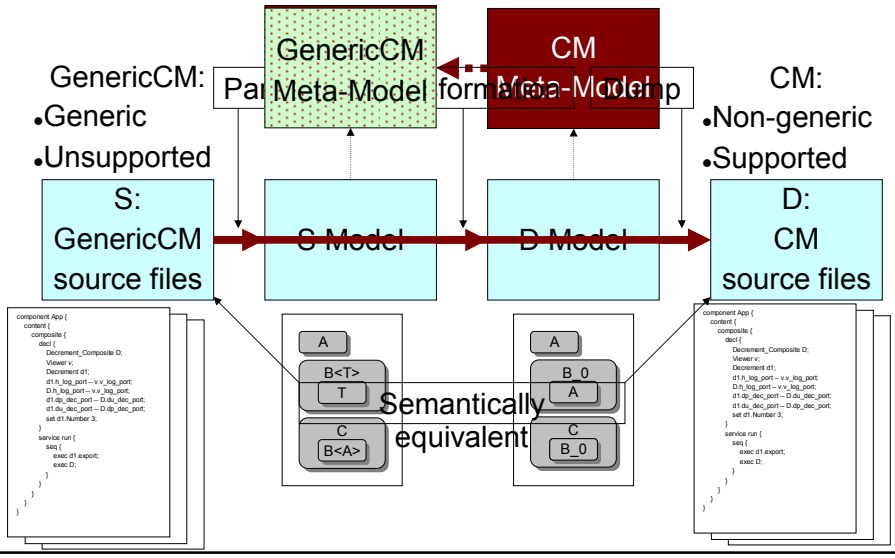
Implementing Generic Components with Model Transformation

Motivating Example: A generic farm



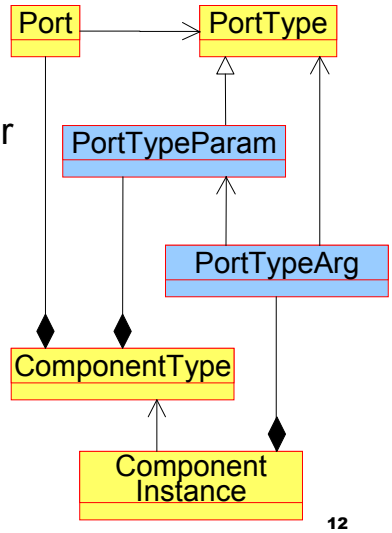
10

Proposed approach:
A Model-Based Text to Text Transformation



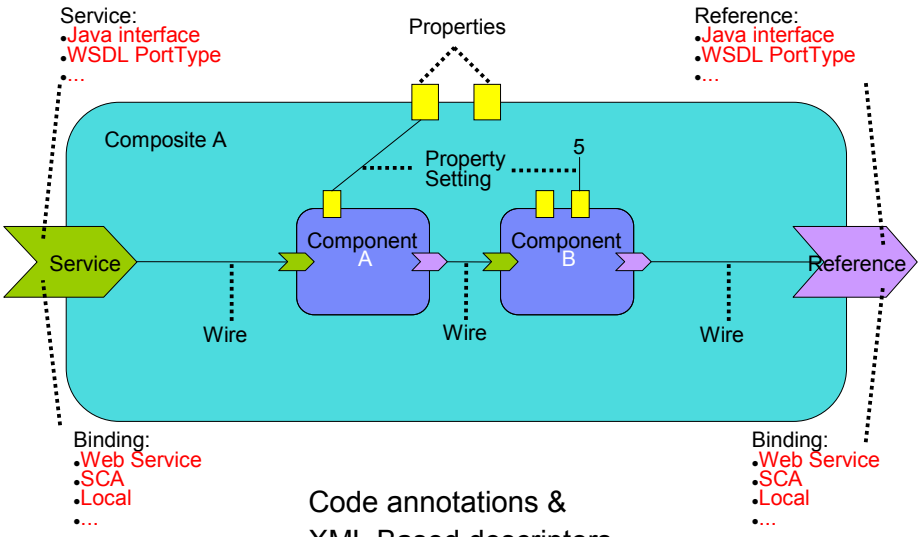
Proposed approach:
Introducing genericity in a meta-model

- Generic Artifact
  - ComponentType
- Artifact usable as Parameter
  - PortType
- Other modifications
  - Default value for parameters
  - Constraints on parameter values
  - Explicit specializations



12

GenericSCA: SCA Overview



Code annotations &
XML Based descriptors

13

GenericSCA: Introduced Features

- Concepts made generic :
  - Composite component implementations
  - Java component implementations
  - Java port interfaces
- Concepts that can be parameters
  - Component implementations
  - Port interfaces
  - Data-types
  - (Data-values) : properties are already part of SCA

14

GenericSCA:The implementation

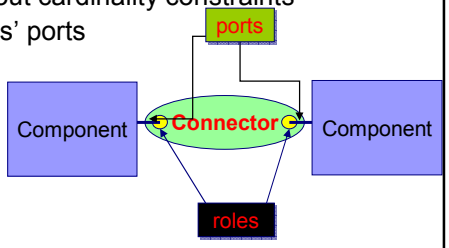
- SCA meta-model
  - Distributed as part of the Eclipse SCA Tool project
  - About 100 annotated meta-classes
  - Autogenerated Model → XML dump
- GenericSCA meta-model
  - SCA meta-model + 18 annotated meta-classes
  - Autogenerated XML → Model parsing
- GenericSCA to Plain SCA transformation
  - QVT (OMG): not mature enough yet
  - Plain java
    - ~750 java lines, mostly copy of attributes
    - ~100 lines for the main logic
    - <<1sec for the Mandelbrot set example

15

Connector-based Composition

Notion of connector

- Introduced in ADL
  - Architecture Description Language
- First class entities
  - List of named roles, with or without cardinality constraints
  - Roles are fulfilled by components' ports
- Instantiated by connection
- Implemented by generator
- Example
  - Connector mpi<role participant>
  - Connector UP<role user
    role provider>
  - Connector consensus<…>

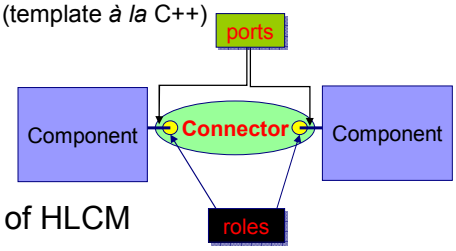


High Level Component Model

Hierarchy, Genericity, Template Meta-Programming & Connectors

High Level Component Model

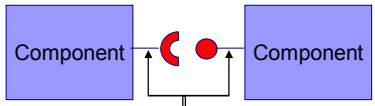
- Major concepts
  - Hierarchical model
  - Generic model
    - Support meta-programming (template *à la* C++)
  - Connector based
    - Primitive and composite
  - Currently static



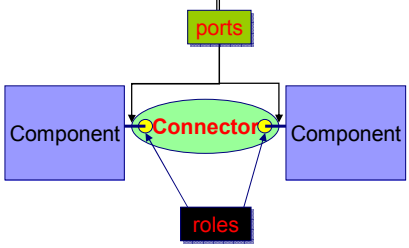
- HLCMi: an implementation of HLCM
  - Model-transformation based
  - Already implemented connectors
    - Use/Provide, Shared Data, Collective Communications, "MxN" RMI, Irregular Mesh

Connectors

- Without connectors
  - Direct connection between ports
  - Limitation to 1-1 connection

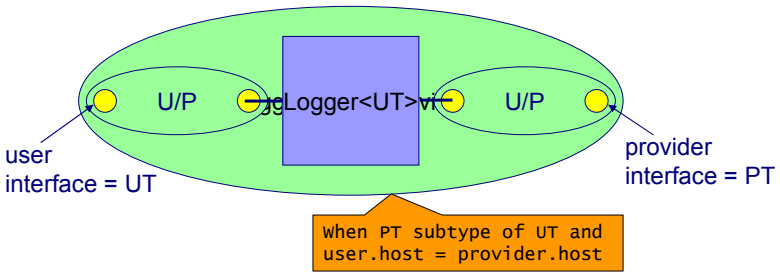


- With connectors
  - Connectors reify connections
    - A name
    - A set of roles
  - Any number of roles
  - Can be 1st class entities
    - Implemented by the user



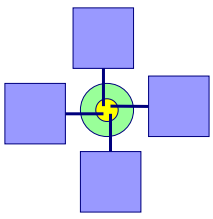
Connector implementations

- Intrinsically generic
  - Types of roles fulfillment ⇔ parameters for the implementation
- 1 connector ⇔ multiple implementations
  - For distinct placement on hardware resources
- Two possible kinds
  - Primitive connectors
    - Directly supported by the model
  - Composite connectors
    - An assembly

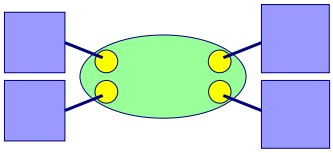


Example of More Complex Interactions as Connectors

- Shared data between components
  - One single role
  - Multiple fulfillments

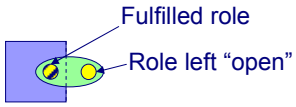


- Parallel method calls
  - Provides the redistribution
  - An example
    - 2x2 Matrix multiplication
    - 2 roles for users (top/bottom)
    - 2 roles for providers (right/left)

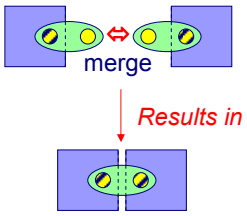


Notion of Open Connections

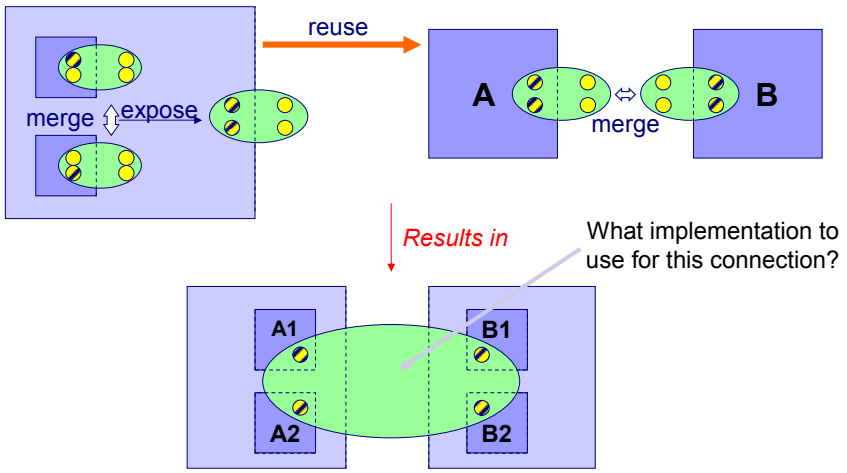
- Components expose “open connections”
  - Some roles fulfilled
  - Some roles left “open”



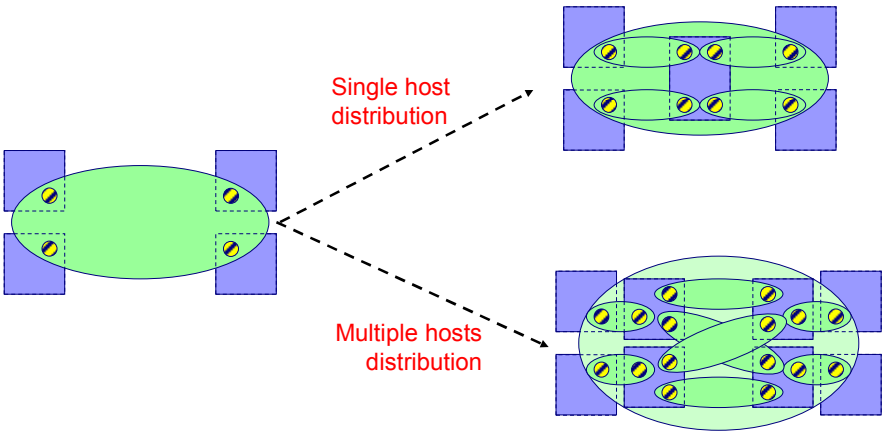
- Interactions are defined by “merging” connections
  - Union of the role fulfillments
  - A single logical connection



Expressing Parallel Matrix Multiplication with HLCM



Connection Implementation: a Planning Choice



Conclusion

- From « simple » to « complex » composition operators
- Need of models with open composition support
  - Component, connector, hierarchy, genericity, etc.
- Need of models/algorithms to derive actual implementation from an abstract declaration
- Need of models/algorithms to support dynamicity
  - Adaptability : reaction to environment modifications
  - « workflow » : reaction to programmed modifications