An AsmL Semantics for Dynamic Structures in UML-RT

Stefan Leue, Alin Ştefănescu, and Wei Wei

Software Engineering Group
University of Konstanz

3–October–2006
Motivation

→ Dynamic reconfigurations are inherent to many applications. Capturing them at design time is not easy.

• the UML-RT profile supports dynamic structures:
  – runtime creation of capsules ⇝ optional capsules
  – multiple containment ⇝ plugin capsules
  – dynamic connections ⇝ unwired ports

• a precise semantics of such concepts is needed for the formal analysis of UML-RT models

We use the AsmL specification framework to give an executable semantics to dynamic structures in UML-RT.
Abstract State Machines (ASM) and AsmL

Abstract State Machines (ASM) used for the semantics

- ASMs are transitions systems with
  - states described by sets with relations and functions
  - transitions described by update rules

- AsmL is a specific implementation for ASMs
  - object-oriented concepts, strong type system, tool support

An executable AsmL specification for UML-RT models:

- semantics of UML-RT specified in AsmL
- translation of a given UML-RT model into AsmL
- analysis of the AsmL specification using the SpecExplorer tool (simulation, model-checking, testing)
UML-RT and Rational Rose Real-Time

supported by Rational Rose Real-Time

In UML 2.0
Structured Classes that may have ports and internal structure
(depicted by Composite Structures diagrams)
Ports and connectors

**class Port**
- name as String
- protocol as Protocol
- isRelay as Boolean
- isWired as Boolean
- isPublic as Boolean
- peerPort as Port or Null

**class ConnectorEnd**
- capsuleRole as CapsuleRole or Null
- port as Port

**class Connector**
- end1 as ConnectorEnd
- end2 as ConnectorEnd

Only ports with **compatible protocols** can be connected.

**Rose RealTime** supports **dynamic connections** of unwired ports.

**In UML 2.0**
**Ports and connectors** were newly introduced in UML 2.0, but with a slightly more general semantics than in UML-RT.
class **CapsuleClass**
- name as String
- super as CapsuleClass
- subcapsuleRoles as Set of CapsuleRole
- ports as Set of Port
- connectors as Set of Connector
- behaviour asStateMachine or Null

class **CapsuleRole**
- name as String
- capsuleClass as CapsuleClass
- parent as CapsuleRole or Null
- kind as **CapsuleRoleKind**
- isSubstitutable as Boolean
- inst as CapsuleInstance or Null

class **CapsuleInstance**
- name as String
- capsuleClass as CapsuleClass
- subcapsules as Set of CapsuleRole
- ports as Set of Port
- importedIn as Set of CapsuleRole
- state as State

with **CapsuleRoleKind** = \{fixed_capsule, optional_capsule, plugin_capsule\}
Dynamic structures in UML-RT

- **fixed capsule role** – incarnated at initialization time of parent
  - In **UML 2.0**: composition (isComposite=true)
  - Graphically: nested box with **solid outline**

- **optional capsule role** – incarnated/destroyed at runtime
  - In **UML 2.0**: composition with **variable multiplicity** for parts

- **plugin capsule role** – filled in at runtime
  - In **UML 2.0**: aggregation (isComposite=false)
  - Graphically: nested box with **dashed outline**
Capsule incarnation

```plaintext
incarnate(capsuleRole as CapsuleRole)

require capsuleRole.inst = null
instance := new CapsuleInstance(...)
createSubcapsuleRoles(instance)
createPorts(instance)
incarnateAllFixedRoles(instance)
connectPorts(instance)
capsuleRole.inst = instance
if capsuleRole.kind = optional  connectPorts(capsuleRole.parent,capsuleRole)
```

In Rose RealTime, optional capsules can be incarnated on different execution threads.
Capsule importation

```javascript
import(instance as CapsuleInstance, pluginRole as CapsuleRole)

require pluginRole.instance = null
require instance <> null
require compatibleCapsuleRoles(pluginRole, instance.capsuleClass)
connectPorts(pluginRole.parent, pluginRole, instance)
pluginRole.inst := instance
add pluginRole to instance.importedIn
```

**Constraint:** A port of the imported capsule instance cannot be simultaneously connected in two different plugin roles.
Capsule deportation and destruction

- Basically, 'undo' of importation and incarnation

- **Deportation** of capsule instance $CI$ from capsule role $CR$:
  - $\rightarrow$ remove all connectors to $CI$ from $CR$.parent
  - $\rightarrow$ remove $CR$ from the set $CI$.importedIn

- **Destruction** of a capsule role $CR$:
  - $\rightarrow$ destroy connections to and in $CR$
  - $\rightarrow$ recursively destroy all owned fixed and optional capsules roles
  - $\rightarrow$ **Semantic variation point**: Destroy or keep the owned subcapsule instances that are imported in another capsule role
Next steps

- integrate the semantics of state machines and communication services (as supported by Rational Rose Realtime tool)
- analysis of generated AsmL code using SpecExplorer (simulation, testing, model checking)
- semantics for the UML 2.0 composite structures

Thank you for attention!