Modeling SW-Architectures using UML-RT/UML 2.0

Ingolf H. Krueger
ikrueger@ucsd.edu
Overview

- UML: Good Enough for Specifying Architectures?
- UML-RT/UML 2.0
  - Overview
  - Capsules
  - Ports and Connectors
  - Protocols
  - Behavior Description
  - Evaluation
- Example: Autonomous Transport System
- Summary and Outlook
Is the UML Good Enough?

- The UML offers a plethora of description techniques for many aspects of software architectures.

- The UML has, however, also significant deficits especially when it comes to modeling complex, service-oriented systems!

- In particular, we miss:
  - An adequate notation for services
  - A non-technical component notion
  - Clear concepts for hierarchy
  - Strong concepts and description techniques for
    - logical component distribution
    - non-technical interfaces
  - Formal means for behavior descriptions with respect to interfaces
Overview

- UML: Good Enough for Specifying Architectures?
- UML-RT/UML 2.0
  - Overview
  - Capsules
  - Ports and Connectors
  - Protocols
  - Behavior Description
  - Evaluation
- Example: Autonomous Transport System
- Summary and Outlook
What is UML-RT?

• Examples of “profiles” of the UML:
  – embedded real-time systems (“UML-RT”)
  – automotive
  – web applications
  – …

• Origin: ROOM [SGW94] + UML

• Focus of UML-RT/ROOM:
  – component-oriented development
  – all components are potentially active units
  – signal-/message-oriented communication
  – time concept
  – quality of service (in preparation)
The Component Model of UML-RT

Description Techniques of UML-RT for Structure and Behavior

- capsules
- ports
- connectors

UML-statecharts

MSCs

structure

behavior
Hierarchical Composition in UML-RT

**Capsule**
- active object or “passive” container
- communication with the environment
  - signal-based (asynchronous message exchange)
  - exclusively via interface objects (ports)
- supports hierarchical composition

**Connector**
- communication link
  - “drives” protocol

**Port**
- concrete realization of an interface
- equipped with a protocol
- list of incoming/outgoing messages/signals
- (signal flow)

**Capsule**
- active object or “passive” container
- communication with the environment
- signal-based (asynchronous message exchange)
- exclusively via interface objects (ports)
- supports hierarchical composition
Example: UML-statecharts

UNLD

h.lock/l.down

wld

l.ready/r.down

wrd

r.ready/h.done

LCKD

r.ready/h.done

wru

l.ready/r.up

wlu

h.unlock/l.up

<<capsule>>

:MotorControl
Signal-Based Communication

- Capsules receive and send **signals** via their **ports**
- Signals, which cannot be processed immediately, are stored in a **queue**
Signalbasierte Kommunikation

- Capsules receive and send **signals** via their **ports**
- Signals, which cannot be processed immediately, are stored in a **queue**
Overview

• UML: Good Enough for Specifying Architectures?
• UML-RT/UML 2.0
  – Overview
  – Capsules
  – Ports and Connectors
  – Protocols
  – Behavior Description
  – Evaluation
• Example: Autonomous Transport System
• Summary and Outlook
Capsules

• Every capsule represents a potentially active object

• Communication between capsule and environment: exclusively via ports
  – no public data
  – no public methods

• Hierarchical decomposition into subcapsules

• Every capsule has (at most) one state automaton describing the capsule’s behavior
  ⇒ capsule is “controller” for its subcapsules
  ⇒ see architectural pattern “recursive control”
Capsules

- Upon its instantiation a capsule builds its internal structure (subcapsules)

- The capsule can change its internal structure over time
  \[\rightarrow\text{Architectural integrity}\]

![Diagram of capsule structure]

<table>
<thead>
<tr>
<th>Name of Capsule</th>
<th>Port Name</th>
<th>Protocol Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>«capsule» CLS</td>
<td>cp</td>
<td>CommandHandler</td>
</tr>
<tr>
<td></td>
<td>mp[2]</td>
<td>Initiator</td>
</tr>
</tbody>
</table>
Overview

- UML: Good Enough for Specifying Architectures?
- UML-RT/UML 2.0
  - Overview
  - Capsules
    - Ports and Connectors
  - Protocols
  - Behavior Description
  - Evaluation
- Example: Autonomous Transport System
- Summary and Outlook
Ports and Connectors

• Port
  – belongs to precisely one capsule
    (the capsule creates and destroys its ports)
  – has identity and state
  – has behavior
  – implements the role of its capsule in a protocol

• Kinds of ports
  – Relay-Ports
    • relay signals between capsules and their subcapsules
    • controlled interface export
  – End-Ports
    • relay signals between capsules and their state automata
    • have queues for signals already received, but not yet processed
Ports and Konnektoren

Simplified Representation

```
«capsule»
CLS

ports

cp : CommandProtocol.Handler;
```
Ports and Connectors

Simplified Representation in Collaboration Diagrams

Port Symbol

Port Name and Protocol-Role
Ports and Connectors

End-Port

Relay-Port

Relay-Port
UML2 Notation
Overview

• UML: Good Enough for Specifying Architectures?
• UML-RT/UML 2.0
  – Overview
  – Capsules
  – Ports and Connectors
  – Protocols
  – Behavior Description
  – Evaluation
• Example: Autonomous Transport System
• Summary and Outlook
Protocols

Example: Simple Communication Protocol

SenderRole

req_start

ack_start

data

ack_data

data

ack_data

req_end

ack_end

ReceiverRole
### Protocols

#### Example: Simple Communication Protocol

<table>
<thead>
<tr>
<th>«protocolRole»</th>
<th>SenderRole</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>incoming</strong></td>
<td></td>
</tr>
<tr>
<td>ack_start</td>
<td></td>
</tr>
<tr>
<td>ack_data</td>
<td></td>
</tr>
<tr>
<td>ack_end</td>
<td></td>
</tr>
<tr>
<td><strong>outgoing</strong></td>
<td></td>
</tr>
<tr>
<td>req_start</td>
<td></td>
</tr>
<tr>
<td>data</td>
<td></td>
</tr>
<tr>
<td>req_end</td>
<td></td>
</tr>
</tbody>
</table>

```
req_start
ack_start
data
ack_data
data
ack_data
req_end
ack_end
```

<table>
<thead>
<tr>
<th>«protocolRole»</th>
<th>ReceiverRole</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>incoming</strong></td>
<td></td>
</tr>
<tr>
<td>req_start</td>
<td></td>
</tr>
<tr>
<td>data</td>
<td></td>
</tr>
<tr>
<td>req_end</td>
<td></td>
</tr>
<tr>
<td><strong>outgoing</strong></td>
<td></td>
</tr>
<tr>
<td>ack_start</td>
<td></td>
</tr>
<tr>
<td>ack_data</td>
<td></td>
</tr>
<tr>
<td>ack_end</td>
<td></td>
</tr>
</tbody>
</table>

```
```
Example: Simple Communication Protocol

<table>
<thead>
<tr>
<th>«protocolRole»</th>
<th>SenderRole</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>incoming</strong></td>
<td></td>
</tr>
<tr>
<td>ack_start</td>
<td></td>
</tr>
<tr>
<td>ack_data</td>
<td></td>
</tr>
<tr>
<td>ack_end</td>
<td></td>
</tr>
<tr>
<td><strong>outgoing</strong></td>
<td></td>
</tr>
<tr>
<td>req_start</td>
<td></td>
</tr>
<tr>
<td>data</td>
<td></td>
</tr>
<tr>
<td>req_end</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>«protocolRole»</th>
<th>ReceiverRole</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>incoming</strong></td>
<td></td>
</tr>
<tr>
<td>req_start</td>
<td></td>
</tr>
<tr>
<td>data</td>
<td></td>
</tr>
<tr>
<td>req_end</td>
<td></td>
</tr>
<tr>
<td><strong>outgoing</strong></td>
<td></td>
</tr>
<tr>
<td>ack_start</td>
<td></td>
</tr>
<tr>
<td>ack_data</td>
<td></td>
</tr>
<tr>
<td>ack_end</td>
<td></td>
</tr>
</tbody>
</table>

“Conjugated” sender role
Protocols

Simplification for Point-to-Point Protocols

<table>
<thead>
<tr>
<th>«protocol» Transmission</th>
</tr>
</thead>
<tbody>
<tr>
<td>incoming</td>
</tr>
<tr>
<td>ack_start</td>
</tr>
<tr>
<td>ack_data</td>
</tr>
<tr>
<td>ack_end</td>
</tr>
<tr>
<td>outgoing</td>
</tr>
<tr>
<td>req_start</td>
</tr>
<tr>
<td>data</td>
</tr>
<tr>
<td>req_end</td>
</tr>
</tbody>
</table>

Base Role for Protocol (Sender View)

Base Role

sp: Transmission

rp: Transmission~

Conjugated (Inverse) Role
Overview

- UML: Good Enough for Specifying Architectures?
- UML-RT/UML 2.0
  - Overview
  - Capsules
  - Ports and Connectors
  - Protocols
    - Behavior Description
    - Evaluation
- Example: Autonomous Transport System
- Summary and Outlook
Behavior Description in UML-RT

- Every capsule that has its own behavior is associated with a UML-statechart
- Max one statechart per capsule
- Hierarchical composition:
  - every sub-capsule can have its own statechart
Doing without AND-states

- Concurrency via separate capsules
- Synchronization via explicit communication
- Result: stronger decoupling
Encapsulation on the Level of States

- States become exchangeable entities
- Helps avoid “stub states”
Overview

- UML: Good Enough for Specifying Architectures?
- UML-RT/UML 2.0
  - Overview
  - Capsules
  - Ports and Connectors
  - Protocols
  - Behavior Description
  - Evaluation
- Example: Autonomous Transport System
- Summary and Outlook
Evaluation

- UML-RT is much better suited for the specification of software architectures and services than “pure” UML:
  - hierarchic component model,
  - precise behavior descriptions
  - interface concept
  - protocols and connectors

- Potentials for improvement (among others):
  - m2m communication instead of p2p
  - association of interaction patterns with ports/connectors
  - methodological guidelines for iterative service development

- Future:
  - (methodological!) treatment of Quality-of-Service aspects
Overview

• UML: Good Enough for Specifying Architectures?
• UML-RT/UML 2.0
  – Overview
  – Capsules
  – Ports and Connectors
  – Protocols
  – Behavior Description
  – Evaluation
• Example: Autonomous Transport System
• Summary and Outlook
Systematic Construction of Reliable SW-Systems

1. Develop/Refine domain model
2. Capture interaction patterns
3. Derive interface specification
   • messages/signals, types
   • behavior
4. Decompose components hierarchically
Example Application: Autonomous Transport System

Holons
- carry workpieces between source and destination
- negotiate for jobs
- hold internal database representing system's status
- process workpieces
- post jobs to be processed ("deliver workpiece")
- negotiate with Holons via broadcasting

OutStorage
- holds workpieces after processing
- holds "production plan/program" (number/kind of workpieces to be processed per day)

InStorage
- holds workpieces yet to be processed

MachineTools
- process workpieces
- post jobs to be processed ("deliver workpiece")
- negotiate with Holons via broadcasting
Example Application: Autonomous Transport System

Architectural Aspects:

- components
- interfaces/behavior
- hierarchy/decomposition

- p2p communication
- broadcasting
Architectural Pattern for Broadcasting

HTS

Disponent

Database

SingleJobControl

HTS_IOSSystem

ProdSys

InStorage

HTS

OutStorage

MachineTool

BroadcastSystem
Sequence Charts for Broadcasting

- **m: Machine Tool**
- **h: HTS**
- **l: HTS**

**Signal Exchange**

- `requestWP()`
- `releaseWP()`
- `jTransporting(jobno)`

**Component Axis**

**Local Action**

- `drive to location 1`
- `update JobStatus()`
- `update JobStatus()`

**Broadcasting**
Sequence Charts for Broadcasting

- `jOrder(jobno)`
- `compute bid`
- `jBid(jobno, h)`
- `jBid(jobno, l)`
- `jEndOfNegotiation(jobno)`
- `store job`
Derivation of Component Structure

Captured scenarios & domain model indicate:

- active vs. passive components
- Point-to-point communication requirements
- broadcasting requirements

```
ProdSys
  InStorage
  OutStorage
  HTS  n
  MachineTool  n
BroadcastSystem
```
Captured scenarios indicate also:

- names and types of signals
- ordering of signal flow
Overview

- UML: Good Enough for Specifying Architectures?
- UML-RT/UML 2.0
  - Overview
  - Capsules
  - Ports and Connectors
  - Protocols
  - Behavior Description
  - Evaluation
- Example: Autonomous Transport System
- Summary and Outlook
Summary and Outlook

• Modeling “in the real world”:
  often – if at all – done using UML/UML-RT/UML 2.0

• UML-RT/UML 2.0 better equipped for modeling software architectures than UML versions < 2.0

• Starting point for component- and service-oriented development: domain model, interaction scenarios

• How to avoid over-modeling and over-engineering?