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)

Read: UML with component notion
The Component Model of UML-RT

Description Techniques of UML-RT for Structure and Behavior

- Structure:
  - capsules
  - ports
  - connectors

- Behavior:
  - UML-statecharts
  - MSCs
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

**Port**
- concrete realization of an interface by means of an object
- equipped with a protocol
  - list of incoming/outgoing messages/signals
  - (signal flow)
- the aid for “encapsulation” and “separation of concerns”

**Connector**
- communication link between ports
- “drives” protocol
Example: UML-statecharts

```
Example: UML-statecharts

wld     l.ready/r.down

h.lock/l.down  wld

wrd     r.ready/h.done

r.ready/h.done  wru

LCKD

wru     l.ready/r.up

UNLD

h.unlock/l.up  wlu

h.ready/h.done

wlu

l.ready/r.down```
Signal-Based Communication

- 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 sub-capsules

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

- Upon its instantiation a capsule builds its internal structure (sub-capsules)

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

![Diagram of capsule structure]
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 sub-capsules
    • 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 Connectors

Simplified Representation

```
«capsule»
CLS

ports

cp : CommandProtocol.Handler;
```

Port Name

Protocol-Role
(qualified with protocol name)
Ports and Connectors

Simplified Representation in Collaboration Diagrams

```
«capsule» :CommandSource

lm:Motor

mp[1]: MotorProtocol.Initiator

cp:CommandProtocol.Handler

mp[2]: MotorProtocol.Initiator

rm:Motor

Port Symbol

Port Name and Protocol-Role
```
Ports and Connectors

End-Port

Relay-Port

Relay-Port

«capsule»
:CLS

«capsule»
:CommandHandler

«capsule»
:Motor

«capsule»
:MotorControl

«capsule»
:Motor

«capsule»
:Motor
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

```
<<protocolRole>>
SenderRole

incoming
ack_start
ack_data
ack_end

outgoing
req_start
data
req_end

<<protocolRole>>
ReceiverRole

incoming
req_start
data
req_end

outgoing
ack_start
ack_data
ack_end
```
Protocols

Example: Simple Communication Protocol

```
«protocolRole»
SenderRole

incoming
ack_start
ack_data
ack_end

outgoing
req_start
data
req_end

«protocolRole»
ReceiverRole

incoming
req_start
data
req_end

outgoing
ack_start
ack_data
ack_end

“Conjugated” sender role
```
Protocols

Simplification for Point-to-Point Protocols

<table>
<thead>
<tr>
<th>Role</th>
<th>Transmission</th>
</tr>
</thead>
<tbody>
<tr>
<td>incoming</td>
<td>ack_start, ack_data, ack_end</td>
</tr>
<tr>
<td>outgoing</td>
<td>req_start, data, req_end</td>
</tr>
</tbody>
</table>

Base Role for Protocol (Sender View)

Base Role

sp: Transmission

Conjugated (Inverse) Role

rp: Transmission~
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
Behavior Description in UML-RT

Doing without AND-states

- Concurrency via separate capsules
- Synchronization via explicit communication
- Result: stronger decoupling
Behavior Description in UML-RT

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 with OutStorage and MachineTools for jobs
  - hold internal database representing the system’s status

- **InStorage**
  - holds workpieces yet to be processed

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

- **MachineTools**
  - process workpieces
  - post jobs to be processed (“deliver workpiece“)
  - negotiate with Holons via broadcasting

- **OutStorage**
  - holds workpieces after processing
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

n

OutStorage

MachineTool

BroadcastSystem
Sequence Charts for Broadcasting

m: MachineTool
h: HTS
l: HTS

drive to location 1

requestWP()
releaseWP()

jTransporting(jobno)

update JobStatus()
update JobStatus()

component axis

local action

signal exchange

broadcasting

time
Sequence Charts for Broadcasting

state label

jOrder(jobno)

compute bid

jBid(jobno, h)

jEndOfNegotiation(jobno)

store job

h:HTS

waiting

jBid(jobno, l)

l:HTS

waiting

waiting

waiting
Derivation of Component Structure

Captured scenarios & domain model indicate:

- active vs. passive components
- Point-to-point communication requirements
- broadcasting requirements
Derivation of Interface Behavior

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?