Towards an open source MDE tool infrastructure for the Internet of Things

Juergen Dingel

EXE 2017
3rd International Workshop on Executable Modeling
September 18, 2017, Austin, Texas
co-located with MODELS 2017
The Internet of Things (IoT)

Technology for collection, aggregation, and analysis of data from a range of devices to optimize operation of a system in different domains, including buildings, traffic, health care, energy, business, industry.

- And, please, don’t forget to buy milk again!
- You should walk these short distances in the future.
- Can you turn me around? I look fuller from the other side.
**IoT: Core Characteristics**

- timed
- reactive
- concurrent
- distributed
- open
- heterogenous
- self-optimizing
- context-aware
- autonomous
- adaptive
- large scale
- available
- reliable
MDE

- MDE = notations, techniques, tools to leverage abstraction and automation for system development

- Examples for abstraction and automation
  - Virtual memory [Denning 1970]
  - Internet Protocol [Cerf 2017]

- Examples for MDE
  - Robotics software [SPARC 2016]
  - Industrial DSLs (e.g., at Ericsson)
  - Game development (e.g., in Unity)

Overview of Talk

timed reactive concurrent
distributed
open heterogenous
self-optimizing context-aware autonomous adaptive
large scale available reliable

UML-RT with Papyrus-RT
MQTT
Connecting Debugging Changing @ runtime
monitoring animation steering adaption runtime verification

EXE'17
Goal of Talk

- Inform
  - MDE with UML-RT and Papyrus-RT and extensions

- Inspire
  - Use, extend, participate

Open Source!
Background

MDE with UML-RT and Papyrus-RT

Overview of Talk

- timed
- reactive
- concurrent
- distributed
- open
- heterogenous
- self-optimizing
- context-aware
- autonomous
- adaptive
- large scale
- available
- reliable

UML-RT with Papyrus-RT
UML-RT: History

- Real-time OO Modeling (ROOM)
  - ObjecTime, early 1990 ties
- Major influence on UML 2
  - E.g., StructuredClassifier
- “RT subset of UML”
- Tools
  - ObjecTime Developer
  - IBM Rational RoseRT
  - IBM RSA-RTE
  - Protos ETrice
  - Eclipse Papyrus-RT

**UML-RT: Characteristics**

- **Domain-specific**
  - Embedded systems with soft real-time constraints
- **Graphical, but textual syntax exists**
- **Small, cohesive set of concepts**
- **Strong encapsulation**
  - Actors (active objects)
  - Explicit, typed interfaces
  - Message-based communication
- **Event-driven execution**
  - State machines
- **Lots of analysis opportunities**

> “UML-RT has features that appeal to the formalist”
> [Whittaker et al 2000]


J. Dingel  
EXE’17
UML-RT: Core Concepts (1)

- **Types**
  - Capsules (active classes)
    - Capsule instances (parts)
  - Passive classes (data classes)
    - Objects
  - Protocols
  - Enumerations

- **Structure**
  - Attributes
  - Ports
  - Connectors

- **Behaviour**
  - Messages (events)
  - State machines

- **Grouping**
  - Package

- **Relationship**
  - Generalization
  - Associations
UML-RT: Core Concepts (2)

- **Model**
  - Collection of *capsule* definitions
  - ‘Top’ capsule containing collection of parts (capsule instances)

- **Capsules**
  - May contain
    - Attributes, ports, or other parts
  - Behaviour defined by *state machine*

- **Ports**
  - Typed over *protocol* defining input and output messages

- **State machine**
  - Transition triggered by incoming messages
  - Action code can contain send statements that send messages over certain ports
Capsules (1)

- Kind of active class
  - Attributes, operations
  - Own, independent flow of control (logical thread)
- May also contain
  - Ports over which messages can be sent, received
  - Parts (instances of other capsules) and connectors
- Creation, use of instances tightly controlled
  - Created by runtime system (RTS)
  - Cannot be passed around
  - Stored in attribute of another capsule (part)
  - Information flow only via messages sent to ports
    \[\Rightarrow\] better concurrency control and encapsulation
- Behaviour defined by state machine
Protocols

- Provide types for ports
- Define
  - Input messages
    - Services provided by capsule owning port
  - Output messages
    - Services required by capsule owning port
  - Input/output messages
- Messages can carry data
Ports

- “Boundary objects” owned by capsule
- Typed over a protocol P
- Have ‘send’ operation
  
  \[
  \text{portName.msg(arg1,...,argn).send()}
  \]
- E.g., in \texttt{Pinger}
  
  \[
  \text{pingPort.ping().send()}
  \]
Ports: External, Internal, Relay

- **External behaviour**
  - Provides (part of) externally visible functionality \((\text{isService}=\text{true})\)
  - Incoming messages passed on to state machine \((\text{isBehaviour}=\text{true})\)
  - Must be connected \((\text{isWired}=\text{true})\)

- **Internal behaviour**
  - As above, but not externally visible \((\text{isService}=\text{false})\)
  - Connect state machine with a capsule part

- **Relay**
  - Pass external messages to and from capsule parts
Ports: System

- Connects capsule to Runtime System (RTS) library via corresponding system protocol
- Provides access to RTS services such as
  - **Timing**: setting timers, time out message
    - `timer2Port.informIn(UMLRTTimespec(10, 0));`
      // set timer that will expire in 10 secs and 0 nanosecs
    - When timer expires, 'timeout' message will be sent over `timer2Port`
  - **Log**: sending text to console
    - `logPort.log("Ready to self-destruct")`
Ports: SPP and SAP

- So far, only **wired ports**
  - Connected automatically when instances are created

- **Unwired ports**
  - Original intent: ‘layered’ design
  - Connected at run-time
    - Port on provider: Service Provision Point (SPP)
    - Port on user: Service Access Point (SAP)
    - Register with RTS using unique service name (manually or automatic)
Example: Ping Pong
Example: Ping Pong

```java
// Protocol definition
protocol PingPongProtocol {
  out ping()
  in pong()
}

// Source code
Controller "DefaultController"
Pinger: ready
Pinger: sending ping
Ponger: ready
Ponger: receiving ping
Pinger: sending pong
Pinger: receiving pong
Ponger: sending pong
Ponger: receiving pong
Pinger: sending pong
Pinger: receiving pong
Ponger: sending pong
Ponger: receiving pong
Pinger: sending pong
Pinger: receiving pong
Ponger: sending pong
Ponger: receiving pong

log.log("Pinger: received pong");
log.log("Pinger: sending pong");
pingPort.ping().send();

// Diagram

log.log("Pinger: ready");
log.log("Pinger: sending ping");
pingPort.ping().send();

// Diagram

log.log("Ponger: received ping");
log.log("Ponger: sending pong");
pongPort.pong().send();

// Diagram

log.log("Ponger: ready");
```
Papyrus-RT: Overview

- **Papyrus for Real-Time** industrial-grade, complete modeling environment for the development of complex, software intensive, real-time, embedded, cyber-physical systems.

- **Part of PolarSys**
  - Eclipse Working Group
  - Open source for embedded systems

- **Building on**
  - Eclipse Modeling Framework (EMF), Xtext, Papyrus

- **History**
  - 2015: V0.7.0
  - March 2017: v0.9
  - Fall 2017: v1.0

[https://wiki.eclipse.org/Papyrus-RT](https://wiki.eclipse.org/Papyrus-RT)
Resources: UML-RT and Papyrus-RT

- **UML-RT**
  - Papers:
  - Tutorials:

- **Papyrus-RT**
  - Distribution: [https://eclipse.org/papyrus-rt](https://eclipse.org/papyrus-rt)
  - Wiki: [https://wiki.eclipse.org/Papyrus-RT](https://wiki.eclipse.org/Papyrus-RT)
  - Overview: [https://www.youtube.com/watch?v=UqefL7-ZPYo](https://www.youtube.com/watch?v=UqefL7-ZPYo)

21 EXE'17
Ongoing Research 1

Connecting

Overview of Talk

- timed
- reactive
- concurrent
- distributed
- open
- heterogenous
- self-optimizing
- context-aware
- autonomous
- adaptive
- available
- reliable
- large scale

UML-RT with Papyrus-RT

MQTT

monitoring
animation
steering
adaption
runtime verification
From Isolated to Connected

Model M

C/C++
cgen

Animation 1
C/C++
cgen
Animation 2
Runtime Verification
Monitoring, learning, adapting
MQTT
Simulation
System Component 1
System Component 2

J. Dingel
EXE'17
Mechanism 1: Gateway Capsule

- **Using SAP/SPP**
  - Protocol defines incoming/outgoing messages
  - Automatic registration

- **Bi-directional**
  - Incoming messages can trigger transitions

---

![Diagram showing bi-directional communication between a traffic light and an observer, with an external tool as a communication channel.](image-url)
Gateway Capsule: Example

```java
Event e1;
e1.setParam("id", "top");
e1.setParam("color", "#ef0404");
e1.setParam("cmd", "set_color");
observation.event(e1).send();
timer.informIn(UMLRTTimespec(5,0));
log.show("Switched to red")
```

![State diagram of a traffic light](image)

![Java code snippet](image)

![UML diagram](image)

![File structure](image)
Gateway Capsule: Examples

- Monitoring and steering
  - Parcel routing system
  - https://www.youtube.com/watch?v=EbMIgEX9O58

- Animation and simulation using Unity

[Diagnostics courtesy Michal Pasternak]
Mechanism 2: MQTT

- Message Queue Telemetry Transport (MQTT)
  - Publish/subscribe protocol
  - Light-weight, low resource requirements
  - Easy to use:
    - (dis-)connect, (un-)subscribe, publish
  - Standardized

- Implementations
  - E.g., Eclipse Paho

- Brokers
  - E.g., Eclipse Mosquitto

<table>
<thead>
<tr>
<th>Topic</th>
<th>Subscribers</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Temperature/bedroom”</td>
<td>Component 2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
MQTT Support in Papyrus-RT

- **In model**
  - Subscribe(t) in capsule C
    - register unwired port of C as SAP under name t
  - Publish(t,m) in capsule C
    - send m to port of C associated with t

- **RTS**
  - Maintains connection to broker(s) and topic/broker table
  - Sends published messages to corresponding broker(s)
  - Periodically checks brokers for incoming messages
  - Sends incoming message m to port associated with m
Ongoing Research 2

Debugging

Overview of Talk

- timed
- reactive
- concurrent
- distributed
- open
- heterogenous
- self-optimizing
- context-aware
- autonomous
- adaptive
- large scale
- available
- reliable

Connecting

MQTT
- monitoring
- animation
- runtime verification

UML-RT with Papyrus-RT

Debugging

steering
adaption
Model Debugging: Two Approaches

1. Interpretation
   - Pros
     - Easier to integrate into MDE environment
   - Cons
     - Two semantics: Interpreter vs code generator
     - Two platforms: Modeling platform vs target platform

2. Executing generated code on target platform
   - Pros
     - One semantics, one platform
   - Cons
     - How to implement?
Existing Approaches

Models
- Debug Command or Data at Model Level
  - Mapping

Code
- Debug Command or Data at Code Level
  - Mapping

Target Platform
- Attach
  - Binary with Debugging Symbols
  - Program Debugger
  - Debug Data

Languages
- C++
- JAVA
- Others

Platforms
- GDB
- DBG

[Diagram courtesy Mojtaba Bagherzadeh]

Consequences?
**Our Approach**

**Key idea:** Use model transformation to enrich model to allow it to support debugging operations:

- Execution stop and resume (breakpoints),
- variable access,
- collection of execution traces
Example: Ping Pong

[Diagram courtesy Mojtaba Bagherzadeh]
Transformation of Structure
Transformation of Behaviour

User-defined model

Instrumented model

[Diagram courtesy Mojtaba Bagherzadeh]
Overview

Pre runtime

At runtime

[Diagram courtesy Mojtaba Bagherzadeh]
Example: Command Line Interface

List running capsules:
#list

Step execution
#next -c capsule1

View last 10 events
#view -c capsule1 -n 10

Generate sequence diagram:
#seq -c capsuleName

[Diagram courtesy Mojtaba Bagherzadeh]
Evaluation

- Instrumentation time
  ~40sec for model with 400 transitions

- Program size
  comparable with existing approaches

- Runtime performance overhead
  microseconds per transition
Resources: Model-level Debugging

- **Paper**

- **Videos**
  - CLI: [https://www.youtube.com/watch?v=UJ4BYSOrTOQ](https://www.youtube.com/watch?v=UJ4BYSOrTOQ)
  - GUI: [https://www.youtube.com/watch?v=PvPbV5QkQ9Y&t=8s](https://www.youtube.com/watch?v=PvPbV5QkQ9Y&t=8s)

- **Code with tutorial**
  - [https://github.com/emoji1/MDebugger](https://github.com/emoji1/MDebugger)

- **Virtual Box image**
  - [https://github.com/emoji1/MDebugger](https://github.com/emoji1/MDebugger)
Ongoing Research 3

Changing @ runtime

Overview of Talk

- timed reactive concurrent
- distributed
- open heterogeneous
- self-optimizing context-aware autonomous adaptive
- large scale available reliable
- UML-RT with Papyrus-RT
- Connecting
  - monitoring animation
  - steering adaptation
- Debugging
- Changing @ runtime
- MQTT runtime verification
Supporting Modifications at Runtime

- A.k.a., “hot patching/loading”, dynamic software updating
- As in, e.g., Erlang, Java Hotswap, Unreal engine, MS VS Recode
Supporting Modifications at Runtime (Cont’d)

- Use shared, dynamically loaded and linked objects
  - Compile dynamically modifiable capsules into shared objects
  - Whenever capsule changes,
    - recompile and relink, and
    - transfer state

- Challenge
  - State transfer can lead to inconsistencies

- Demo
  - https://youtu.be/FrJm9NTR-bc

- Ongoing
  - Minimizing inconsistencies
  - Roll back
Ongoing Research 4

Demonstrator: PolarSys Rover
- **PolarSys Rover**
  - 2 motors, motor controller
  - Line sensor, ultrasonic detection sensor, camera
  - [https://www.polarsys.org/projects/polarsys.rover](https://www.polarsys.org/projects/polarsys.rover)

- **Raspberry Pi 3 Model B**
  - 1.2GHz 64-bit Quad-core, 1GB RAM
  - WLAN, Bluetooth, 4 USB, HDMI, Ethernet

[UML-RT slides courtesy of Nicolas Hili]

https://www.youtube.com/watch?v=2kLhRUHGLB4
Conclusion
Conclusion

MDE with UML-RT and Papryrus-RT

+ Connecting (gateway, MQTT)
+ Debugging
+ Change at RT
+ Distribution
+ Deployment (MARTE)
+ Schedulability
+ Testing
+ ?

= Open source MDE tool infrastructure for (certain kinds of) IoT applications
Resources

Acknowledgements

- Nicolas Hili, PDF
- Mojtaba Bagherzadeh, PhD
- Karim Jahed, PhD
- Reza Ahmadi, PhD
- Michal Pasternak, MSc
- Harshith Vasanth Gayathri, MSc
- Sudharshnan Gopikrishnan, MSc
Thank you ...  

... for your attention

Hey you shrub, do you always have to have the last word?