TTool Training

II. The TURTLE Profile

Ludovic Apvrille
ludovic.apvrille@telecom-paris.fr
Eurecom, Office 223
I. Introduction

- UML Profile
- The TURTLE Profile
- Design with TURTLE
- Analysis with TURTLE
- Deployment with TURTLE
UML Profiles

- UML Profiles are defined as a formal part of the UML 1.4 specification

- Specific way to define the use of the UML
  - Subset of the UML model elements,
  - Specializations of UML concepts,
  - Limitations and specific requirements for the used concepts,
  - Extra (meta)attributes that can be added to the UML models

- Must be defined within a metamodel
UML profiles: Understanding Diagrams

• Common syntax: UML syntax
  - *Except for new elements*

• But various semantics

• Various way of making these diagrams
  - *Methodology*
    - RUP
    - ROPES
    - etc.
Profile for Performance, Scheduling and Time

- Profile defined at the OMG
- Addresses more specifically real-time systems

Rose RT Profile

- Toolkit
  - Capsules
  - Ports
  - Protocols
  - Communication channels

- Methodology
  - RUP

TAU G2

- Toolkit based on UML 2.0 elements issued from SDL
- Methodology
I. Introduction

- UML Profile
- The TURTLE Profile
- Design with TURTLE
- Analysis with TURTLE
- Deployment with TURTLE
Context

- Design of real-time embedded system is complex
  - Equipments’ heterogeneity
  - Functionalities to offer are more and more complex

- Actual methodologies
  - Are informal (e.g. UML)
    - No formal validation
  - Take into account a limited amount of constraints
    - Real-time constraints

- Formal methods
  - Hardly no industrial use
Propositions

- **Idea:** let us enrich UML
  - *UML operators are informal*
  - *UML lacks advanced temporal operators such as time intervals*
  - *UML has no methodology (no validation)*

- **Proposition:** Semi-formal UML-based environment
  - *Semantics given by mapping to a Formal Description Technique*

- **What formal language?**
  - *Well-defined formal semantics*
  - *Logical and temporal operators*
  - *Tools*

=> TURTLE UML profile (Timed UML and RT-LOTOS Environment)
Methodology

Translation to RT-LOTOS

Validation (RTL)
- Intensive simulation
- Reachability analysis
- Formal proofs

Detailed design
- Activity diagram
  (Behaviour)

Design
- Class diagram
  (Architecture)

Analysis
- Scenarios
**TURTLE: Comparison with UML 1.5**

### UML 1.5

- **Class diagram**
  - Parallelism is implicit
  - Associations = documentation

- **Behavior diagram**
  - Operation calls
  - Delay with pre-determined duration

- **Industrial tools**
  - Implementation-oriented simulation
  - Sequence diagram based testing

### TURTLE

- **Extended class diagram**
  - Explicit parallelism
  - Explicit association between classes (parallelism, synchronization through gates, etc.)

- **Extended activity diagrams**
  - Data sending/receiving on gates
  - Advanced temporal operators
    - Time intervals

- **Tools**
  - TTool + RTL + Aldebaran / CADP
  - Generation of reachability graphs
Chronology of TURTLE

1999
- First definition of operators

2000 - 2001
- Definition of a methodology supporting validation
- Modeling and translation rules
- Translation from TURTLE to RT-LOTOS partially implemented

2002
- New operators (temporal operators, new diagrams)
- Methodological extensions

2003
- First release of the TURTLE toolkit (Ttool)

2004
- TURTLE 2.0
  - UML 2.0-based extensions

2005
- TURTLE analysis
- TURTLE deployment
- Code generation
  - Java
Labs and People Involved in TURTLE

- LAAS / CNRS
  - Jean-Pierre Courtiat

- ENSICA
  - Pierre de Saqui-Sannes

- Concordia University
  - Ferhat Khendek

- ENST
  - Ludovic Apvrille

- ENST Bretagne
  - Christophe Lohr

- Alcatel Space Industries
  - Thesis
References

Definition of the profile


Use of the profile

Online Documentation

http://labsoc.comelec.enst.fr/turtle/HELP/

- Installing TTool
- Using TTool
- Examples of TURTLE modeling
I. Introduction

- UML Profile
- The TURTLE Profile
- Design with TURTLE
- Analysis with TURTLE
- Deployment with TURTLE
A TURTLE Design

Class diagram
- Architecture of the system
  - Instances
    - Tclasses
    - Tobjects
  - Relations between these classes / objects

Activity diagram
- Behavior of classes
Tclasses and Gates
Example of Tclasses

```
TrainingTTYool
+ counter = 4 : Natural;
- test : Boolean;
+ hello : Gate;
- bye : OutGate;
```

```
TrainingTTYool
+ counter = 4 : Natural;
- test : Boolean;
+ hello : Gate;
- bye : OutGate;
```

Ludovic Apvrille - UML - 2005. Slide #18
Relations between Tclasses: TURTLE’s Composition Operators

- **Default relation**
  - Parallel

- **Communication relations**
  - Synchro
  - Invocation
  - Note: Tclasses exchange information exclusively through communication gates

- **Others**
  - Sequence
  - Preemption

- There can be only one composition relation between two tclasses
Parallel Composition Operator
Synchronization Composition Operator

- Synchronization between 2 gates of two different tclasses
- Data can be exchanged when synchronization occur
- A synchronization gate can be involved in only one synchronization relation
- For example, let’s assume that T1.g1 is synchronized with T2.g2
  - g1 can synchronize with g2
  - g1!1 can synchronize with g2?x:nat
  - g1!x1 can synchronize with g2!1
  - g1!x1?y1:nat can synchronize with g2?x2:nat!y2
Synchronization Composition Operator

double click!
Sequence Composition Operator

Semantics

- $T1 \rightarrow seq \rightarrow T2$ means that $T2$ executes once $T1$ has terminated its execution
  - A new instance of $T2$ is executed

Note: the association must be directed to the created instance

no “start”!
Sequence Composition Operator (Cont.)

- Note: T2 on previous slide had no “start”
- If T2 has a “start”
  - **When the system is started**
    - An instance of T1 is started
    - An instance of T2 is started
    - There is no relation between these two instances -> they execute in parallel
  - **Once T1 has terminated**
    - Another instance of T2 is started
    - There is no relation between the two instances of T2
Semantics

- $T1 \rightarrow_{preempt} T2$ means that, when $T2$ can perform one of its first action, $T1$ is terminated and $T2$ executes

Note: the association must be directed from the preempted instance to the executed one

No “start”!
Invocation Composition Operator

- **Modeling of an operation call**
  - *Caller is suspended until the callee unblocks it*
    - Operation call
    - Return from operation call

- **Data can be exchanged**
  - *From the caller to the callee when the operation call is performed*
  - *From the callee to the caller when returning from operation call*

- **Example: a basic calculator**
  - *Experimentation with your first activity diagram!*

Ludovic Apvrille - UML - 2005. Slide #26
Invocation Composition Operator: Example
Invocation Composition Operator: Example

T1

1. \text{subtractRequest}(5, 7) \rightarrow \text{result}
2. \text{printResult}(\text{result})

Calculator

1. \text{subtract}(x: \text{nat}, y: \text{nat})
2. \text{result} = x - y
3. \text{subtract}(\text{result})
4. \text{printResult}(\text{result})
Activity Diagrams

- An activity diagram must be provided for each Tclass
- TURTLE activity diagrams extend UML activity diagrams with two main features
  - Synchronization operators
  - Temporal operators
Activity diagrams: Logical and Temporal Operators
TURTLE Types

### Boolean
- `not :bool->bool`
- `and :bool,bool->bool`
- `or :bool,bool->bool`

### Natural
- `+ :nat,nat->nat`
- `- :nat,nat->nat`
- `* :nat,nat->nat`
- `min :nat,nat->nat`
- `max :nat,nat->nat`
- `< :nat,nat->bool`
- `> :nat,nat->bool`
- `<= :nat,nat->bool`
- `>= :nat,nat->bool`
- `== :nat,nat->bool`
- `div :nat,nat->nat`
- `mod :nat,nat->nat`
- `divs :nat,nat->nat`
Example: Enhancing the Calculator

- The calculator must be able to perform several operations
  - Subtract operation on subtract gate
  - Add operation on addition gate
- Subtract and Add can be performed at the same time
- Two subtract operations cannot be performed at the same time
- Two add operations cannot be performed at the same time
  - T1 makes subtract operations
  - T2 make add operations
- An add operation takes between 5 and 6 time units
- A subtract operation takes exactly 10 time units
- Model T1, T2 and Calculator
Enhancing the Calculator: Class Diagram

T1
<<start>>
+ result : Natural;
- printResult : Gate;
+ subtractRequest : Gate;

Calculater
<<start>>
+ x : Natural;
+ y : Natural;
+ result : Natural;

Invocation

T1.subtractRequest = Calculator.subtract

T2
<<start>>
+ result : Natural;
- printResult : Gate;
+ addRequest : Gate;

Invocation

T2.addRequest = Calculator.add
Enhancing the Calculator: Activity Diagrams
Using Operators of Activity Diagrams

\[ \text{action1} \xrightarrow{\text{delay1}} \text{action2} \xrightarrow{\text{latency2}} \text{delay2} \]

\[ x \xrightarrow{\text{[]}} \text{action1} \rightarrow \text{action3} \]
\[ y \xrightarrow{\text{[]}} \text{action2} \rightarrow \text{action3} \]

\[ x \xrightarrow{\text{[action3]}} \text{action1} \rightarrow \text{action3} \]
\[ y \xrightarrow{\text{[action3]}} \text{action2} \rightarrow \text{action3} \]

\[ \text{[a<5]} \xrightarrow{\text{[]}} \text{action1} \rightarrow \text{action3} \]
\[ \text{[a>2]} \xrightarrow{\text{[]}} \text{action2} \rightarrow \text{action3} \]

\[ \text{[>]} \xrightarrow{\text{[]}} \text{action1} \rightarrow \text{action3} \]
\[ \text{[>]} \xrightarrow{\text{[]}} \text{action2} \rightarrow \text{action3} \]
Advanced Concepts on Composition Operators

- Use of composition operators might be ambiguous
  - Instances created at startup
    - “start” stereotype
    - For each tclasses pointed out by preemption relations
  - Instances created at run time
    - Sequence relations
  - On which instances exactly are applied those composition operators?

- Problematic
  - Multiple compositions operators
  - Priorities between composition operators
  - Tinstances vs. Tclasses

- Examples on next slides!
Multiple Preemption Relations
Priorities of Composition Operators
Use of Multiple Sequence Operators

Diagram showing interactions between different sequence operators and synchronization points.
Use of Multiple Sequence Operators (Cont.)

T4
<<start>>
+ g4 : Gate;
+ g4bis : Gate;

T1
<<start>>
+ g1 : Gate;

T2

+ g2 : Gate;

{ T4.g4 = T1.g1 }

Sequence

Synchro

{ T4.g4bis = T2.g2 }
Using Tobjects instead of Tclasses

- TURTLE Class diagram
  - *Describe the static architecture of the system under design*
  - *But: describe also the dynamics of the systems -> notion of instances*

- For describing one instance of a Tclass -> use of a tclass

- For describing several instances of the same tclass -> use of tobjects

- Example on next slides!
Use of Tobjects: Example

- Webserver having several clients
- Clients can connect to the web server
- Each client can be distinguished with an identifier
- Request of clients are conveyed through a medium
- Modeling of the system: 3 clients, a webserver, and a medium
Webserver: Activity Diagram (WebClient)
Webserver: Activity Diagram (Medium and Webserver)
Advanced Data types: Tdatas

- TURTLE supports two types
  - Natural
  - Boolean

- Data structures: Tdatas!
  - Set of Natural and Boolean

- Using Tdatas
  - Declared as other attributes
  - Used as in C language
    - c.field1 = 5
Example on Tdatas

send!pdu if client has an attribute names pdu of type PDU
I. Introduction

- UML Profile
- The TURTLE Profile
- Design with TURTLE
- Analysis with TURTLE
- Deployment with TURTLE
Methodology

Translation to RT-LOTOS

Validation (RTL)
- Intensive simulation
- Reachability analysis
- Formal proofs

Detailed design
- Activity diagram
  (Behaviour)

Design
- Class diagram
  (Architecture)

Analysis
- Scenarios

Synthesis
A TURTLE Analysis

- **Purpose**
  - *Exemplify very basic scenarios*
  - *Nominal scenarios*
  - *Error scenarios*

- **Interaction Overview Diagram**
  - *Linking between scenarios*

- **Sequence Diagrams**
  - *Scenarios*
  - *Message exchange*
  - *Timing constraints*
TURTLE's IOD
Example
Using Choices
TURTLE’s Sequence Diagrams
Message Semantics

- Synchronous message
  - *Sender and receiver must synchronize*

- Asynchronous message
  - *Sender writes message on a channel*
  - *Receiver reads message from the channel*

- Various possible semantics for channels

- Default semantics
  - *No delay*
  - *Total ordering*
  - *FIFO buffer at receiver’s side*
  - *1 channel is settled for each trio (sender, receiver, message)*

Ludovic Apvrille - UML - 2005. Slide #55
Absolute and Relative Time Constraints
Simulating with Time Constraints

<table>
<thead>
<tr>
<th>Time</th>
<th>TURTLE action(s)</th>
<th>Values</th>
<th>RT-LOTOS action</th>
<th>Action No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>end_tc_0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.2</td>
<td>M.OUT b.out</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3</td>
<td>go_tc_0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4</td>
<td>Channel_b.i2.M.IN b</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.5</td>
<td>Channel_b.i2.M.OUT b</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ATC.i.end_tc</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.6</td>
<td>Channel_a.i1.M.IN a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.7</td>
<td>Channel_a.i1.M.OUT a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.8</td>
<td>Channel_a.i1.M.IN a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ATC.i.go_tc</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ludovic Apvrille - UML - 2005. Slide #57
Using Timers

- ref sendData
  - ref dataOk
  - ref DataResend

- timer=timer1, duration=10
- computeData

- data

Ludovic Apyrille - UML - 2005. Slide #58
Using Timers (Cont.)

{timer=timer1}
dataRetransmit

dataOk

Ludovic Apvrille - UML - 2005. Slide #59
Projection on all gates except the one used for the management of time constraints.
Non-Implementability Issue

- Temporal constraints may reduce possible paths
  - No path at all!
  - Temporal inconsistencies

- Instances execute their events on their own
  - Distributed system
  - At choice node, they may not all execute the same scenario leading to deadlock situations
Temporal Constraints Reducing Logical Paths

Deadlock situation if action "b_in" is fired after 13 time units: action "b_out" never happens
Deadlock situation: action “b_out” never happens
Non-Implementability due to Logical Constraints
Non-Implementability due to Logical Constraints (Cont.)
Non-Implementability due to Logical Constraints (Cont.)
I. Introduction

- UML Profile
- The TURTLE Profile
- Design with TURTLE
- Analysis with TURTLE
- Deployment with TURTLE
Methodology with Deployment

(1) Analysis: IOD + SDs
   - Formal validation

(2) Design: CD + ADs
   - Formal validation
   - Automatic synthesis

(3) Deployment: components + DD
   - Code generation
   - Formal validation

(4) Code (Java)
   - Execution

Generation and execution of Java code
What is a UML Deployment diagram?

- Set of execution nodes
  - nodes may host artifacts
- Links between nodes

client:Client

server:Server

Ludovic Apvrille - UML - 2005. Slide #69
TURTLE Deployment diagrams

- **TURTLE artifacts**
  - Set a classes modeled in a TURTLE designs

- **TURTLE Deployment diagrams**
  - **Execution nodes**
    - May hosts TURTLE artifacts
  - **Links between nodes**
    - Interconnection of Artifacts’ gates
    - Formal specification
      - Parameter: delay, loss rate
      - Pseudo FIFO
        - Actions in the same time slot may be reordered
    - For Java code generation
      - Protocol: UDP, TCP, RMI
      - Ports
Example of TURTLE Deployment Diagram

Artifact \textit{PkgClient} is defined here, and used there.