Towards Fixing Sketchy UML Models by Leveraging Textual Notations
Application to Real-Time Embedded Systems

Frédéric Jouault
frederic.jouault@eseo.fr

Jérôme Delatour
Jerome.delatour@eseo.fr
Overall Objective & Problems

- Our main objective
  - From models created using UML (or SysML) tools such as Papyrus, Rhapsody, etc.
  - Generate verifiable Fiacre specifications
    - Fiacre is a french acronym for “Intermediate Format for the Embedded Distributed Component Architectures”
    - Notably for exhaustive model exploration using tools such as OBP explorer (see: http://www.obpcdnl.org/doku.php)

- Notable problems
  - Preciseness & completeness of UML models necessary for formal models generation
  - Semantic mismatches between UML & Fiacre
  - Variability in UML metamodel(s): abstract syntax, concrete syntax, but most importantly semantics
UML Subset

- We model at **general design** level
  - Active objects
    - instances of active classes with state machines
  - Communication via asynchronous signals
    - Interconnections between active objects modeled as composite structure
- Only three diagrams, typical of embedded systems modeling
  - Class
  - Composite structure
  - State
- **Precise** and **unambiguous** semantics
Approach Overview

- **Architecture:**
  - Front end: handles UML tools variability
  - Back end: handles Fiacre model generation
  - Eclipse UML 2.4 metamodel used as reference & pivot abstract syntax
    - *Native* tools such as Papyrus can be directly supported

```
UML Tools → Front end → Eclipse UML → Back end → Fiacre verification tools
```
Approach (front end)

- Adapt alternative tools (such as Rhapsody) using transformations
  - Takes care of abstract & concrete syntax variability at relatively low cost
- Evaluate model preciseness & completeness
  - Using extended UML model validation (i.e., more OCL constraints)
  - Rendering UML models in several ways for review by experts
- Model semantic variability
  - Within UML standard
  - Across UML tools (including non-conforming tools, which are unfortunately numerous)
Introducing tUML

- Problems with graphical UML tools
  - No global model view
  - Hidden “details” (accessible as “properties” via several clicks)

- Approach: a textual syntax for this UML subset
  - A single homogeneous rendering of the complete model
  - Visible “details”
  - Close to UML metamodel
  - Reduced redundancy
  - Full-text search and replace

- tUML is just another view on the model, not intended to replace diagrams
tUML

or \{tvt\} ?

(textual, verifiable by translation)
class Button behavesAs SM implements I_Controls
receives
  problems_detected_R(problems_detected) {
private controlsManager[1-1] : ControlsManager in ControlsManager_Button;
private id[1-1] : Integer;

stateMachine SM {
  region MainRegion {
    Initial -> RELEASED;
    RELEASED -> PRESSED : button_pressed_SE /
    opaqueBehavior = 'send pressed(id) to controlsManager;' in ABCD;;
    PRESSED -> PRESSED : after100ms /
    opaqueBehavior = 'send pressed(id) to controlsManager;' in ABCD;;
    PRESSED -> RELEASED : problems_detected_SE /;
    PRESSED -> RELEASED : button_released_SE /;
    initial pseudoState Initial;
  }
}
}
tUML Transformations

Input from (graphical) UML modeling tool

- Rhapsody
- tUML editor
- tUML

- tUML2Diagnostic
- tUML2Fiaacre
- tUML2PlantUML

- Diagnostic
- Fiacrete
- PlantUML

Validation | Visualization
---|---
tUML Editor

- Defined with TCS (xText-like)
Visualization Example: Cruise Control System
Visualization Example: Context Diagram
Profiles/Stereotypes Example

```uml
model ProfilesTest {
    <<pathmap://SysML_PROFILES/SysML.profile.uml#
        //Blocks/Block>>

class A {}
    <<pathmap://Papyrus_PROFILES/MARTE.profile.uml#
        //MARTE_DesignModel/HLAM/RtUnit(
            isDynamic=false,
            srPoolSize=5,
            operationalMode=m,main=o)>>

class B extends A implements I {
    operation o();
}

interface I {}
    <<pathmap://Papyrus_PROFILES/MARTE.profile.uml#
        //MARTE_DesignModel/HLAM/RtUnit(
            isDynamic=false,
            srPoolSize=5,
            main=B::o)>>

    <<pathmap://SysML_PROFILES/SysML.profile.uml#
        //Blocks/Block>>

class C extends B {}

class D {}

activity m {}
```
Possible tUML Extensions

• Extend supported UML subset
  – Improved State Machines support: composite states
  – Activities
  – Improved communications: operations (asynchronous calls, synchronous calls), ports (lightweight)
  – Non-opaque guards & effects → Activity and Actions subset (fUML, Alf, etc.)
  – Alternative semantics for UML variation points

• But we must make sure we can verify models using these additional features.
Conclusion

- In a UML model verification context
- For critical embedded systems
- Leveraging a specific textual syntax
- Applied to a PaceMaker Case Study
  - Originally as a contemplative model (i.e., nice pictures)
  - That we tried to verify
  - That we tried to “fix” using graphical editors
  - For which tUML simplifies the fixing work
Identifying Issues in tUML Editor

- Problem markers
Possible Extensions of this Work

- Systematic study of problems that can be fixed more easily by using tUML
- Partial textual view instead of whole-model view
- Abstract-Concrete syntaxes synchronization
- Comparing to other ways to address the problem of graphical UML editors, e.g.:
  - Leveraging the new Papyrus customization features (hiding elements in the palette, selecting which property show up in property views, etc.)
Thanks for listening!

Non contractual pictures.