Automatic UML transfo. 4
Context-aware Verification

Frederic.Jouault
Ciprian.Teodorov
Jerome.Delatour
Luka.Le_Roux
Philippe.Dhaussy

@eseo.fr  @ensta-bretagne.fr
Outline

● Context
  ○ UML
  ○ Context-based verification & OBP
● Transformation Overview
● UML Subset
● tUML
● ABCD
● Conclusion & Perspectives
Context

- Complex systems with many interacting components
- Verification by Exhaustive Reachability Analysis
  - Automatic
  - Safety and Liveness
  - Counter-example
UML

- A modeling language
  - Graphical, with more than a dozen diagrams
  - Able to model structural and behavioral aspects
  - Semi-formal
- Usable for
  - Requirements
  - Specification
  - Design
- Facilitates dialogue between experts
- Extensible via profiles, notably:
  - MARTE for real-time & embedded systems
  - SysML for systems engineering
Context-aware Verification

- Explicit environment modeling (contexts)
  - Operating modes
  - External actors
  - Environmental conditions
- Requirements associated to contexts

System-under-study || Context
in Fiacre || in CDL
Transformation Overview

- Model in UML, convert to Fiacre, verify
  - Front end: handles UML tools variability
  - Back end: handles Fiacre model generation
  - Eclipse UML 2.4 metamodel used as reference & pivot abstract syntax
UML Subset

- Only three diagrams, typical of embedded systems modeling
  - Class
  - Composite structure
  - State
- Precise and unambiguous semantics
- We model at general design level
  - Active objects
  - Communication via asynchronous signals
UML Subset Example

I l_Controls
<<signal>> button_pressed
<<signal>> button_released

C |Button|
id
<<signal>> problems_detected

buttons[*]
<<use>>

I l_ControlsManager
<<signal>> pressed

C ControlsManager

controlsManager

SUS

theOffButton : Button
theOnButton : Button

ControlsManager_OffButton
ControlsManager_OnButton

theControlsManager : ControlsManager

button_pressed / send pressed({ id: id}) to controlsManager;
problems_detected / button_released

RELEASED

PRESSED

after 100ms / send pressed({ id: id}) to controlsManager;
tUML

● Problem 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
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;
    }
}
ABCD

- Intermediate format
- Halfway between UML & Fiacre
  - Reuses Fiacre syntax, but notably adds UML-like event-pool

- Architecture
- Behaviors
- Communications
- Data
ABCD Example
Class Diagram

type ButtonEvents is union
    problems_detected  \localy defined event
    button_pressed
    button_released
end union

type ControlsManagerEvents is union
    pressed of int
end union

channeltype ToControlsManager is eventpool buffersize: 2 of ControlsManagerEvents
channeltype ToButton is eventpool buffersize: 2 of ButtonEvents
ABCD Example
Composite Structure Diagram

channel itsOnButton : ToButton
channel itsOffButton : ToButton
channel itsControlsManager : ToControlsManager
par
  [itsOnButton] theOnButton:Button(1) [itsControlsManager]
|| [itsOffButton] theOffButton:Button(2) [itsControlsManager]
|| [itsControlsManager] theControlsManager:ControlsManager
end par
process Button
port
myEventPool : in ButtonEvents;
itsControlsManager : out ControlsManagerEvents;
param
   id : int
is
init to RELEASED
from RELEASED
receive ButtonEvents:button_pressed from myEventPool;
send ControlsManagerEvents:pressed(id) to itsControlsManager;
to RELEASED
from PRESSSED
select
   wait[100,100];
send ControlsManagerEvents:pressed(id) to itsControlsManager;
to PRESSSED
[] receive ButtonEvents:problems_detected from myEventPool;
to RELEASED
[] receive ButtonEvents:button_released from myEventPool;
to RELEASED
end select
from UnsetSetting
  defer = {evtRequestEngagement}
select
...
[] receive evtSpeedA(speed) from eP;
...
end select;

from UnsetSetting
select
...
[] case (not (empty myEventPool)) of
  true ->
    myEventPool :=
      preprocessDeferred_ToActuation(
        myEventPool, [true, false, false, …]);
  case (first myEventPool) of
    evtSpeedA (speed) ->
      myEventPool := (dequeue myEventPool)
      end case
  end case
end case;
...
end select;
function preprocessDeferred_ToActuation
eventPool = ToActuation, deferMask,
deferMask(toActuation), ToActuation is var
deferred: ToActuation = [[]],
result: ToActuation = [[]],
continue: bool = true
begin
while (not (empty eventPool)) and continue do
  case [first eventPool] of
    eventPool := (dequeue eventPool)
      def(DeferredEventAction := (deferMask[2]))
eventPool := (dequeue eventPool)
      else
        continue := false
        end if
    | evCloseAppAction := (deferMask[3])
eventPool := (dequeue eventPool)
      else
        continue := false
        end if
    | evCloseAppAction := (deferMask[4])
eventPool := (dequeue eventPool)
      else
        continue := false
        end if
    | evSetA := (deferMask[5])
eventPool := (dequeue eventPool)
      else
        continue := false
        end if
    | evIncA := (deferMask[6])
eventPool := (dequeue eventPool)
      else
        continue := false
        end if
    | evDecA := (deferMask[7])
eventPool := (dequeue eventPool)
      else
        continue := false
        end if
    | evPause := (deferMask[8])
eventPool := (dequeue eventPool)
      else
        continue := false
        end if
    | evProc := (deferMask[9])
eventPool := (dequeue eventPool)
      else
        continue := false
        end if
    | evSpeedR := (deferMask[10])
eventPool := (dequeue eventPool)
      else
        continue := false
        end if
    | evSetB := (deferMask[11])
eventPool := (dequeue eventPool)
      else
        continue := false
        end if
    | evIncB := (deferMask[12])
eventPool := (dequeue eventPool)
      else
        continue := false
        end if
    | evDecB := (deferMask[13])
eventPool := (dequeue eventPool)
      else
        continue := false
        end if
    | evPause := (deferMask[14])
eventPool := (dequeue eventPool)
      else
        continue := false
        end if
    | evProc := (deferMask[15])
eventPool := (dequeue eventPool)
      else
        continue := false
        end if
    | evSpeedR := (deferMask[16])
eventPool := (dequeue eventPool)
      else
        continue := false
        end if
    | evSetC := (deferMask[17])
eventPool := (dequeue eventPool)
      else
        continue := false
        end if
    | evIncC := (deferMask[18])
eventPool := (dequeue eventPool)
      else
        continue := false
        end if
    | evDecC := (deferMask[19])
eventPool := (dequeue eventPool)
      else
        continue := false
        end if
    | evPause := (deferMask[20])
eventPool := (dequeue eventPool)
      else
        continue := false
        end if
    | evProc := (deferMask[21])
end case
end while;
if (not (empty eventPool)) then
  result := [enqueue(result, [first eventPool])];
eventPool := (dequeue eventPool)
end if;
while (not (empty deferred)) do
  result := [enqueue(result, [first deferred])];
  deferred := [dequeue deferred]
end while;
while (not (empty eventPool)) do
  result := [enqueue(result, [first eventPool])];
eventPool := (dequeue eventPool)
end while;
return result
end
Conclusion

- Our tool
  - helps creating correct models
  - enables verification of UML models
    - By transformation to Fiacre
    - Supporting contexts by using OBP
  - can be extended to additional input formalisms
    - By targeting either UML/tUML or ABCD
Conclusion

UML 2 Fiacre
Application to
Cruise-Control System (CCS)

CCS

360 lines

460 lines

1560 lines

(XMI: 4330 lines)

CDL

OBP Explorer

Requirements

CCS

Rhapsod

CCS

tUML

CCS

ABCD

CCS

Fiacre
Thanks!

Questions?
SYLLABUS OVERVIEW  (see http://www.mdd4dres.org/program/ for details)
Lectures given by F. Bordeleau, M. DiNatale, M. Egea, R. France, P. Fritzson, J.M. Jézéquel, G. Karsai, P.A. Muller, B. Schätz, B. Selic, J. Sztipanovits, H. Vangheluwe, J.L. Voirin, ...

And a Work in Progress Session and Tools Session

Apply online http://www.mdd4dres.org before August 23th

Get a flyer (http://www.mdd4dres.org/flyer.pdf) and help us spread the word!
You can even get a printed flyer at Neptune 2014.