Reehan Shaikh

MSDL Summer Presentations
School of Computer Science
McGill University

August 27th, 2009
COMP – 304, School of Computer Science, McGill University


Outline

- Introduction
- Requirements of the DigitalWatch
- Solution to the DigitalWatch
- Code synthesis
  - Class diagram
  - Statecharts
  - DEVS
  - Demo
- Conclusion + Future work
Introduction

- **What: Goals are:**
  - To synthesize code from a statechart
  - Use this code to model a behaviourally equivalent DEVS system

- **Why:** allows for truly multi-formalism modelling since DEVS can be used as a semantic domain for DAEs, Motif, etc...

- **How:** Syntax check and compile (to nice, readable, executable code)
Requirements

- We should know these like the back of our hands by now, after taking COMP 304/522/763.
- Just as a recap (for visitors):
- Given a set of requirements, an API and some pre-defined events, create a statechart that meets those requirements
Given a class diagram describing the system structure, where each class may have an associated statechart describing its behaviour, we must analyze and compile this model:

- First, check validity of class diagram as a whole (at least one class, unique names, single default class)
- Second, process each association and inheritance
- If there were no errors, process each class. If the class has no errors and has a statechart, process the statechart
Statecharts

- Verify that there are no empty components
- Verify that each component doesn't contain two or more children with identical names
- Each component must contain a single default state
- Verify correct statechart semantics
- Calculate transition data (LCA, enter/exit actions)
- Synthesize code
DigitalWatchBehaviour

Attributes:
- behaviour :: CDV3_DChart_TYPE
- i :: Integer
- m :: CDV3_Method_TYPE
So, what will the synthesize code look like?

```python
class DigitalWatchBehaviour:
    Basic0 = 0
    Basic1 = 1

    def init(self, controller, loopMax):
        "initialize statechart variables..."
        i = 0 "user defined variable"
        def enterActionBasic0(self):
            "enter action for Basic0"
```
def exitActionBasic1(self)
    "exit action for Basic1"
...
def m(self, parameters)
    "body of user defined method"
def event(self, event, time, *args)
    "add event to object's event queue"
def getEarliestEvent(self)
    "return absolute time"
...
And now the core behaviour:

def transition(self, event, parameters):
    if currentState == Basic0:
        if event == "normal":
            self.exitActionBasic0()
            print "trigger normal"
            self.enterActionBasic1()
    if currentState == Basic2:
        if event == "abnormal":
            self.exitActionBasic2()
            print "trigger abnormal"
            self.enterActionBasic3()
CD + SC example

- Two helper functions
  - microstep to calculate current events to be processed and continuously call transition until stable, unchanged state
  - step to calculate AFTER events and continuously call microstep
- Now, how would an application run, suppose there were many classes and statecharts?
We need a controller to control the various activities. We will discuss the DEVS controller in particular.

- 4 functions to be filled
  - External transition
  - Internal transition
  - Output function
  - Time advance
Each controller needs to keep track of all object instances that entail a statechart

External transition
- Interrupts are given to a DEVS model via ports
- An external event is packaged as the event "portName:event" for the statechart and each statechart's step function is called

Internal transition
- Each statechart's step function is called
Output function

- If a statechart puts an output event, of the form "portName:event", on the controllers output queue, it is taken care of when the output function is called.

Time advance

- Get each statechart's earliest event time
- The earliest time of those is returned
- Otherwise INFINITY is returned, meaning statecharts are finished processing or awaiting external interrupt
So, we synthesized code for a statechart

Essentially, we used DEVS as a particular controller, but other controllers are available

- Real-time or simulated time
- Time-sliced or event-scheduled

Future work

- Add a mechanism to generate Tkinter events
- Finish bug testing for the above controllers
!!!Thank you!!!

???Questions???