Abstract State Machines

IRMS Presentation by
Eman Elghoneimy
Ph.D. Student
January 18th, 2005

Based on CMPT 745 lecture notes, Fall 2004, by Dr. Uwe Glasser

Contents

- Software modeling
- Formal methods
- ASM and ground models
- Executable specifications
- ASML
- DASM
- References

Software modeling

- Modeling is the process of turning product ideas and specs into software requirements.
    - What product are we building?
- Requirements are refined into implementation
- Verification
    - Are we building the product right?
- Validation
    - Are we building the right product?

Formal methods

- Any attempt to use mathematics in defining a computer-based system.
- Can be used in:
  - Verification
    - Requirement spec
    - Design spec
    - Code spec
  - Study of key properties
  - Refutation

ASM Ground model

- Model reflects the system, establish the correctness and completeness of system by observing and experimenting the model
- Model used to
  - Clarifying requirements
  - Turning English into mathematics
  - Discovering ambiguities, inconsistencies and loose ends in informal descriptions

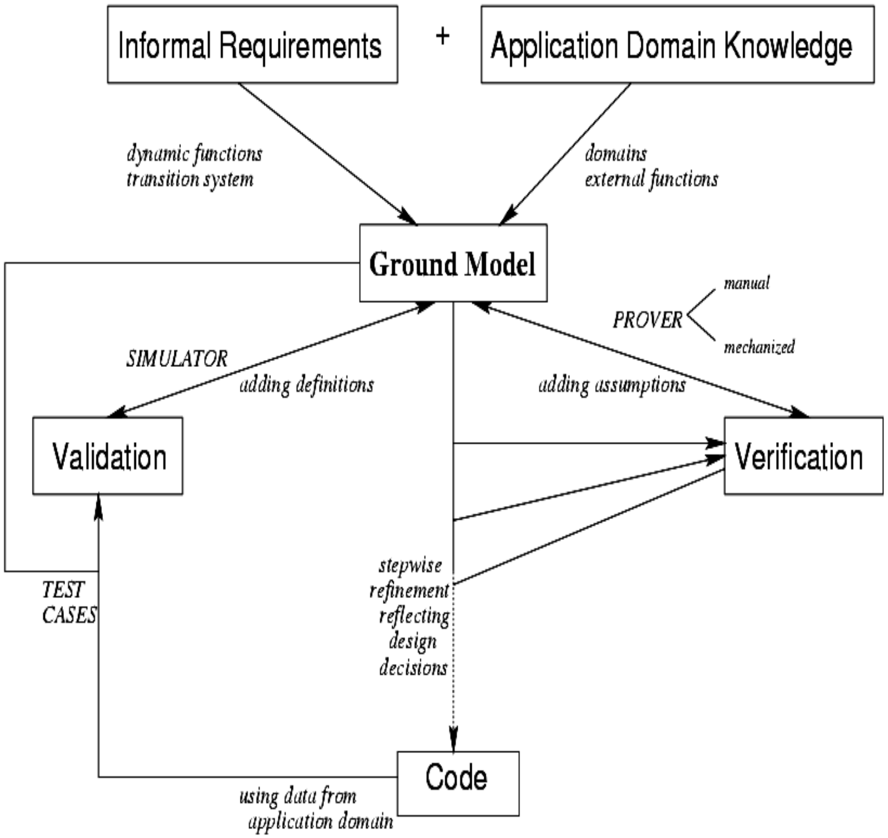
ASM ground models (cont.)

- Establish system correctness by experimental, mathematical and conceptual justification of the model.
- Bridge the gap between specification, implementation and verification.

Ground Model characteristics

- *Precise*
- *Concise*
- *Abstract*
- *Checkable*
- *Revisable*
- *Refinable*
- *Formal*

Integrated development process [2]



Executable specification

- Model-based specification: abstract encoding
  - Explore design choices
  - Experimental validation of key properties
  - Discover undesirable behaviours

ASML

- ASML (Abstract State Machine Language) is an executable specification language based on the ASM theory.
- ASML is available for free download from the Microsoft research website. The ASML package contains useful tutorials and references to the language.
- ASML supports specification and rapid prototyping of object-oriented and component-oriented software.

ASML: states and updates

- States:
  - The model state is represented by the interpretation of state vocabulary symbols
- Updates:
  - Total update by replacing the value of the vocabulary symbol
  - Partial update by adding or removing elements
  - All updates fired simultaneously (if consistent)

ASML: Statements

- If-then-else
- forall
- choose

Example: Sorting

choose $x,y \in$ Index : $x < y \cap a(x) > a(y)$

do in-parallel

$a(x) := a(y)$

$a(y) := a(x)$

Example: GCD

- $gcd(a; b) = gcd(b; a \bmod b)$

```
if output = undef then
   if (a mod b = 0) then output := b
   else
      a := b
      b := a mod b
```

DASM

- Autonomous operating agents, each with its own program
- Run one step of all agents, or choose subset of agents to run
- Interaction by reading and writing to shared locations of global machine states

DASM: Example

- step until fixpoint
  RunAgents()

- RunAgents()

```
step
    //   forall a in ChooseSubset({a|a in JAgentSet where a.IsAvailable()})
       forall a in JAgentSet where a.IsAvailable()
         a.Program()

    //   forall m in ChooseSubset({m|m in MAgentSet where m.IsAvailable()})
       forall m in MAgentSet where m.IsAvailable()
         m.Program()
```

- FlipCoin() as String

```
    choose x in {"heads","tails"}
      return x
```

- ChooseSubset(elems as Set of JobAgent) as Set of JobAgent

```
    return {e | e in elems where FlipCoin() = "heads"}
```

DASM: Example

- Operations

head: ItemList →Item
tail: ItemList →ItemList
newItem: →Item

- ProducerProgram ≡

itemList := *itemList* ∩ *newItem*

- ConsumerProgram ≡

if *itemList* ≠ *empty* **then**
item := *head*(*itemList*)
itemList := *tail*(*itemList*)

Light control DASM

- Three submachines
  - Automatic control
  - Manual control
  - Malfunction
- Automatic and manual control alternate, with malfunction submachine executed in-between.

Example: Light control (ASMGopher)

Room_wall_button(room, lightgroup) =
if lightgroup_wall_button_pressed(room, lightgroup) **then**
  if lightgroup_is_completely_on(room, lightgroup)
  then
    Switch_lightgroup_off(room, lightgroup)
  else
    Switch_lightgroup_completely_on(room, lightgroup)

Switch_lightgroup_off(room, lightgroup) =
  mode(room) := Manual
  forall light $\in$ lights_in_group(room, lightgroup)
    Switch_light(room, light, minDimValue)

Switch_lightgroup_completely_on(room, lightgroup) =
  mode(room) := Manual
  forall light $\in$ lights_in_group(room, lightgroup)
    Switch_light(room, light, maxDimValue)

Additional requirement

U1Req	It is safe to allow a person who wants to rest in a room to choose a light scene in which all the lights are switched off and the room is dark.
U3Req	Instead of establishing the chosen light scene we use the last light scene.
U10Req	If the ceiling lights do not enter explicitly the lights to be turned on for the given light scene, they are set to minDimValue.
NF5aReq	Ceiling lights in a hallway section are \not controllable manually" if at least one hallway button is defective.
NF5bReq	If a motion detector is defective, its sensor value behaves as if there is motion.
PushButtonReq	Consistency of simultaneous pushing on different wall buttons (fixed priority or hardware solution).
RoomOccupationReq	A reasonable definition for a location to be not occupied is that there has been no motion for a period of max_quiet_time.
MotionDetectorReq	The motion sensor detects motion when users push buttons.
LightSceneReq	The function lights to turn on computes an ordered set containing all lights that should be switched on together with their dim values. The order of the set is the order in which the lights should be turned on.
HallwayReq	The requirements FM1 and NF3 are useless for hallways if these are without windows.
OutdoorSensorReq	The sensor value of an outdoor light sensor remains constant if the sensor does not work correctly.
DefaultLightSceneReq	We do not commit to any particular definition of default light scene

References

1. CMPT 745 lecture notes, Fall 2004, U. Glaesser.
2. E. Börger, E. Riccobene and J. Schmid. Capturing Requirements by Abstract State Machines: The Light Control Case Study. Journal of Universal Computer Science 6(7), pages 597-620, July 2000.
3. U. Glässer, Y.Gurevich and M. Veanes: Abstract Communication Model for Distributed Systems. IEEE Transactions on Software Engineering, vol. 30, no. 7, pages 458-472, July 2004.
4. ASML website: http://research.microsoft.com/fse/asml/
5. ASM website: http://www.eecs.umich.edu/gasm/
6. D. Berry. Formal Methods: The Very Idea. *Science of Computer Programming,* 42(1): 11-27 (2002)
7. Y. Gurevich. Sequential Abstract State Machines Capture Sequential Algorithms, ACM Transactions on Computational Logic, vol. 1, no. 1, July 2000, pages 77-111
8. E. Börger. “The ASM Ground Model Method as a Foundation For Requirements Engineering”. [Online document], http://www.di.unipi.it/~boerger/Papers/SwEngg/GroundModMethod.pdf