

An Executable UML Virtual Machine

Marc J. Balcer

SMUG 2002

Westward Look Resort • Tucson, Arizona



ModelCompilers.com
code at a higher level

Outline

- Background
- Actions in XML
- A simple VM



Origins of the work

- Book
 - What to do about the action language?
 - Detailed reading of the action semantics
- XP and test-first development
 - I wanted a “scripted model verifier”
- Show that model compilers aren't just 3GL code generators



UML action semantics

- OMG effort to define
 - a set of fundamental actions
 - precise semantics for those actions
- Procedures composed of actions used for
 - state procedures
 - constraints
 - derived attributes and associations
 - bridging domains*



Object actions

- Actions that operate on objects / attributes / links
 - createObject (class) → object
 - createLink (object, object, association)
 - addAttributeValue (object, attribute, value)
 - readLink (object, associationEnd) → object
 - sendSignal (object, signal, parameters)



Control actions

- Actions that operate on collections
 - filterAction
 - mapAction
 - reduceAction
- Actions that execute selectively
 - conditionalAction
- Actions that execute repeatedly
 - loopAction



Other actions

- Read and write variable values
- Literal values
- Apply functions
 - No operations or types are “built into” Executable UML
 - Instead, there is a general philosophy of extension



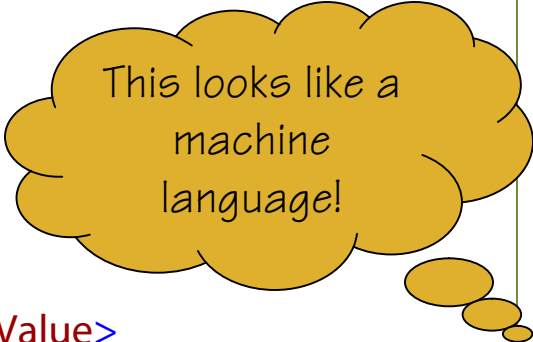
Procedures and actions

- The original problem was
“what to do about an action language”
- First priority
“must be UML Action Semantics – compliant”
- Task
“procedures as fundamental actions”



Procedures composed of actions

```
<procedure>
  <createObject name="Publisher" />
  <addVariableValue name="newPublisher" />
  <createObject name="Book" />
  <addVariableValue name="newBook" />
  <readVariable name="newBook" />
  <literalValue type="string">Modeling the World in Data</literalValue>
  <addAttributeValue name="title" />
  <readVariable name="newBook" />
  <literalValue type="string">0-378-78972-X</literalValue>
  <addAttributeValue name="isbn" />
  <readVariable name="newPublisher" />
  <readVariable name="newBook" />
  <createLink name="R1" />
  <createObject name="Author" />
  <addVariableValue name="newAuthor" />
  <readVariable name="newBook" />
  <readVariable name="newAuthor" />
  <createLinkObject name="R2" />
</procedure>
```



This looks like a machine language!



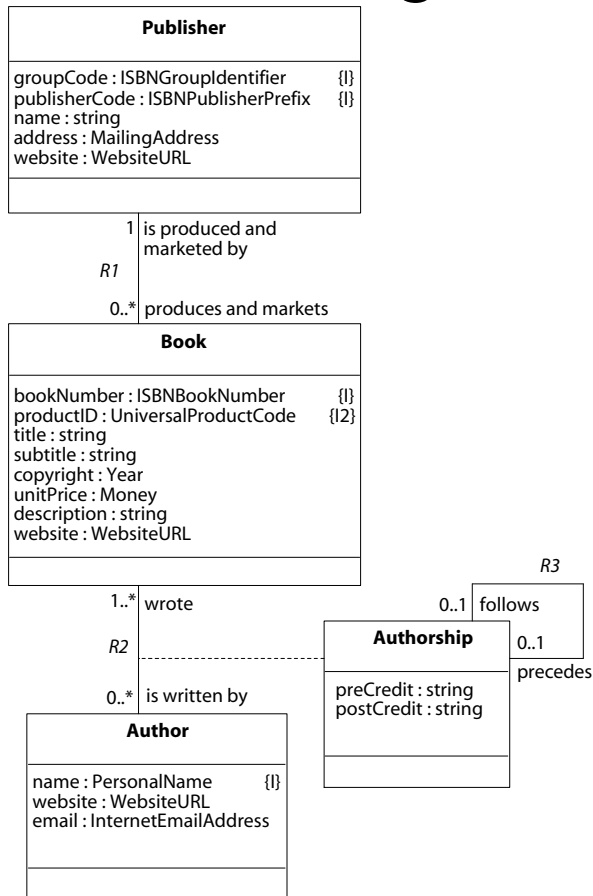
The Executable UML Machine

- Programs in XML
 - model
 - procedures
- Many machines for different architectures
 - run identical programs
 - can exploit concurrency
- Viably demonstrate implicit bridging
 - the simple simulator is an example



Model in XML

- Model as a diagram...



- Or in text (XML)

```
<class name="Publisher">
  <attribute name="groupCode"
    type="ISBNGroupIdentifier">
    <description>
      The ISBN registry group who
      has assigned the Publisher its
      publisherCode. Normally there
      is one ISBN registry group per
      country, region, or language.
    </description>
  </attribute>
  <attribute name="publisherCode"
    type="ISBNPublisherPrefix">
```



Actions in XML

- State procedures have actions in an action language

```
self.code = rcvd_evt.publisherCode;
```



is compiled into

- But the language is just syntax for the fundamental actions

```
<state>  
  <procedure>  
    <readSelf/>  
    <readParam  
      name="publisherCode"/>  
    <setAttribute name="code"/>  
  </procedure>
```



Architecture

- Domain
 - the whole thing – the structure
- Called Procedure
 - state procedure, derived attribute, constraint check
 - parameters and local variables
- Execution Context
 - single sequence of actions
 - stack



Execution Context

- Individual execution contexts run subparts of a procedure
 - each instance in a filterAction
 - condition vs. body in a conditionAction
- Each Execution Context has its own stack
- Each Execution Context could run concurrently (depending upon the action's rules)



Single-operand stack machine

- One fixed operand per operation
- Input values popped off stack
- Result value pushed onto stack

Action	Operand	Input(s)	Output
createObject	class name	none	object reference
setAttributeValue	attribute name	object reference new value	none
readLink	association end	object reference	object reference



Typed Stack

- Each value on the stack has a type
 - Model data type
 - base type
 - domain-specific type
 - Execution data type
 - @instance (object reference)
 - @instanceSet (object reference set)
 - @valueSet



Variables

- Procedure may store values in variables
- Akin to “named data flows” in action semantics
- Variables are typed
 - model type
 - execution type



Implementation-free semantics

- Each action may be realized differently by different software architectures
 - createObject (class) → object
 - Allocate memory
 - Create a row in a database table
 - Create a new database table
 - readLink (object, associationEnd) → object
 - Get the pointer to another object
 - Do a database join query



Demo 1: Model and Procedure

- Load a domain model
- Procedure: Create some initial instances
- Procedure: Figure the total value of the books



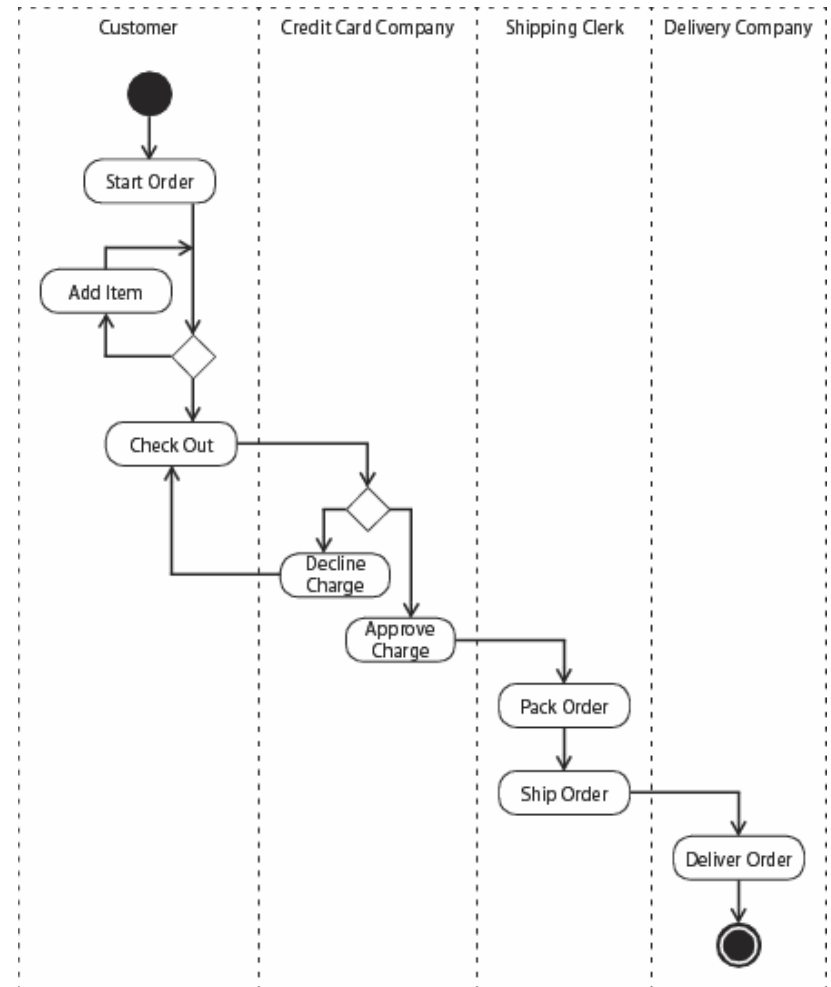
State Machines

- Each state machine instance has its own set of pending events
- Many state machine instances may execute concurrently
- Easy to fire in a single signal
- How to script a sequence of signals and state machine transitions?



Scripting state machines

- Scenario as activity diagram
- Bubbles represent use cases
 - initiating event
 - completion criteria (perhaps alternatives)



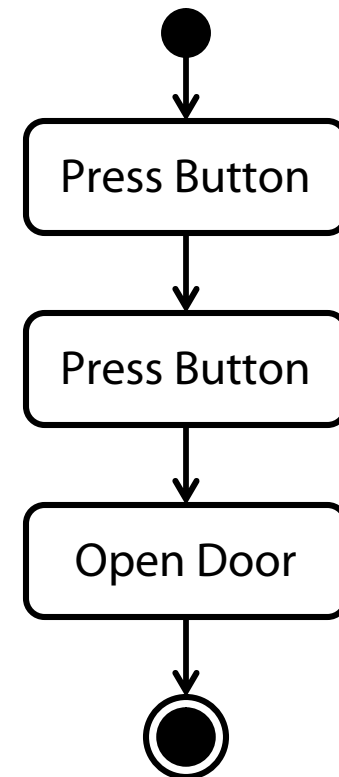
"WaitFor" in scenario

- Execution of scenario waits until the condition is true

```
<readVariable name="myMicrowave" />  
<sendSignal name="buttonPressed" />  
<readVariable name="myMicrowave" />  
<waitFor stateName="Cooking" />
```

```
<readVariable name="myMicrowave" />  
<sendSignal name="buttonPressed" />  
<readVariable name="myMicrowave" />  
<waitFor stateName="Cooking Extended" />
```

```
<readVariable name="myMicrowave" />  
<sendSignal name="doorOpened" />  
<readVariable name="myMicrowave" />  
<waitFor stateName="Cooking Interrupted" />
```



Demo 2: State Machine

- Microwave Oven
- Script signals the oven
- Oven receives events; initiates state procedures
- Script waits for oven
 - to be in specified state
 - with procedure complete

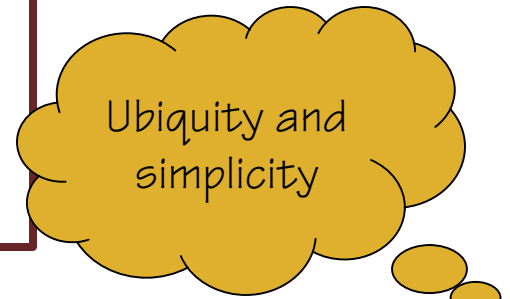
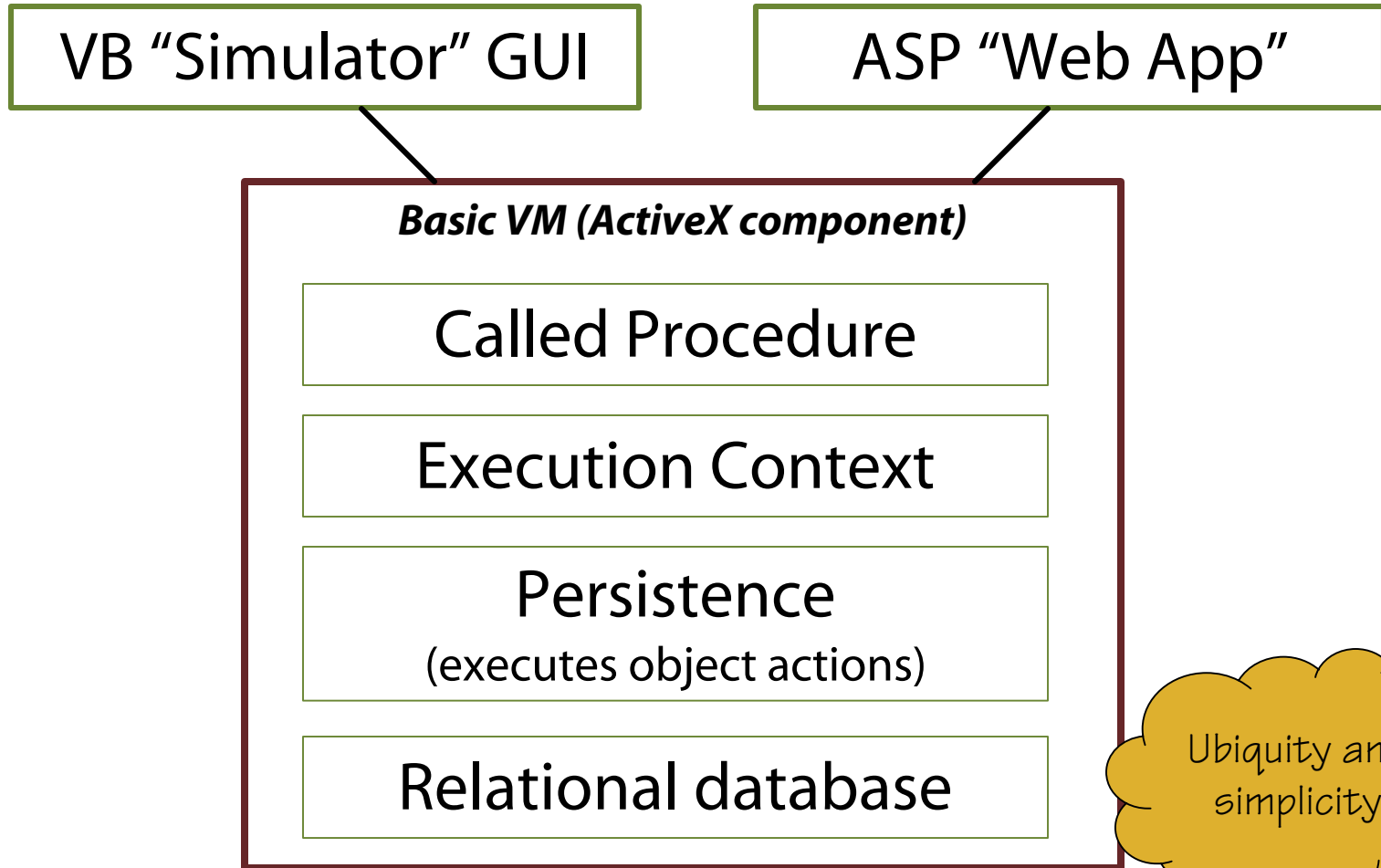


Simple VM

- Fixed relational database
- Single-tasking
- Single processor



Structure of this VM



Implicit Bridging

The displays of states, instance values etc. here...

VB "Simulator" GUI

Basic VM (ActiveX component)

Called Procedure

Execution Context

Persistence
(executes object actions)

Relational database

*...are done
without any code
in the app
domain
models—
the GUI
subscribes to
occurrences in
the app domain*



So what's there?

- XML model representation
 - structure
 - actions
- Archetypes to generate .dxml and .pxml files
- VM as
 - ActiveX component
 - Simulator (GUI atop component)
 - ASP web example



Next?

- XML standard
 - metamodel
 - interchange format for modeling tools
- Higher-performance VMs
 - all-in-memory (for you embedded folks)
 - bytecode instruction set
 - big database
- XSLT codegen archetypes



Alternative to 3GL code generation

- MC-2010, etc. transform models into 3GLs (third-generation languages) like C and C++
 - you're competing with hand-coding
 - archetypes are complex
- A UMLVM model compiler is much simpler
 - the target language is the model
 - no competition with hand-coding
- Greater market acceptance of VMs (Java!)

