An Executable UML Virtual Machine

Marc J. Balcer

ModelCompilers.com

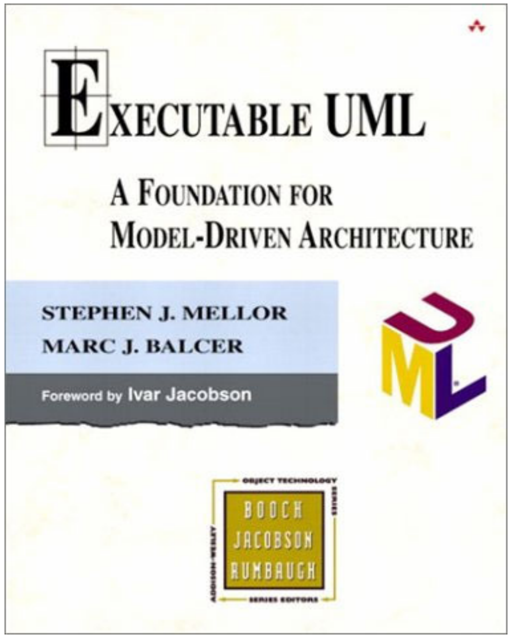
code at a higher level

Contents

Project Origins

what to do about an action language?

we need to make sure it's compliant



EXECUTABLE UML

A FOUNDATION FOR MODEL-DRIVEN ARCHITECTURE

STEPHEN J. MELLOR
MARC J. BALCER

Foreword by Ivar Jacobson

BOOCH JACOBSON RUMBAUGH

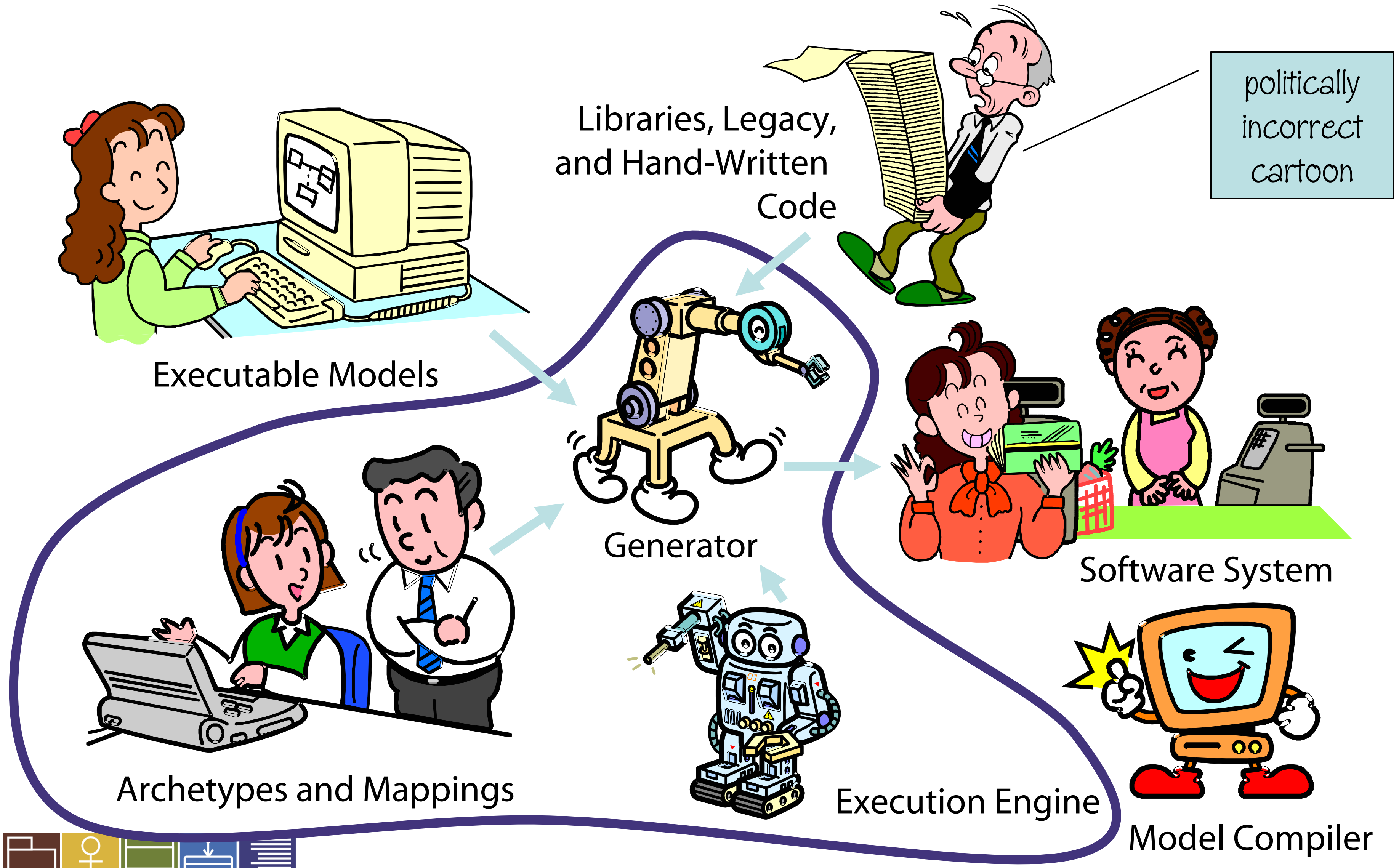
We have these new UML Action Semantics

OBJECT MANAGEMENT GROUP

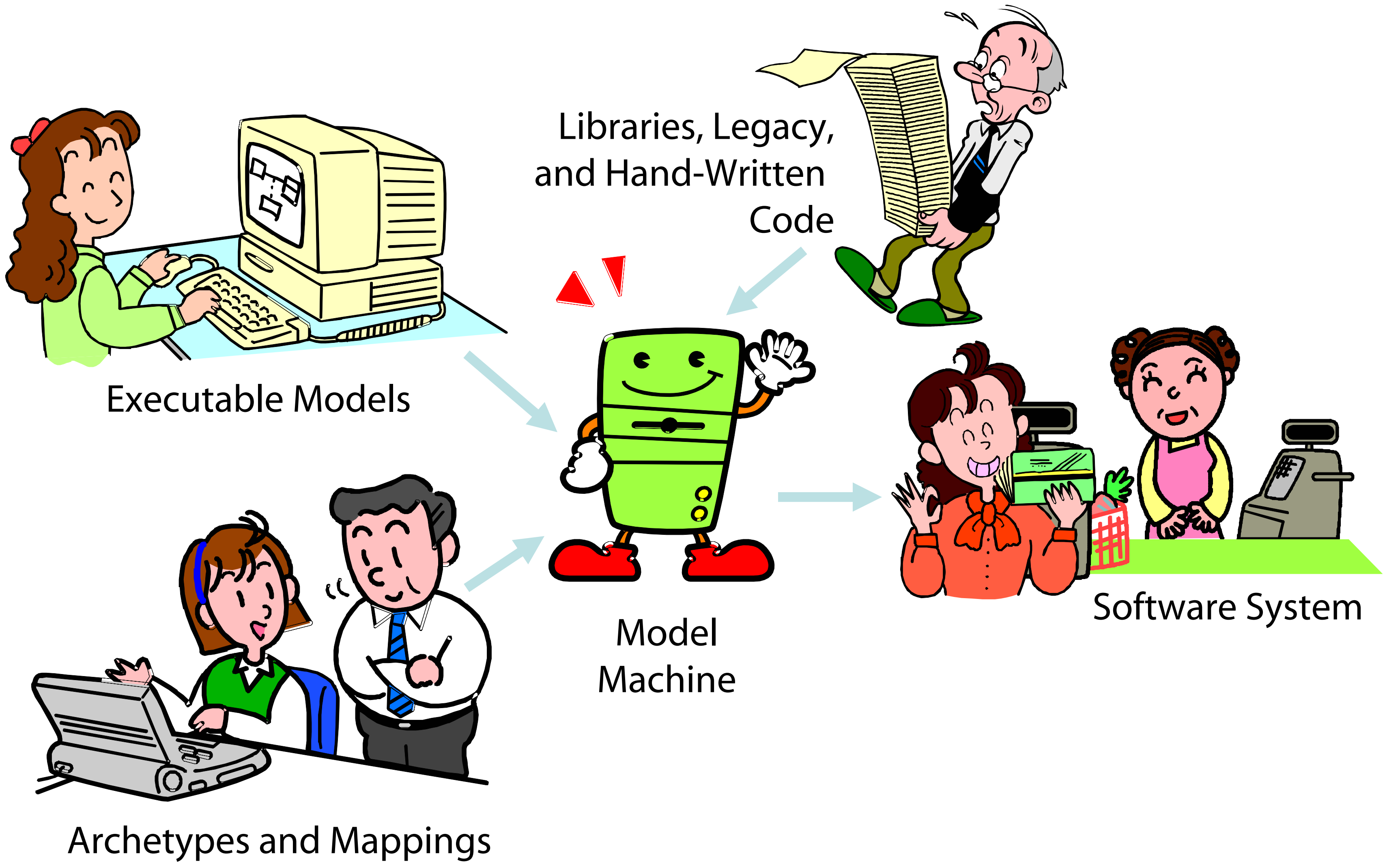
Executable UML is more than code generation

I want a scripted model verifier

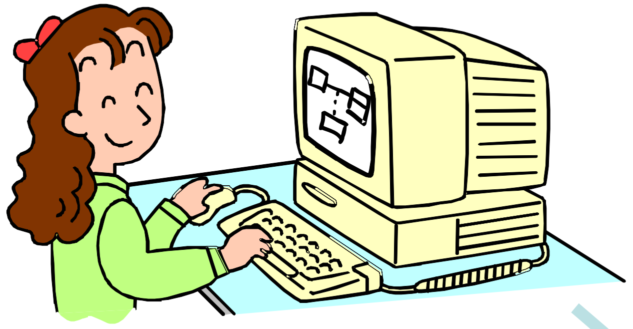
Executable UML != Code Generation



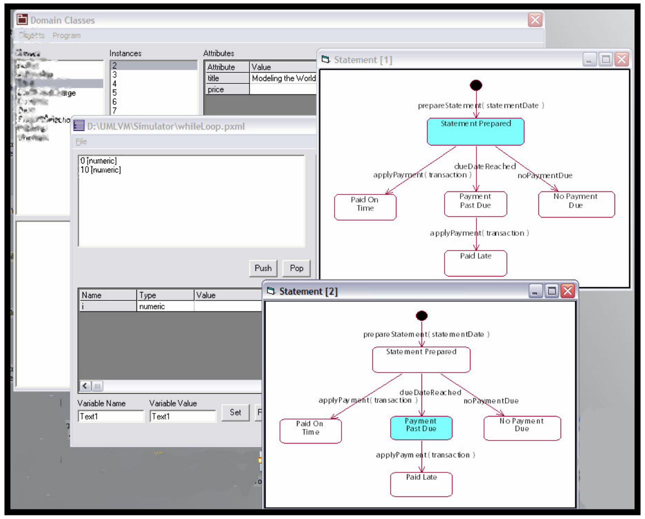
Executable UML != Code Generation



Scripted Model Verifier



Executable Models

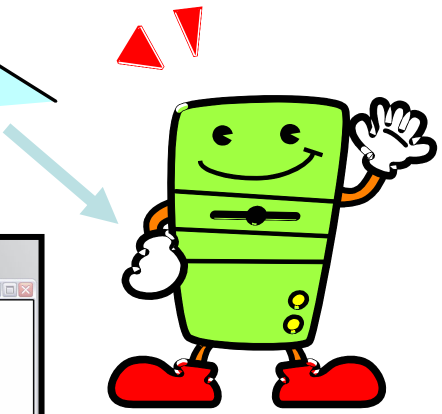


Attribute	Value
title	Modeling the World
price	

0 [numeric]
10 [numeric]

Name	Type	Value
i	numeric	

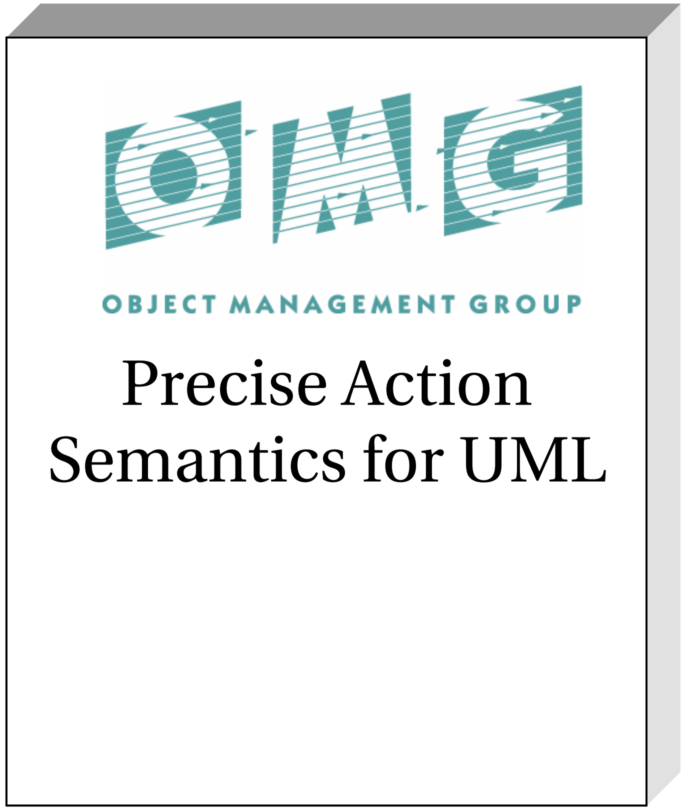
Variable Name: Text1
Variable Value: Text1



Model Machine

- Test-first development with models
- Simple single-processor architecture

UML Action Semantics


OBJECT MANAGEMENT GROUP

Precise Action Semantics for UML

- Integrated into UML 1.5
- Fundamental actions on UML elements (classes, associations, …)
- Foundations for writing processing in an executable model
  - in the problem domain
  - does not presume an implementation

Action Language & Semantics

- This action language

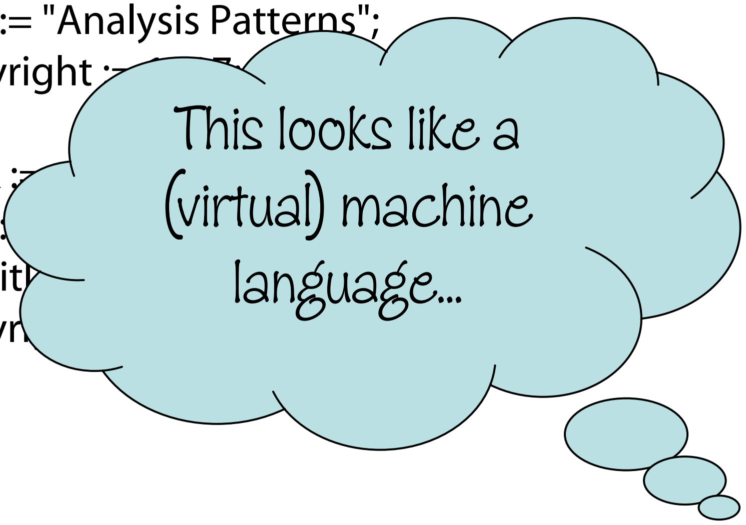
```
create object p of Publisher;
p.name := "Addison-Wesley";
create object b of Book;
relate p to b across R1;
b.title := "Analysis Patterns";
b.copyright := 1997;

create object b of Book;
b.title := "Refactoring";
b.subtitle := "Improving the Design …";
b.copyright := 2001;
```

Action Language & Semantics

- Or this action language

```
p := new Publisher {
    .name := "Addison-Wesley";
    -> Book := new Book {
        .title := "Analysis Patterns";
        .copyright :=
    };
    -> Book :=
        .title :
        .subtitl
        .copyr
    }
}
```

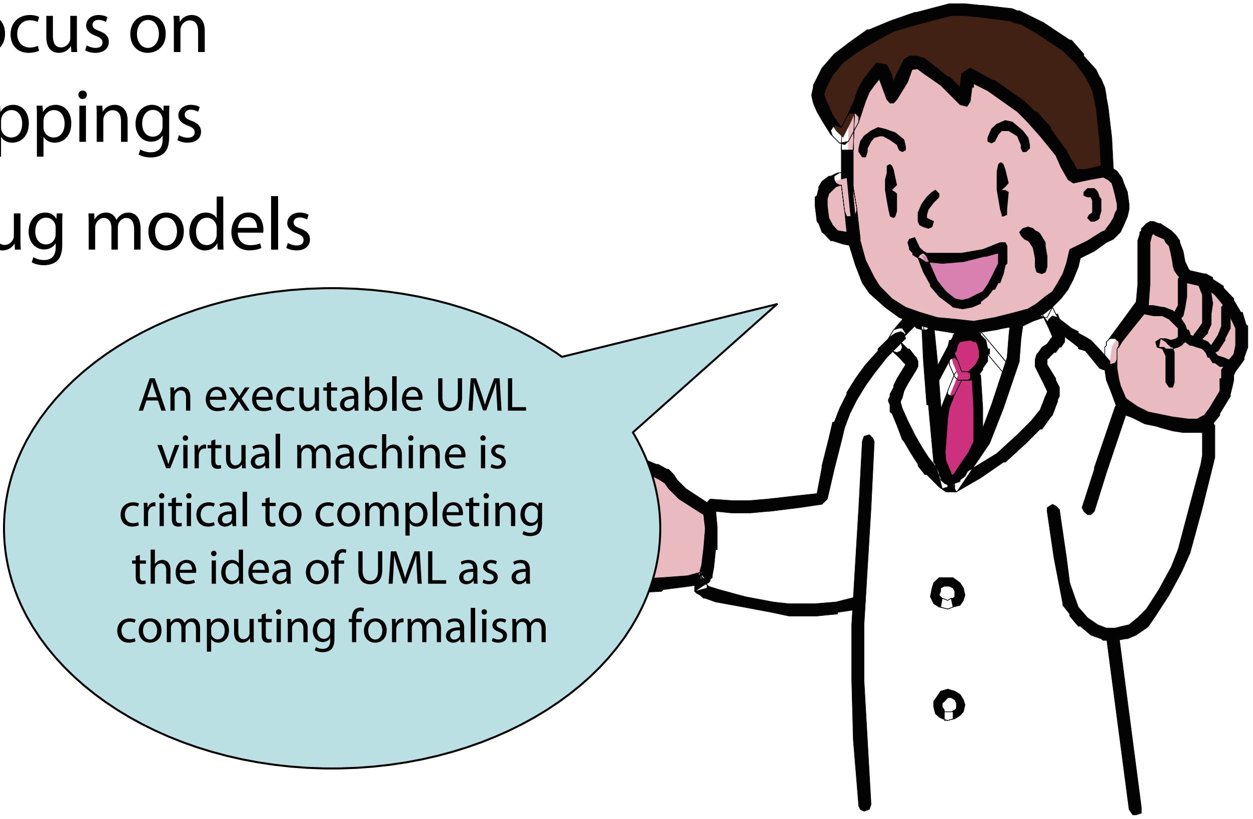


- compiles to actions

```
createObject “Publisher”
addVariableValue "p"
groupAction {
    readContext
    literalValue "Addison-Wesley"
    addAttributeValue “name"
    readContext
    createObject "Book"
    createLink "Book"
    groupAction {
        readContext
        literalValue “Analysis Patterns"
        addAttributeValue “title"
        readContext
        literalValue "1997"
        setAttributeValue “copyright"
    } . . .
```

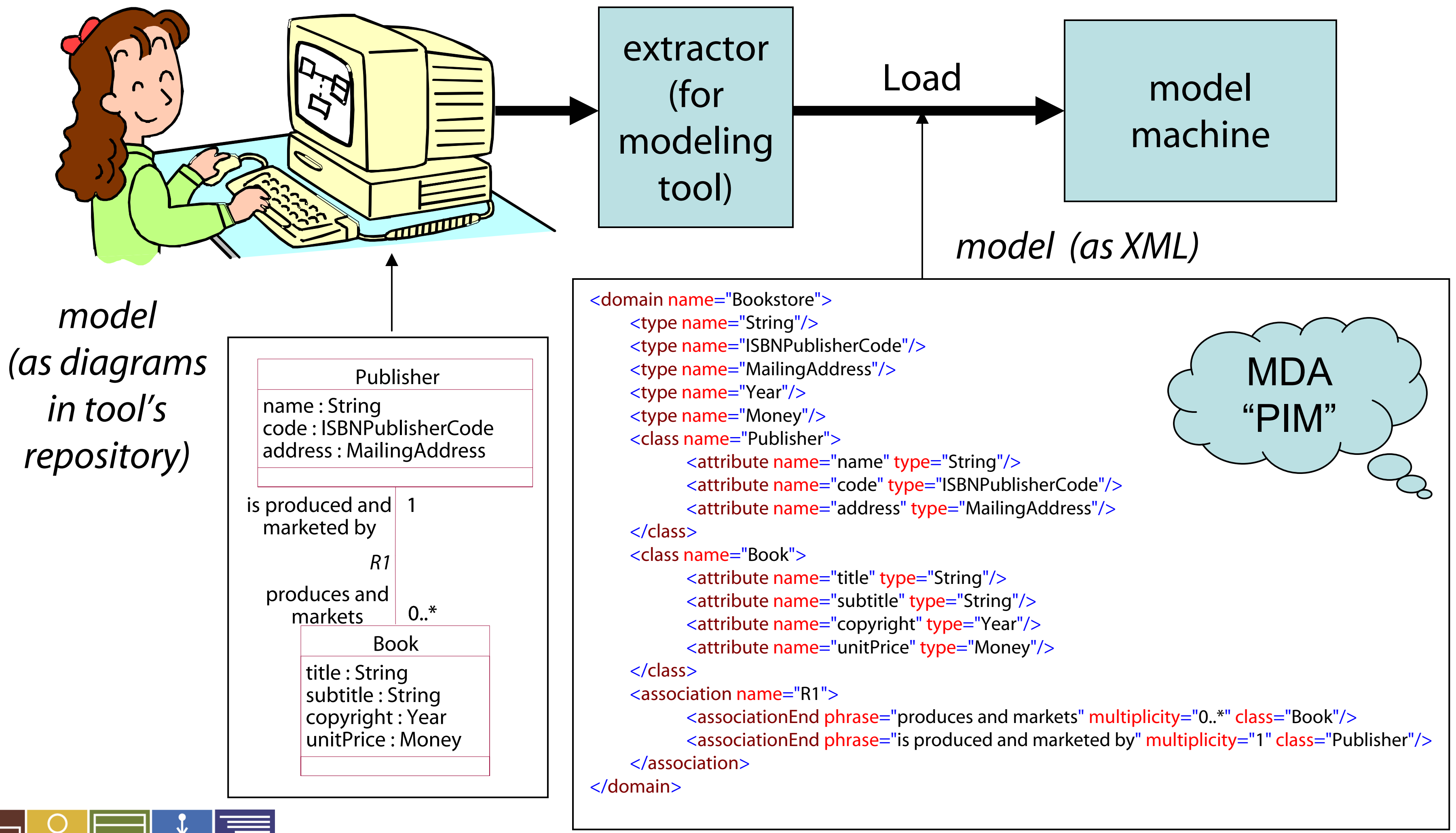
Aside: Why a Virtual Machine?

- Java made VMs palatable
- Project can focus on models & mappings
- Easier to debug models at the model level


An executable UML virtual machine is critical to completing the idea of UML as a computing formalism

Contents

Static Model Import



```xml
<domain name="Bookstore">
       <type name="String"/>
       <type name="ISBNPublisherCode"/>
       <type name="MailingAddress"/>
       <type name="Year"/>
       <type name="Money"/>
       <class name="Publisher">
              <attribute name="name" type="String"/>
              <attribute name="code" type="ISBNPublisherCode"/>
              <attribute name="address" type="MailingAddress"/>
       </class>
       <class name="Book">
              <attribute name="title" type="String"/>
              <attribute name="subtitle" type="String"/>
              <attribute name="copyright" type="Year"/>
              <attribute name="unitPrice" type="Money"/>
       </class>
       <association name="R1">
              <associationEnd phrase="produces and markets" multiplicity="0..*" class="Book"/>
              <associationEnd phrase="is produced and marketed by" multiplicity="1" class="Publisher"/>
       </association>
</domain>
```

Aside: Why Not XMI?

```
<domain name="Bookstore">
    <type name="String"/>
    <type name="ISBNPublisherCode"/>
    <type name="MailingAddress"/>
    <type name="Year"/>
    <type name="Money"/>
    <class name="Publisher">
        <attribute name="name" type="String"/>
        <attribute name="code" type="ISBNPublisherCode"/>
        <attribute name="address" type="MailingAddress"/>
    </class>
    <class name="Book">
        <attribute name="title" type="String"/>
        <attribute name="subtitle" type="String"/>
        <attribute name="copyright" type="Year"/>
                   Price" type="Money"/>
    <
                        produces and markets"
                      class="Book"/>
                    ase="is produced and marketed by"
        multiplicity="1" class="Publisher"/>
    </association>
</domain>
```

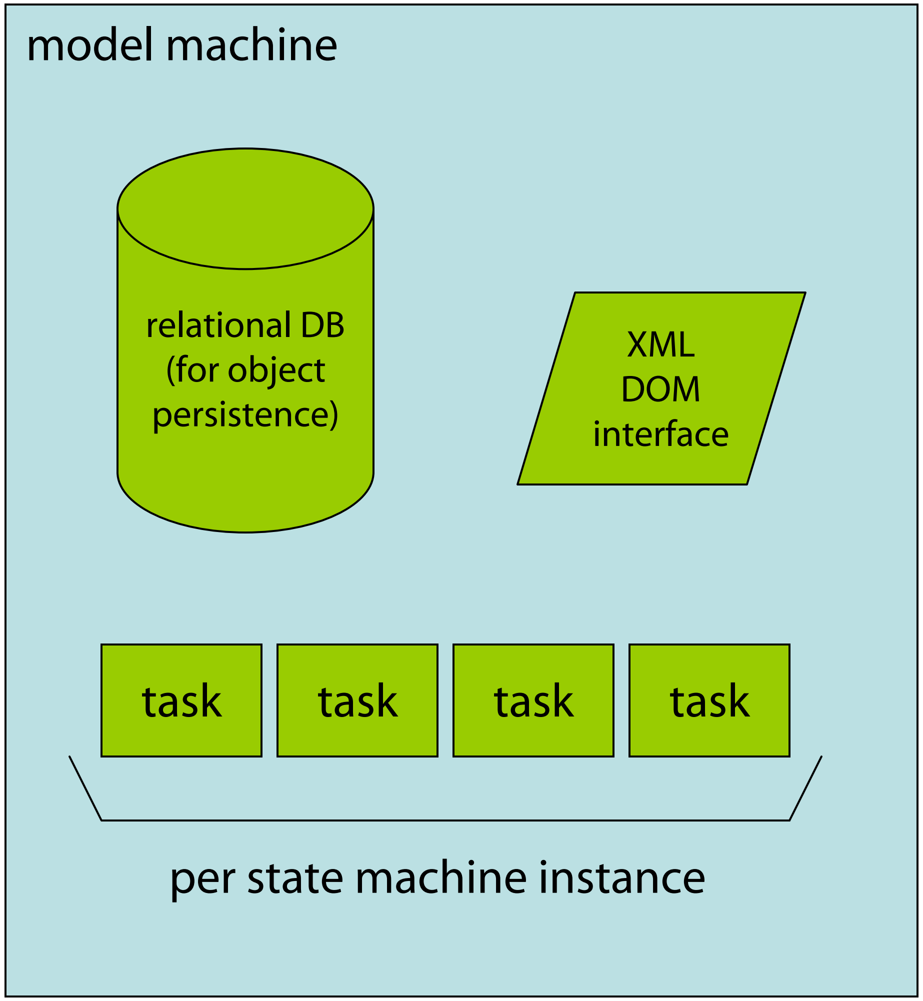
This is human-readable

XMI is

- Too much
  - Only want what's in the executable profile
  - Models, not diagrams
- Not enough
  - No XMI for actions

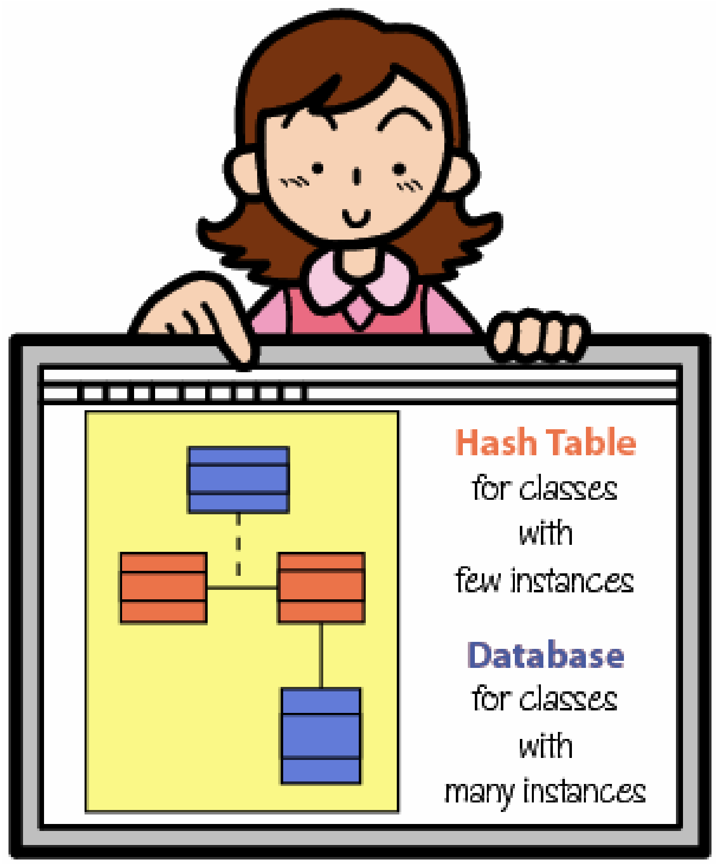
Besides XMI ← → MXML is just a mapping problem!

Internal Representations


model machine
relational DB (for object persistence)
XML DOM interface
task
task
task
task
per state machine instance

- Everything needed to load and run a model
- Of course, this is not the only way
  - there can be multiple VMs
  - a single VM may support multiple approaches

Mappings



- Design-time assignment of different realizations to distinct elements in the model
- Separate from the model and the machine
- “Uniformity != Rigidity”

Mappings

model

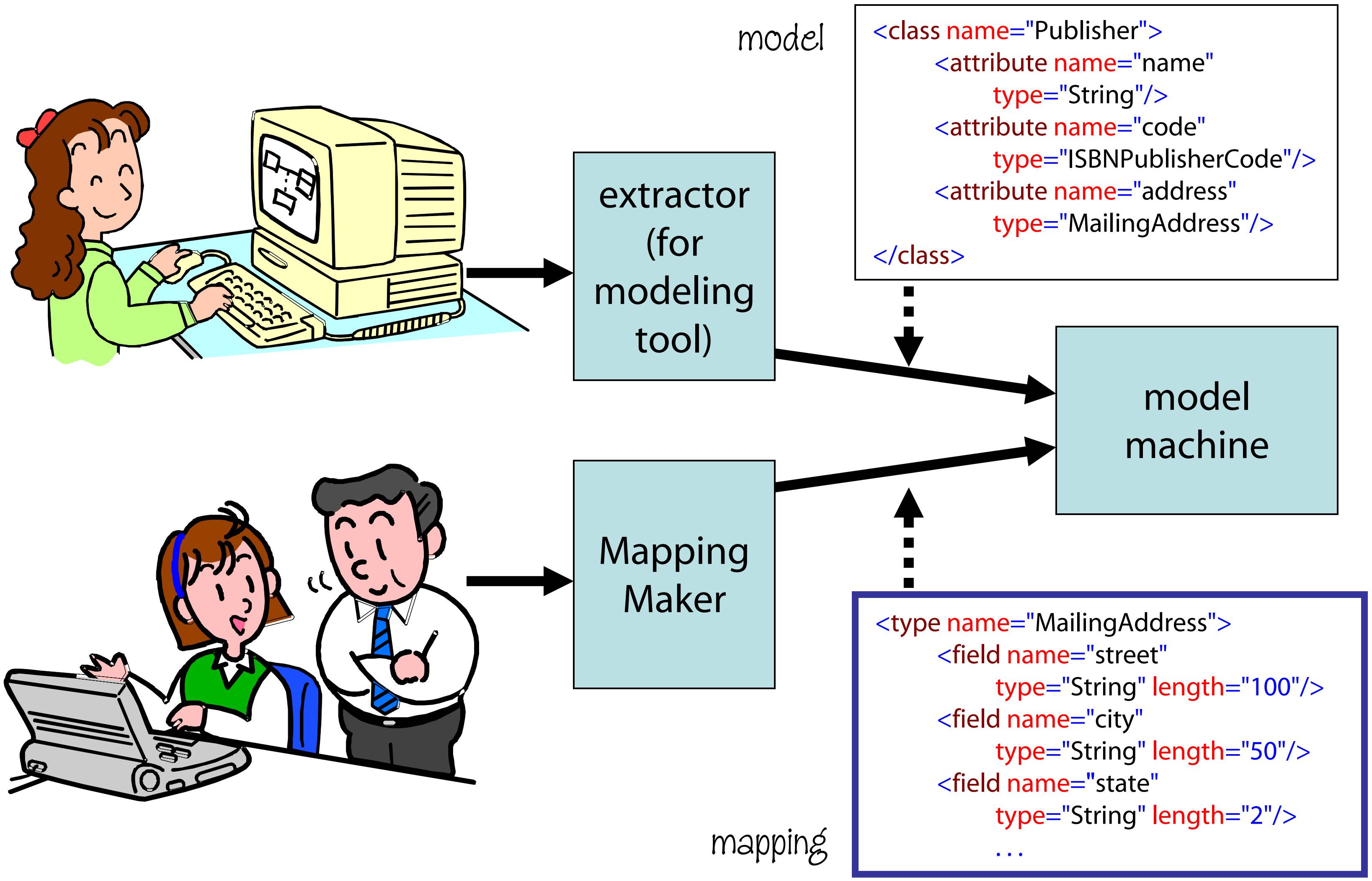
```
<class name="Publisher">
    <attribute name="name"
        type="String"/>
    <attribute name="code"
        type="ISBNPublisherCode"/>
    <attribute name="address"
        type="MailingAddress"/>
</class>
```

```
<type name="MailingAddress">
    <field name="street"
        type="String" length="100"/>
    <field name="city"
        type="String" length="50"/>
    <field name="state"
        type="String" length="2"/>
        . . .
```

type mapping

- Some things are not in the model
  - Model just declares a type
  - How should the machine realize it?
- A mapping directs the VM how to realize something in the model
  - Not part of the application model
  - May be changed separately from the model

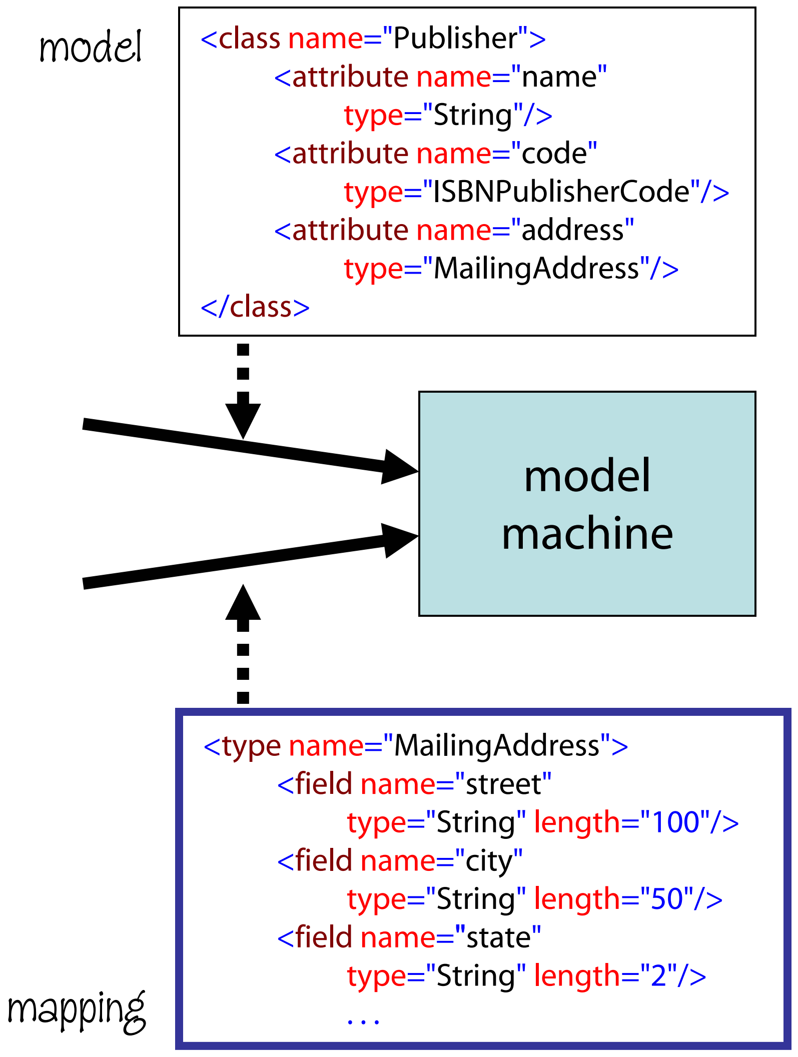
Loading a Model



```
<class name="Publisher">
        <attribute name="name"
                type="String"/>
        <attribute name="code"
                type="ISBNPublisherCode"/>
        <attribute name="address"
                type="MailingAddress"/>
</class>
```

```
<type name="MailingAddress">
        <field name="street"
                type="String" length="100"/>
        <field name="city"
                type="String" length="50"/>
        <field name="state"
                type="String" length="2"/>
                . . .
```

Using the Loaded Model

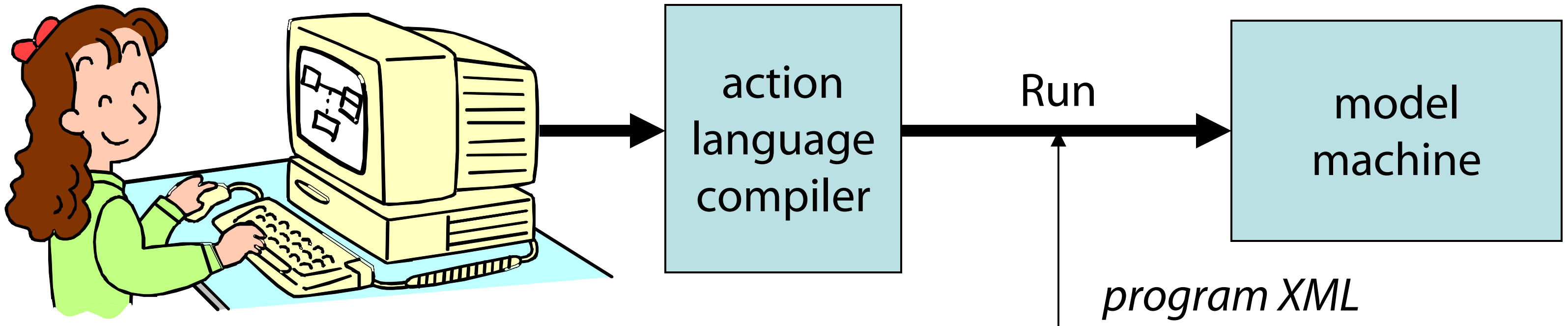


```
<class name="Publisher">
        <attribute name="name"
                type="String"/>
        <attribute name="code"
                type="ISBNPublisherCode"/>
        <attribute name="address"
                type="MailingAddress"/>
</class>
```

```
<type name="MailingAddress">
        <field name="street"
                type="String" length="100"/>
        <field name="city"
                type="String" length="50"/>
        <field name="state"
                type="String" length="2"/>
                . . .
```

- Got a model loaded
- Now what to do?
  - load objects
  - operate on them
  - run the state machines
  - connect the model to the rest of the world

Single Program



```
p := new Publisher {
    .name := "Addison-Wesley";
    -> Book := new Book {
        .title := "Analysis Patterns";
        .copyright := 1997;
    };
    -> Book := new Book {
        .title := "Refactoring";
        .subtitle := "Improving the Design …";
        .copyright := 2001;
    }
}
```

```
<program source="codePath">
    <createObject>
        <createClass name="Publisher"/>
    </createObject>
    <duplicate/>
    <addVariableValue name="p"/>
    <groupAction>
        <readContext/>
        <literalValue value="Addison-Wesley" type="string"/>
        <setAttributeValue name="name"/>
        <readContext/>
        <createObject>
            <createClass name="Book"/>
        </createObject>
        <duplicate/>
        <createLink class="Book"/>
        <groupAction>
            <readContext/>
            <literalValue value="Analysis Patterns" type="string"/>
            <setAttributeValue name="title"/>
            <readContext/>
            <literalValue value="1997" type="integer"/>
            <setAttributeValue name="copyright"/>
        </groupAction> . . .
```

Programs

```
<program source="codePath">
    <createObject>
        <createClass name="Publisher"/>
    </createObject>
    <duplicate/>
    <addVariableValue name="p"/>
    <groupAction>
        <readContext/>
        <literalValue value="Addison-Wesley" type="string"/>
        <setAttributeValue name="name"/>
        <readContext/>
        <createObject>
            <createClass name="Book"/>
        </createObject>
        <duplicate/>
        <createLink class="Book"/>
        <groupAction>
            <readContext/>
            <literalValue value="Analysis Patterns" type="string"/>
            <setAttributeValue name="title"/>
            <readContext/>
            <literalValue value="1997" type="integer"/>
            <setAttributeValue name="copyright"/>
        </groupAction> . . .
```

- Executed for
  - external activity
  - state procedures
  - derived attributes
  - constraint checks
- Written in fundamental actions

Single Program

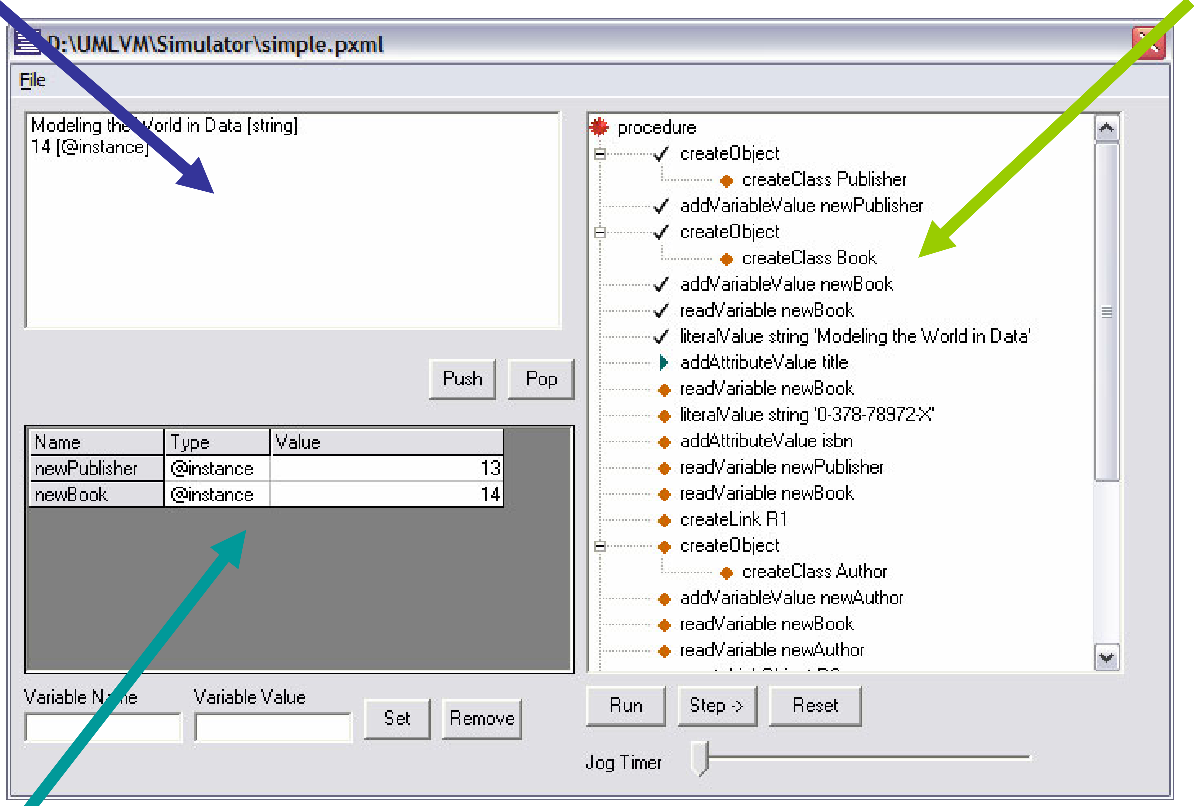
```
p := new Publisher {
    .name := "Addison-Wesley";
    -> Book := new Book {
        .title := "Analysis Patterns";
        .copyright := 1997;
    };
    -> Book := new Book {
        .title := "Refactoring";
        .subtitle := "Improving the Design …";
        .copyright := 2001;
    }
}
```

```
<program source="codePath">
    <createObject>
        <createClass name="Publisher"/>
    </createObject>
    <duplicate/>
    <addVariableValue name="p"/>
    <groupAction>
        <readContext/>
        <literalValue value="Addison-Wesley" type="string"/>
        <setAttributeValue name="name"/>
        <readContext/>
        <createObject>
            <createClass name="Book"/>
        </createObject>
        <duplicate/>
        <createLink class="Book"/>
        <groupAction>
            <readContext/>
            <literalValue value="Analysis Patterns" type="string"/>
            <setAttributeValue name="title"/>
            <readContext/>
            <literalValue value="1997" type="integer"/>
            <setAttributeValue name="copyright"/>
        </groupAction> . . .
```

Execution Context

Stack

Program (as actions)



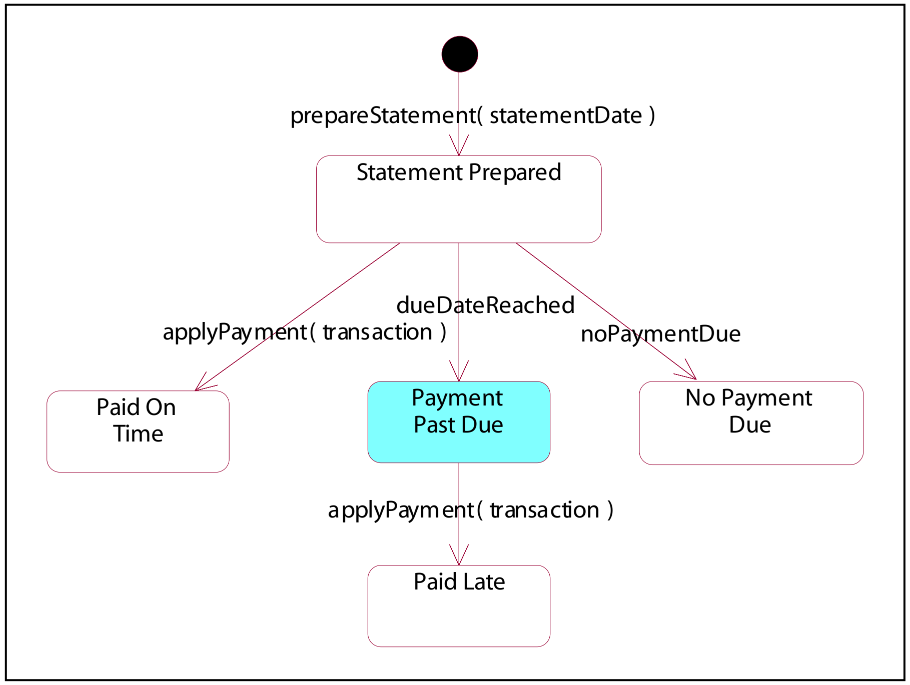
```
Modeling the World in Data [string]
14 [@instance]
```

Name	Type	Value
newPublisher	@instance	13
newBook	@instance	14

```
procedure
  ✓ createObject
      ◆ createClass Publisher
  ✓ addVariableValue newPublisher
  ✓ createObject
      ◆ createClass Book
  ✓ addVariableValue newBook
  ✓ readVariable newBook
  ✓ literalValue string 'Modeling the World in Data'
  ▸ addAttributeValue title
  ◆ readVariable newBook
  ◆ literalValue string '0-378-78972-X'
  ◆ addAttributeValue isbn
  ◆ readVariable newPublisher
  ◆ readVariable newBook
  ◆ createLink R1
  ◆ createObject
      ◆ createClass Author
  ◆ addVariableValue newAuthor
  ◆ readVariable newBook
  ◆ readVariable newAuthor
```

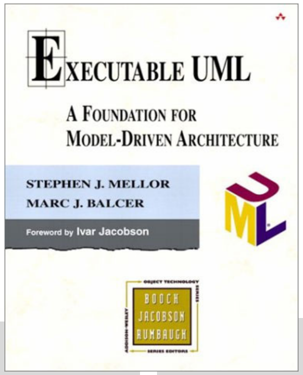
Local Variables (named data flows)

State Machines



- Each object has its own independently-executing state machine
- State machines respond to events sent by actions
- State machines are *the* mechanism for concurrency
- Can visualize with sequence diagrams

Time & Event Rules



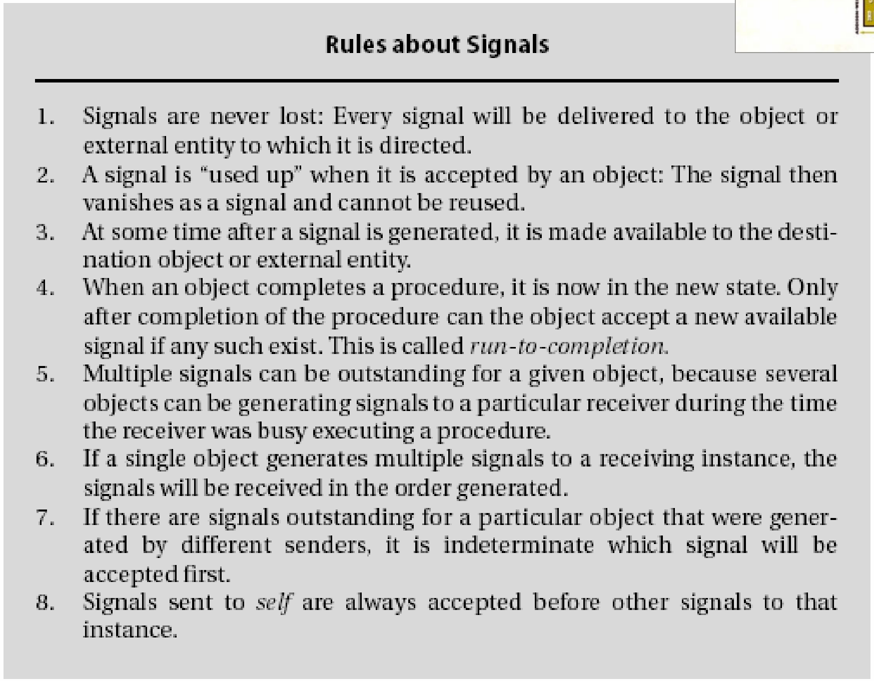
EXECUTABLE UML

A FOUNDATION FOR MODEL-DRIVEN ARCHITECTURE

STEPHEN J. MELLOR
MARC J. BALCER

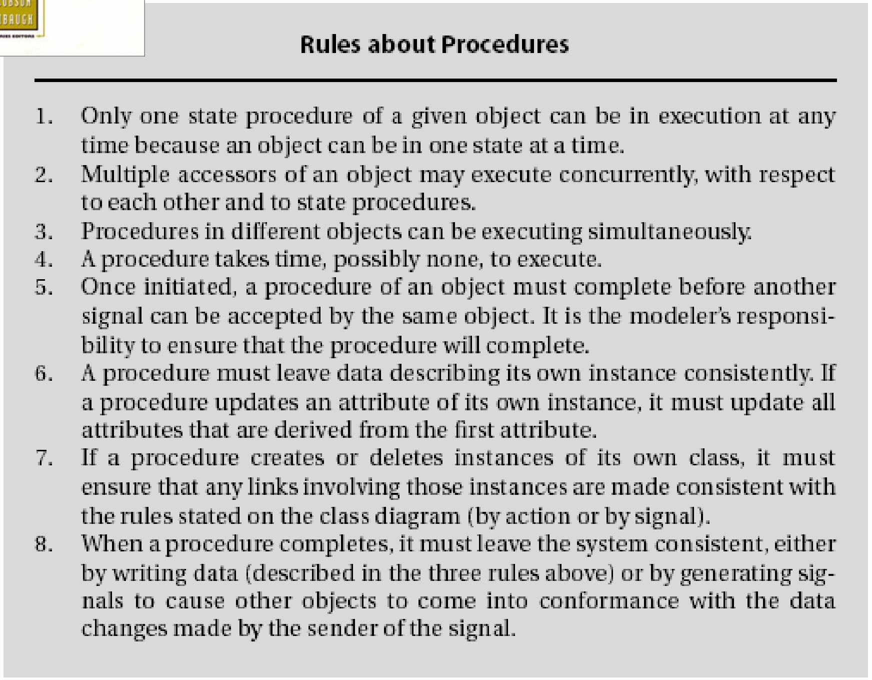
Foreword by Ivar Jacobson

BOOCH JACOBSON RUMBAUGH



Rules about Signals

1. Signals are never lost: Every signal will be delivered to the object or external entity to which it is directed.
2. A signal is "used up" when it is accepted by an object: The signal then vanishes as a signal and cannot be reused.
3. At some time after a signal is generated, it is made available to the destination object or external entity.
4. When an object completes a procedure, it is now in the new state. Only after completion of the procedure can the object accept a new available signal if any such exist. This is called *run-to-completion.*
5. Multiple signals can be outstanding for a given object, because several objects can be generating signals to a particular receiver during the time the receiver was busy executing a procedure.
6. If a single object generates multiple signals to a receiving instance, the signals will be received in the order generated.
7. If there are signals outstanding for a particular object that were generated by different senders, it is indeterminate which signal will be accepted first.
8. Signals sent to *self* are always accepted before other signals to that instance.

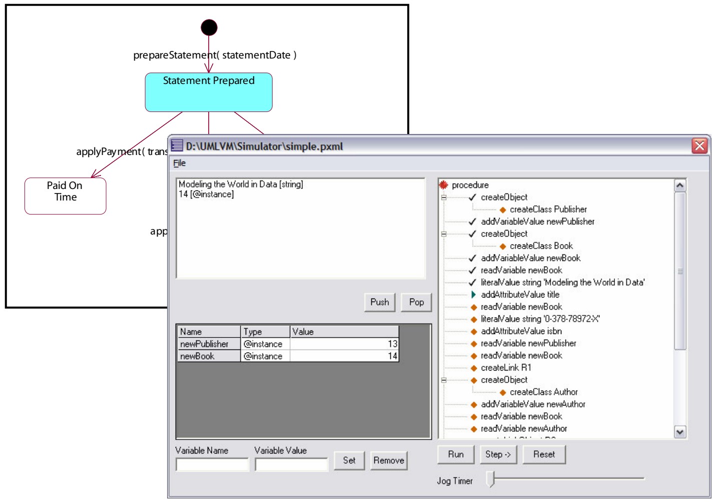


Rules about Procedures

1. Only one state procedure of a given object can be in execution at any time because an object can be in one state at a time.
2. Multiple accessors of an object may execute concurrently, with respect to each other and to state procedures.
3. Procedures in different objects can be executing simultaneously.
4. A procedure takes time, possibly none, to execute.
5. Once initiated, a procedure of an object must complete before another signal can be accepted by the same object. It is the modeler's responsibility to ensure that the procedure will complete.
6. A procedure must leave data describing its own instance consistently. If a procedure updates an attribute of its own instance, it must update all attributes that are derived from the first attribute.
7. If a procedure creates or deletes instances of its own class, it must ensure that any links involving those instances are made consistent with the rules stated on the class diagram (by action or by signal).
8. When a procedure completes, it must leave the system consistent, either by writing data (described in the three rules above) or by generating signals to cause other objects to come into conformance with the data changes made by the sender of the signal.

Concurrency

- Each object has its own execution context



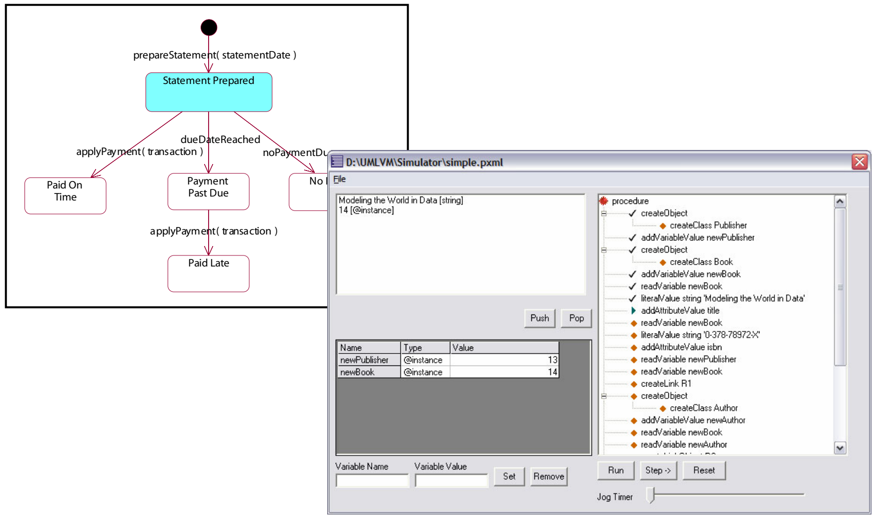
D:\UMLVM\Simulator\simple.pxml

File

```
Modeling the World in Data [string]
14 [@instance]
```

Name	Type	Value
newPublisher	@instance	13
newBook	@instance	14

```
procedure
  ✓ createObject
      ◆ createClass Publisher
  ✓ addVariableValue newPublisher
  ✓ createObject
      ◆ createClass Book
  ✓ addVariableValue newBook
  ✓ readVariable newBook
  ✓ literalValue string 'Modeling the World in Data'
  ▶ addAttributeValue title
  ◆ readVariable newBook
  ◆ literalValue string '0-378-78972-X'
  ◆ addAttributeValue isbn
  ◆ readVariable newPublisher
  ◆ readVariable newBook
  ◆ createLink R1
  ◆ createObject
      ◆ createClass Author
  ◆ addVariableValue newAuthor
  ◆ readVariable newBook
  ◆ readVariable newAuthor
```


prepareStatement(statementDate)
Statement Prepared
applyPayment(transaction)
dueDateReached
noPaymentDu
Paid On Time
Payment Past Due
No
applyPayment(transaction)
Paid Late

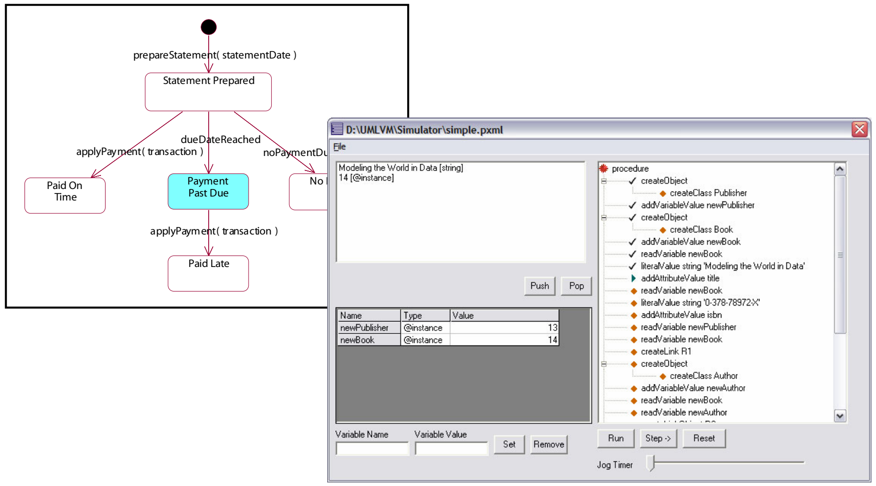
D:\UMLVM\Simulator\simple.pxml

File

Modeling the World in Data [string]
14 [@instance]

Name	Type	Value
newPublisher	@instance	13
newBook	@instance	14

procedure
createObject
createClass Publisher
addVariableValue newPublisher
createObject
createClass Book
addVariableValue newBook
readVariable newBook
literalValue string 'Modeling the World in Data'
addAttributeValue title
readVariable newBook
literalValue string '0-378-78972-X'
addAttributeValue isbn
readVariable newPublisher
readVariable newBook
createLink R1
createObject
createClass Author
addVariableValue newAuthor
readVariable newBook
readVariable newAuthor


prepareStatement(statementDate)
Statement Prepared
applyPayment(transaction)
dueDateReached
noPaymentDu
Paid On Time
Payment Past Due
No
applyPayment(transaction)
Paid Late

D:\UMLVM\Simulator\simple.pxml

File

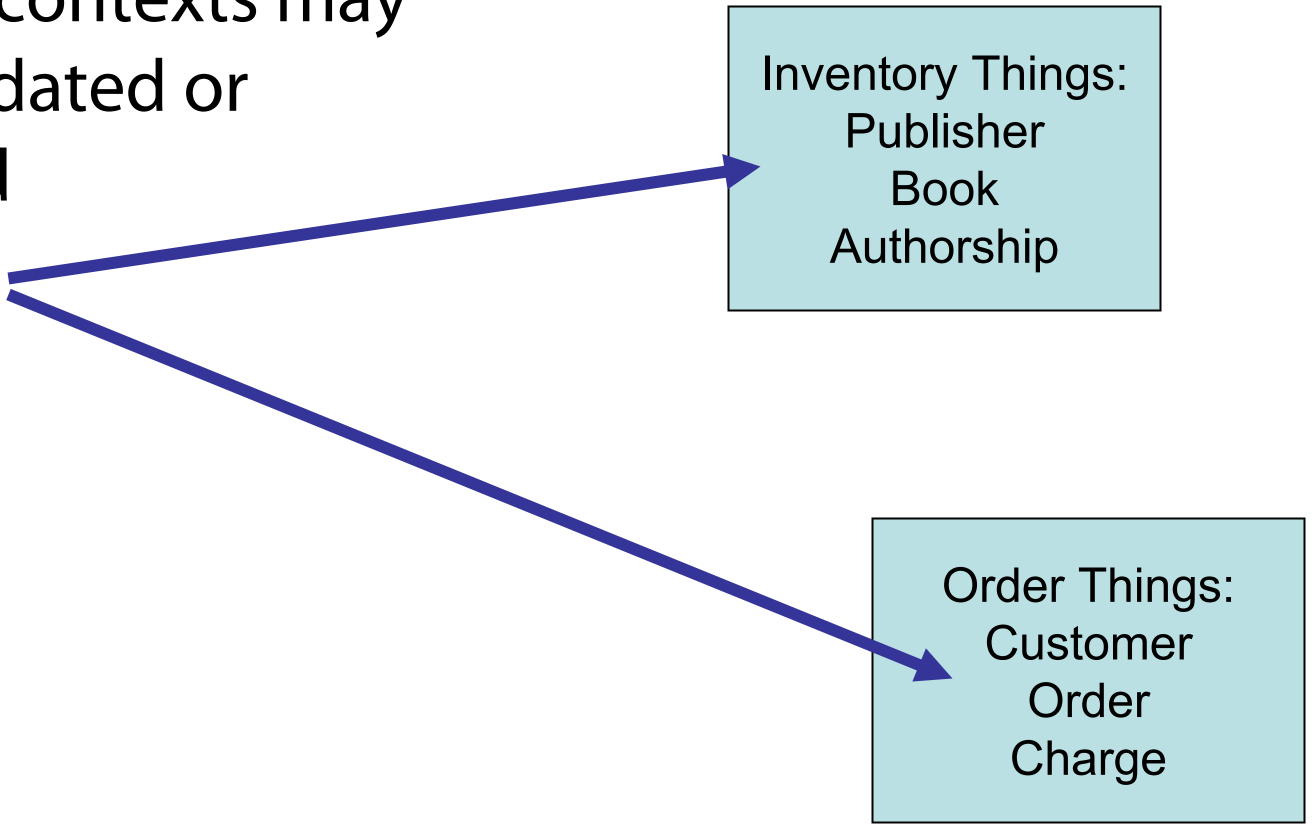
Modeling the World in Data [string]
14 [@instance]

Name	Type	Value
newPublisher	@instance	13
newBook	@instance	14

procedure
createObject
createClass Publisher
addVariableValue newPublisher
createObject
createClass Book
addVariableValue newBook
readVariable newBook
literalValue string 'Modeling the World in Data'
addAttributeValue title
readVariable newBook
literalValue string '0-378-78972-X'
addAttributeValue isbn
readVariable newPublisher
readVariable newBook
createLink R1
createObject
createClass Author
addVariableValue newAuthor
readVariable newBook
readVariable newAuthor

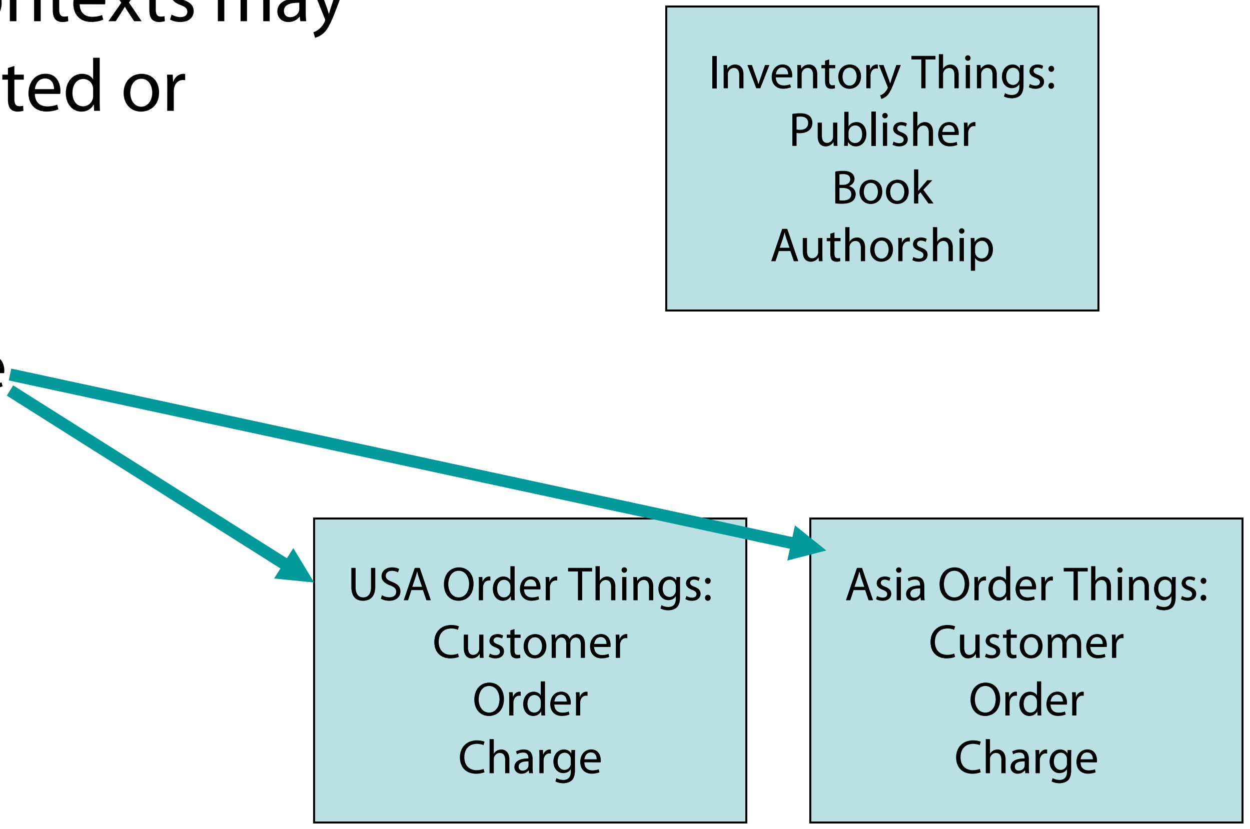
Distribution

- Execution contexts may be consolidated or distributed
  - by class



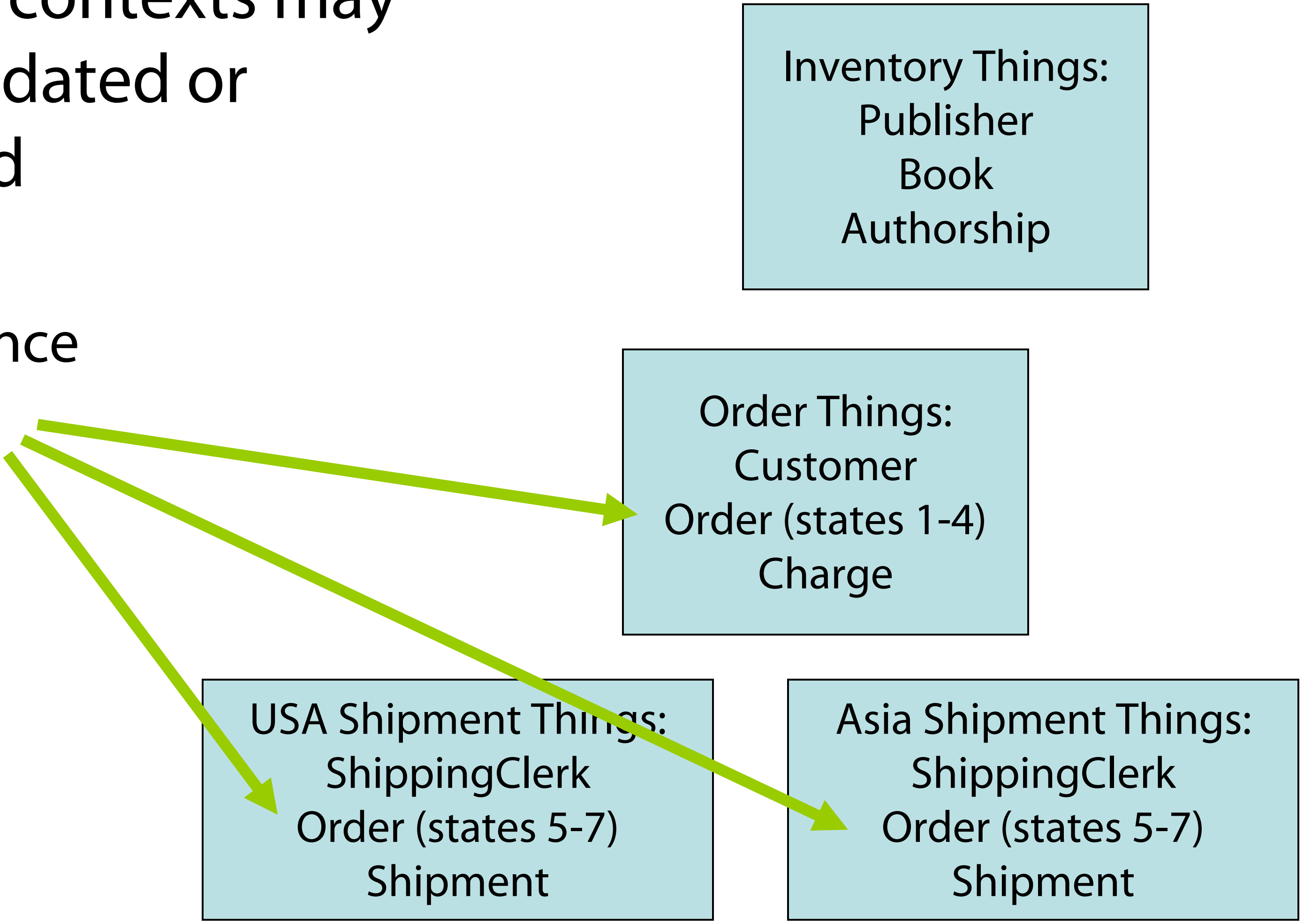
Distribution

- Execution contexts may be consolidated or distributed
  - by class
  - by instance



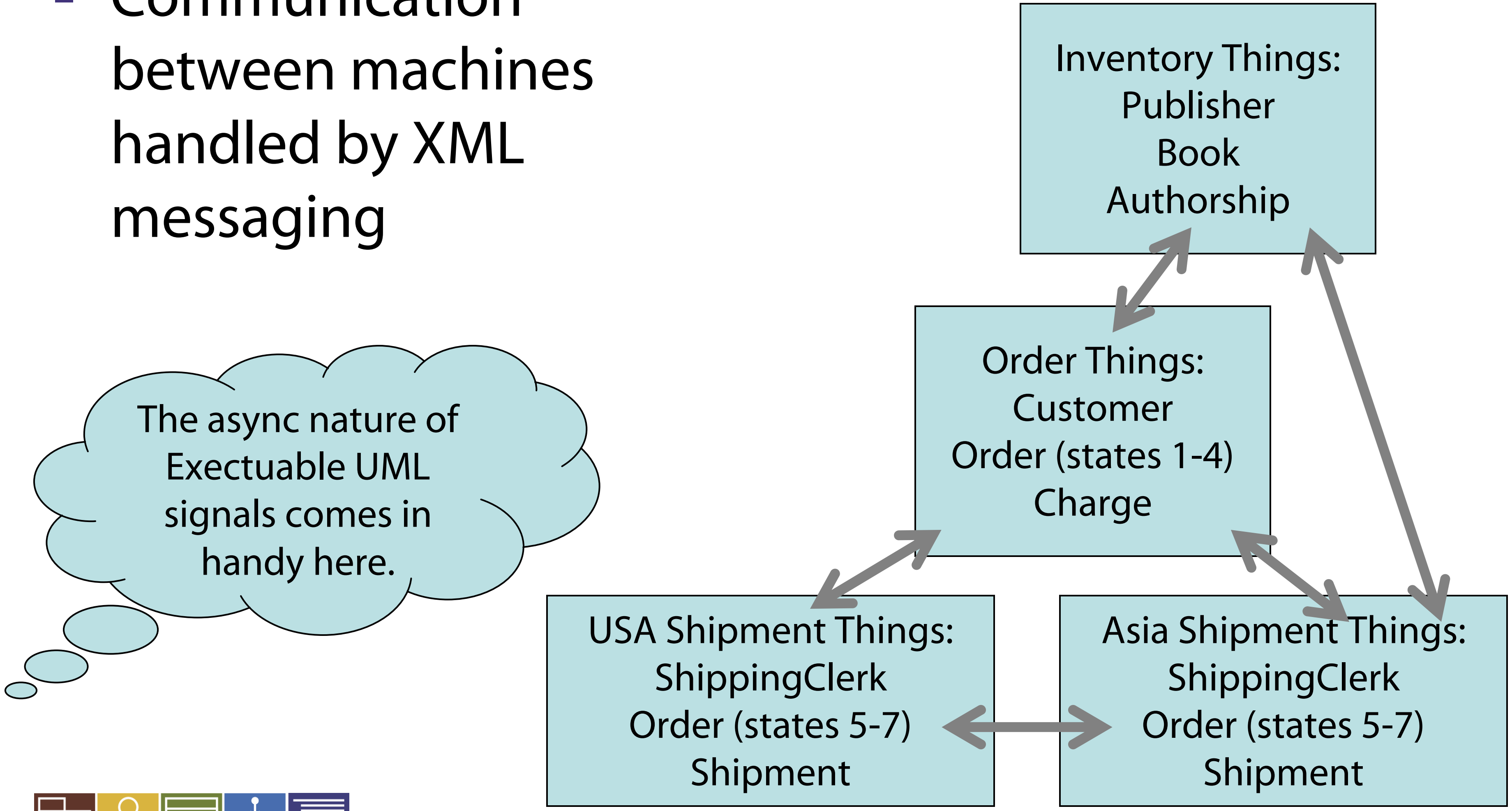
Distribution

- Execution contexts may be consolidated or distributed
  - by class
  - by instance
  - by state

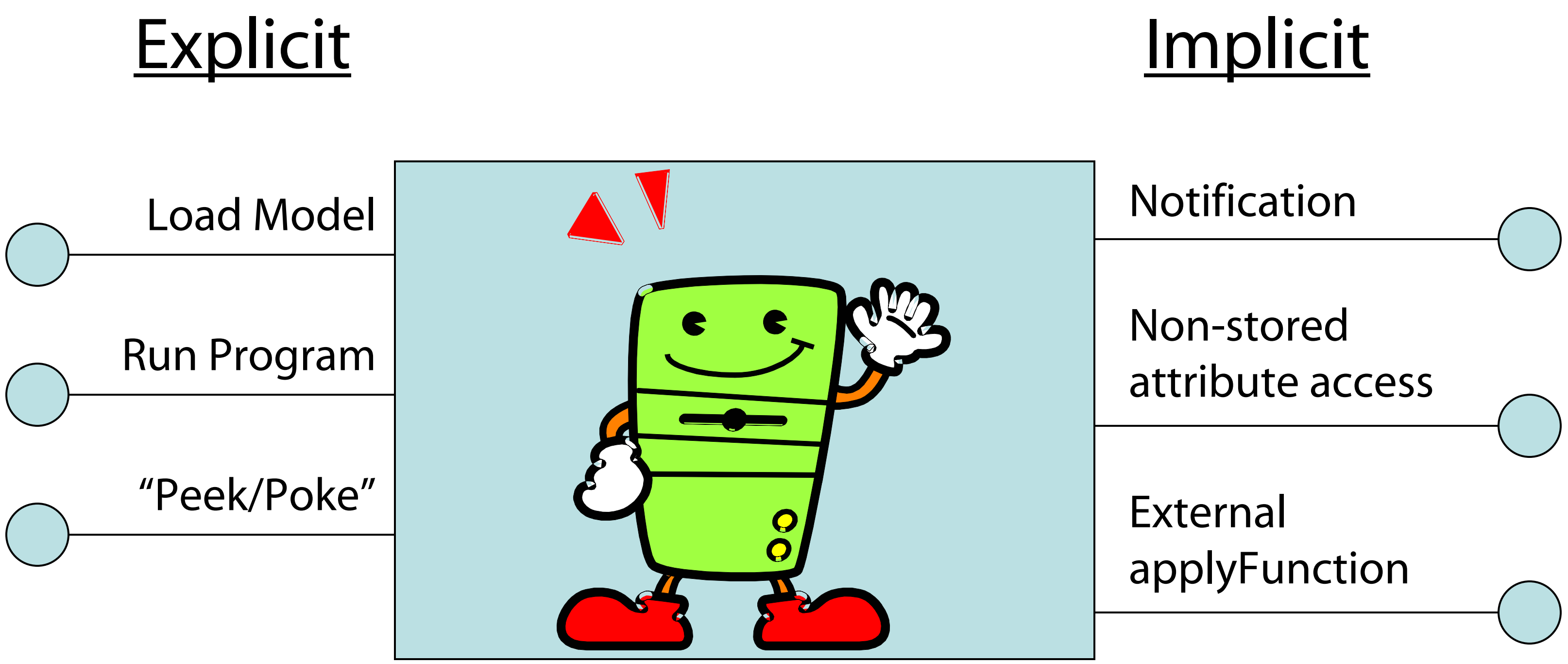


Distribution

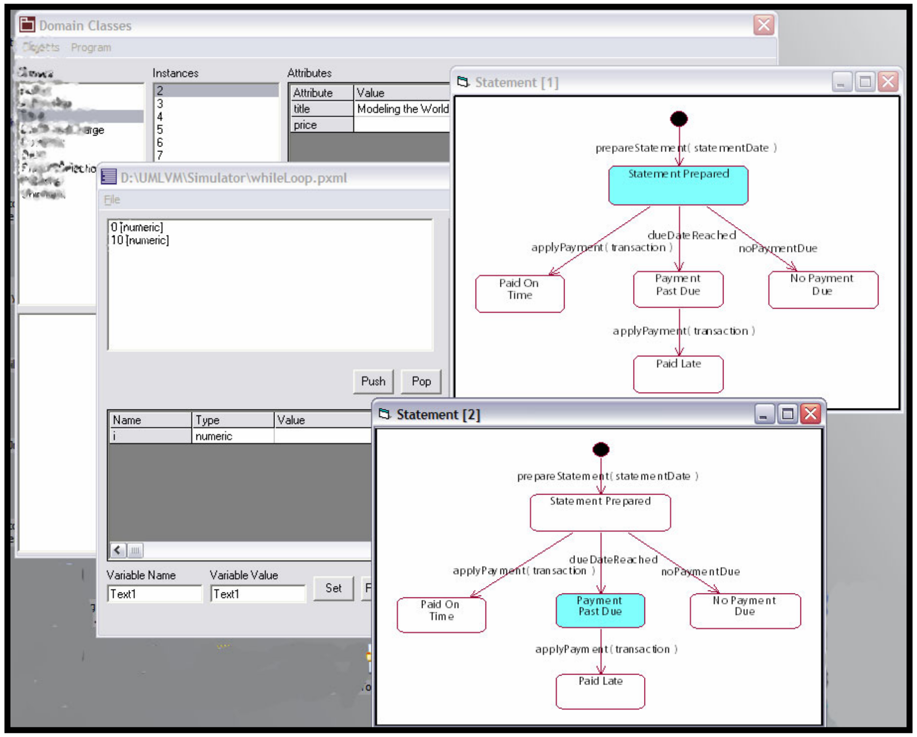
- Communication between machines handled by XML messaging


The async nature of Exectuable UML signals comes in handy here.
Inventory Things: Publisher Book Authorship
Order Things: Customer Order (states 1-4) Charge
USA Shipment Things: ShippingClerk Order (states 5-7) Shipment
Asia Shipment Things: ShippingClerk Order (states 5-7) Shipment

Connecting to the World



Example Applications



- Scripted Model Verifier
  - Windows application
  - Use notifications to show changes in real-time

Example Applications

- Web Application Engine
  - COM component
  - Callable from within an ASP program
  - No use of notifications
  - Form postings compiled to inbound programs

Contents

- Concept
- Architecture
  - Static Model Import
  - Single Program
  - State Machines
  - Connecting to the World
- Outcomes
- Epilogue

Outcomes

- TALL action language
- Metamodel
- XML for Executable UML
- Virtual Machine Zero
- Development Tools

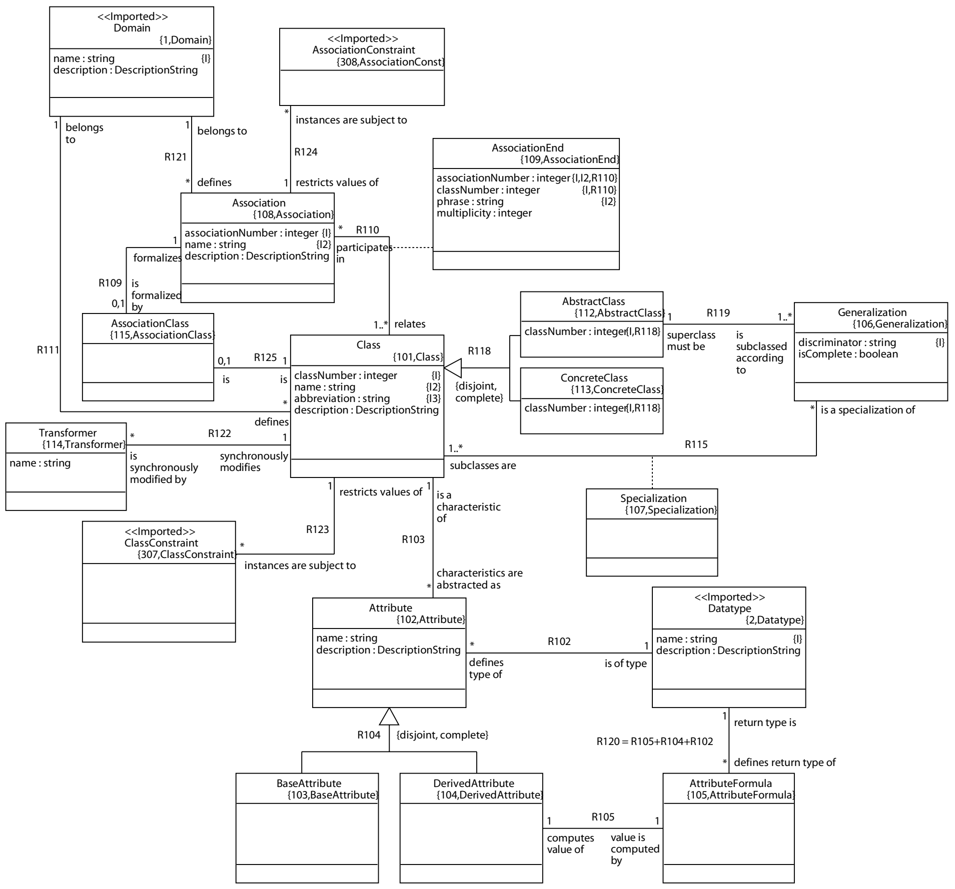
TALL action language

```
p := new Publisher {
    .name := "Addison-Wesley";
    -> Book := new Book {
        .title := "Analysis Patterns";
        .copyright := 1997;
    };
    -> Book := new Book {
        .title := "Refactoring";
        .subtitle := "Improving the Design …";
        .copyright := 2001;
    }
}
```

```
foreach readyOrder in Order[.approved = true] {
    ^readyOrder.packAndShip();
}
```

- Functional
- True to action semantics
- Traditional-looking syntax

Metamodel



<<Imported>> Domain {1,Domain}
name : string {I}
description : DescriptionString
<<Imported>> AssociationConstraint {308,AssociationConst}
instances are subject to
R124
restricts values of
belongs to
R121
defines
belongs to
R111
Association {108,Association}
associationNumber : integer {I}
name : string {I2}
description : DescriptionString
R110
participates in
AssociationEnd {109,AssociationEnd}
associationNumber : integer {I,I2,R110}
classNumber : integer {I,R110}
phrase : string {I2}
multiplicity : integer
formalizes
R109
is formalized by
AssociationClass {115,AssociationClass}
R125
is
relates
Class {101,Class}
classNumber : integer {I}
name : string {I2}
abbreviation : string {I3}
description : DescriptionString
R118
{disjoint, complete}
AbstractClass {112,AbstractClass}
classNumber : integer {I,R118}
ConcreteClass {113,ConcreteClass}
classNumber : integer {I,R118}
R119
superclass must be
is subclassed according to
Generalization {106,Generalization}
discriminator : string {I}
isComplete : boolean
is a specialization of
R115
subclasses are
Specialization {107,Specialization}
Transformer {114,Transformer}
name : string
R122
is synchronously modified by
synchronously modifies
defines
<<Imported>> ClassConstraint {307,ClassConstraint}
instances are subject to
R123
restricts values of
is a characteristic of
R103
characteristics are abstracted as
Attribute {102,Attribute}
name : string
description : DescriptionString
R102
defines type of
is of type
<<Imported>> Datatype {2,Datatype}
name : string {I}
description : DescriptionString
R104
{disjoint, complete}
return type is
R120 = R105+R104+R102
defines return type of
BaseAttribute {103,BaseAttribute}
DerivedAttribute {104,DerivedAttribute}
R105
computes value of
value is computed by
AttributeFormula {105,AttributeFormula}

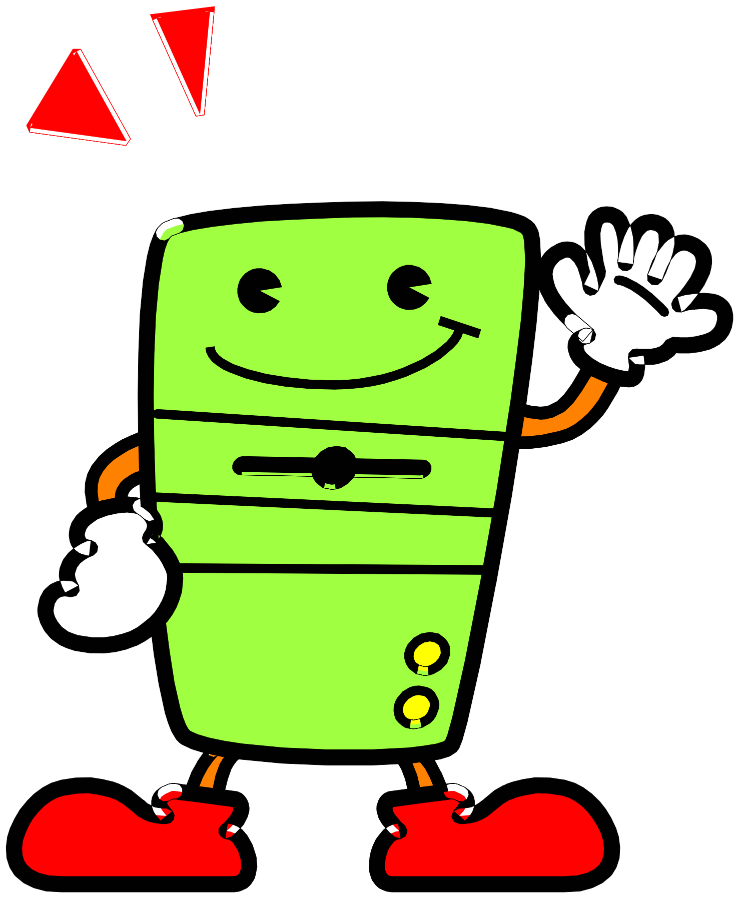
- Fundamental for translation
- Foundation for tools and interchange formats

XML for Executable UML

```
<domain name="Bookstore">
    <type name="String"/>
    <type name="ISBNPublisherCode"/>
    <type name="MailingAddress"/>
    <type name="Year"/>
    <type name="Money"/>
    <class name="Publisher">
        <attribute name="name" type="String"/>
        <attribute name="code" type="ISBNPublisherCode"/>
        <attribute name="address" type="MailingAddress"/>
    </class>
    <class name="Book">
        <attribute name="title" type="String"/>
        <attribute name="subtitle" type="String"/>
        <attribute name="copyright" type="Year"/>
        <attribute name="unitPrice" type="Money"/>
    </class>
    <association name="R1">
        <associationEnd phrase="produces and markets"
            multiplicity="0..*" class="Book"/>
        <associationEnd phrase="is produced and marketed by"
            multiplicity="1" class="Publisher"/>
    </association>
</domain>
```

- Just Enough
  - Only want what’s in the executable profile
  - Models, not diagrams
  - Includes actions

Virtual Machine One



- Single Processor
- Windows
- MS Access / Jet persistence
- → Only because it's cheap & ubiquitous

Development Tools

- Extractors
  - Rose
  - BridgePoint
- Generator
  - based upon domain chart
- Model Verifier
  - based upon VM-zero

Contents

Epilogue

- Building the VM helped to
  - check the action semantics
  - build an Executable UML metamodel
  - start a new business in *this* economy!