Executable and Translatable UML

Stephen J. Mellor

Marc J. Balcer

Contents

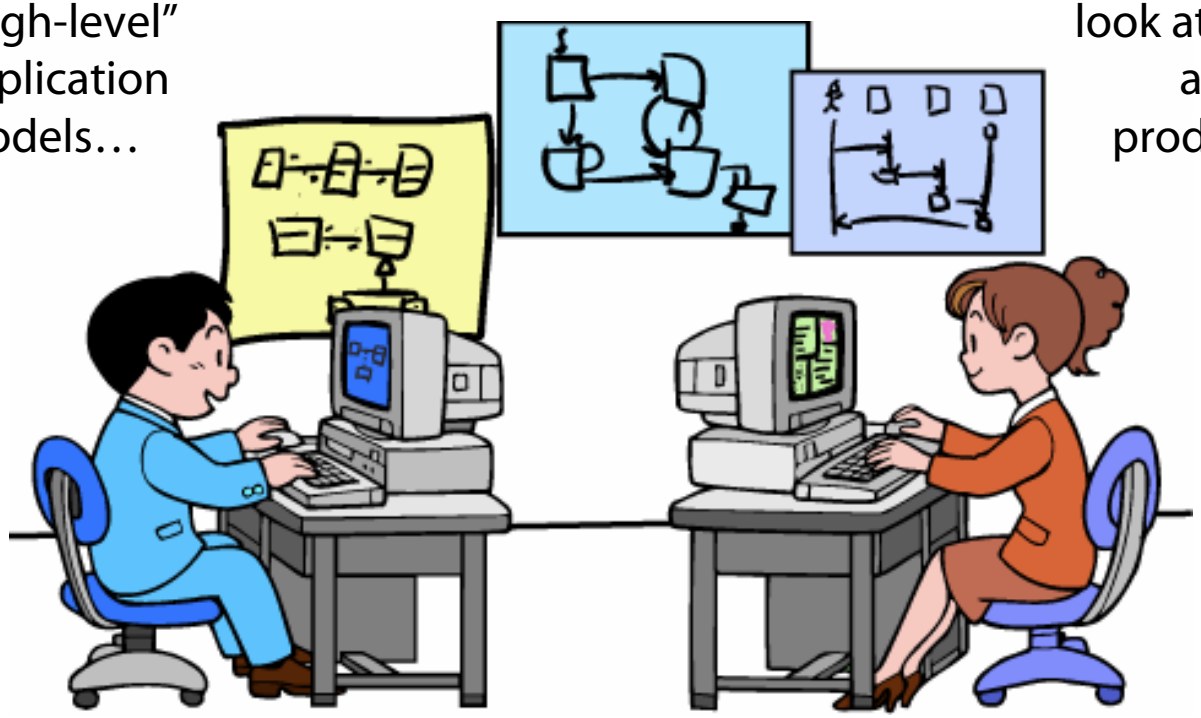
- What's the problem?
- UML Models
- Executable UML elements
- Executable UML behavior
- The Repository
- Model Compiler Concepts
- How model compilers work
- MDA

What's the Problem?

Analysts create "high-level" application models…

…print them out…

… then the developers look at the models and write the production code

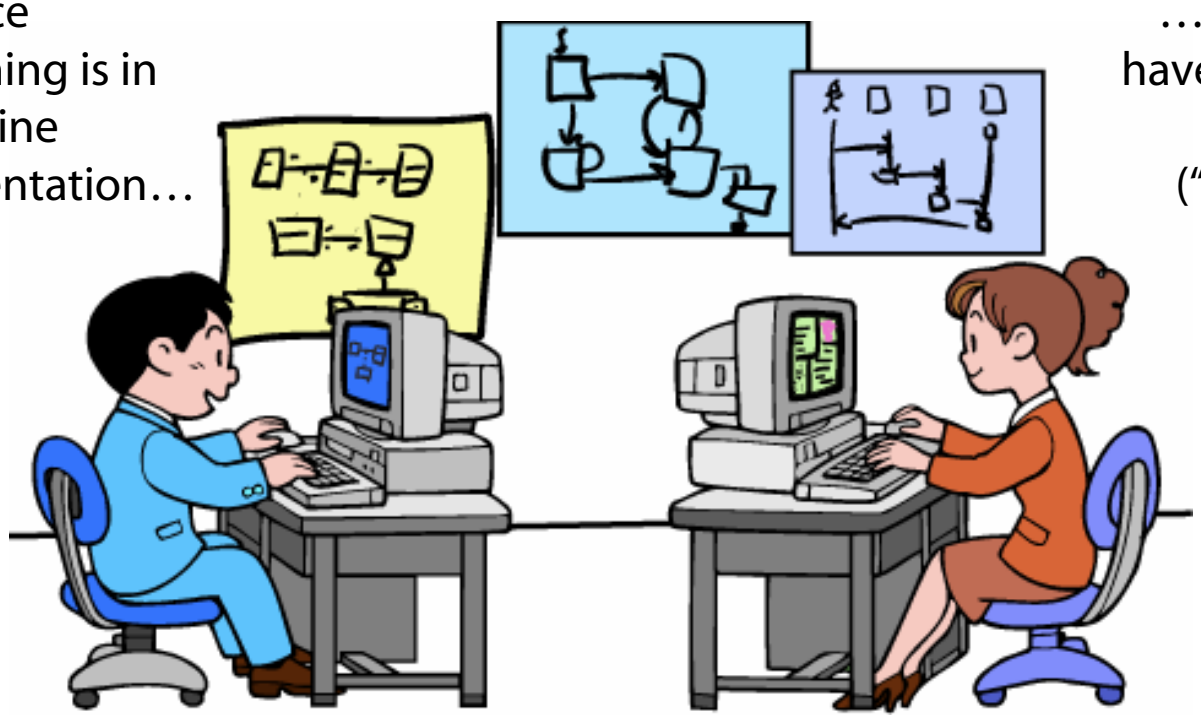


These models are independent of the implementation technology

Coding adds this "design detail"

What's the Problem?

But once something is in a machine representation…



…we shouldn't have to manually reinterpret it ("elaborate it") into another.

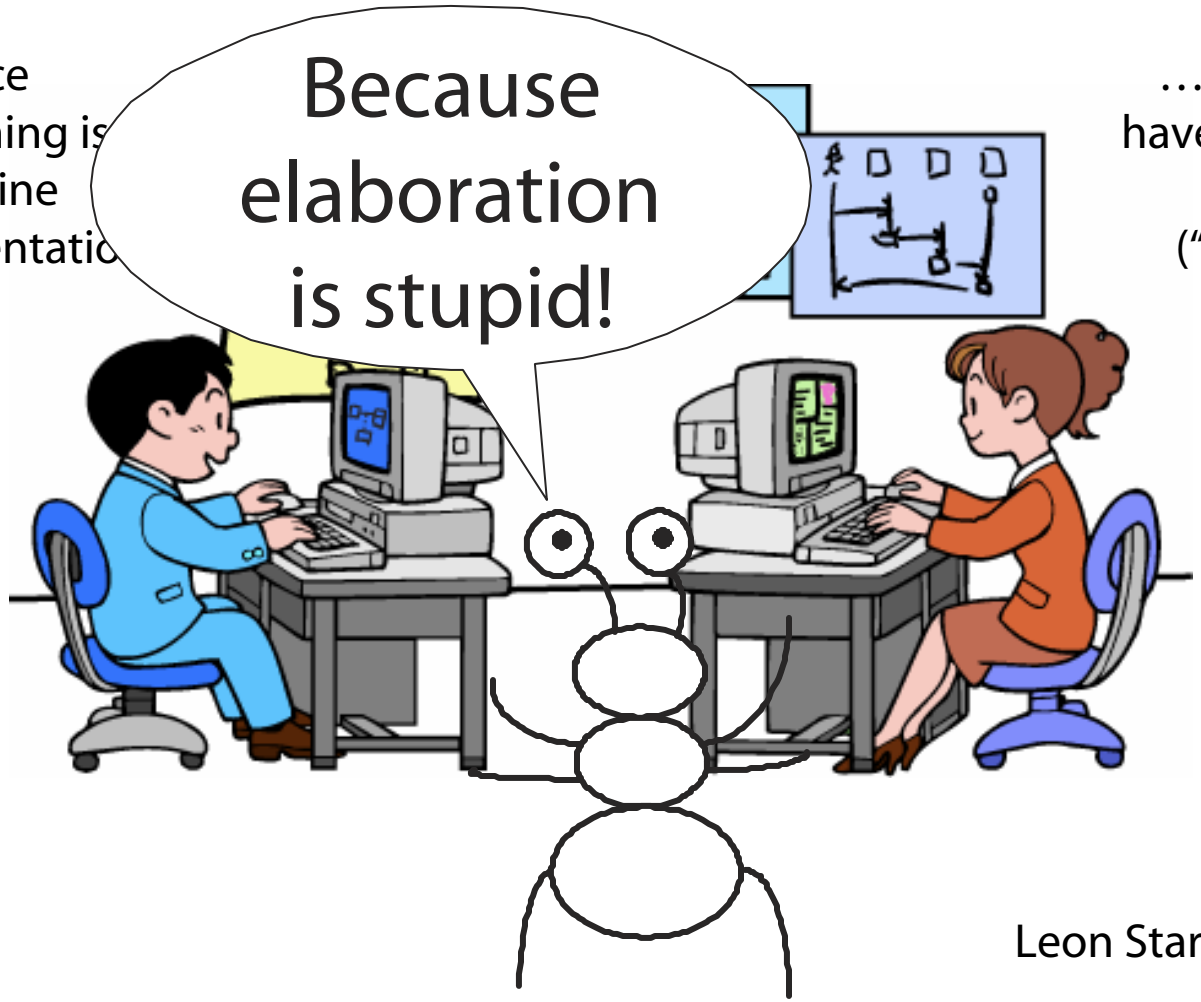
The analyst's diagrams don't have to be complete, correct, or consistent models

Keeping the diagrams in sync with the code is a daunting task.

The rework introduces development errors and obscures analyst errors

What's the Problem?

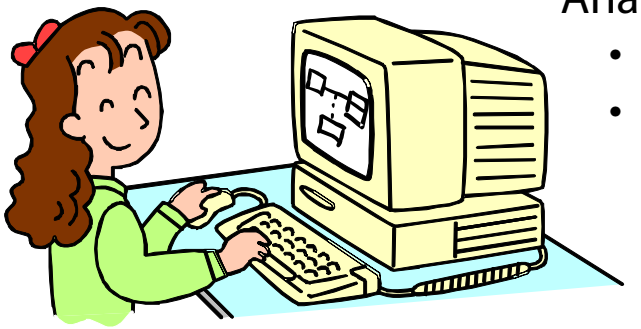
But once something is a machine representatio



…we shouldn't have to manually reinterpret it ("elaborate it") into another.

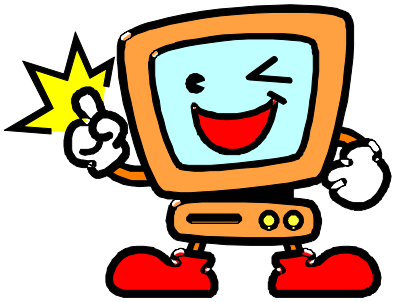
The Ant appears courtesy of Leon Starr, *Executable UML*

Executable Models



Analysts

- create executable models
- test those executable models

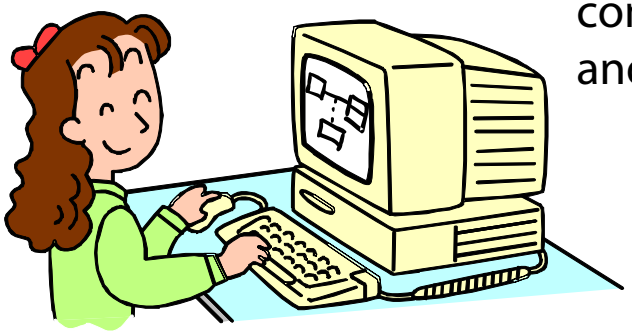


…and compile the models to produce a running system

Developers

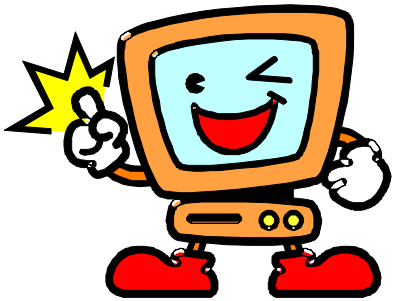
- buy, build, or adapt a model compiler for a software architecture
- map model elements to the architecture…

Executable Models



The analyst's models must be complete, correct, and consistent—and the models are verifiable.

The process always moves forward…



…there's no editing of the translated code, no round-tripping, no reverse-engineering, no getting out of sync.

The software deign decisions are formalized in the mappings and translation archetypes

Contents

- What's the problem?
- UML Models
- Executable UML elements
- Executable UML behavior
- The Repository
- Model Compiler Concepts
- How model compilers work
- MDA

What's UML?

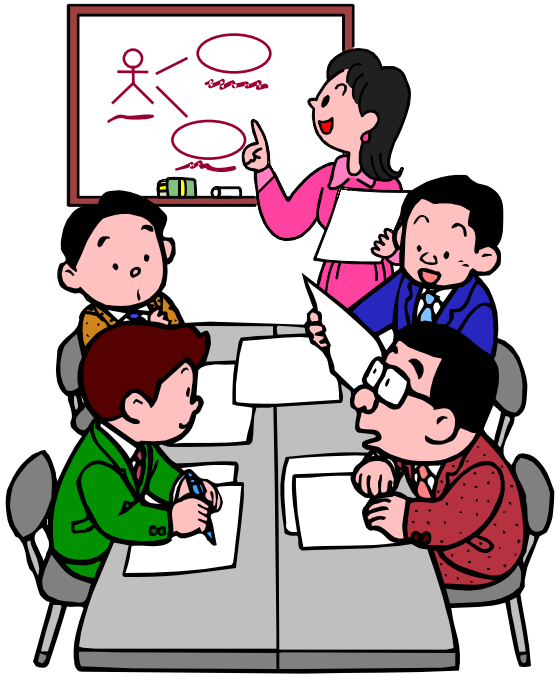
"The Unified Modeling Language is a language for specifying, constructing, visualizing, and documenting artifacts of a software-intensive system"

—The UML Summary

What's UML?

- Common notation for OOA / OOD
- Goal was to eliminate the notation wars
- One notation for many purposes
- Very large
- Concept of "profiles" for different uses

UML in Practice

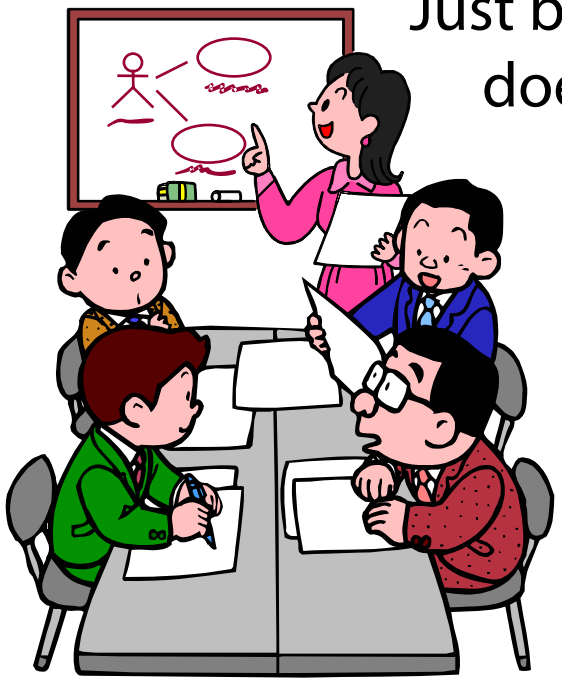


Describing
a problem
“Requirements”

Modeling
a solution
“Analysis”

Documenting
software
“Design”

UML in Practice

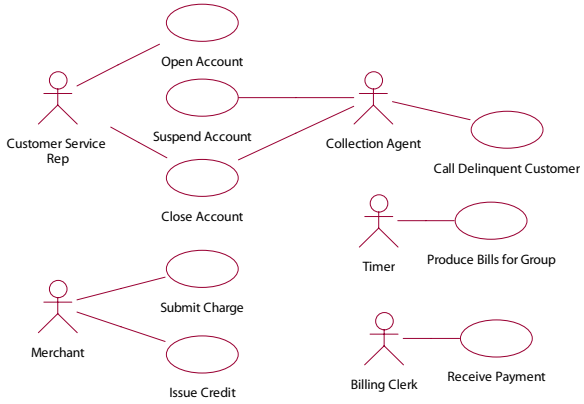


Just because these use the same notation (UML) doesn't mean these are the same problem.

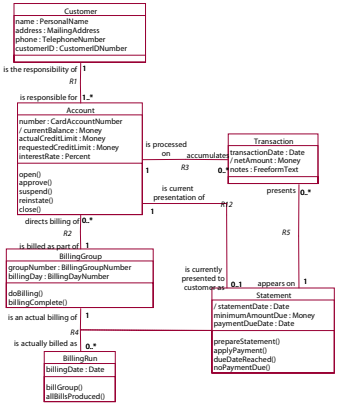
The models are more than just the observable behavior to the user

The software design does not follow from the model of the problem

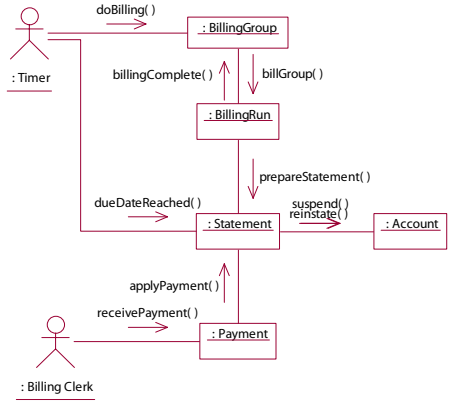
Notation Semantics



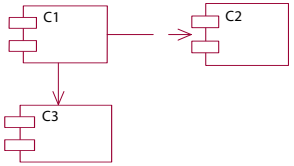
Use Case Diagram



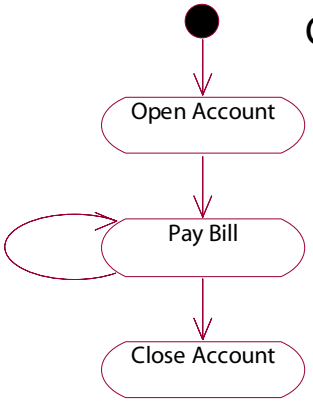
Class Diagram



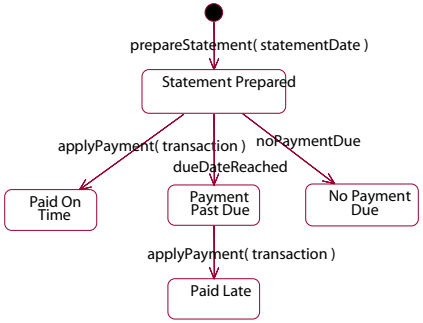
Collaboration Diagram



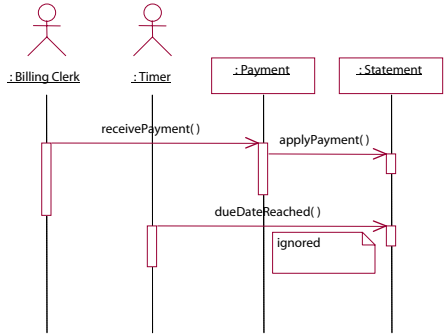
Component Diagram



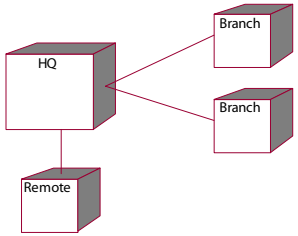
Activity Diagram



State Diagram



Sequence Diagram



Deployment Diagram

Each diagram is well-defined,
but how do they fit together?

Contents

- What's the problem?
- UML Models
- Executable UML elements
- Executable UML behavior
- The Repository
- Model Compiler Concepts
- How model compilers work
- MDA

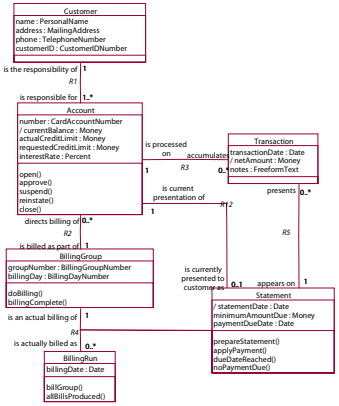
UML in Practice

Describing
a problem
"Requirements"

Modeling
a solution
"Analysis"

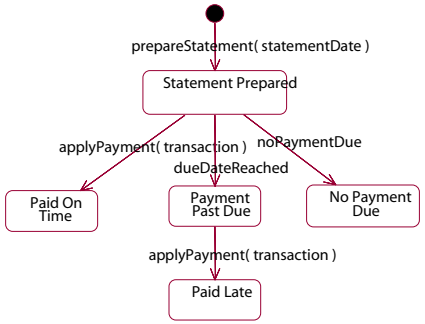
Documenting
software
"Design"

UML Profiles



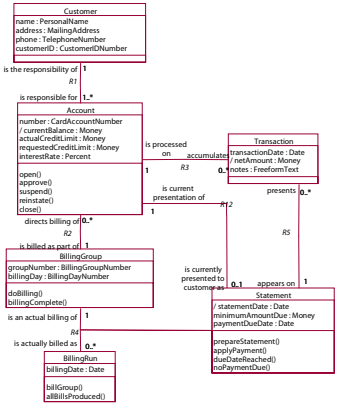
Information Model
(things in the problem and how they are related)

A profile defines how these diagrams represent models



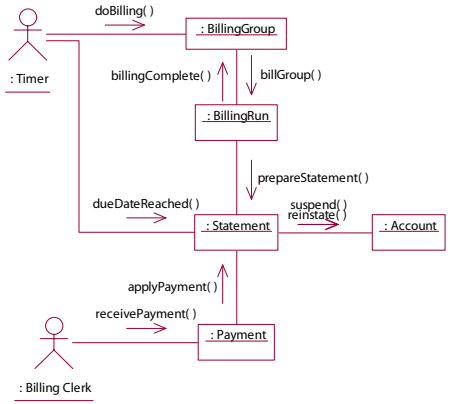
State Model
(lifecycle of an instance of a class)

UML Profiles

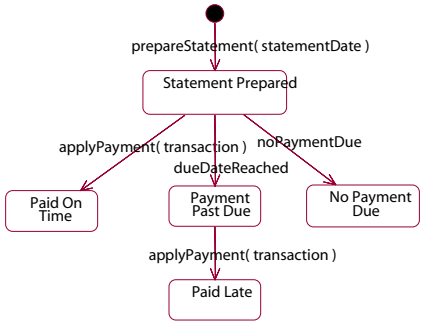


Information Model
(things in the problem and how they are related)

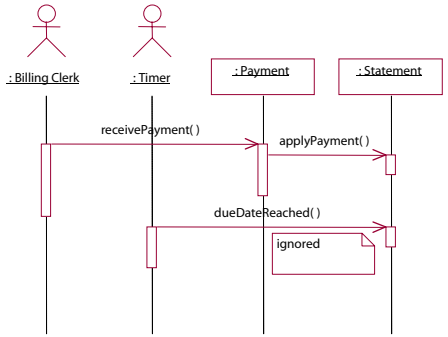
It also defines a set of diagrams that are derived from the basic models



Object Communication Model
(summary of object interactions)

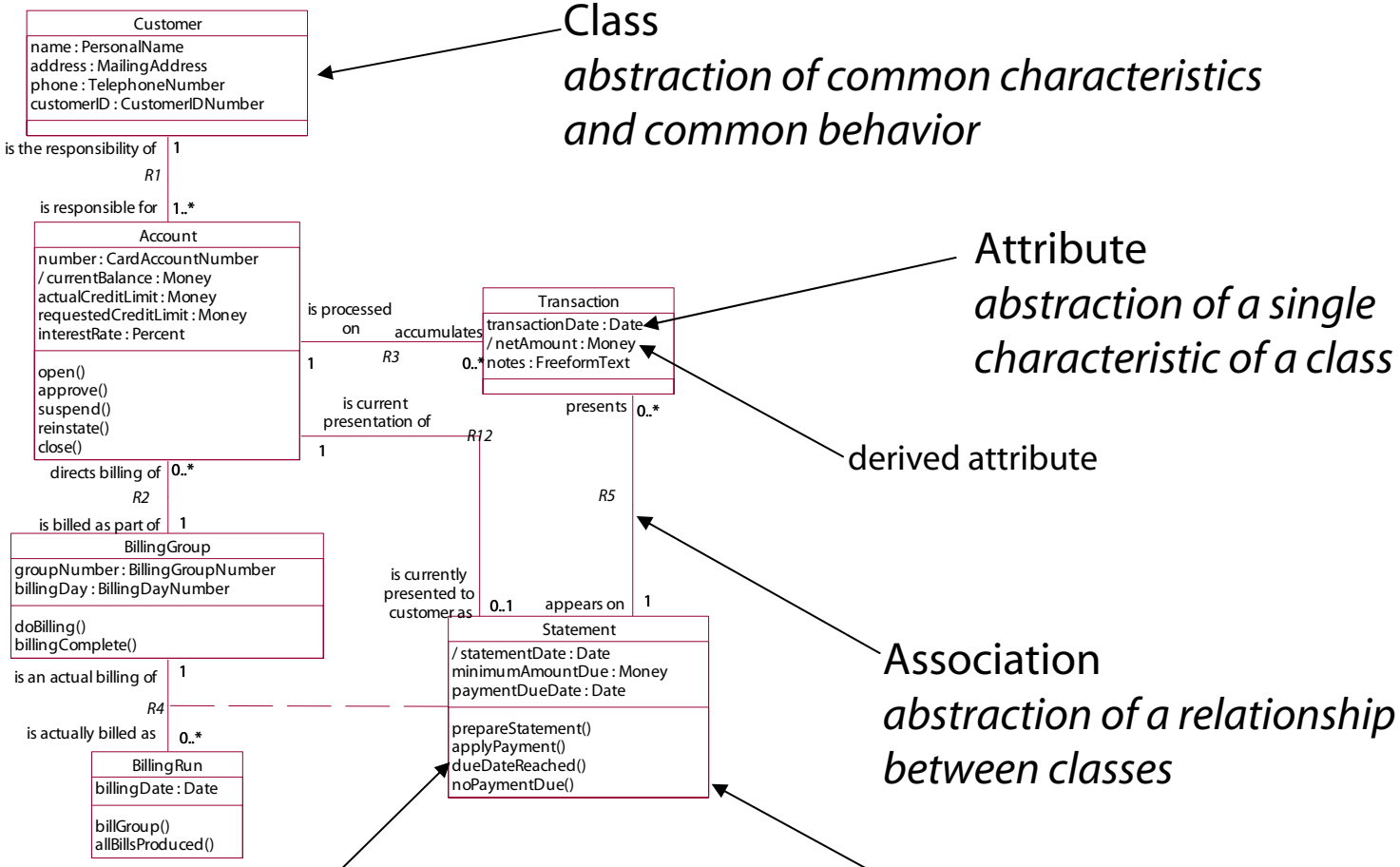


State Model
(lifecycle of an instance of a class)

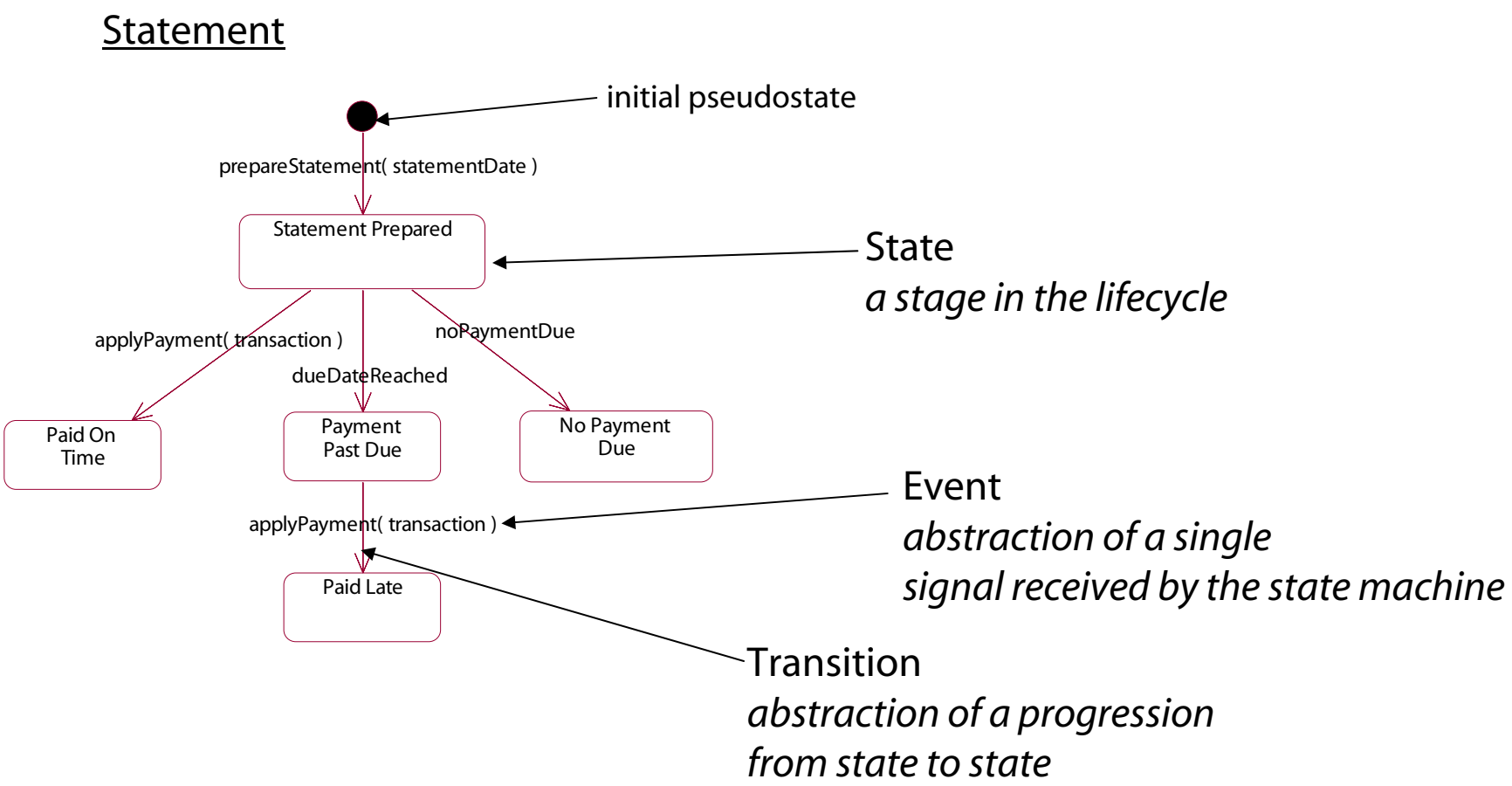


Execution Trace
(system behavior under one scenario)

Classes

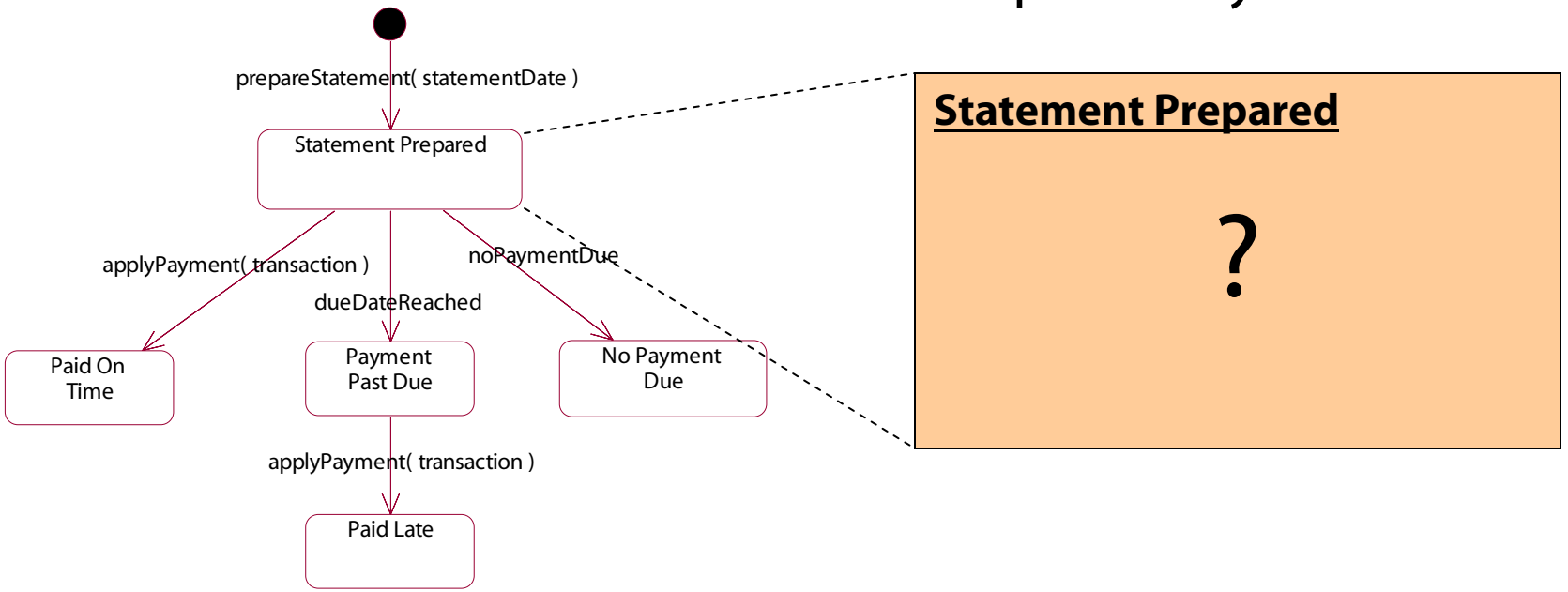


Lifecycles



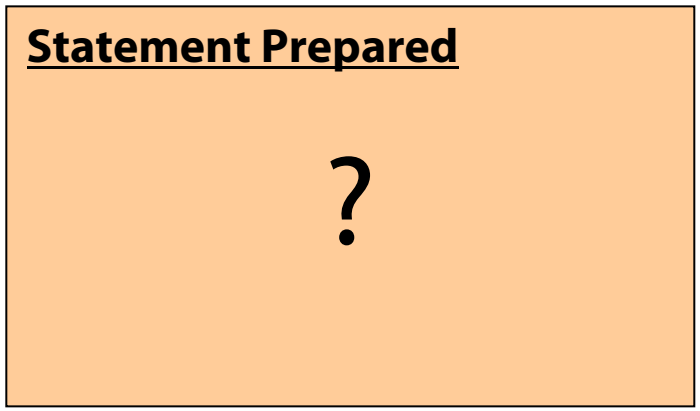
State Procedure & Actions

Each state has a procedure executed upon entry to the state



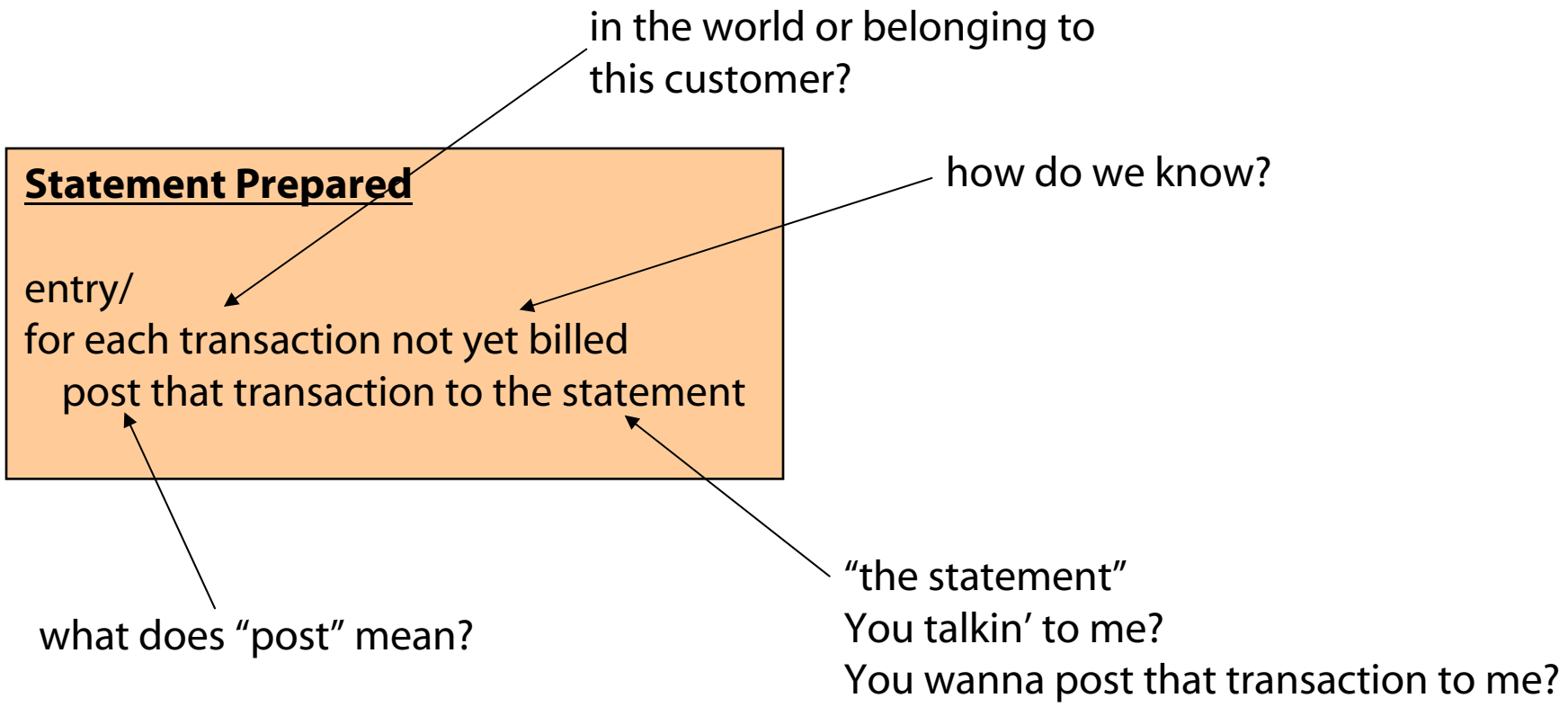
Actions specify what happens in the state procedure

Expressing Actions



- Need to express the semantics of the problem domain
  - in pseudocode
  - in Java itself?
  - in a Java-esque syntax?
  - in a flowchart?
  - in a dataflow diagram?
  - in pipes-and-filters?

Why not pseudocode?



Why not 3GL code?

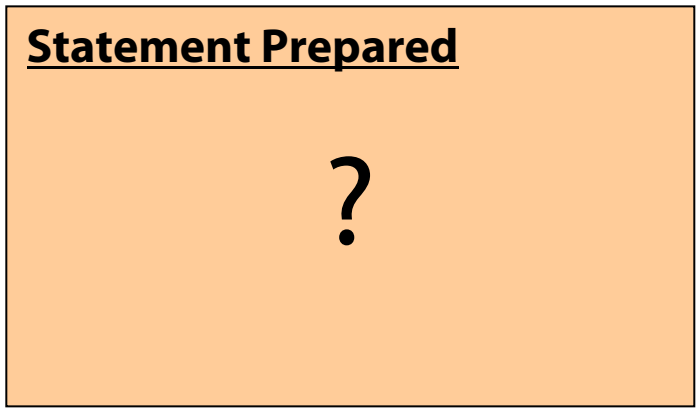
Statement Prepared

```
entry/
query = "SELECT * FROM TBL_TRANS, …
set rst=CurrentDB().open(query)
while not rst.EOF
   rst.Edit
   rst("R5StatementPtr") = statementID
   rst.Update
   rst.MoveNext
wend
rst.Close
```

(Microsoft Visual Basic…sort of)

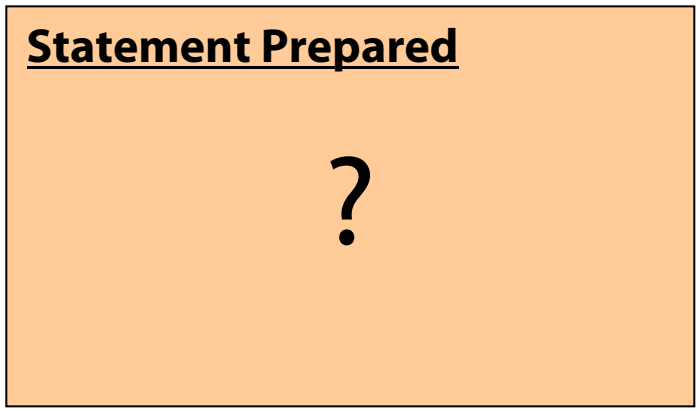
- Binds the model to a particular implementation
- Requires other non-application decisions to be made at the same time

What to do about actions?



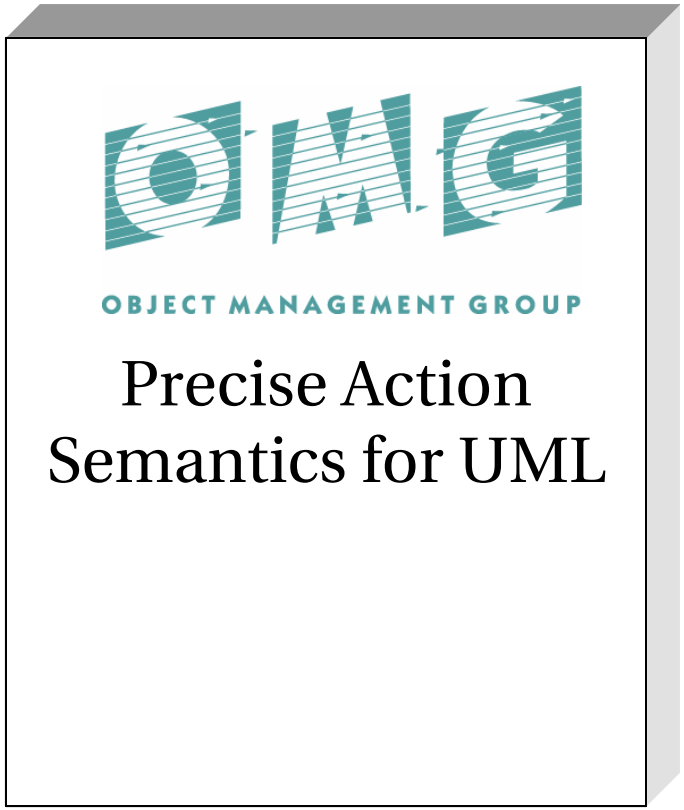
- Pseudocode is not executable
- 3GLs bind model to a particular architecture
- We want to write
  - from a problem perspective
  - in terms of the model
  - not presuming a specific implementation

What about actions?



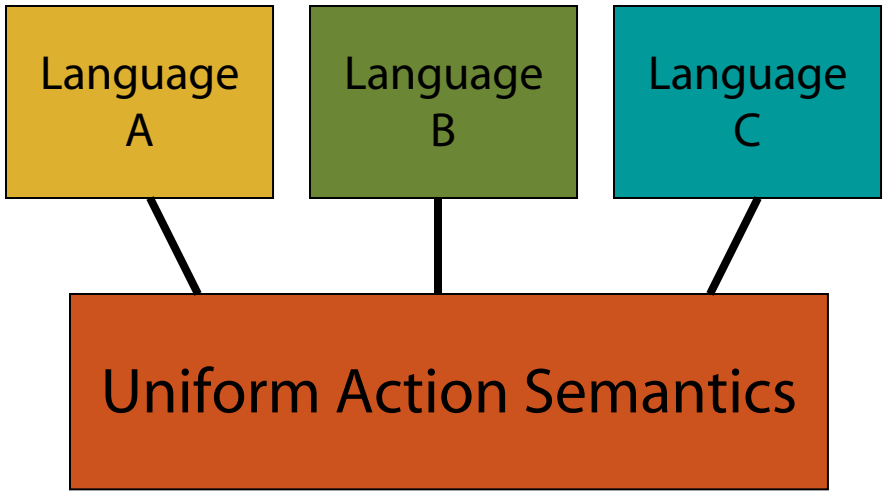
- UML (pre – 1.5) did not have a formalism for actions
- Precise Action Semantics specification supplements the UML
  - functional/dataflow-based
  - established the executable profile
  - essential for executability

UML Action Semantics



- 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

Semantics, not Language



- Why not just design an action language?
  - Highly political
  - Many vendors had some proprietary (but not fully compliant) action languages.
  - Syntax issues would obscure semantic issues
- First define a uniform semantics

Action Languages

TALL

```
foreach transaction in
   self->Account->Transaction[not .isBilled]
   self->Transaction := transaction;
endfor;
```

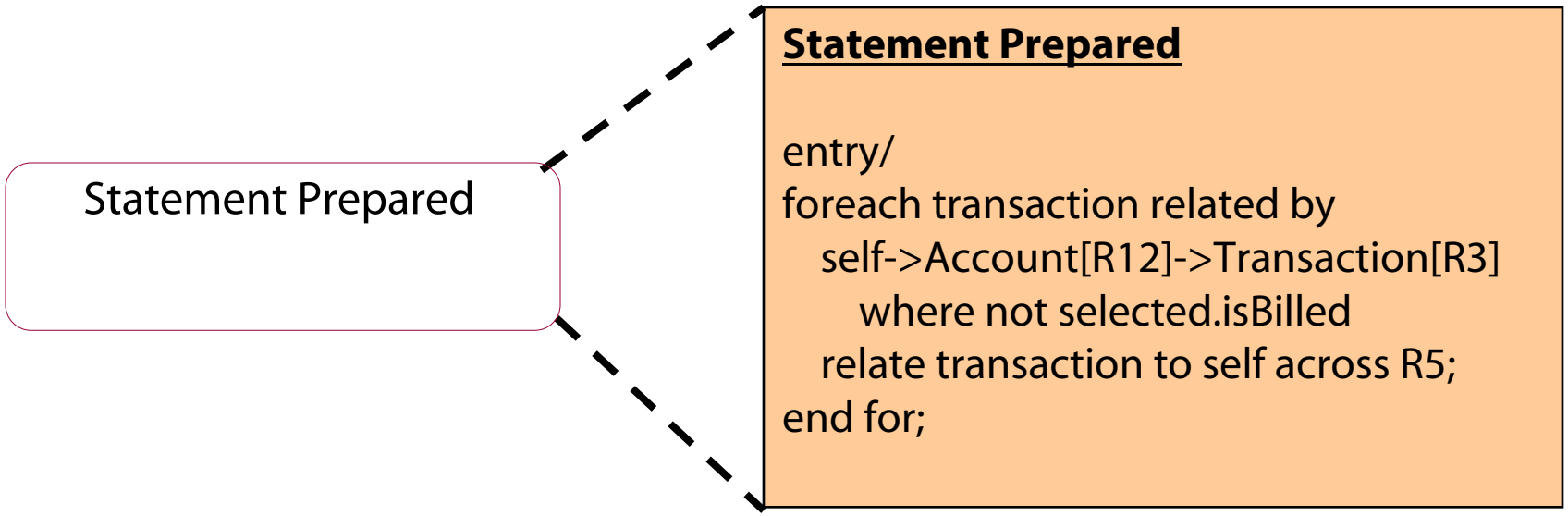
OAL

```
foreach transaction related by
   self->Account[R12]->Transaction[R3]
   where not selected.isBilled
   relate transaction to self across R5;
end for;
```

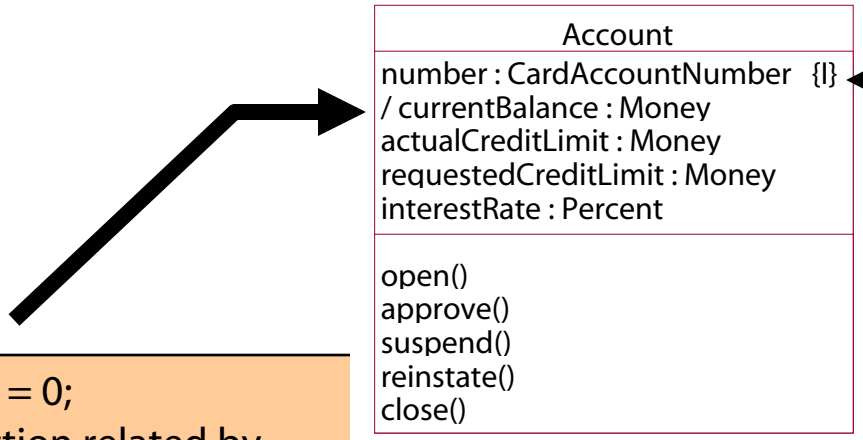
- Several different tool-specific languages
- First step is to make all action-semantics compliant
- Syntax is secondary

Actions

- Specify the logic for each state's action.



Actions – Other Uses



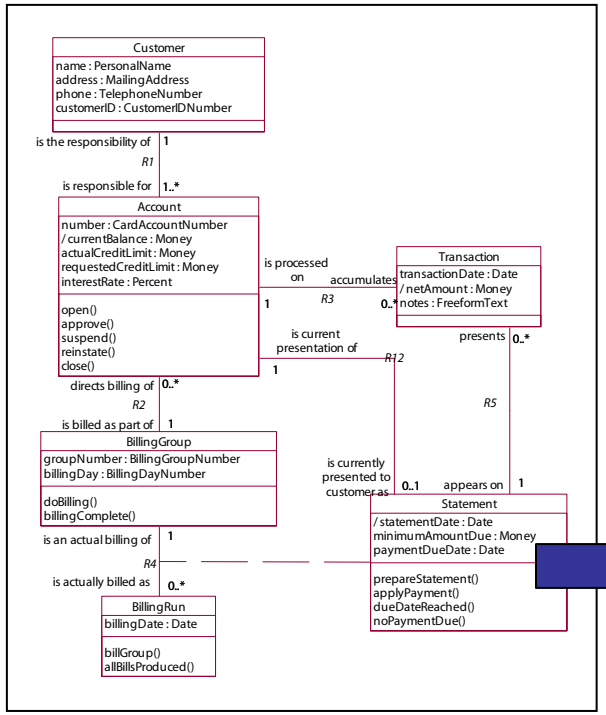

```
currentBalance = 0;
foreach transaction related by
   self->Transaction[R3]
   currentBalance = currentBalance +
      transaction.netAmount;
end for;
```

Derived Attribute

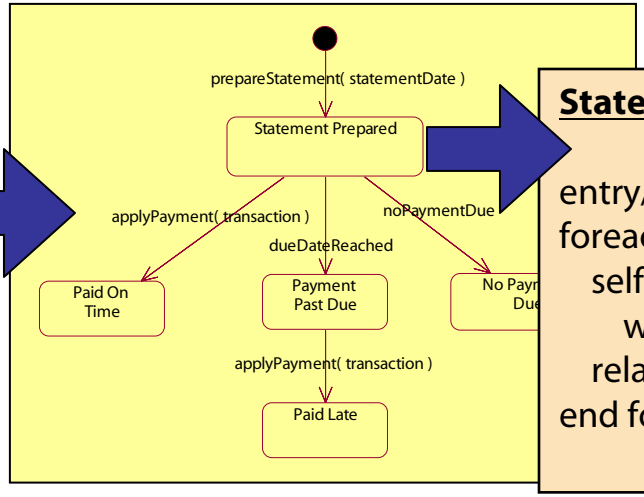
select many *accounts* from instances of *Account*
where *selected.number == self.number;*
return *((*cardinality *accounts) == 1);*

Constraint

An Executable Model



Classes



Lifecycle of Statement

Statement Prepared

entry/
foreach transaction related by
self->Account[R12]->Transaction[R3]
where not selected.isBilled
relate transaction to self across R5;
end for;

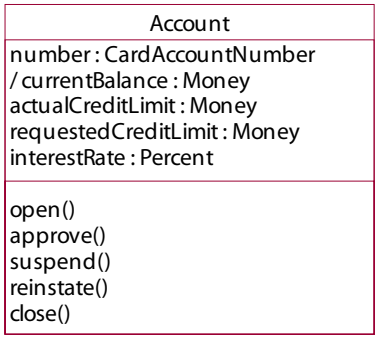
Statement Prepared state procedure

Table of contents

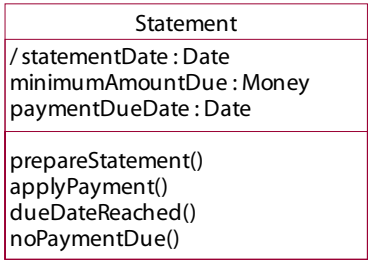
- What's the problem?
- UML models
- Executable UML elements
- Executable UML behavior 
- The repository
- Model compiler concepts
- How model compilers work
- MDA

Instances

An executable model operates on instances.



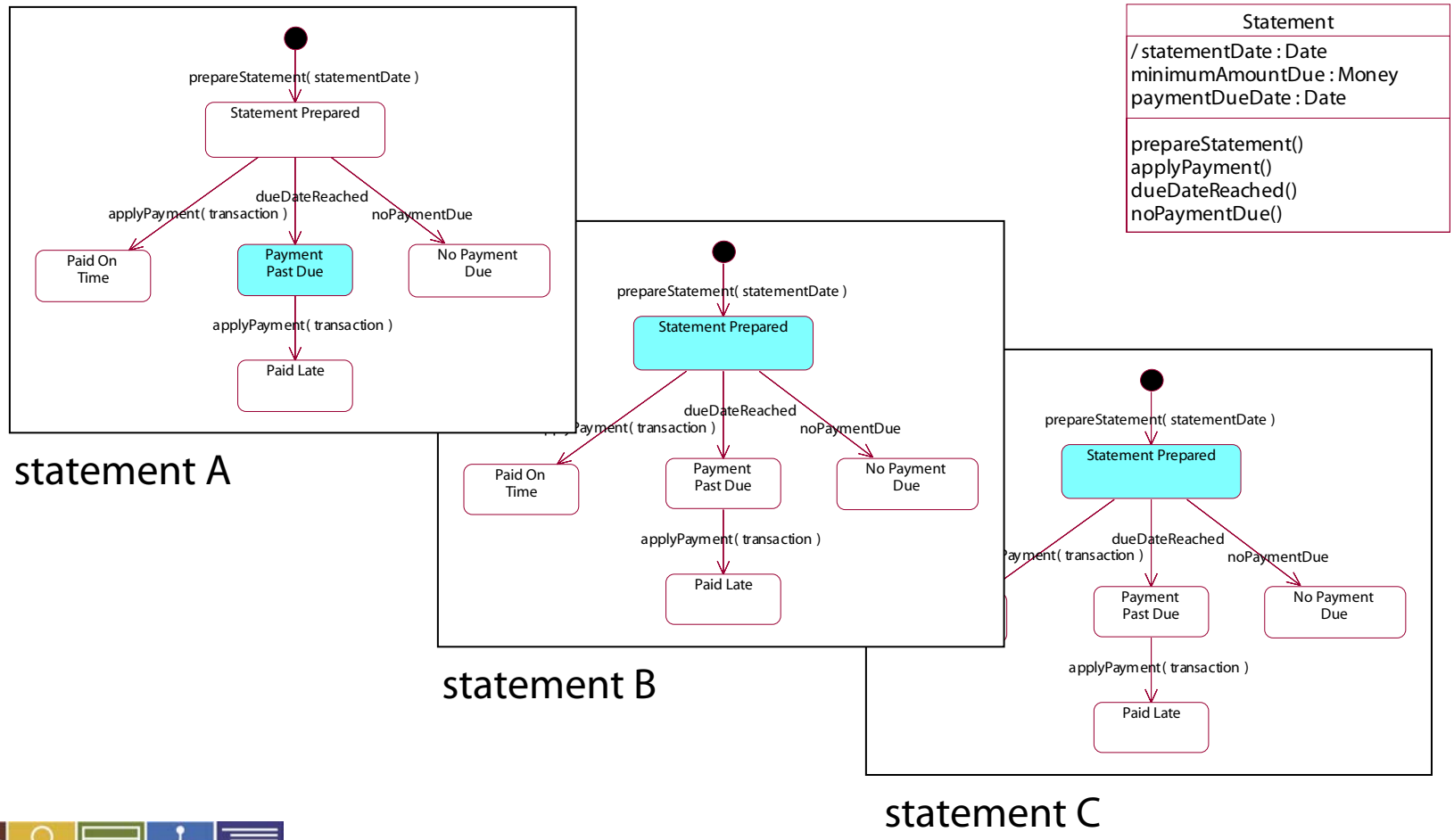
number	currentBalance	actualCredit Limit	requestedCredit Limit	interestRate
231552343	110.23	5000	5000	14%
897899934	0.00	8000	10000	12%
789978994	3423.32	3500	5000	18%



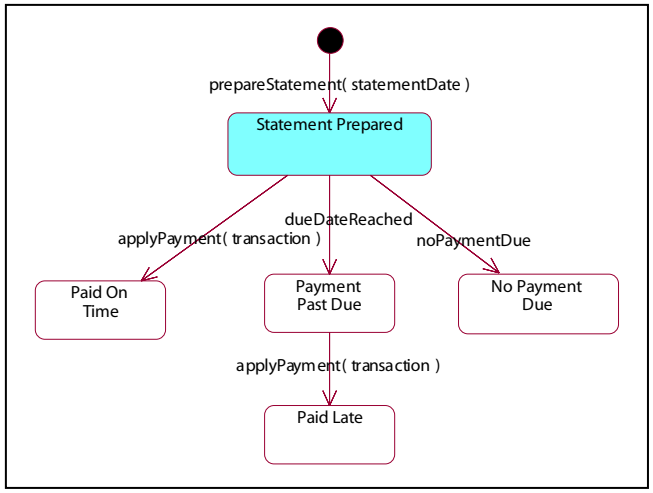
statementDate	minimumAmountDue	paymentDueDate
5/10/2003	10.00	6/1/2003
6/1/2003	0.00	6/21/2003
6/2/2003	68.00	6/21/2003

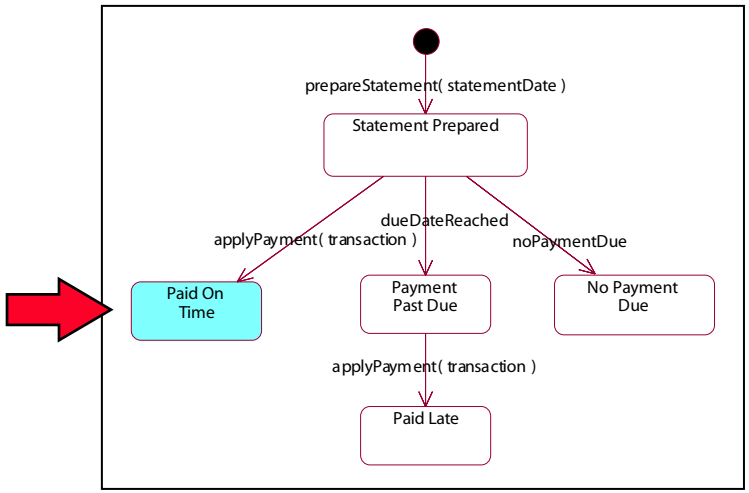
Instances

An executable model operates on instances.



Execution



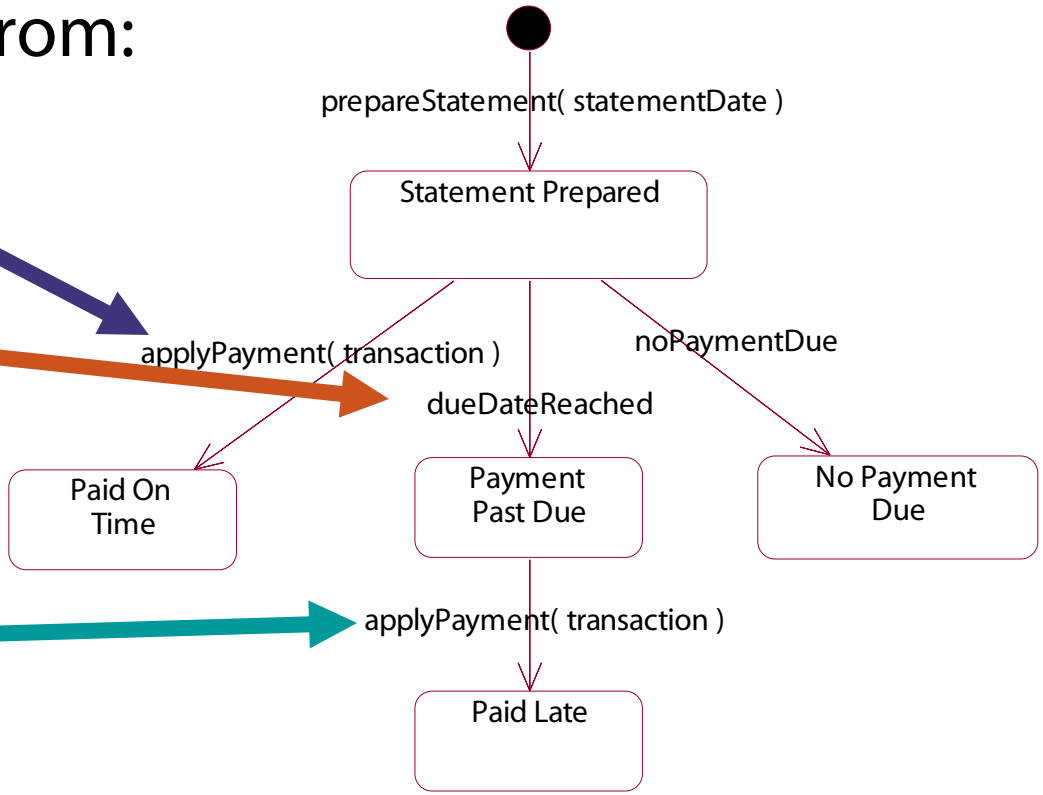


- The lifecycle model prescribes execution

When the customer's payment is received, the statement transitions to the next statement and executes actions.

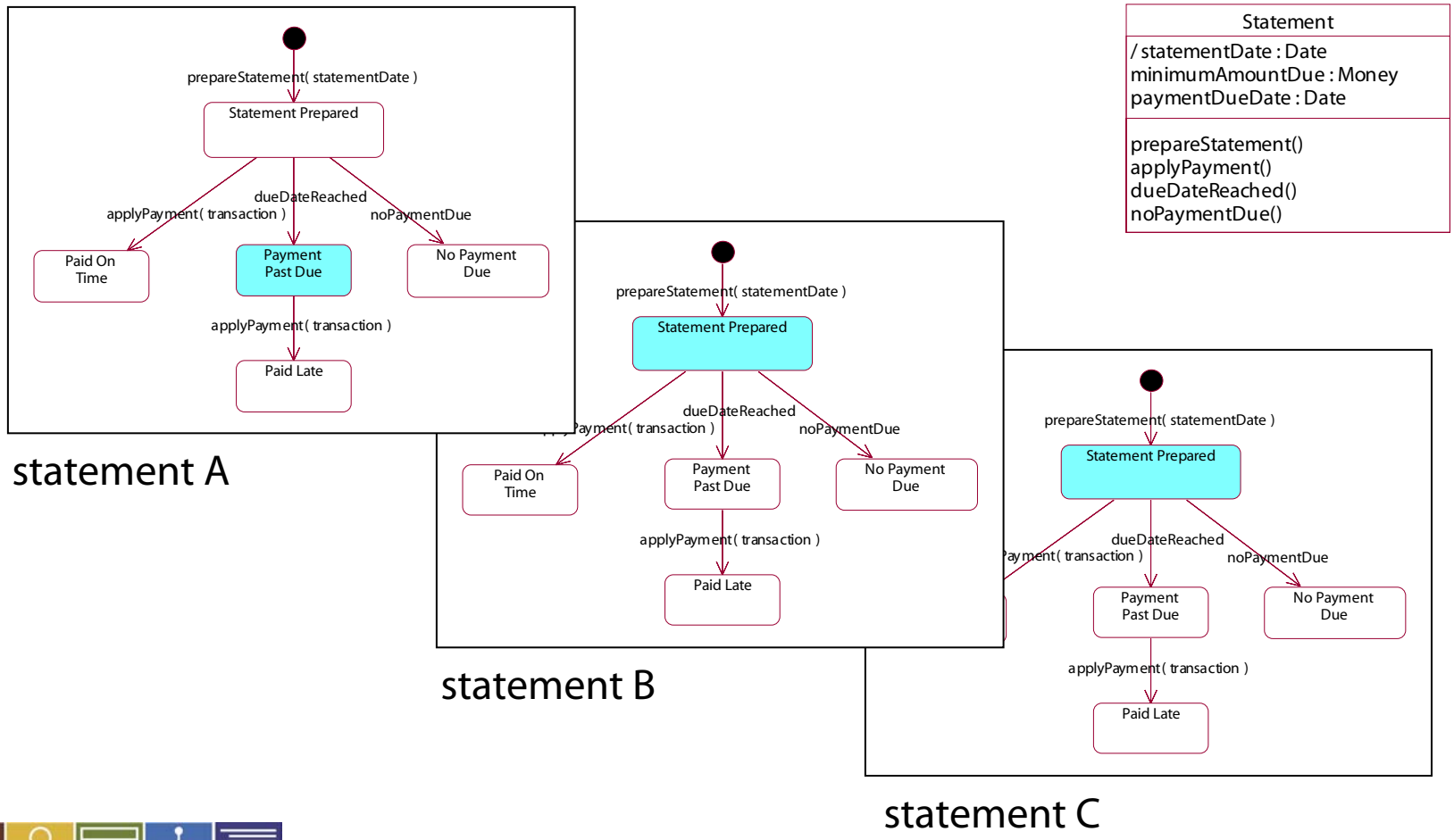
Executing the Model

- The model executes in response to signals from:
  - the outside,
  - timers (delayed events)
  - other instances as they execute

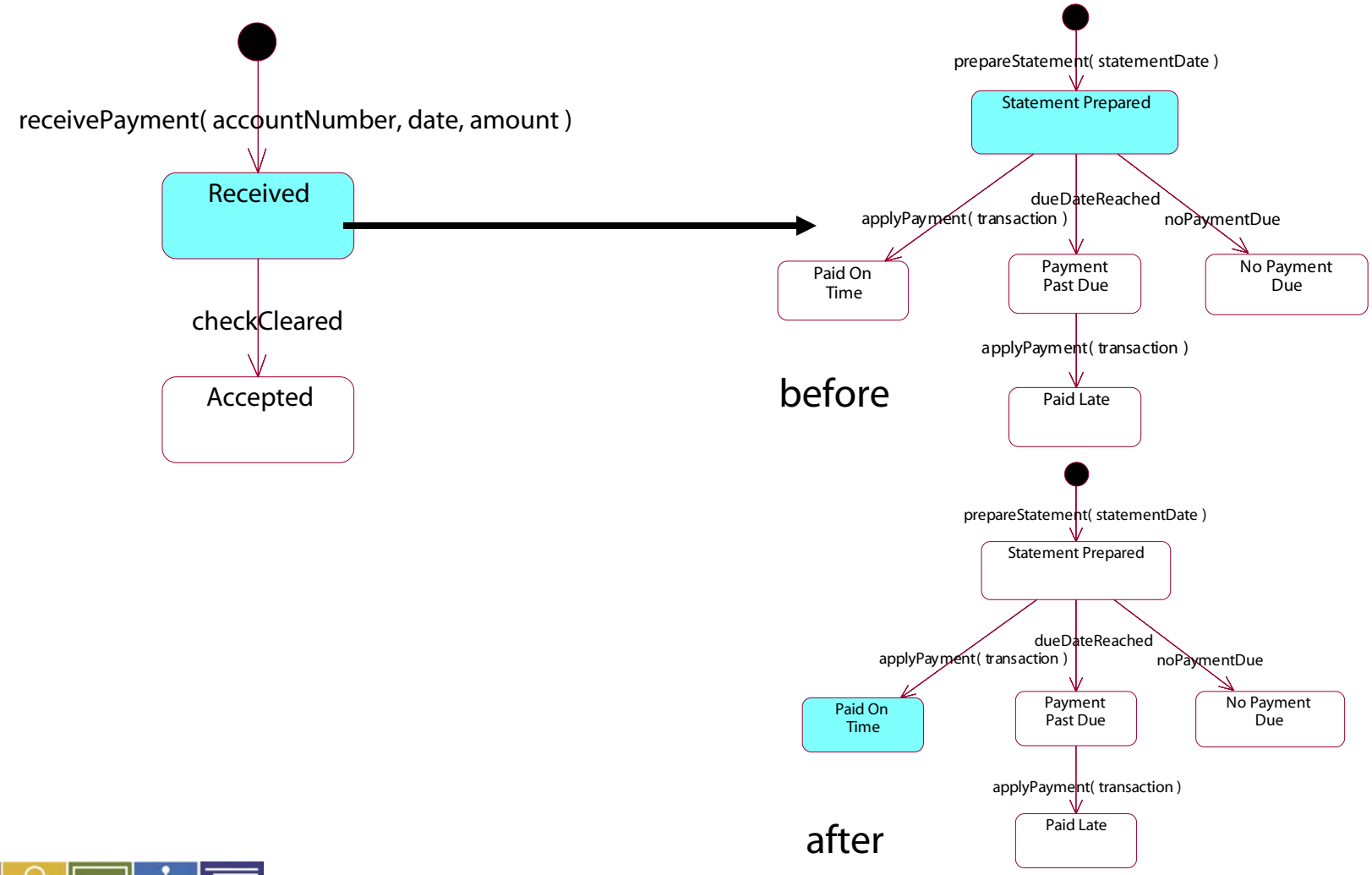


Concurrent Execution

All instances execute concurrently.

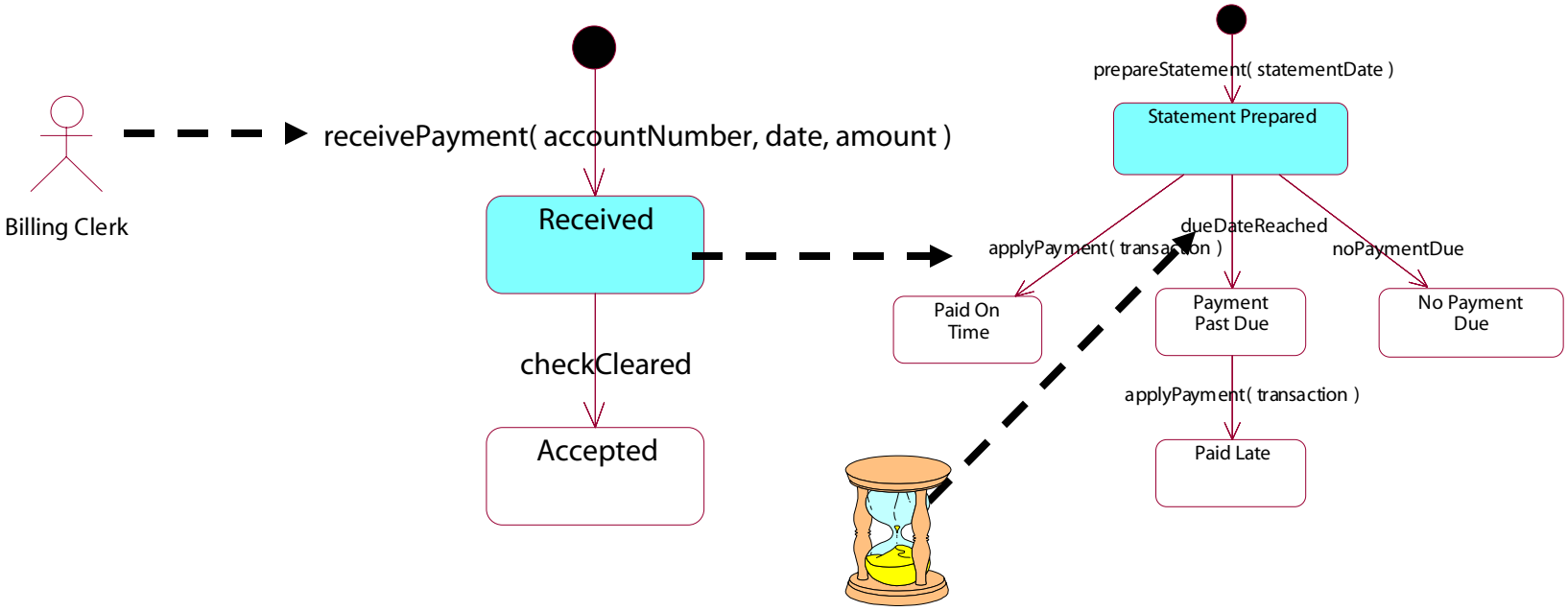


Communication



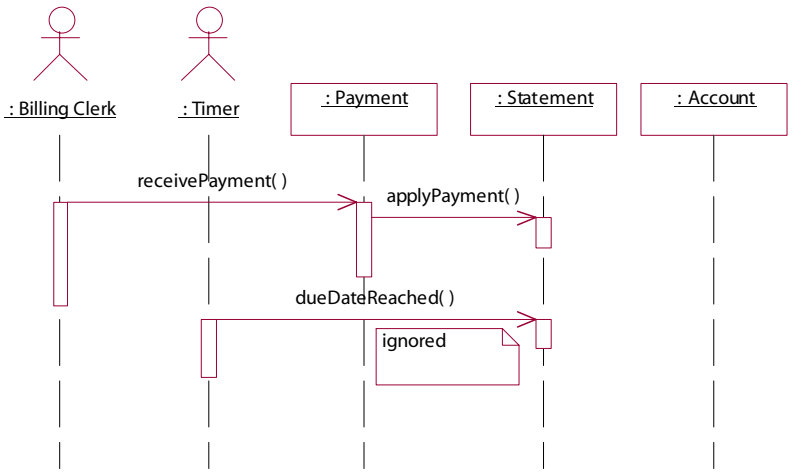
Concurrent Execution

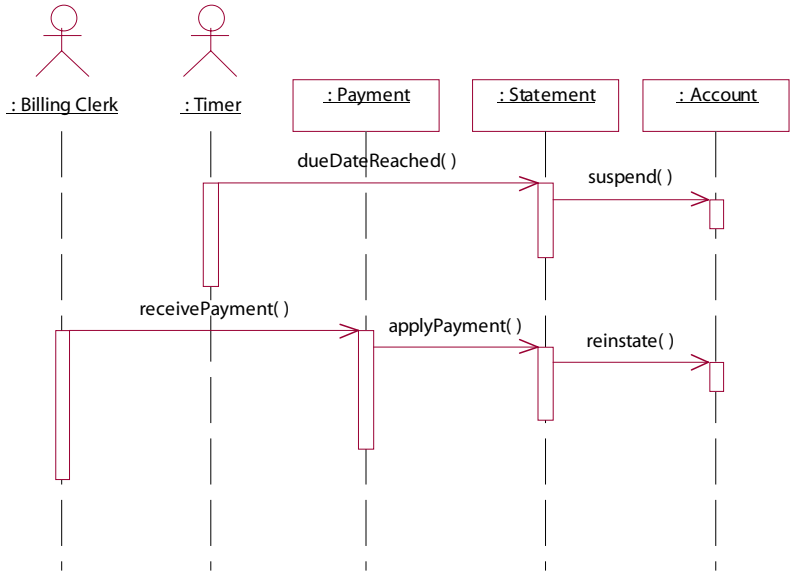
Multiple events may be received.



Concurrent Execution

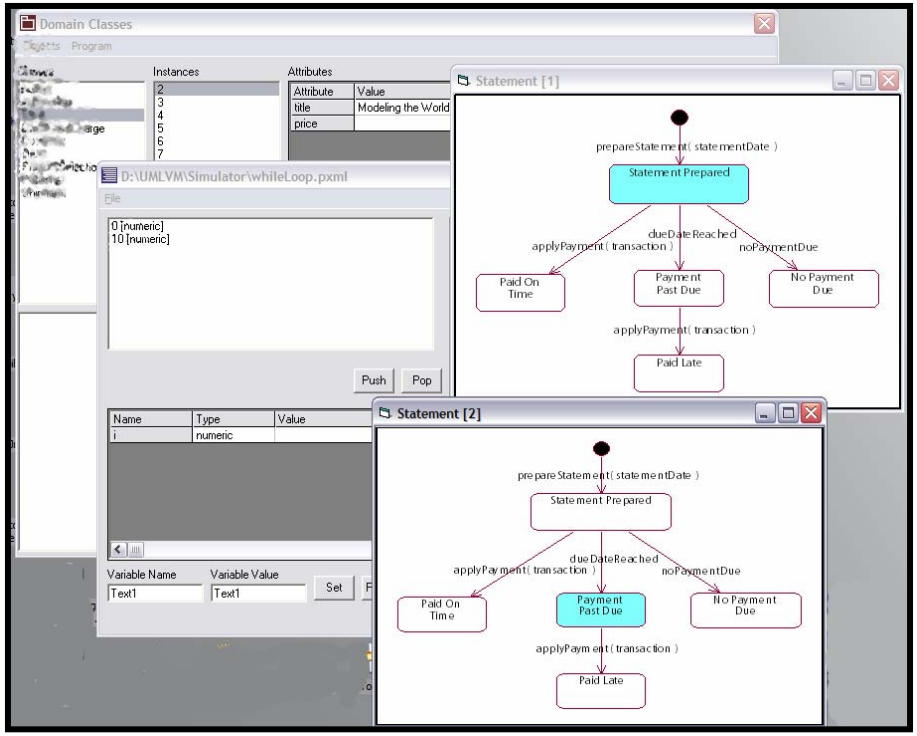
- Concurrent events can produce different outcomes





Two outcomes if the payment is received as the due date passes

Verification

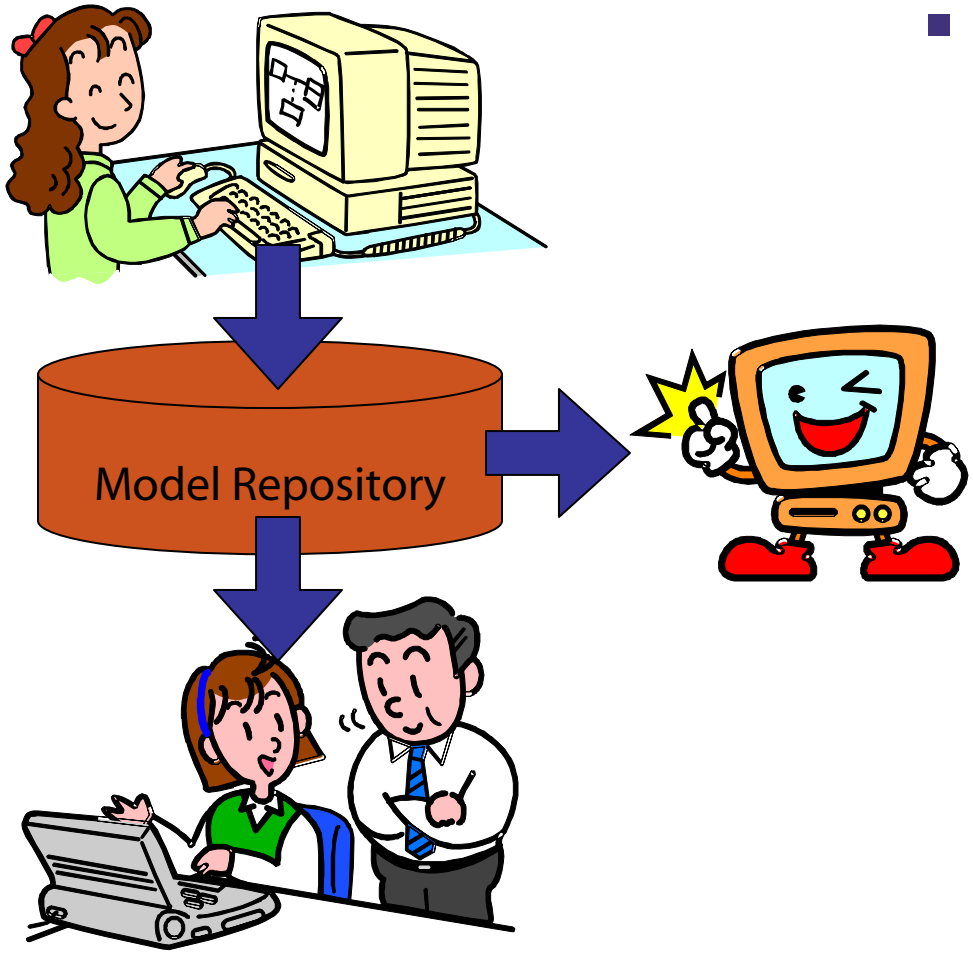


- Test models in the development environment
- Check the models just like code

Table of contents

- What's the problem?
- UML models
- Executable UML elements
- Executable UML behavior
- The repository
- Model compiler concepts
- How model compilers work
- MDA

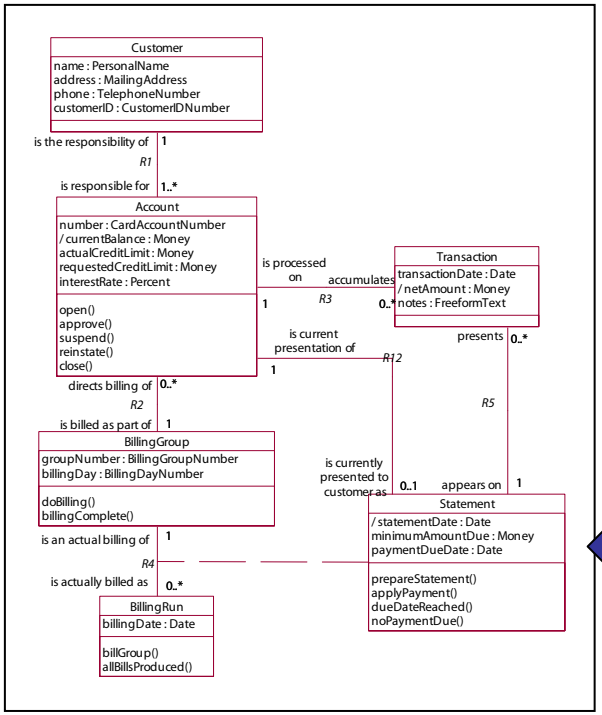
Model Repository

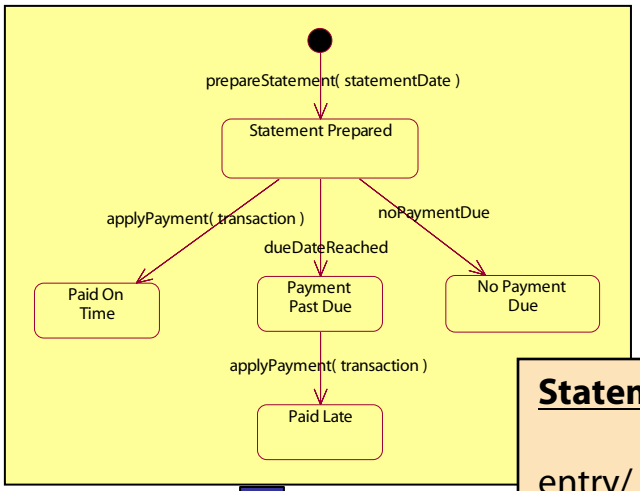


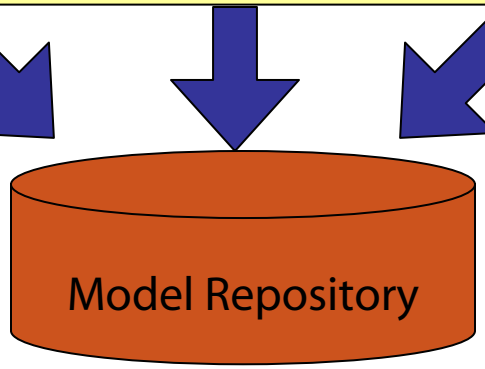
- Compilation requires that the models be available in a metamodel-based model repository

Model Repository

- The repository holds the developers' models.



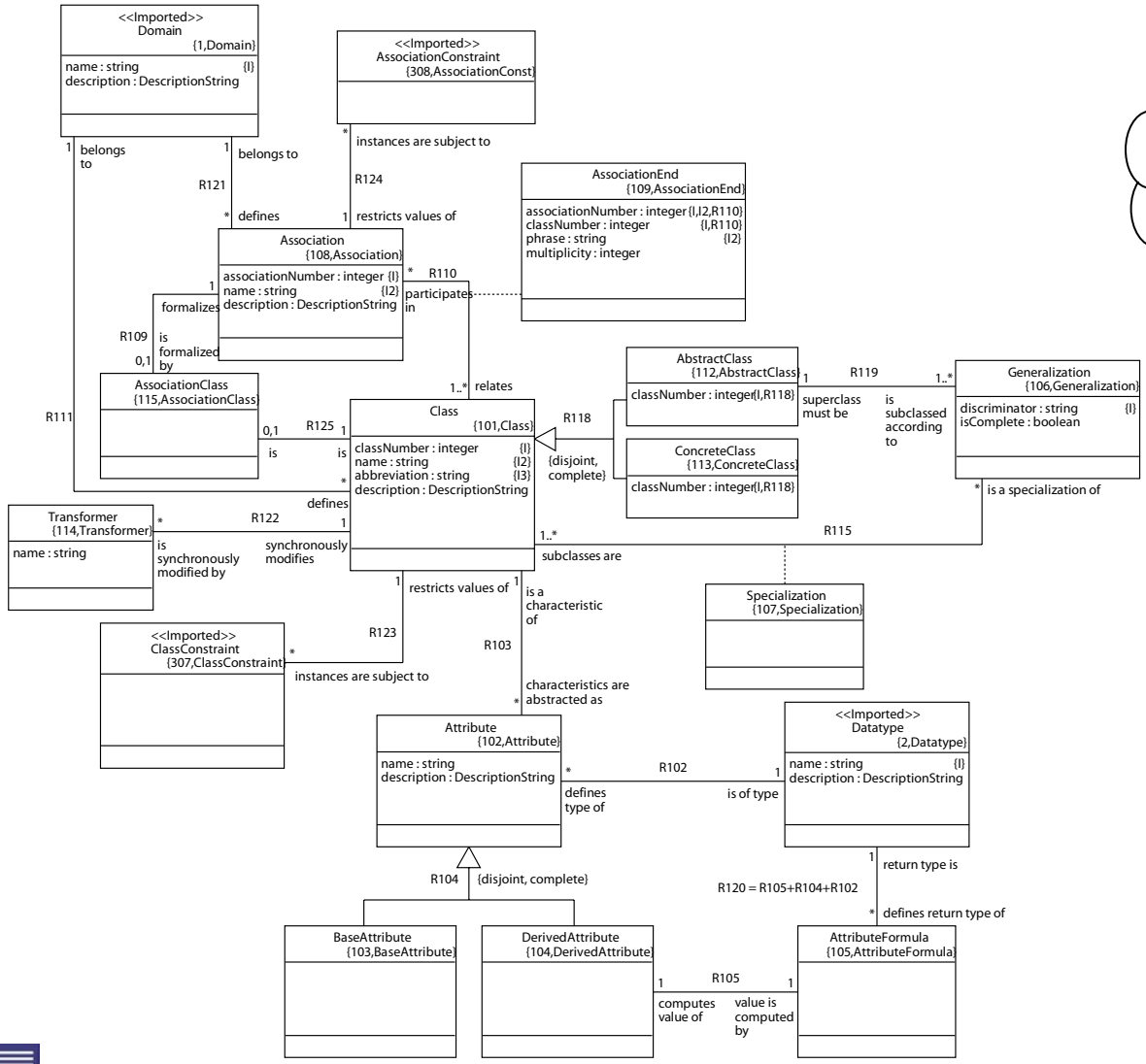


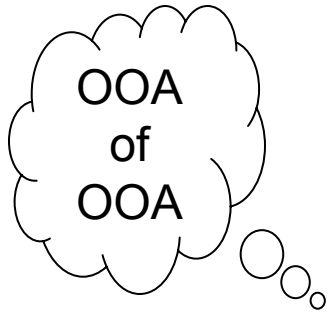


Statement Prepared

entry/
foreach transaction related by
self->Account[R12]->Transaction[R3]
where not selected.isBilled
relate transaction to self across R5;
end for;

Metamodel





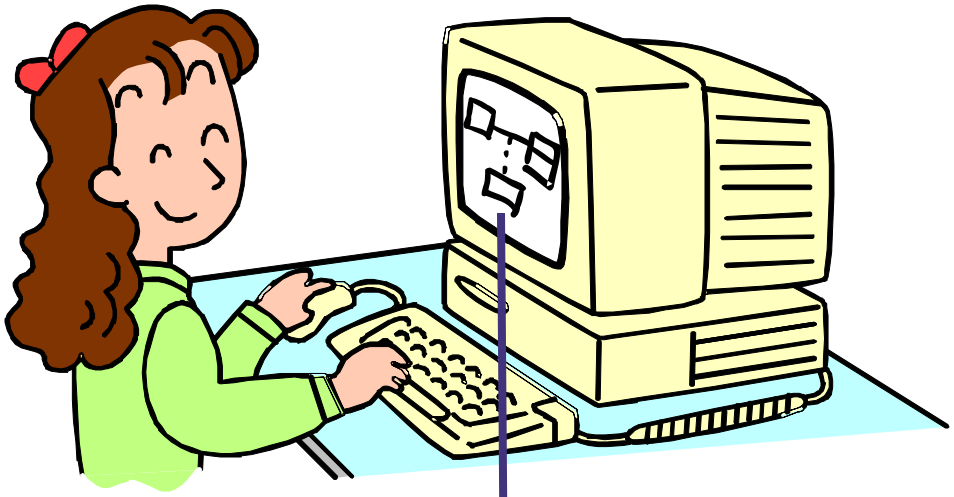
Metamodel Instances

- Just like an application model, the metamodel has instances.

classID	name	description
100	Customer	A Customer is any individual or business…
101	Account	An account represents a customer's financial…
102	BillingGroup	A set of accounts that are all billed periodically…

classID	stateID	stateName
101	1	NewAccount
101	2	In Good Standing
101	3	Suspended
101	4	Closed

Repository Instances



- Modeling tools create instances in the repository

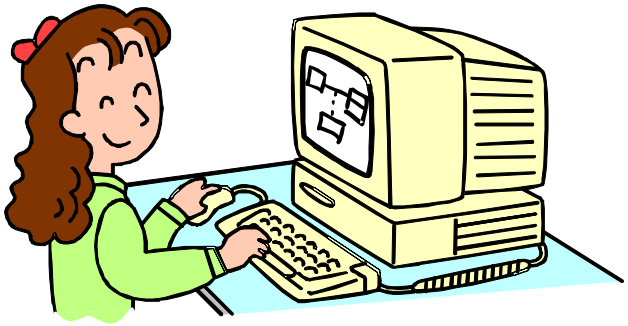
classID	name	description
100	Customer	A Customer is any individual or business…
101	Account	An account represents a customer's financial…
102	BillingGroup	A set of accounts that are all billed periodically…
103	Statement	A statement is a periodic preparation of the …

Table of contents

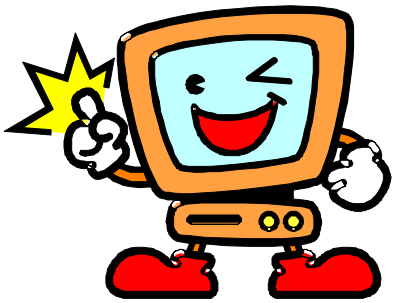
- What's the problem?
- UML models
- Executable UML elements
- Executable UML behavior
- The repository
- Model compiler concepts
- How model compilers work
- MDA

Model Compiler

A model compiler is the automated embodiment of how to turn a model into software under a set of rules.



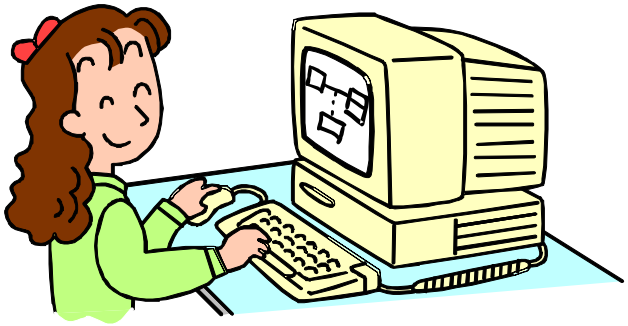
Analysts create models



Model Compiler produces a running system

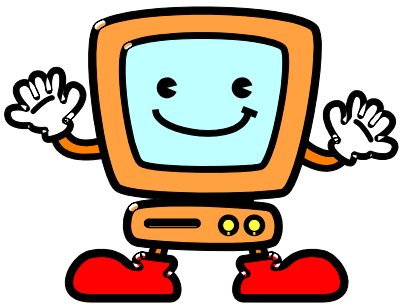
Developers program the rules

Model Compiler



Analysts create models

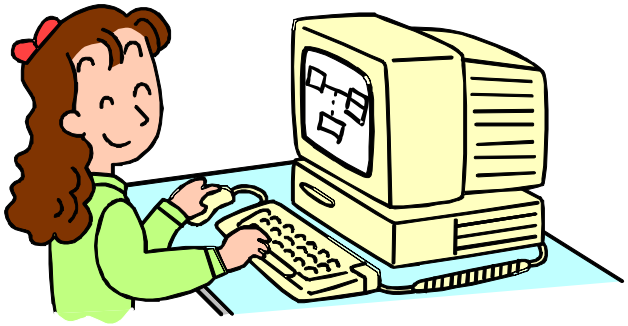
…a program that creates code from models…



Model Compiler produces a running system

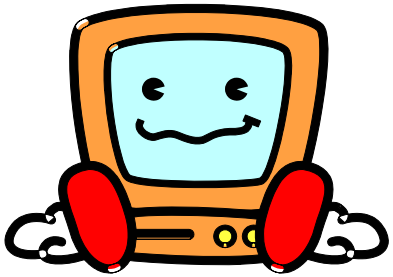
Developers program the rules

Model Compiler



Analysts create models

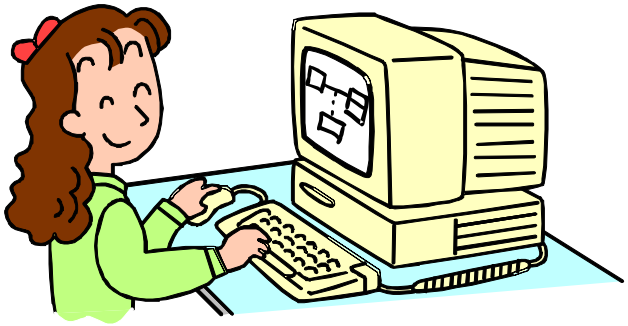
…something that makes models executable…



Model Compiler produces a running system

Developers program the rules

Model Compiler



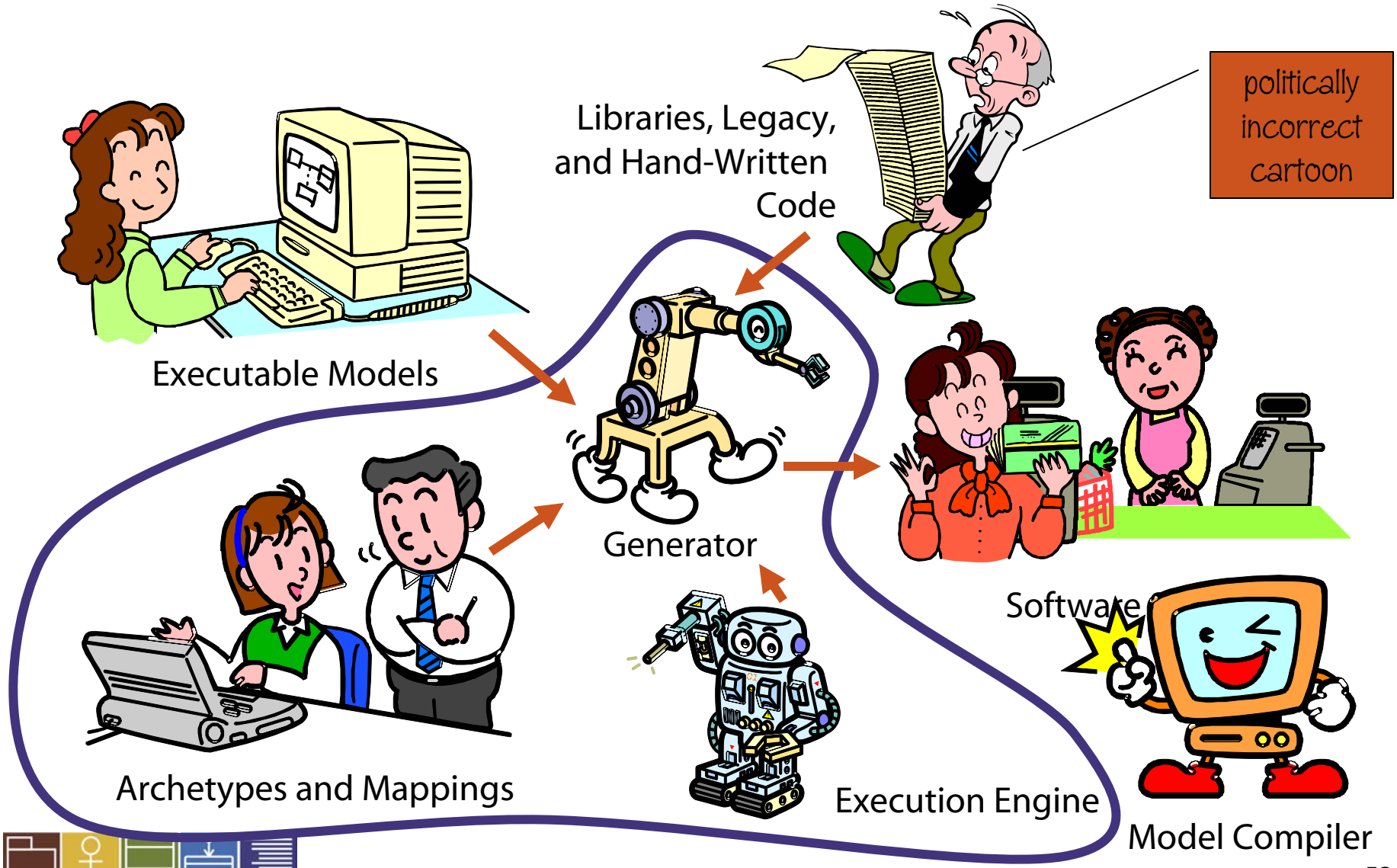
Analysts create models

Developers program the rules

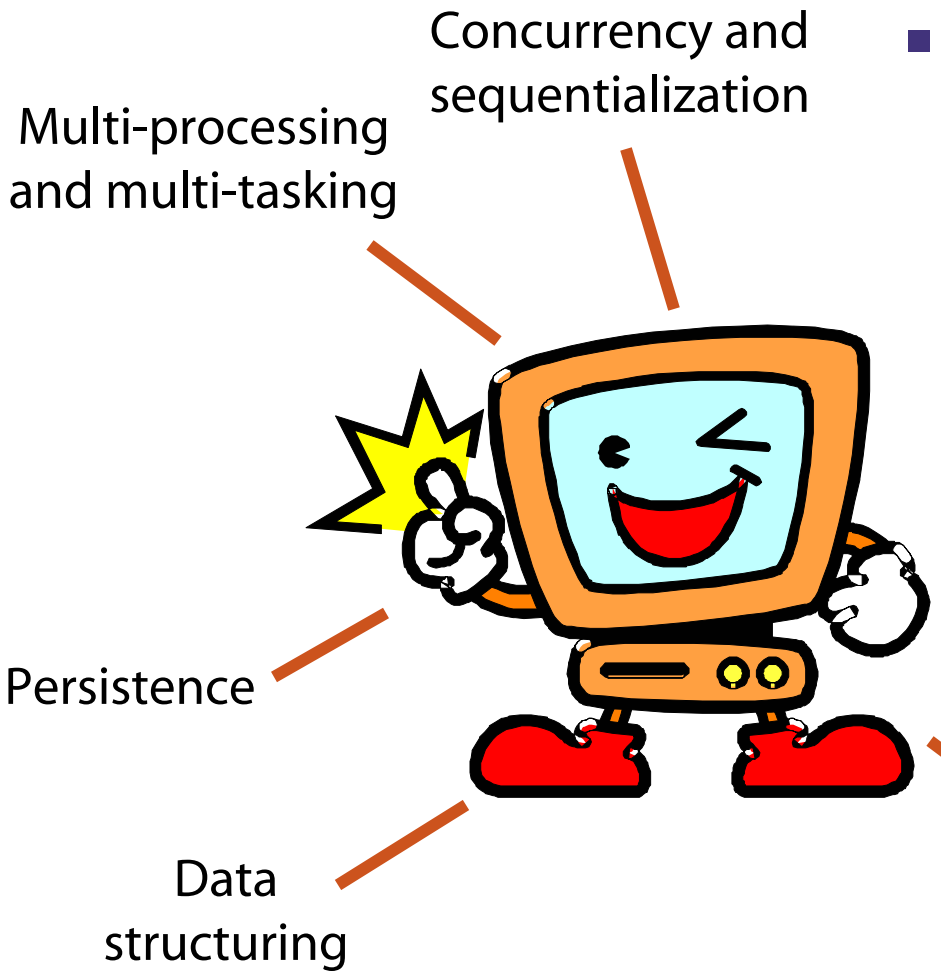
…it's a model compiler.

Model Compiler produces a running system

The Big Picture

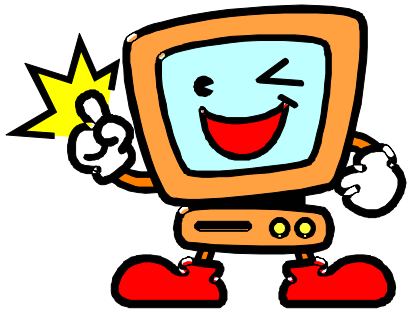


Model Compiler

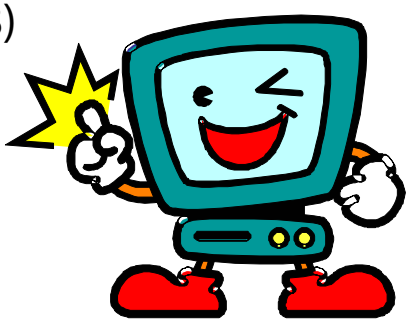


- A model compiler reads executable UML models and creates a system with certain dimensions

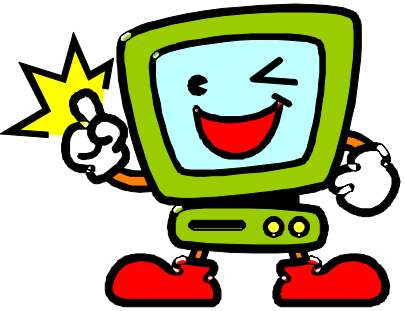
Examples



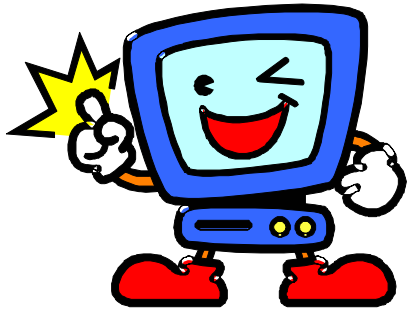
Financial system
- Highly distributed
- Concurrent
- Transaction-safe with rollback
- Persistence with rollback
- J2EE (Java + EJB)



Embedded system
- Single task
- No operating system
- Optimized data access and storage
- C



Telecommunication system
- Highly distributed
- Asynchronous
- Limited persistence capability
- C++

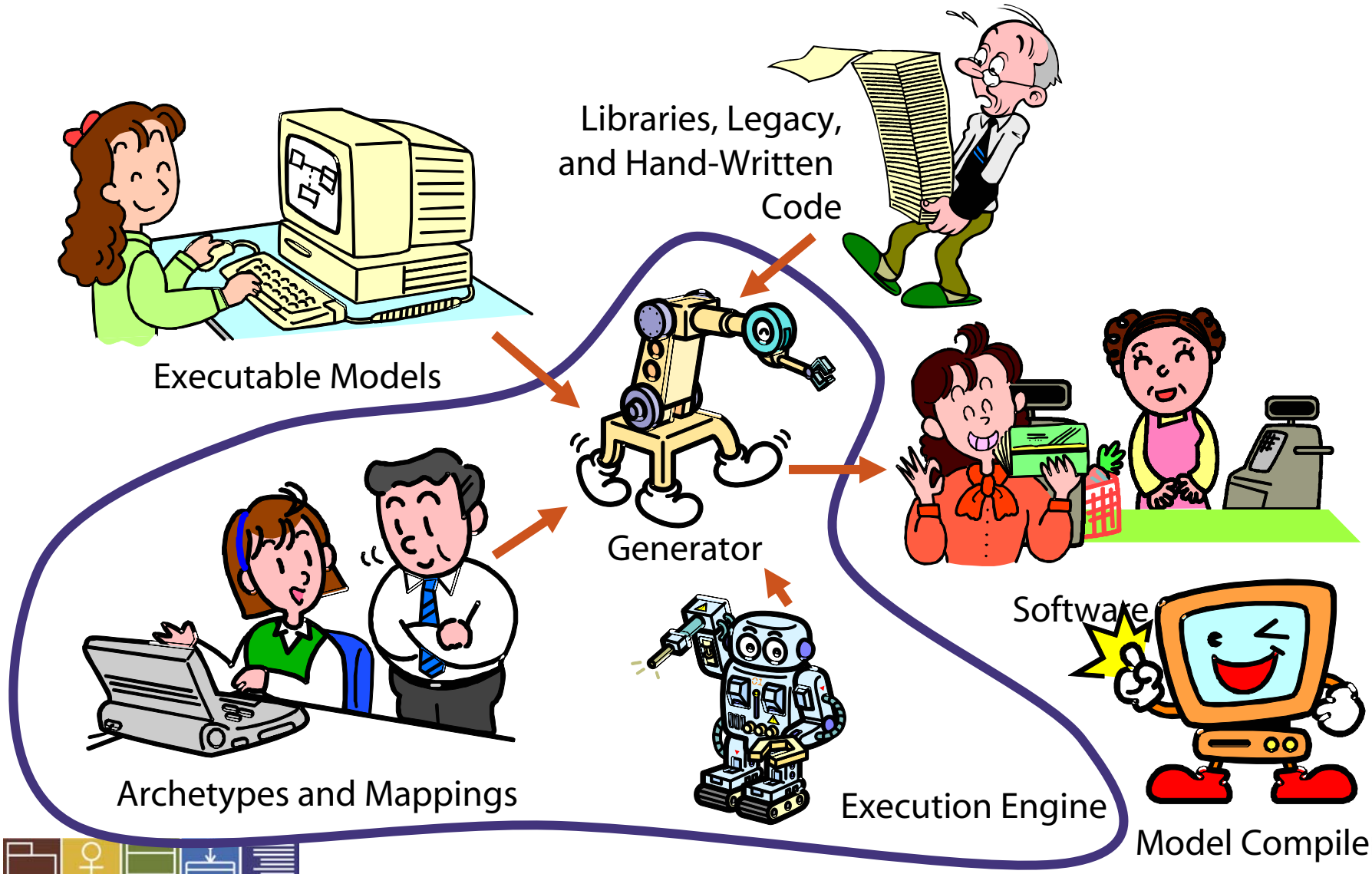


Simulation system
- Mostly synchronous
- Few tasks
- Special-purpose language “Import”

Table of contents

- What's the problem?
- UML models
- Executable UML elements
- Executable UML behavior
- The repository
- Model compiler concepts
- How model compilers work
- MDA

The Big Picture



Model Compiler

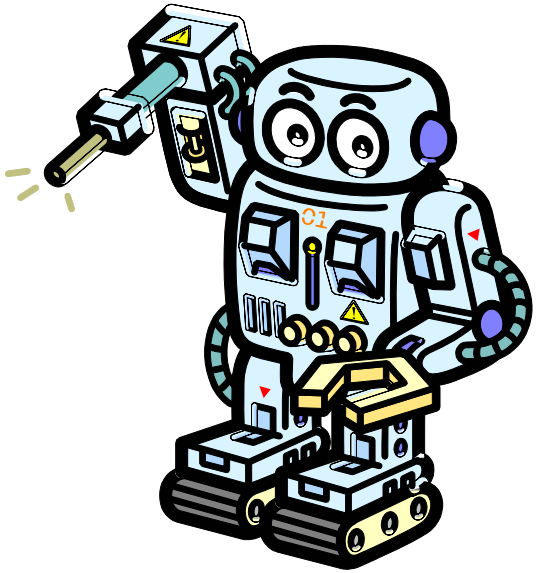
```
.Function Class
 Class ${Class.name} :
  public ActiveInstance {
 private:
 .invoke PrivateDataMember( Class )


};
```

```
.Function PrivateDataMember
.param Class Class
.select many PDM from instances
  of Attribute related to Class
.for each PrivateDataMember
${PDM.Type} ${PDM.Name};
.endfor
```

Archetypes

Translation rules interpreted by a generator against the repository



Execution Engine

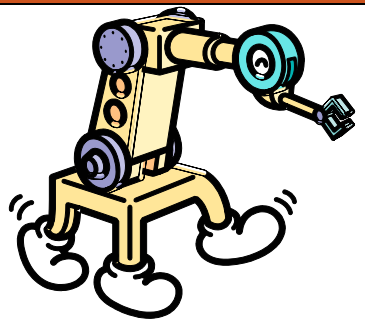
A limited set of reusable components sufficient to execute Executable UML

Archetypes

Archetypes traverse the repository and...

classID	name	description
100	Customer	A Customer is any individual or business…
101	Account	An account r
102	BillingGroup	A set of acco

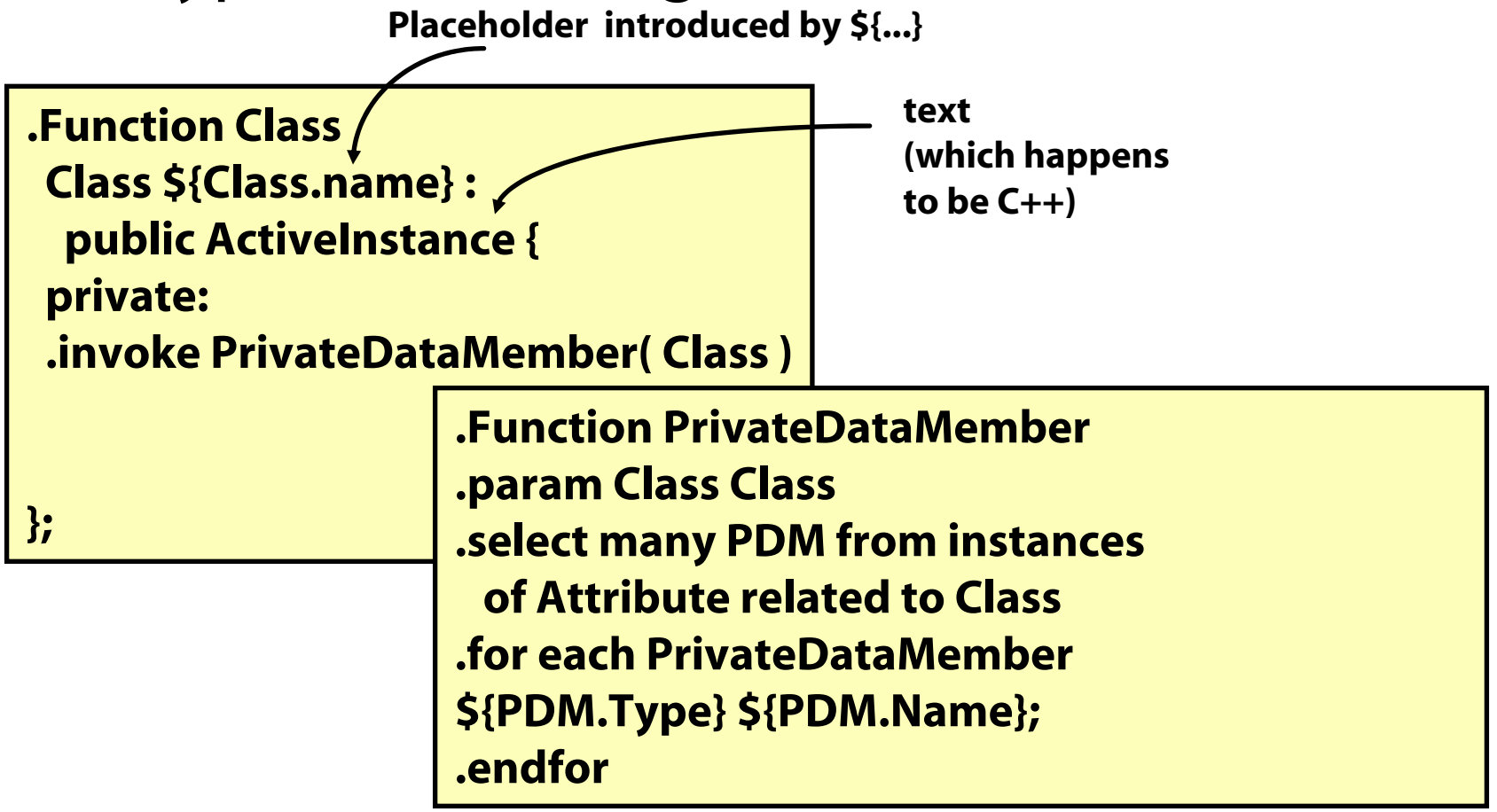
classID	stateID	stateName
101	1	NewAccount
101	2	In Good Standing
101	3	Suspended
101	4	Closed



… output text.

Archetypes

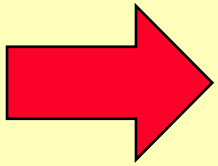
- Archetypes direct the generation of text.



Example

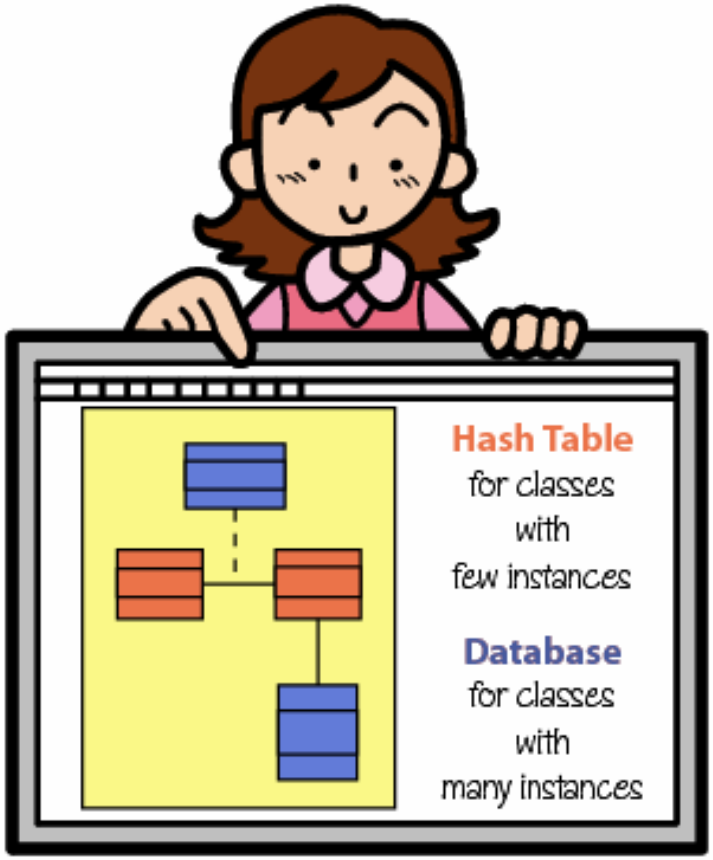
- The archetype language produces text.

```
.select many stateS related to instances of
    class->[R13]StateChart ->[R14]State
       where (isFinal == False)
public:
 enum states_e
  { NO_STATE = 0 ,
.for each state in stateS
    .if ( not last stateS )
       ${state.Name} ,
    .else
       NUM_STATES = ${state.Name}
    .endif
.endfor
};
```



```
public:
   enum states_e
      {  NO_STATE = 0 ,
         NewAccount ,
         InGoodStanding ,
         Suspended ,
         NUM_STATES = Closed
      };
```

Mappings



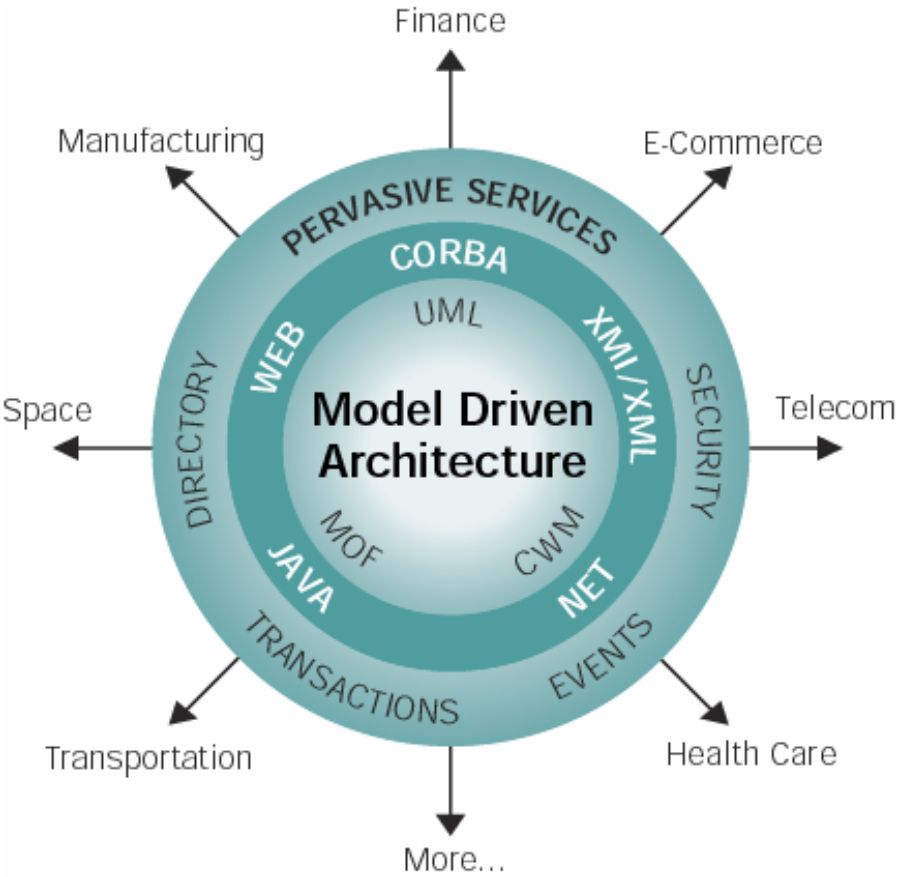
- Design-time assignment of different archetypes to distinct elements in the model
- Separate from the model and the archetypes
- “Uniformity != Rigidity”

Table of contents

- What's the problem?
- UML models
- Executable UML elements
- Executable UML behavior
- The repository
- Model compilers
- Model compiler concepts
- How model compilers work
- MDA

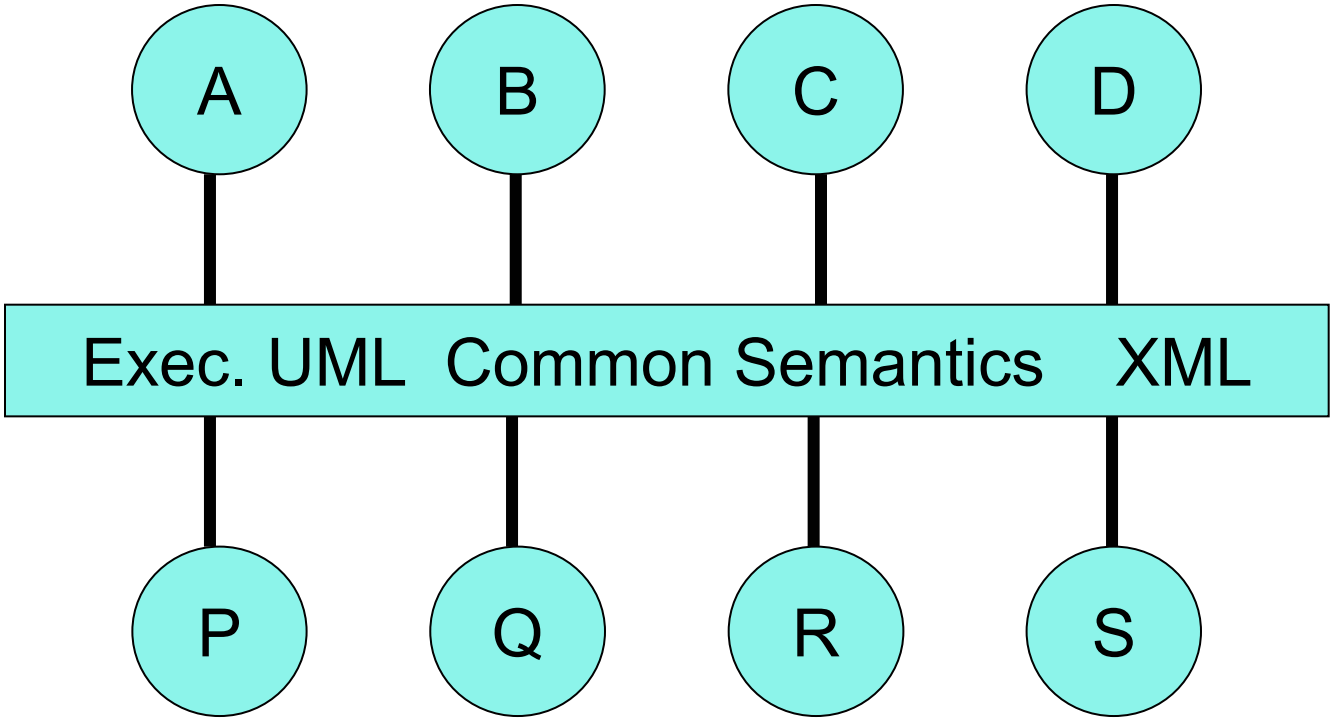
Model Driven Architecture

- A standard for software development using formal, executable and compilable system specifications
- Standards should be based on executable UML including the Action Semantics.
- Interchange can be managed using XML



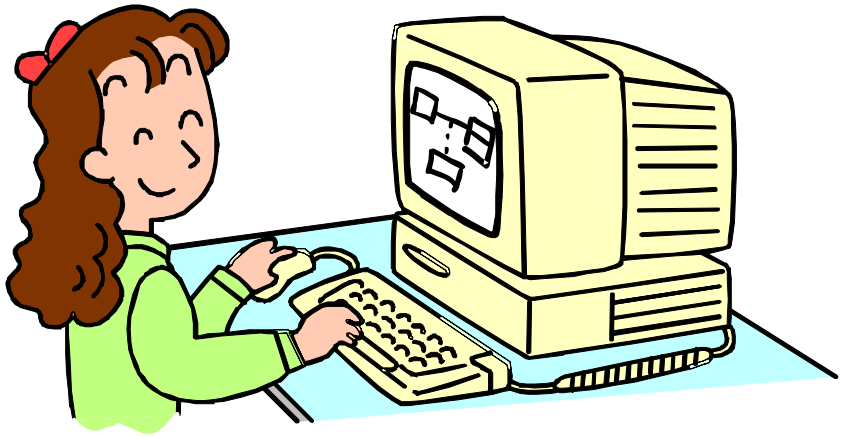
Executable UML

- Entry tools: BridgePoint®, Rose®, others
- Verification tools: T-Vec®, S/R®, COSPAN®



Model Compilers turn Executable UML models into systems.

Building a Market



The models are a valuable corporate resource that are true business models—they describe the business, not the software.

- Design time composability
  - protects IP

Building a Market

Just because we move to a new software platform doesn't mean that we need to recreate the rules of the bank.

- Design time composability
  - protects IP
  - allows IP to be mapped to multiple implementations

Building a Market

We can sell "Billing" models that can run on many platforms and support many different businesses.

- Design time composability
  - protects IP
  - allows IP to be mapped to multiple implementations
  - enables a market in IP in software

The Future

- Executable UML enables a market in model compilers.
- Application experts build models….
  - ….select the appropriate model compiler, and…
  - ….compile the model.
- Each model compiler targets a specific software platform.

Conclusion

- Executable UML, using archetypes to direct the generation, enables:
  - early error detection through verification
  - reuse of the execution engine
  - faster performance tuning
  - faster integration
  - faster, cheaper retargeting

Brought to you by…

PROJECT TECHNOLOGY INC

ModelCompilers.com

code at a higher level