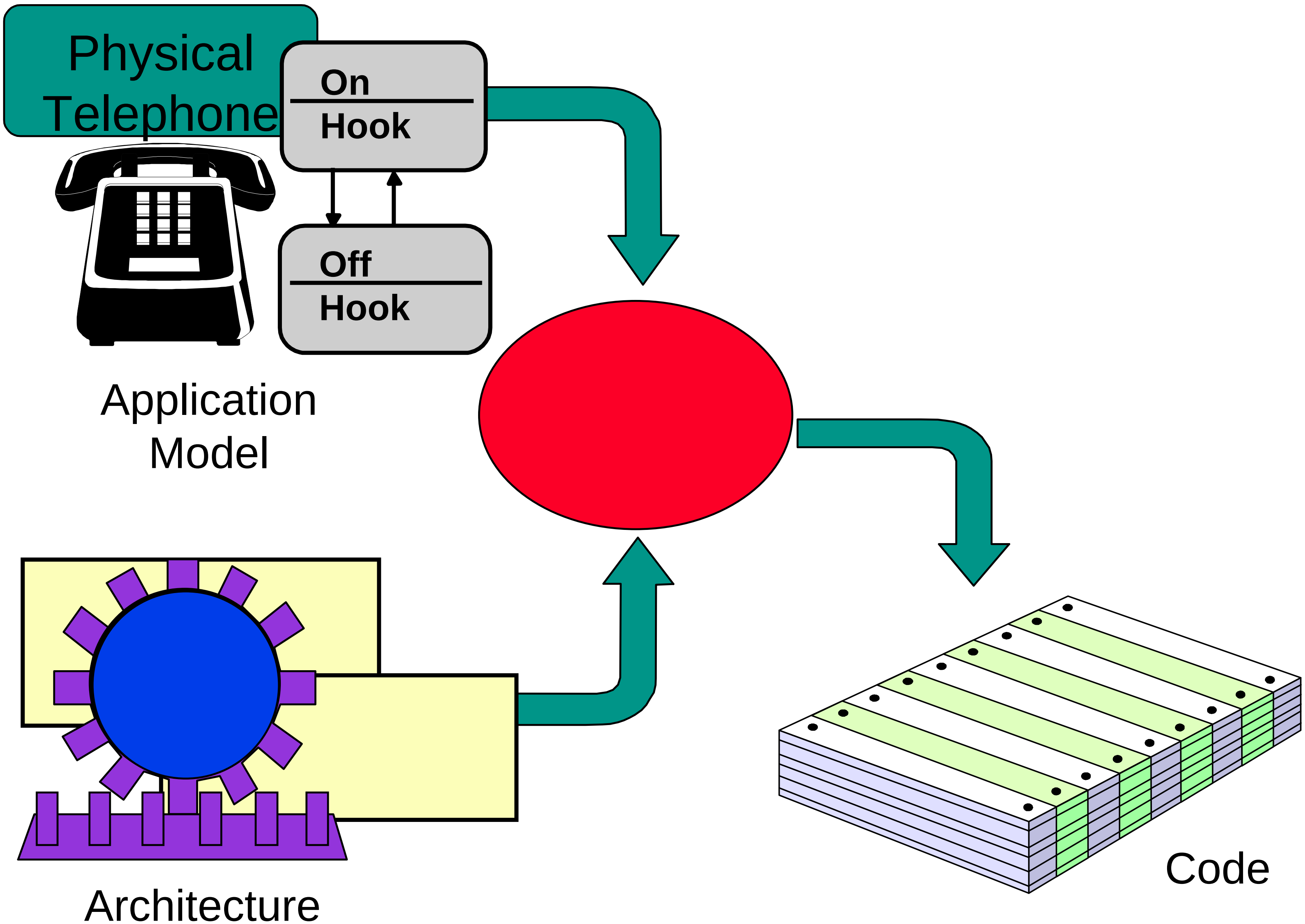
Executable UML

Stephen J. Mellor
Project Technology, Inc.
http://www.projtech.com

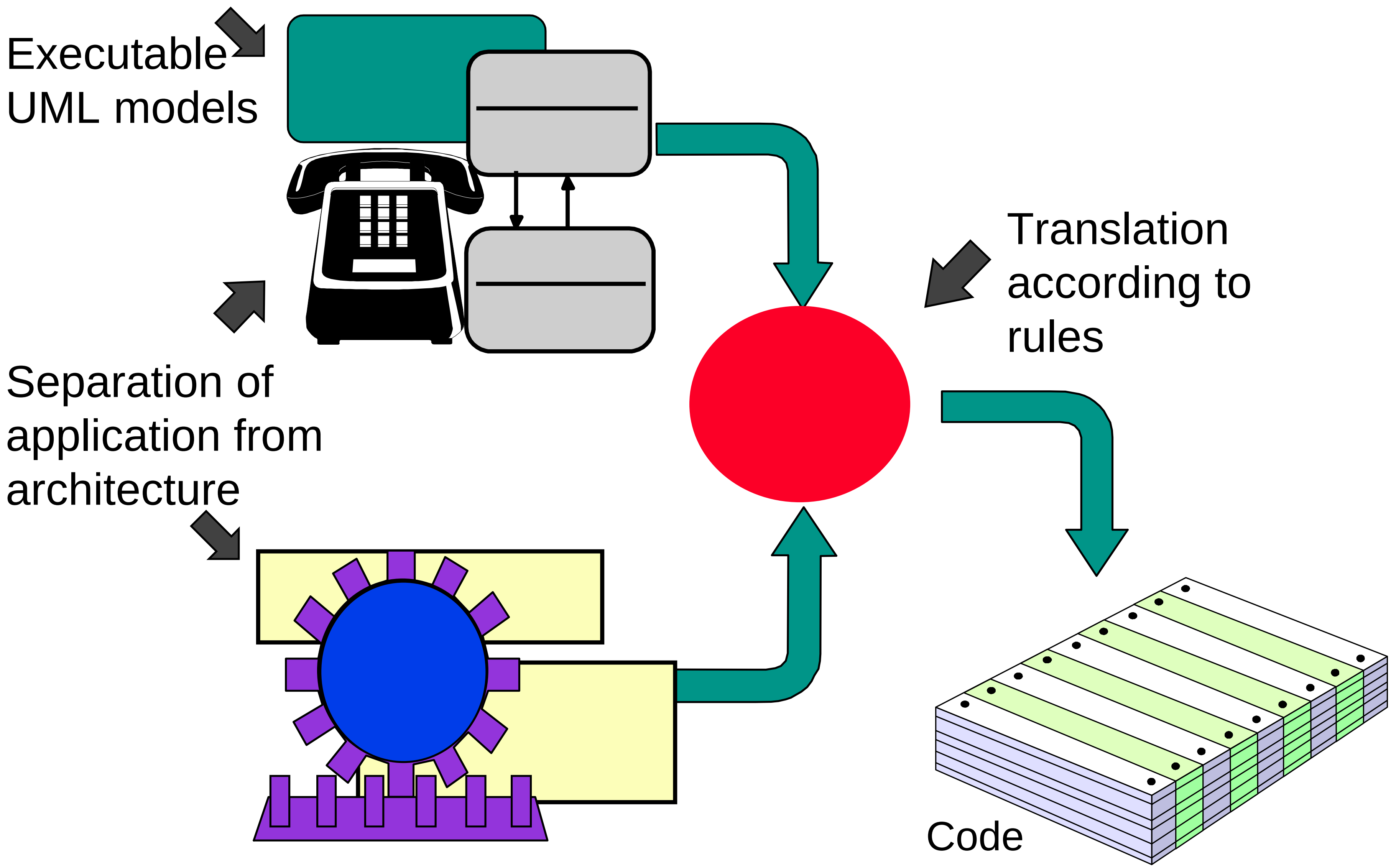
PROJECT TECHNOLOGY INC.

Application-Independent Software Architecture

PROJECT TECHNOLOGY, INC.



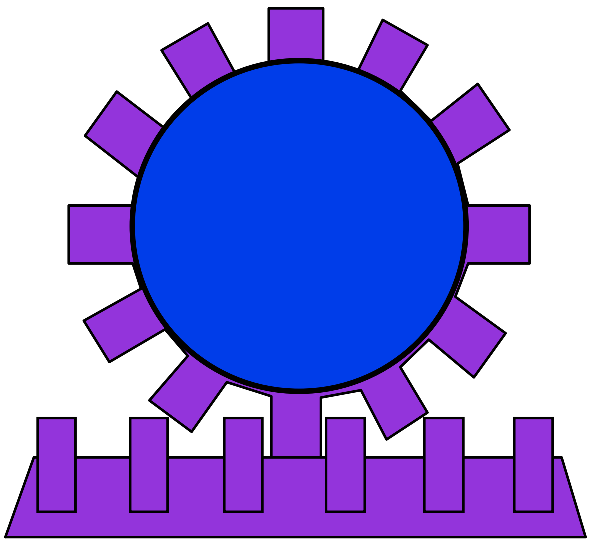
Properties


Executable
UML models
Separation of
application from
architecture
Translation
according to
rules
Code

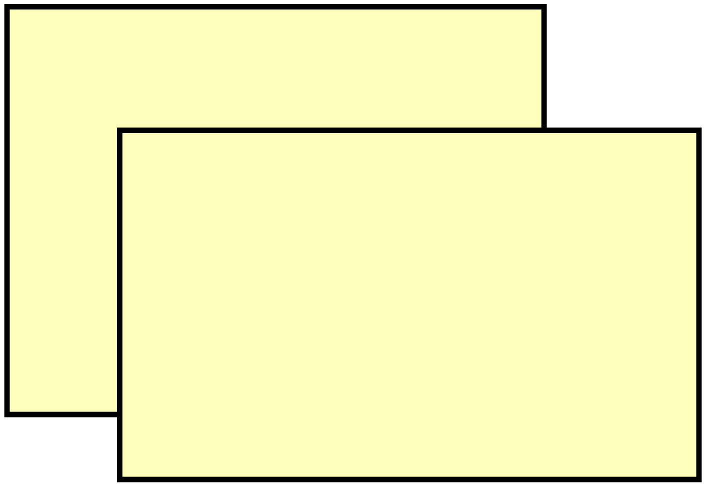
What's in the Architecture?

The architecture comprises:

- an execution engine plus
- a set of archetypes.



Execution Engine



Archetypes

Archetypes

Archetypes define the rules for translating the application into a particular implementation.

placeholder introduced by ${...}

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

};
```

text
(which happens
to be C++)

```
.Function PrivateDataMember( inst_ref class )
.select many PDMs from instances
   of Attribute related to Class
.for each PDM in PDMs
${PDM.Type} ${PDM.Name};
.endfor
```

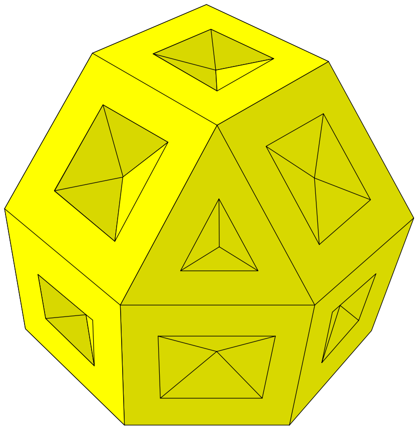
Application-Independent Software Architecture

The software architecture is independent of the semantics of the application.

This offers:

- early error detection through verification
- reuse of the architecture
- faster performance tuning
- faster integration
- faster, cheaper retargeting

The Software Architecture



Challenges of Real-Time Development

How can we both:

- provide required functionality

and

- meet real-time performance constraints?

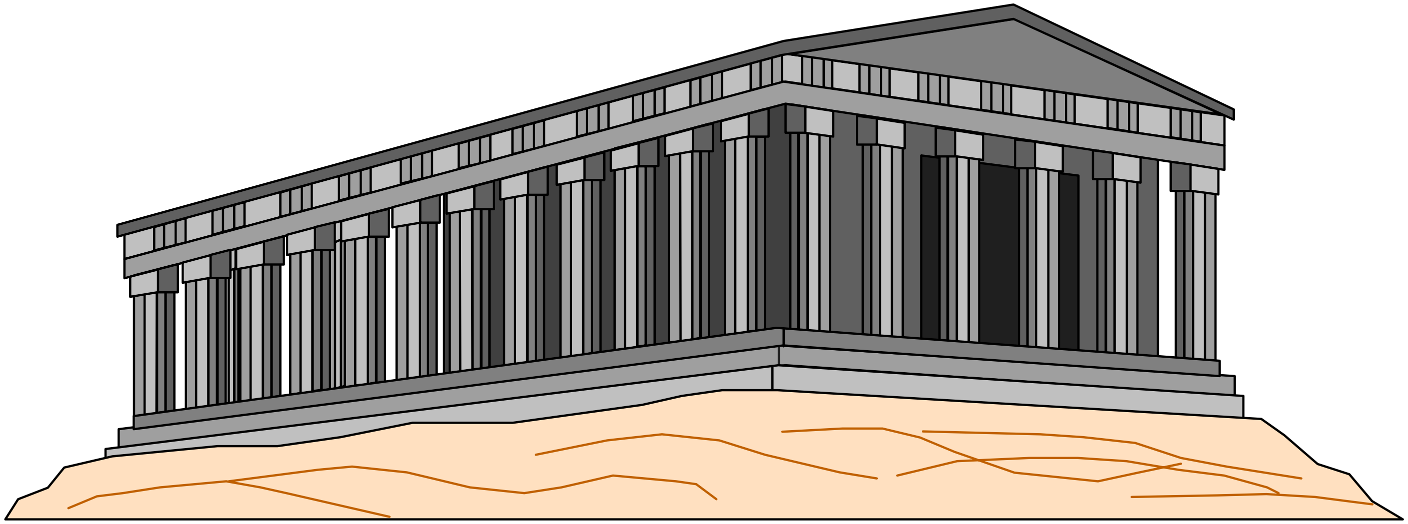
- (Re-)organize the software.

Software Architecture

The abstract organization of software is called the *software architecture*.

It proclaims and enforces system-wide rules for the organization of:

- data
- control
- structures
- time



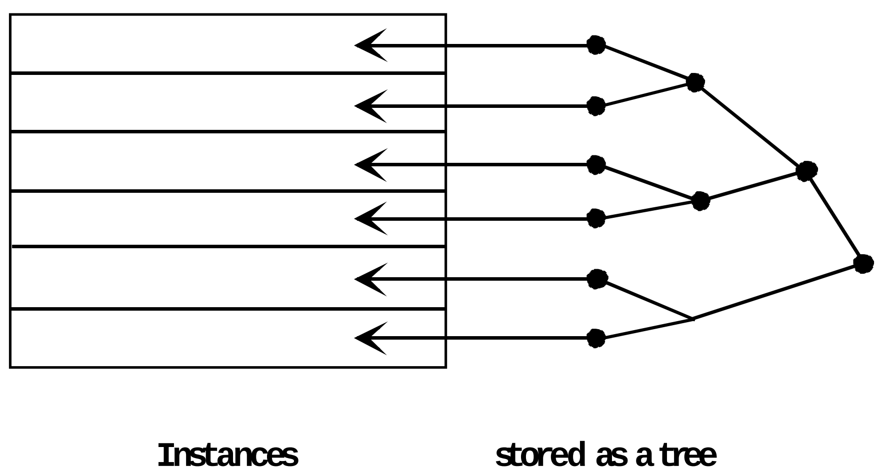
Data

The architect prescribes the *storage scheme* for data elements:

- tables or arrays?
- special purpose structures such as trees, linked lists?
- independent?

and *access* to them:

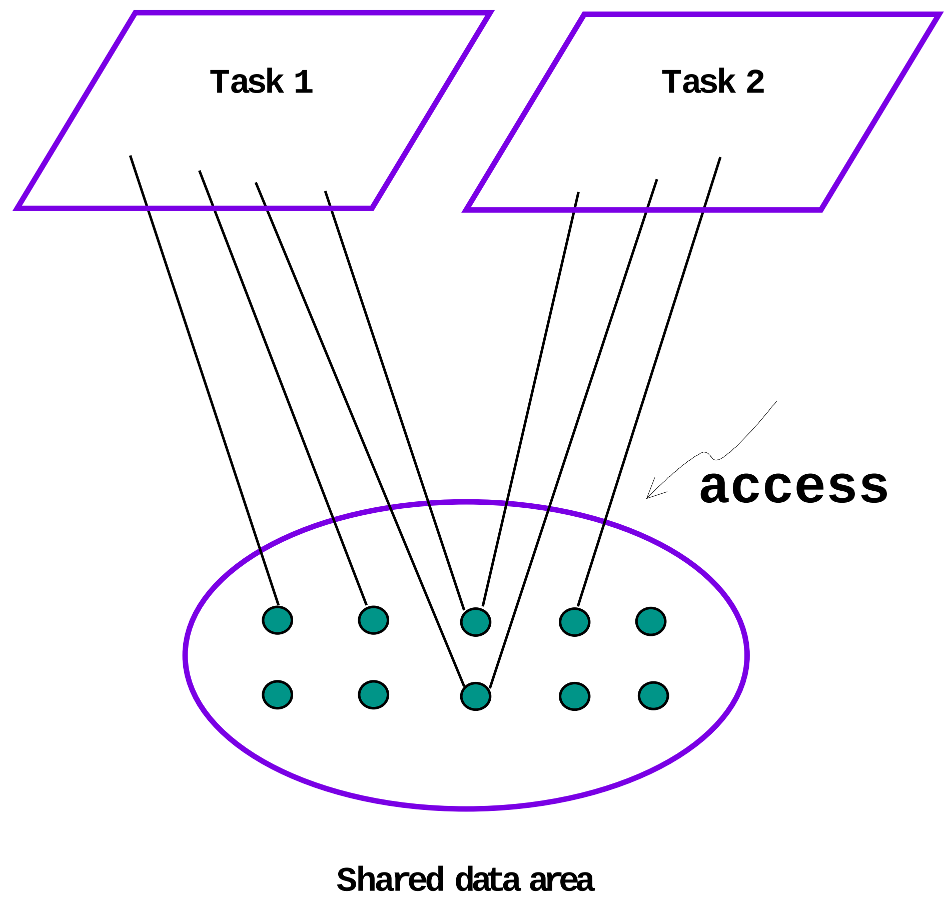
- direct access by name or pointer?
- indirect access through functions that encapsulate the data structure?



Control

The architect prescribes control:

- what causes a task to execute?
- what causes a task to relinquish control?
- what is the next function to execute within a task?
- how to coordinate multiple tasks accessing common data to ensure data consistency?

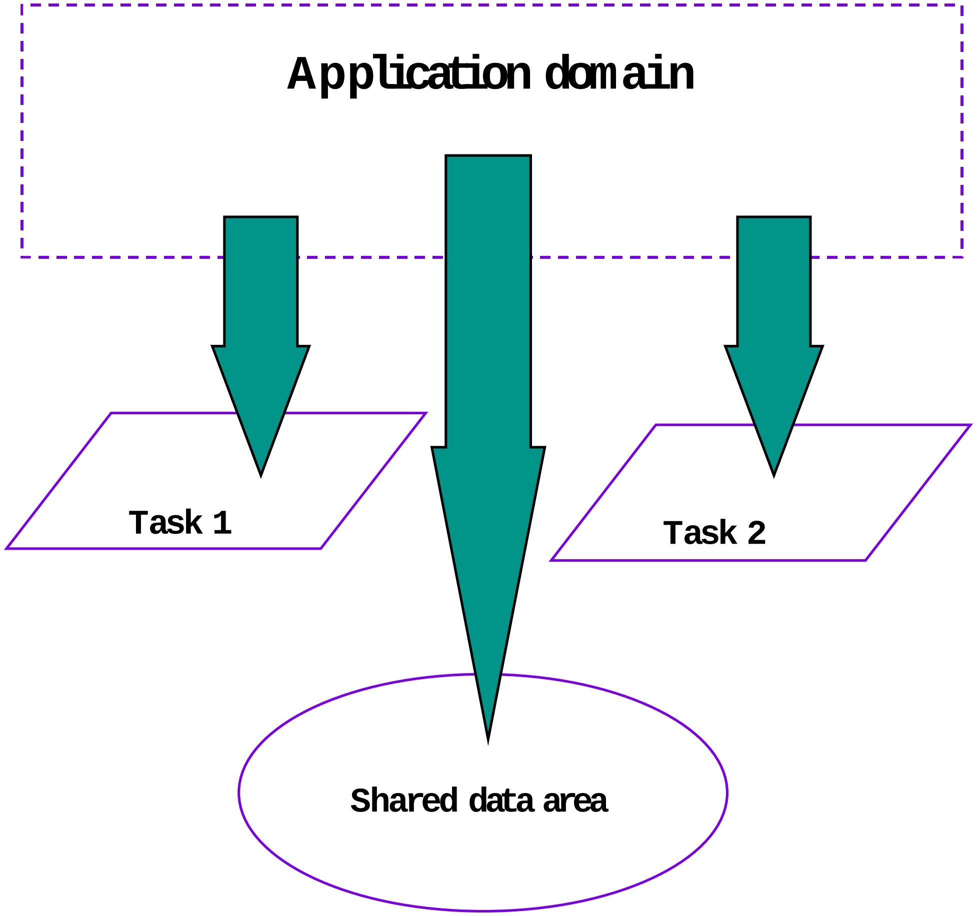


Structures

The architect prescribes how *to package* code and data in:

- tasks?
- functions?
- shared data areas?
- classes?

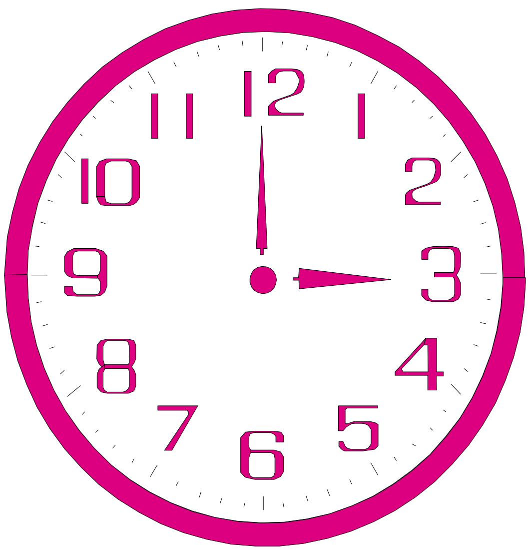
and the *allocation criteria* for allocating parts of the application to these structures.

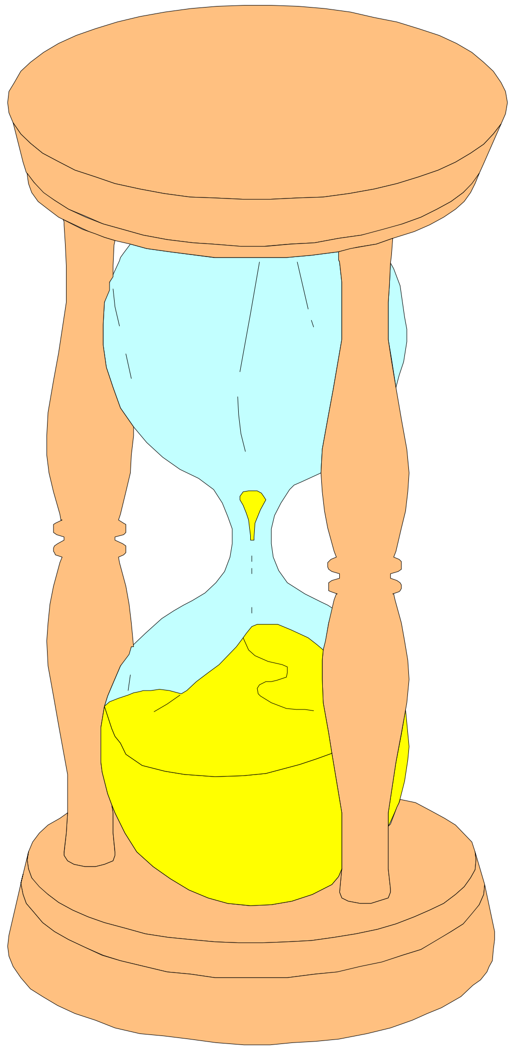


Time

The software architect prescribes how to provide time-related services:

- absolute time
- relative time

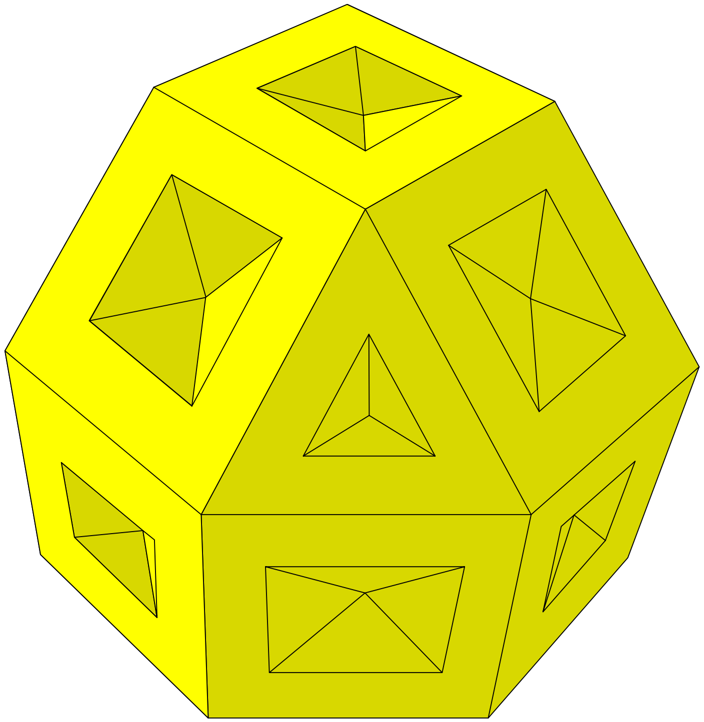




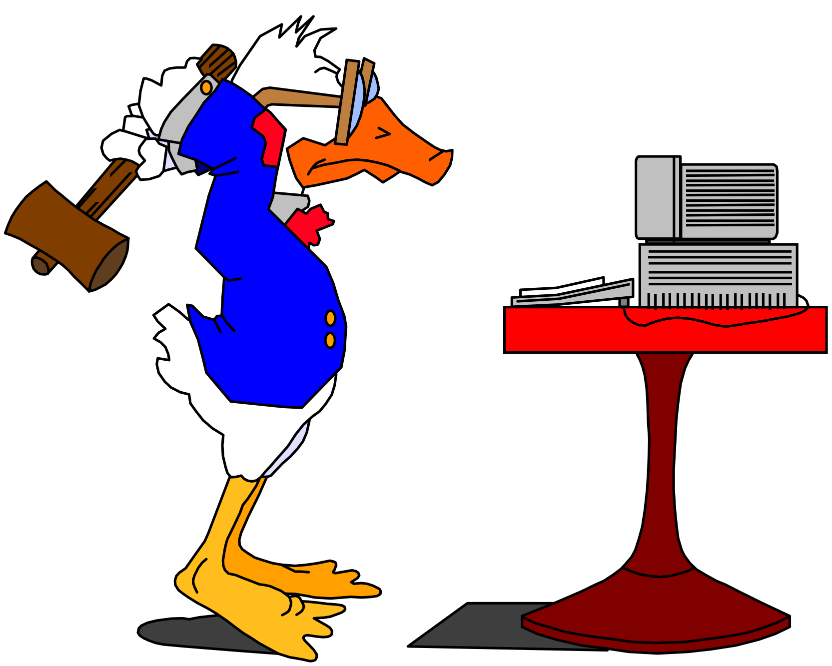
Uniformity

A minimal, uniform set of organization rules:

- reduces cost of understanding, building, and maintaining the software
- decreases integration effort
- leads to smaller, more robust code



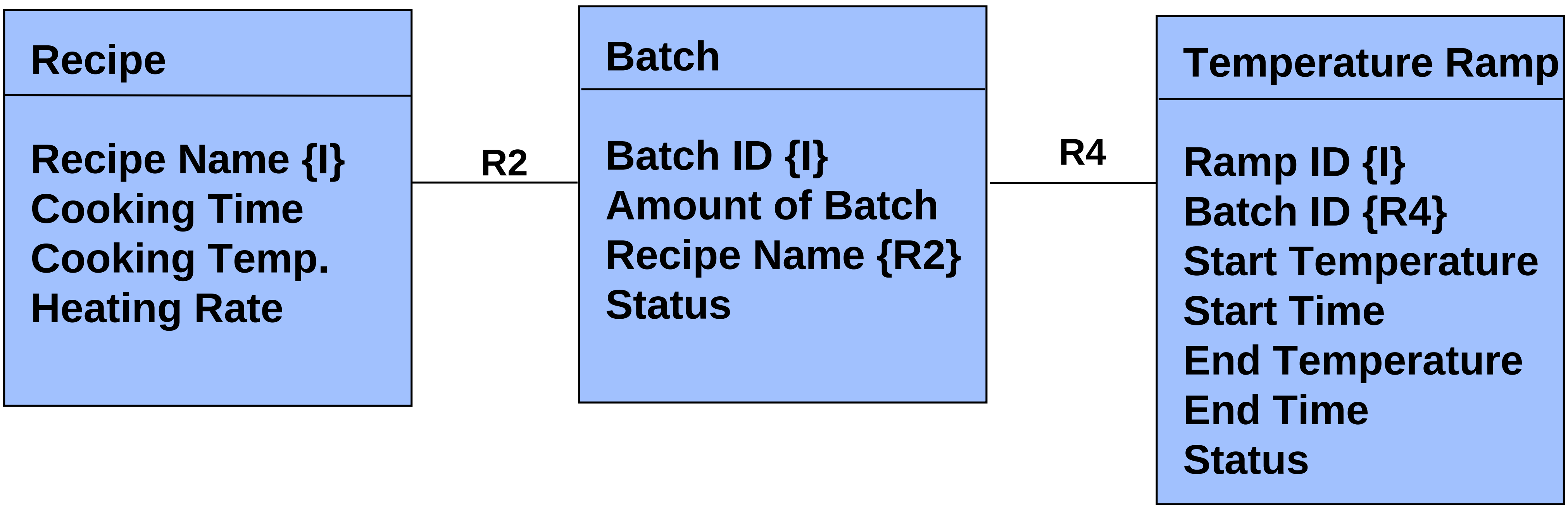
Executable Domain Models



Class Diagram

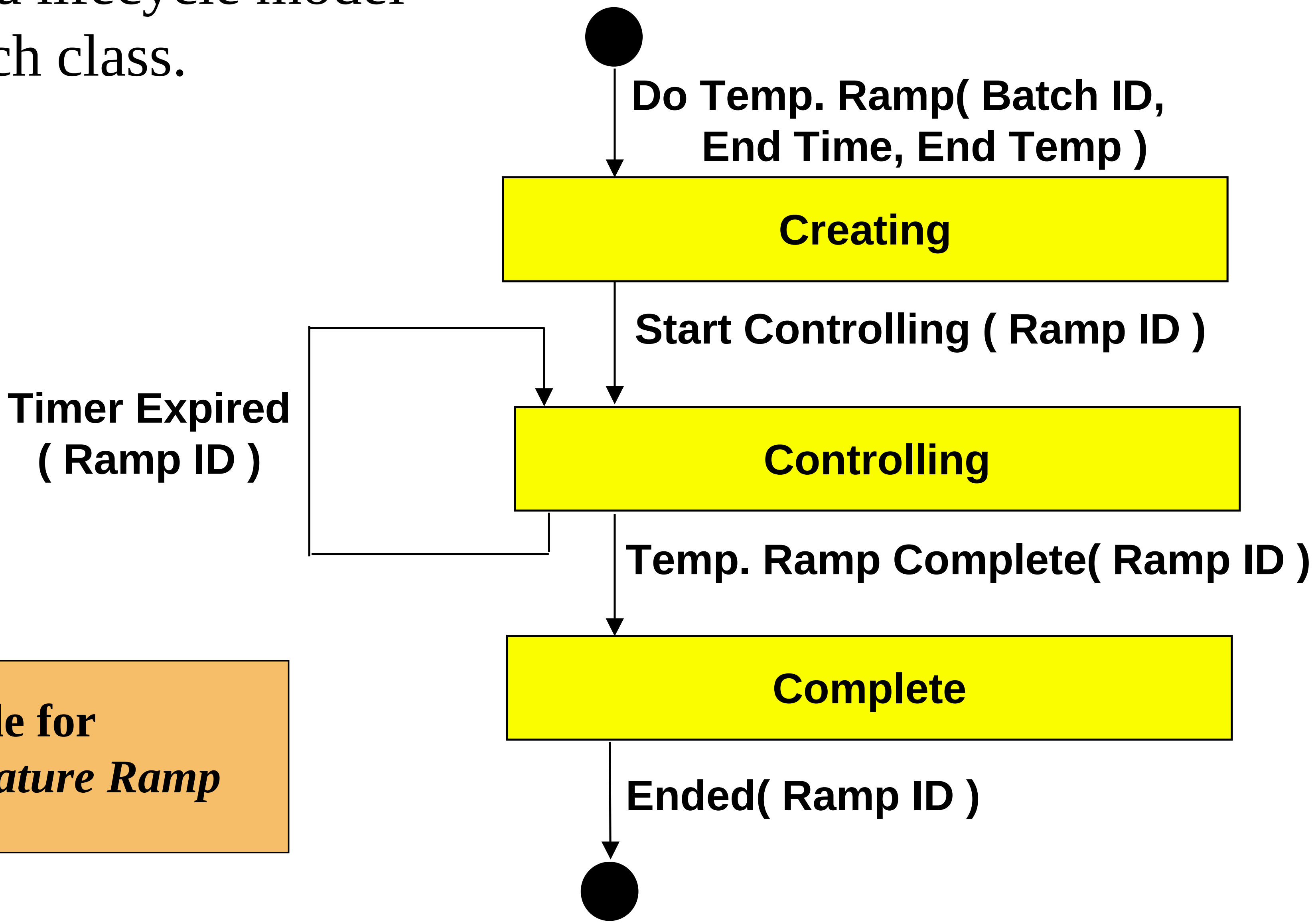
Abstract classes based on both:

- data, and
- behavior



Lifecycles

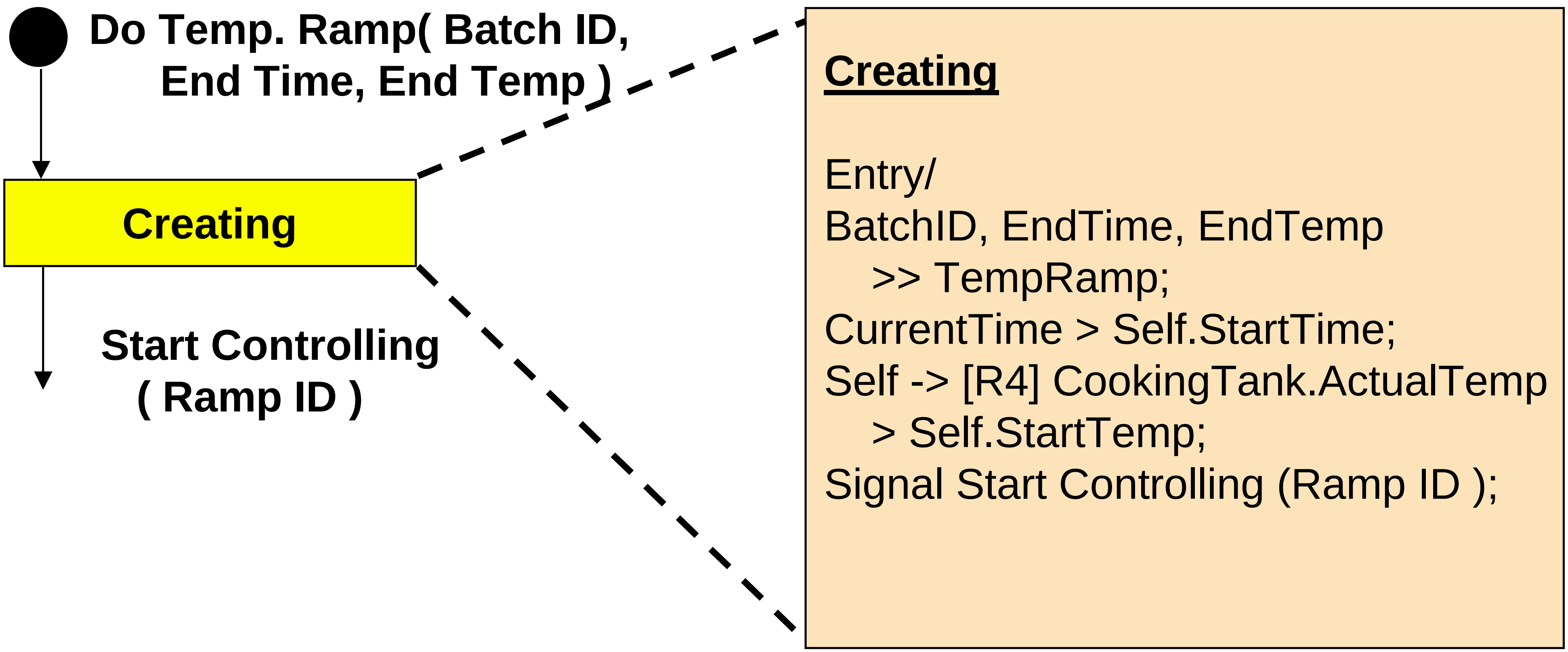
Build a lifecycle model for each class.



Lifecycle for *Temperature Ramp*

Actions

Specify the logic for each state's action.



Action Semantics

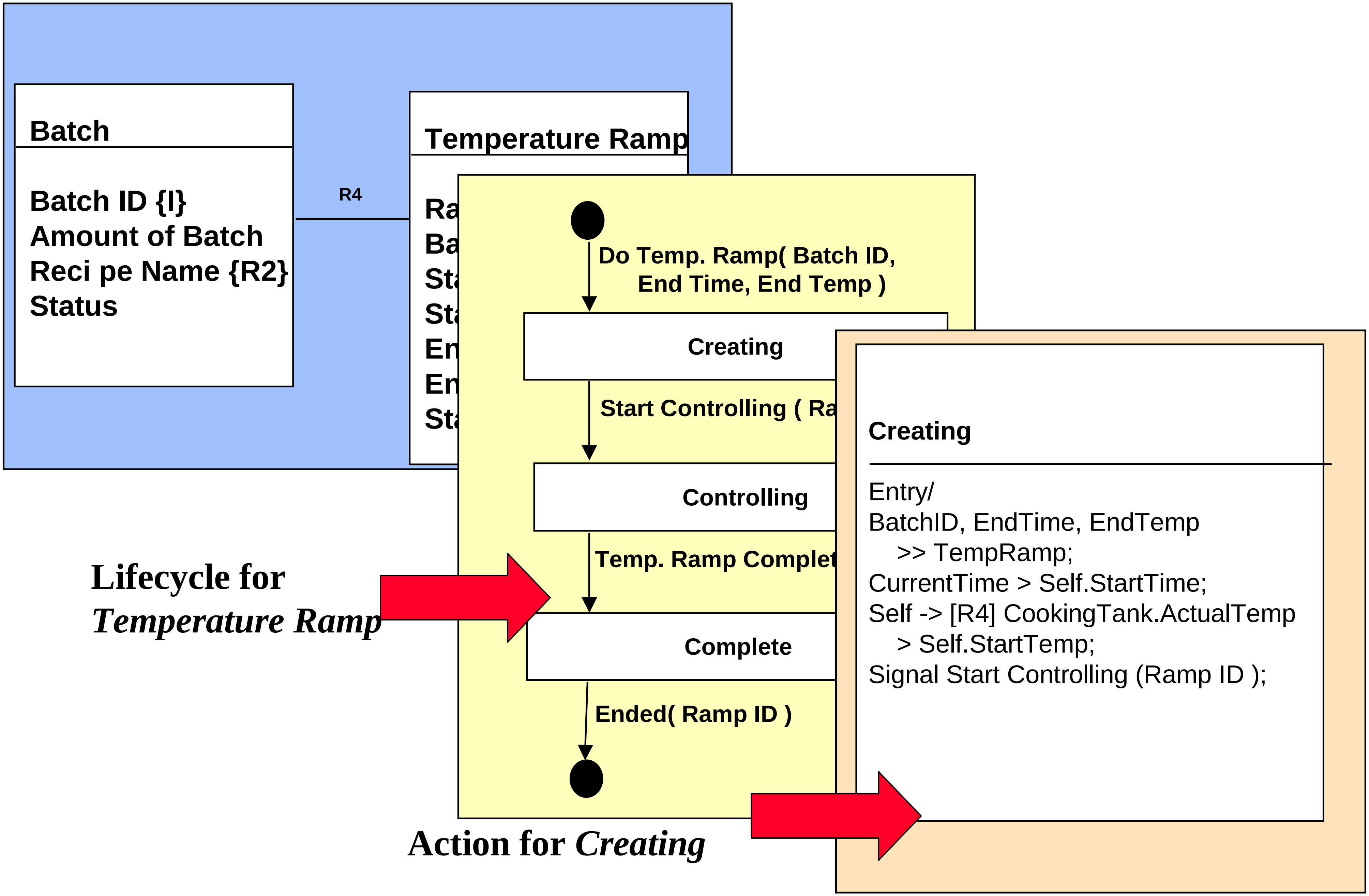
The action semantics should:

- not over-constrain sequencing
  - i.e concurrency & data flow
- separate computations from data access
  - to make decisions about data access without affecting algorithm specification
- manipulate only UML elements
  - to restrict the generality and so make a specification language

<u>Creating</u>

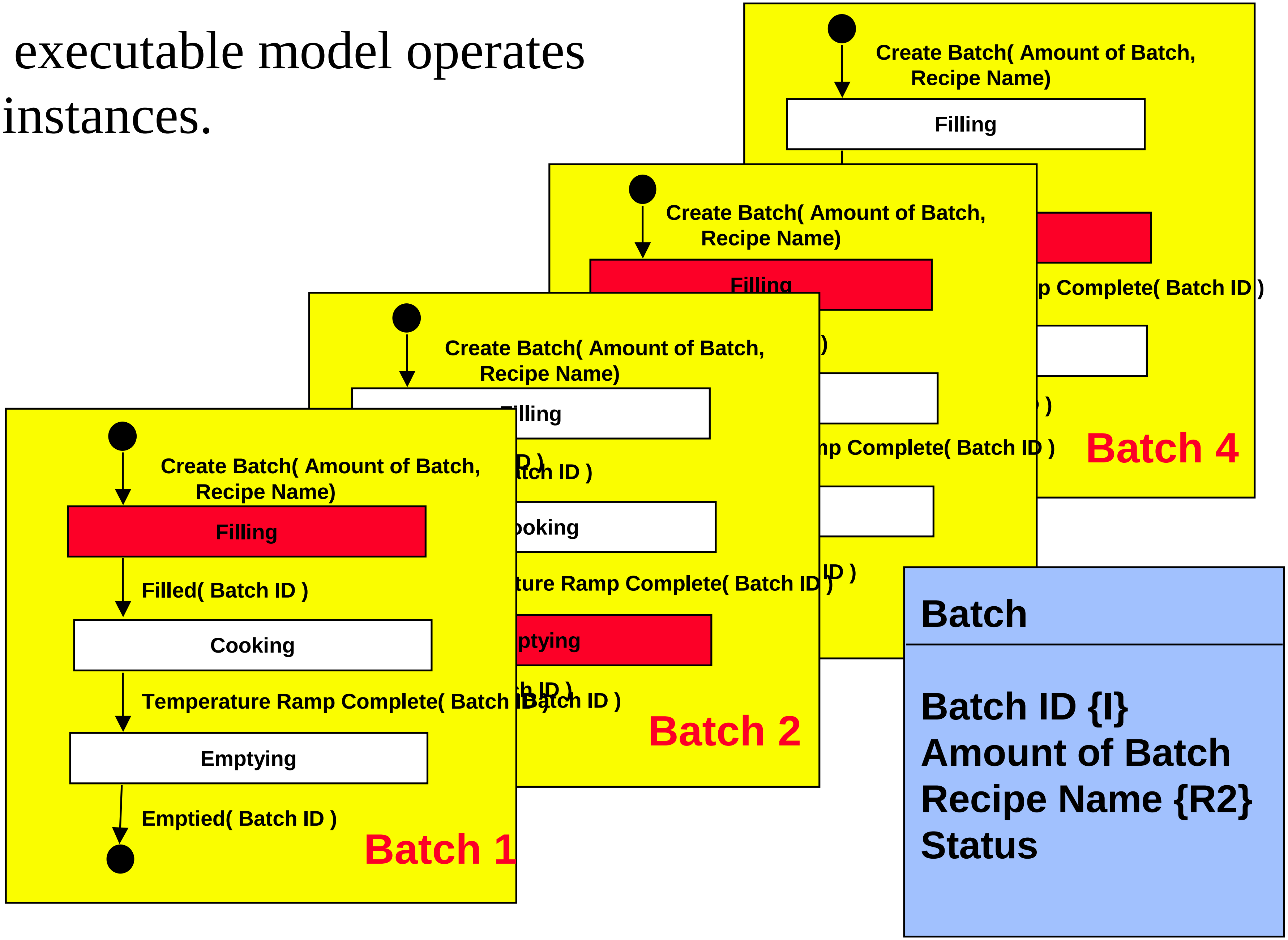
Entry/
BatchID, EndTime, EndTemp
    >> TempRamp;
CurrentTime > Self.StartTime;
Self -> [R4] CookingTank.ActualTemp
    > Self.StartTemp;
Signal Start Controlling (Ramp ID);

An Executable Model



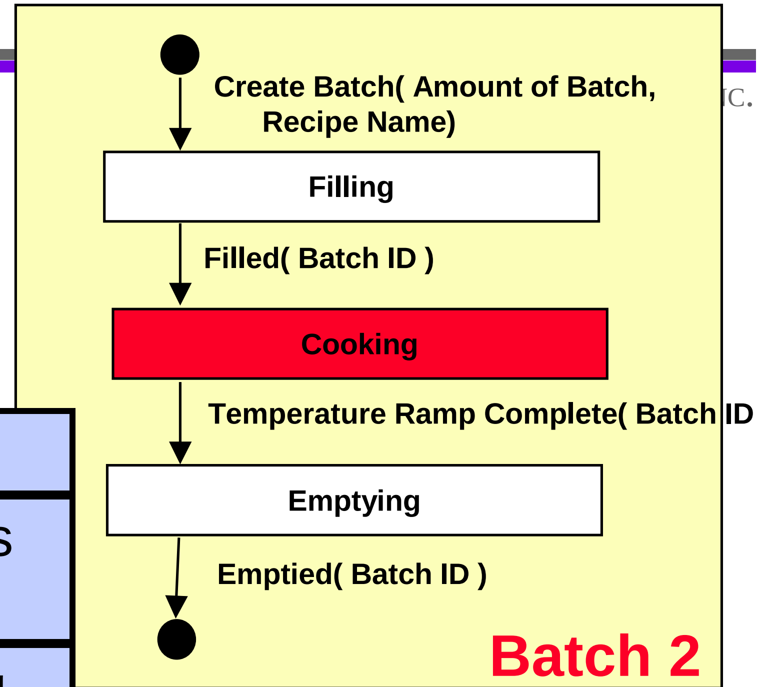
Instances

An executable model operates on instances.

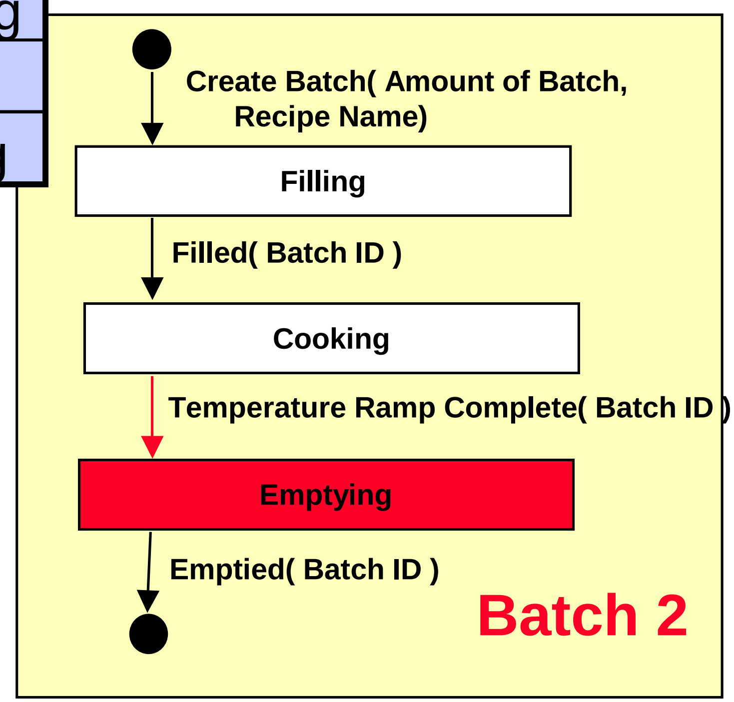


Execution

The lifecycle model prescribes execution.



Batch			
Batch ID	Amount of Batch	Recipe Name	Status
1	100	Nylon	Filling
2	127	Kevlar	Emptying
3	93	Nylon	Filling
4	123	Stuff	Cooking

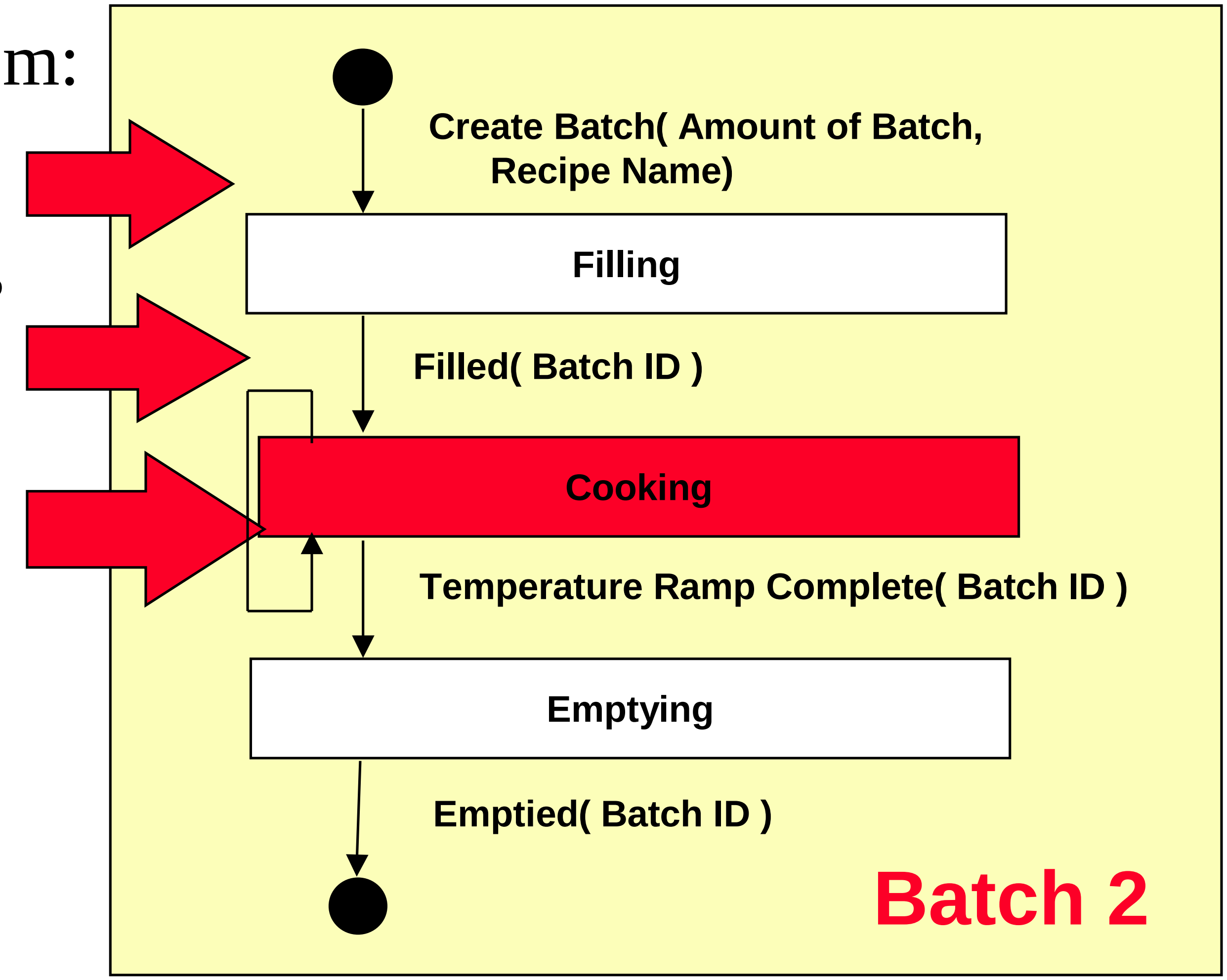


When the Temperature Ramp is complete, the instance moves to the next state....and executes actions.

Executing the Model

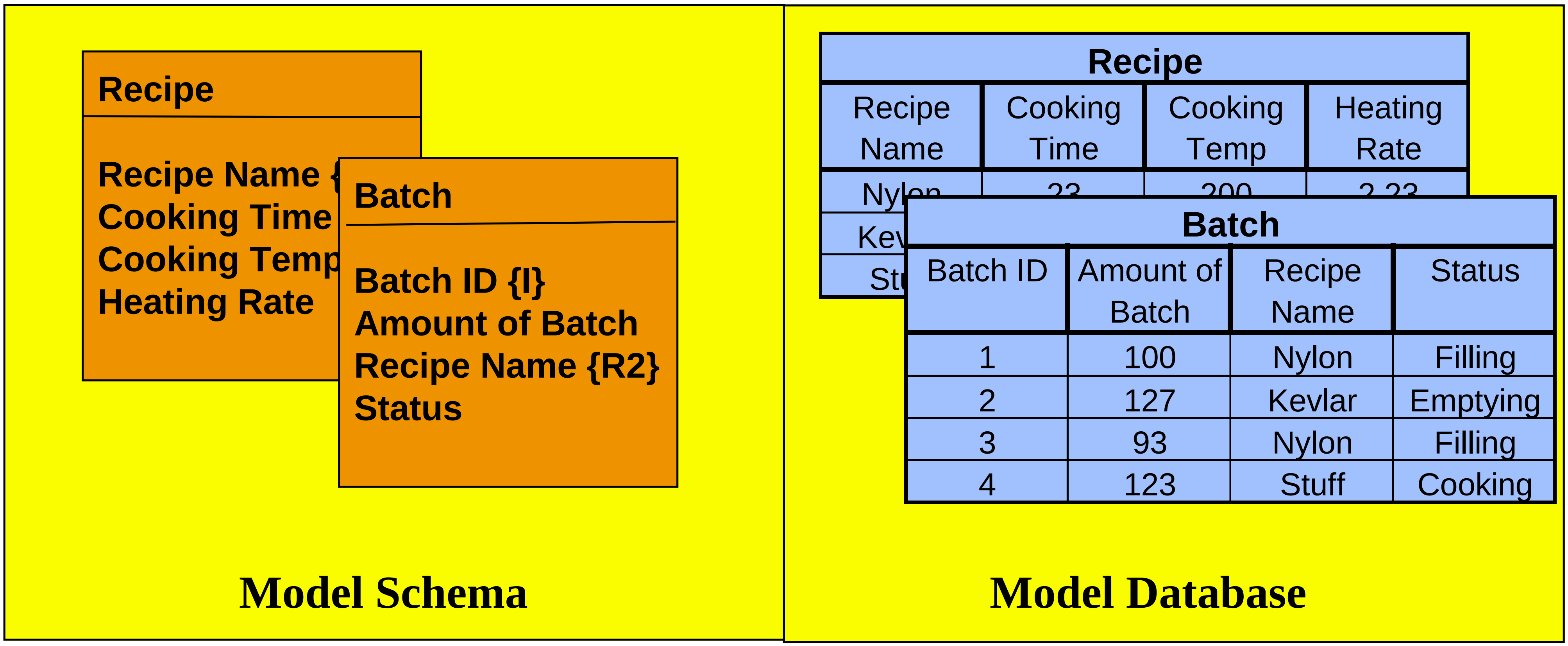
The model executes in response to signals from:

- the outside,
- other instances as they execute
- timers

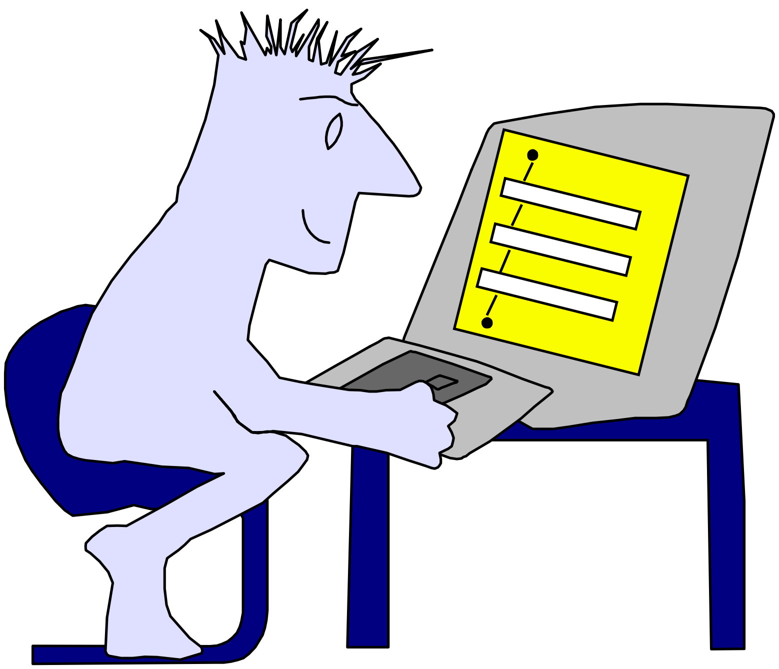


Model Database

Each schema has a corresponding database for instances.



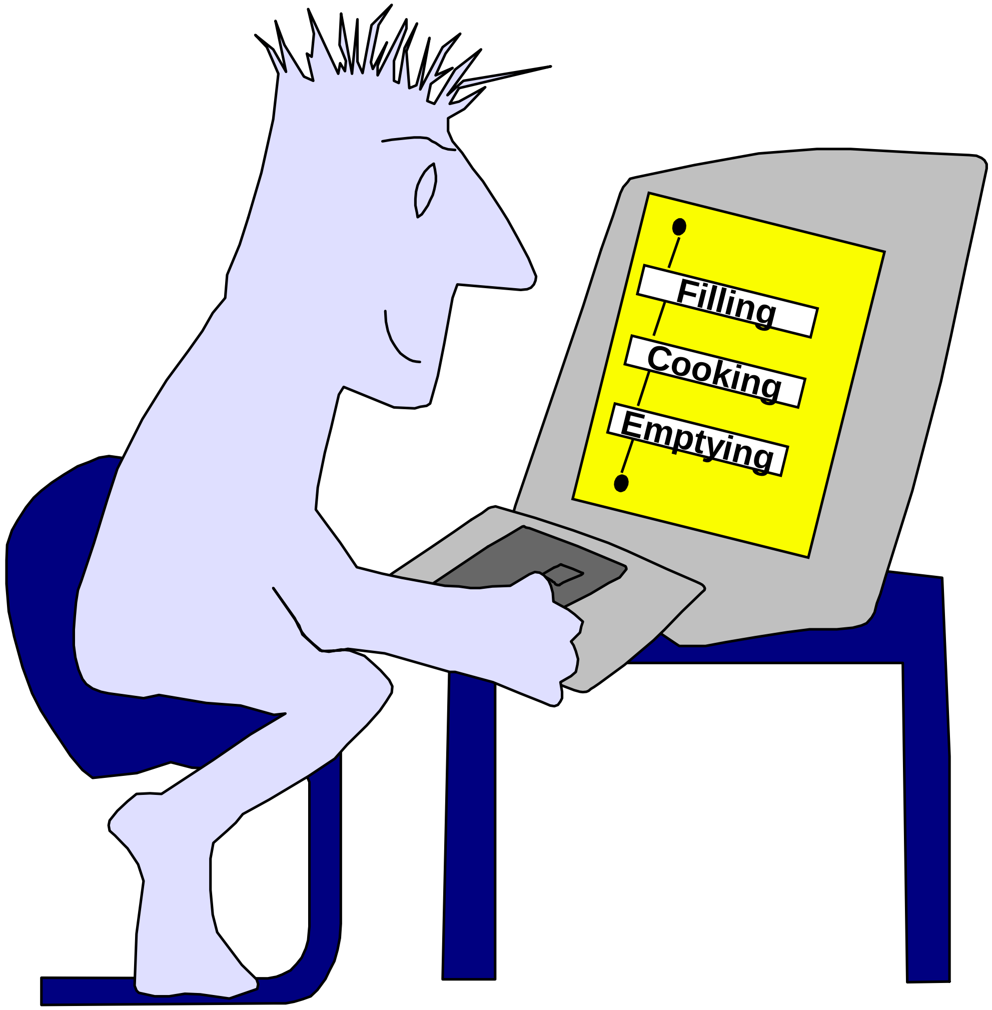
Model Capture



Model Repository

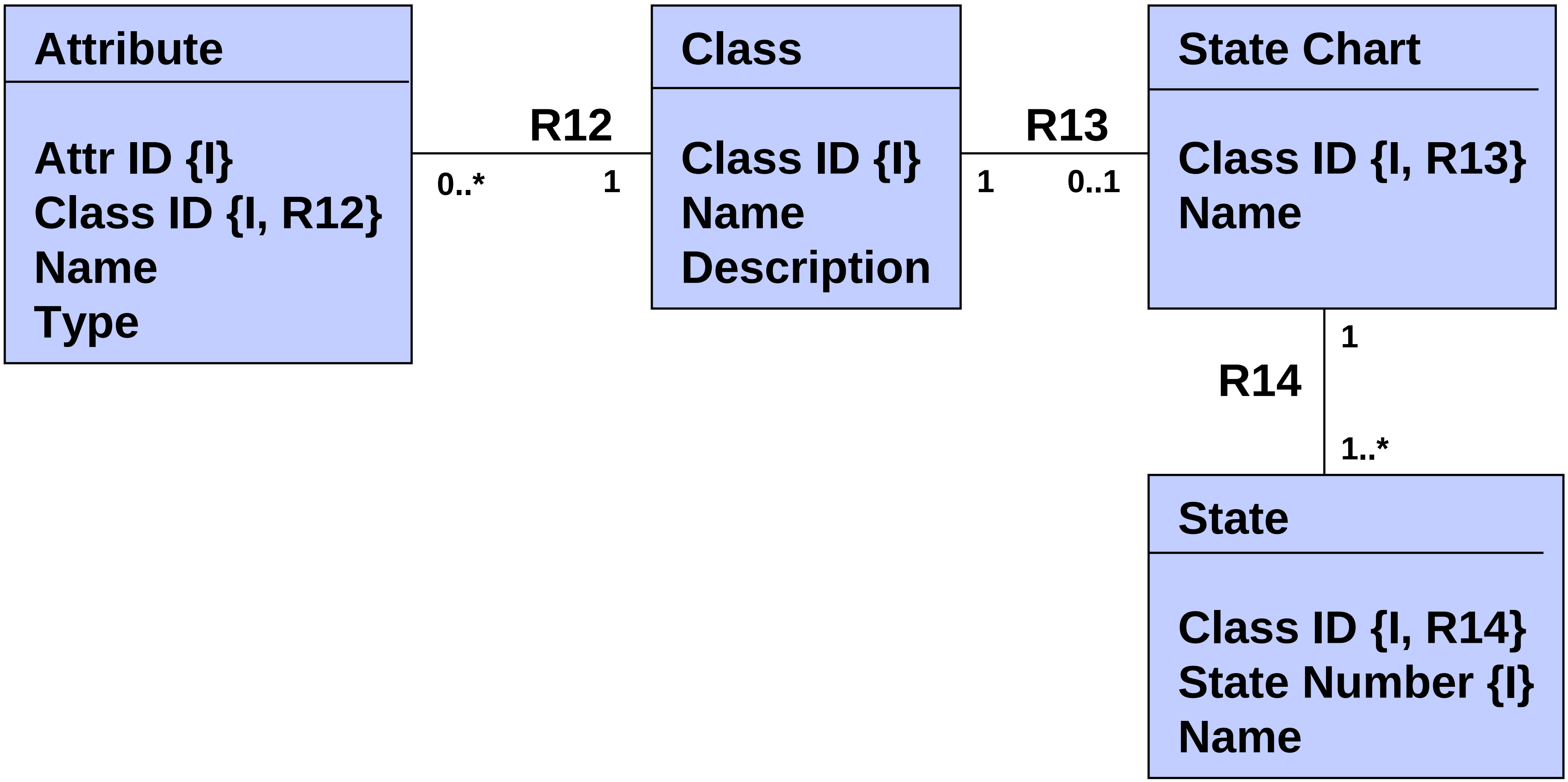
Capture the model in a model repository.

What is the structure of the repository?



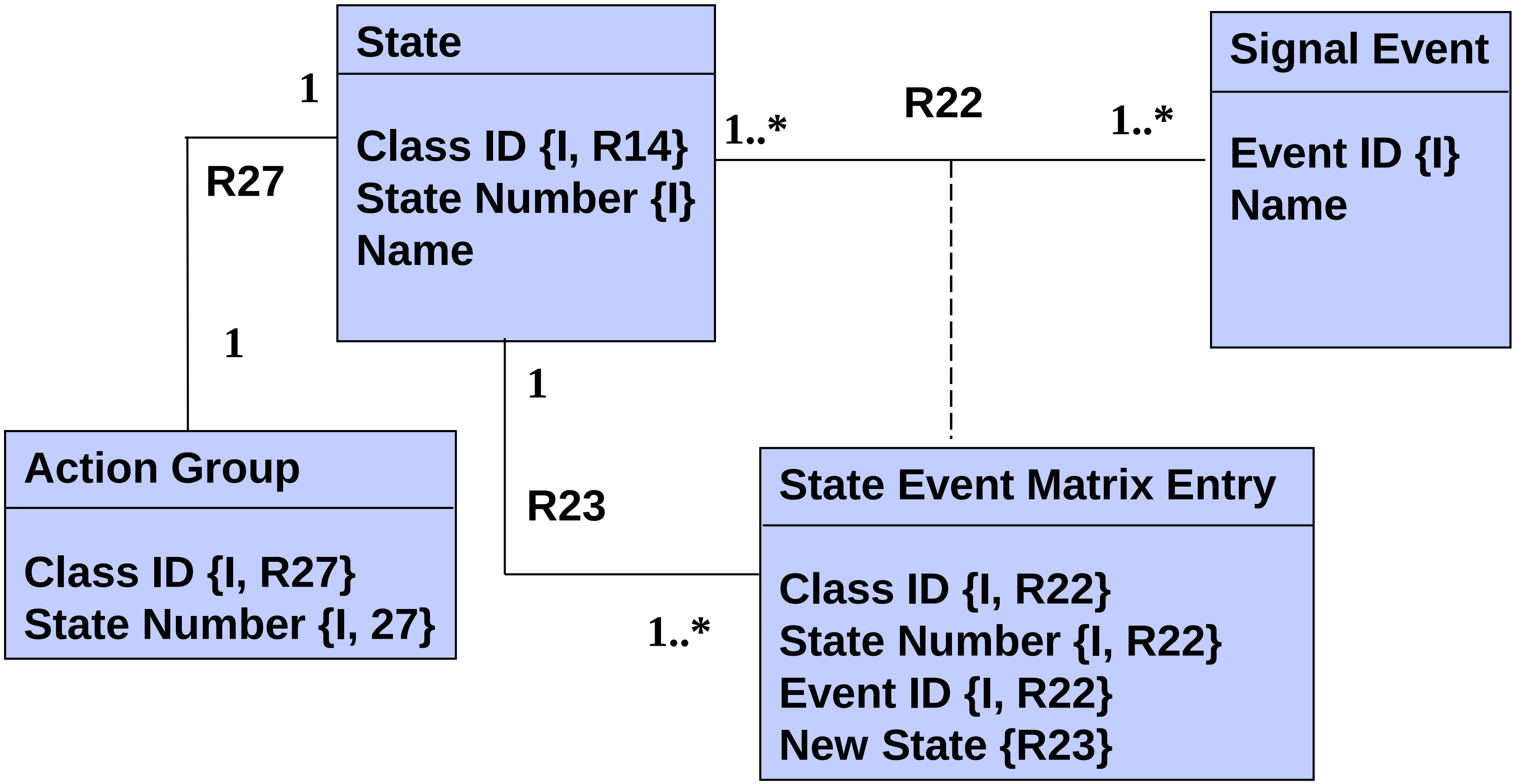
Model Structure

A *meta-model* defines the structure of the repository.



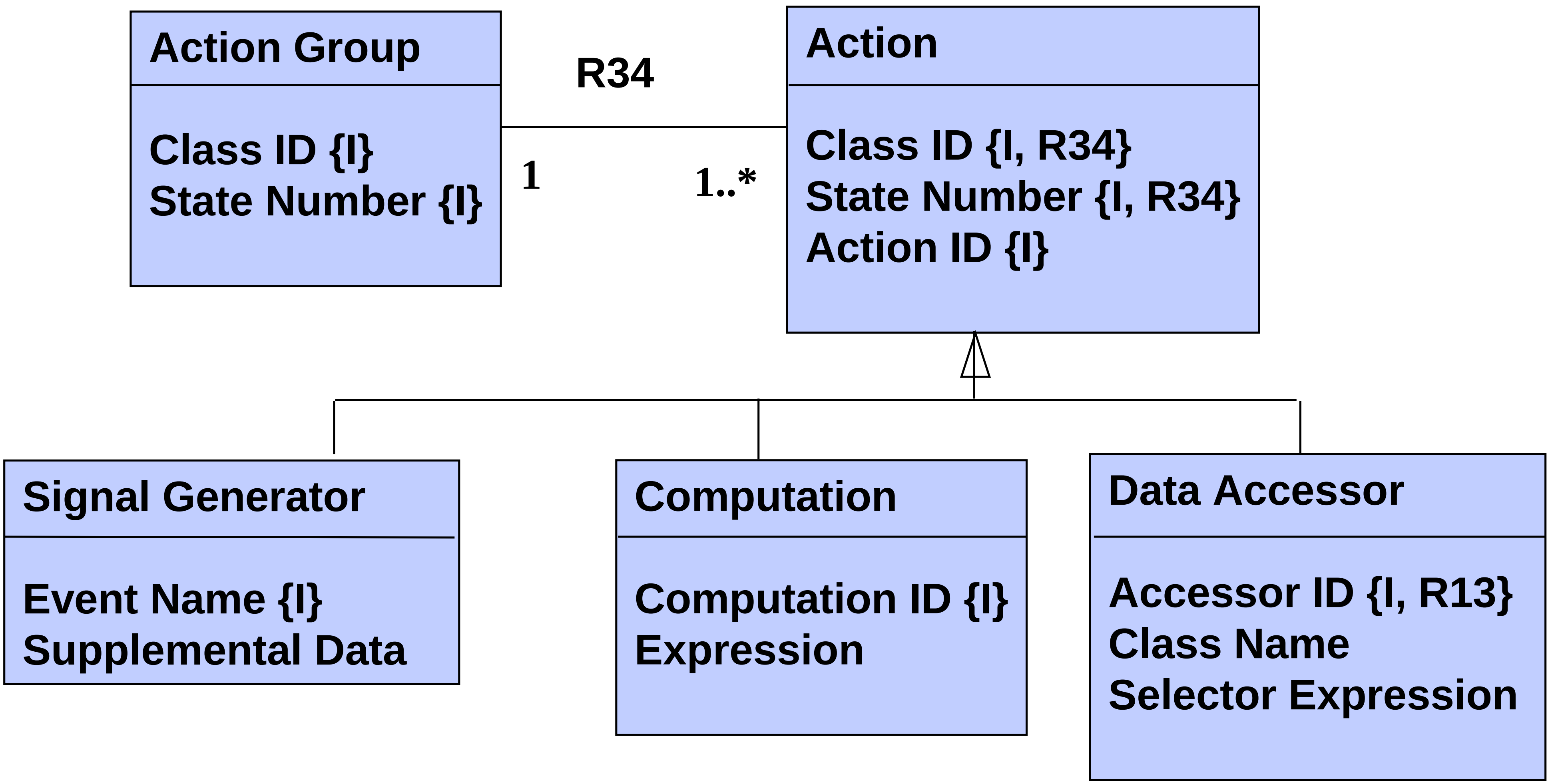
Model Structure

A *meta-model* defines the structure of the repository.



Model Structure

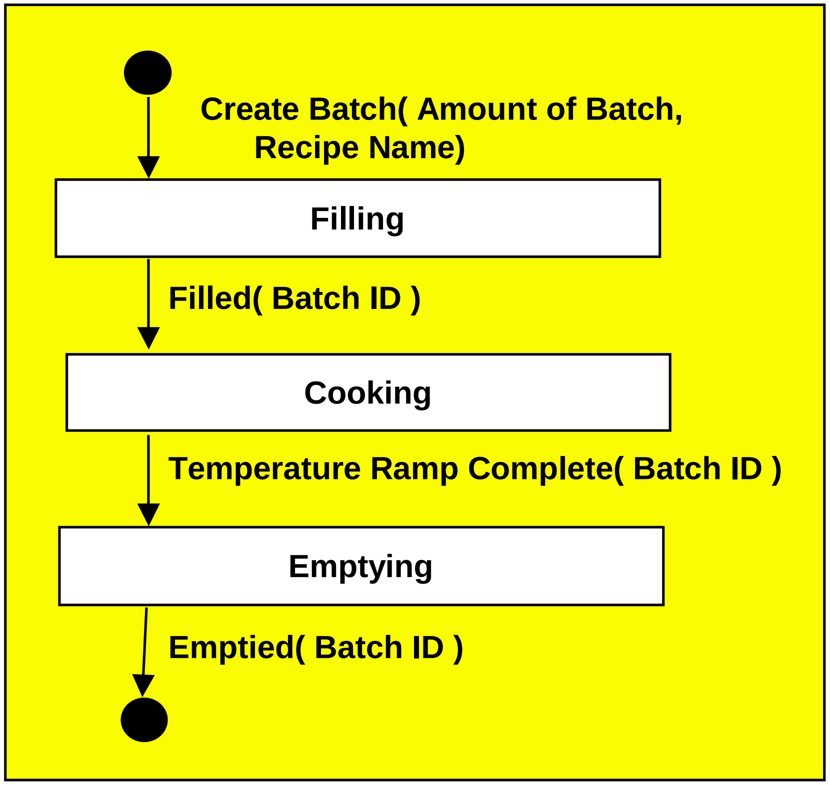
A *meta-model* defines the structure of the repository.



Meta-Model Instances

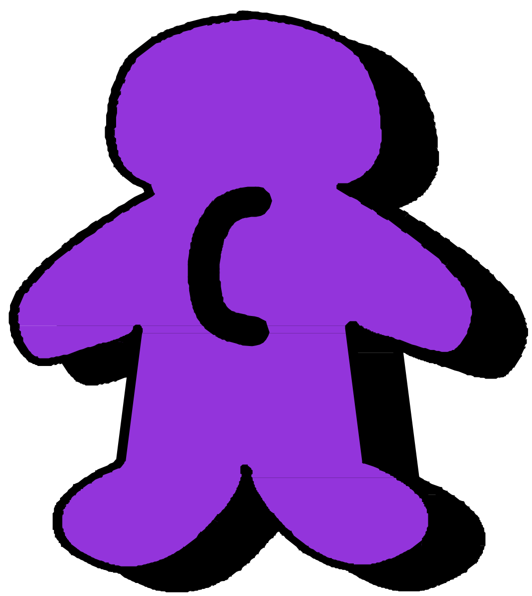
Just like an application model, the meta-model has instances.

Class		
Class ID	Name	Descr'n
100	Recipe	.....
101	Batch	.....
102	Temp Ramp	.....



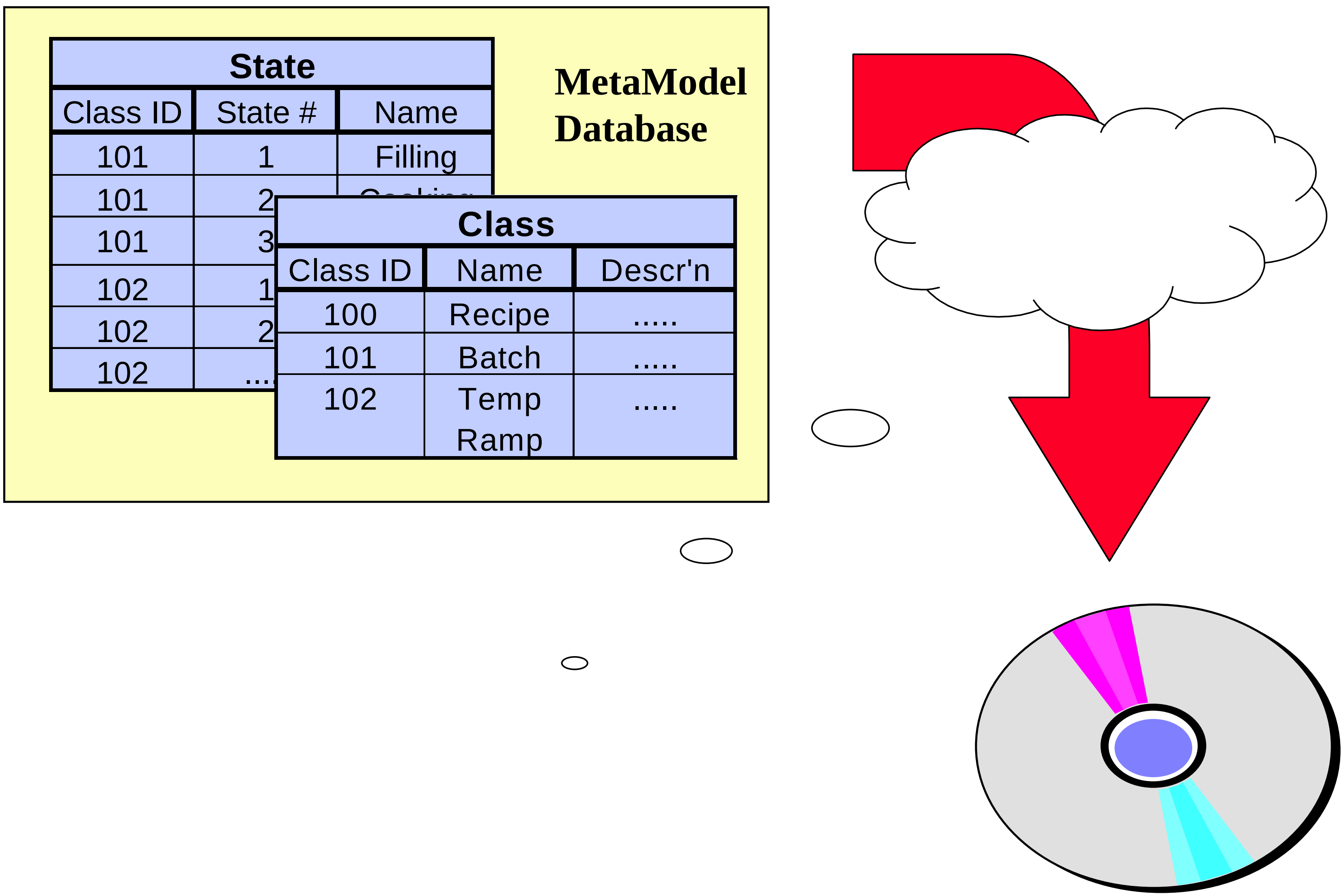
State		
Class ID	State #	Name
101	1	Filling
101	2	Cooking
101	3	Emptying
102	1	....
102	2	.....
102	.....	.....

Archetype Language



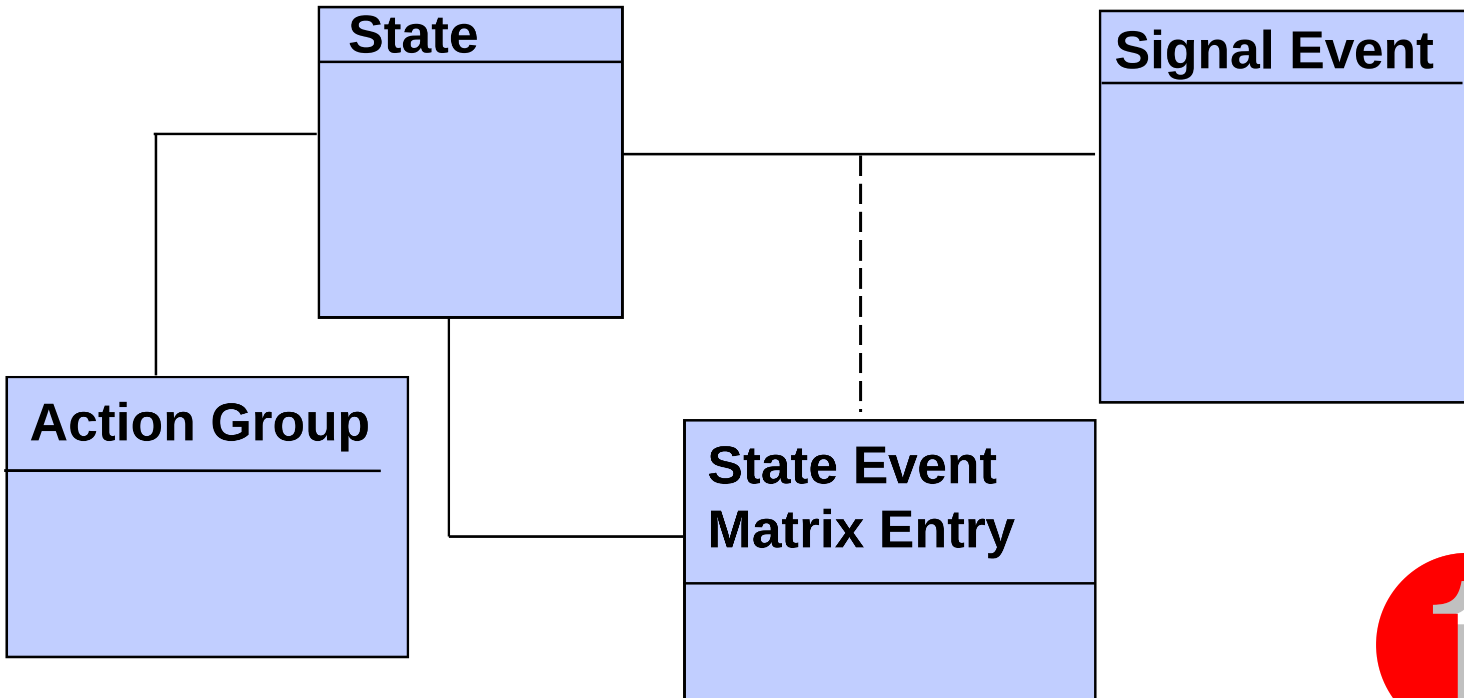
Purpose

To generate code….



Purpose

….traverse the repository and...



… output text.

Example

The archetype language produces text.

```
.select many stateS related to instances of
      class->State->StateChart
          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 ,
       Filling ,
       Cooking ,
       NUM_STATES = Emptying
    };
```

Text

To generate text:

The quick brown fox jumped over the lazy dog.

```
The quick brown fox jumped over the lazy dog.
```

Data Access

To select any instance from the repository:

.select any *class* from instances of <u>Class</u>

`Instance reference`

Class		
Class ID	Name	Descr'n
100	Recipe	.....
101	Batch	.....
102	Temp Ramp	.....

Substitution

To access attributes of the selected instance….

${*class*.Name}

Class		
Class ID	Name	Descr'n
100	Recipe	.....
101	Batch	.....
102	Temp Ramp	.....

`Batch`

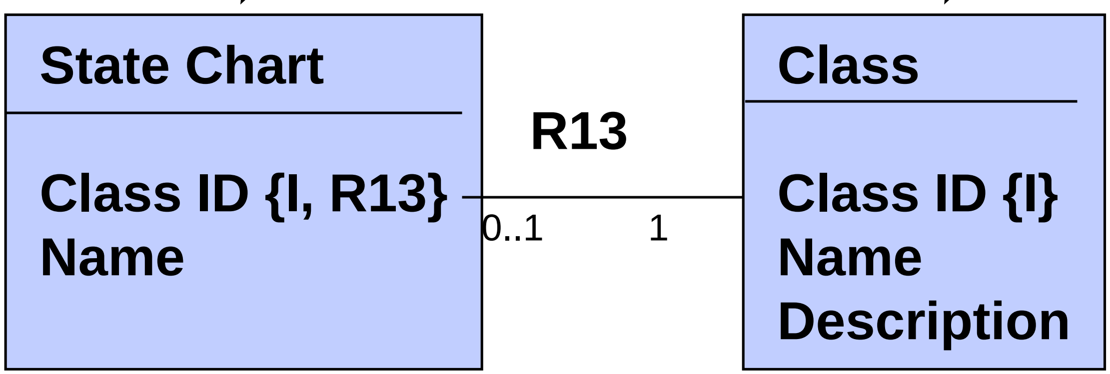
`just text`

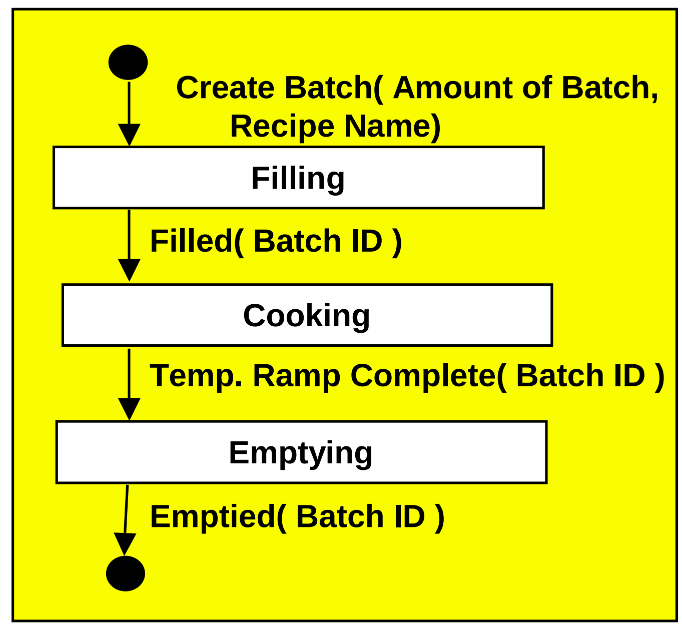
Association Traversal

To traverse an association…..

Not just any one--
the one that's associated

.select one *StateChart* related to instances of *class*->StateChart

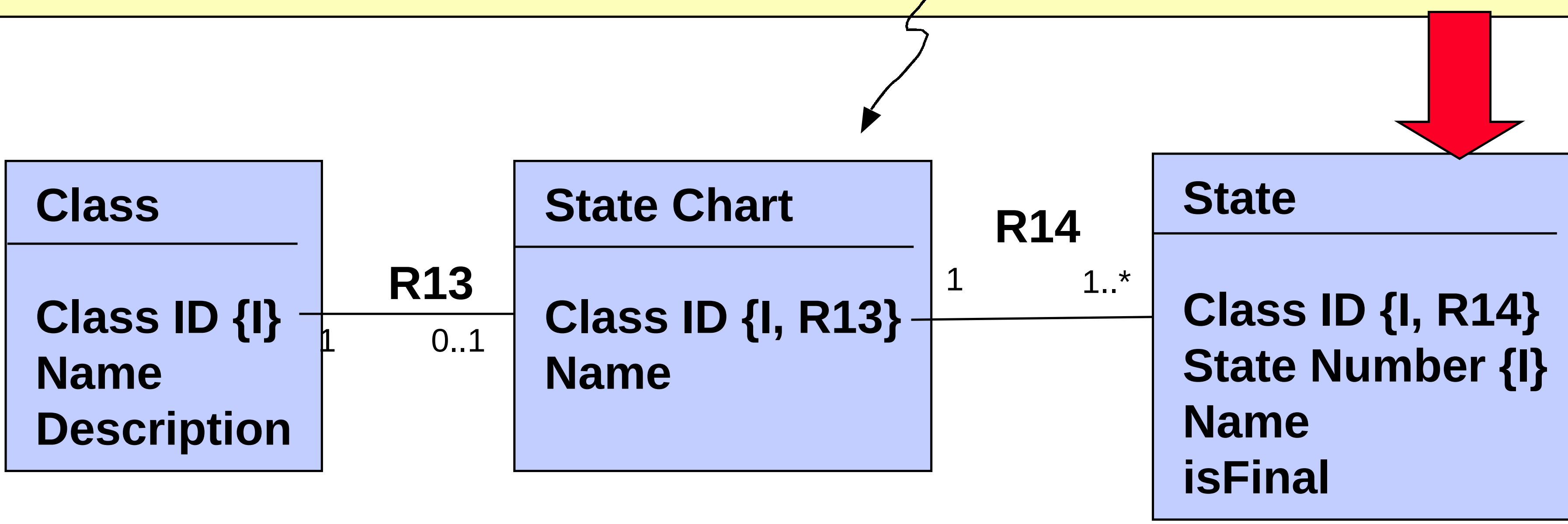




Arbitrary Instance

To select an arbitrary one….

.select any *state* related to instances of *StateChart*->State



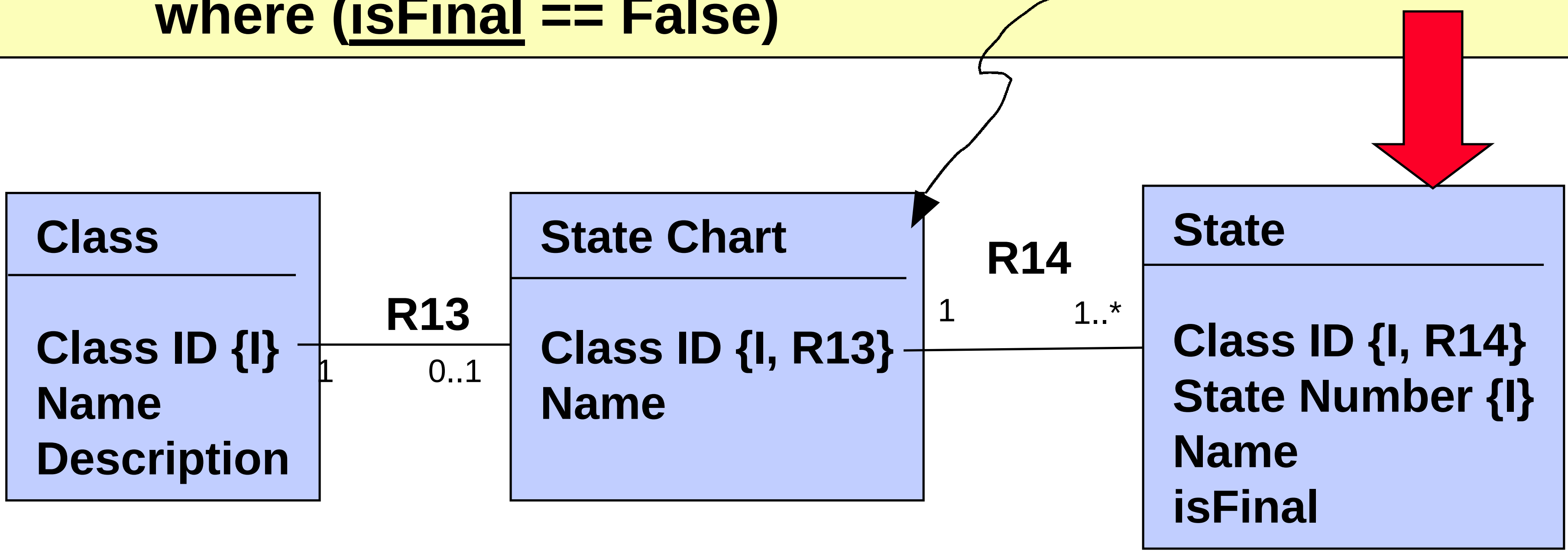
Or...

.select any *state* related to instances of *Class*->StateChart->State

Complex Traversals

To qualify the selection...

.select any *state* related to instances of *StateChart*->State
where (isFinal == False)



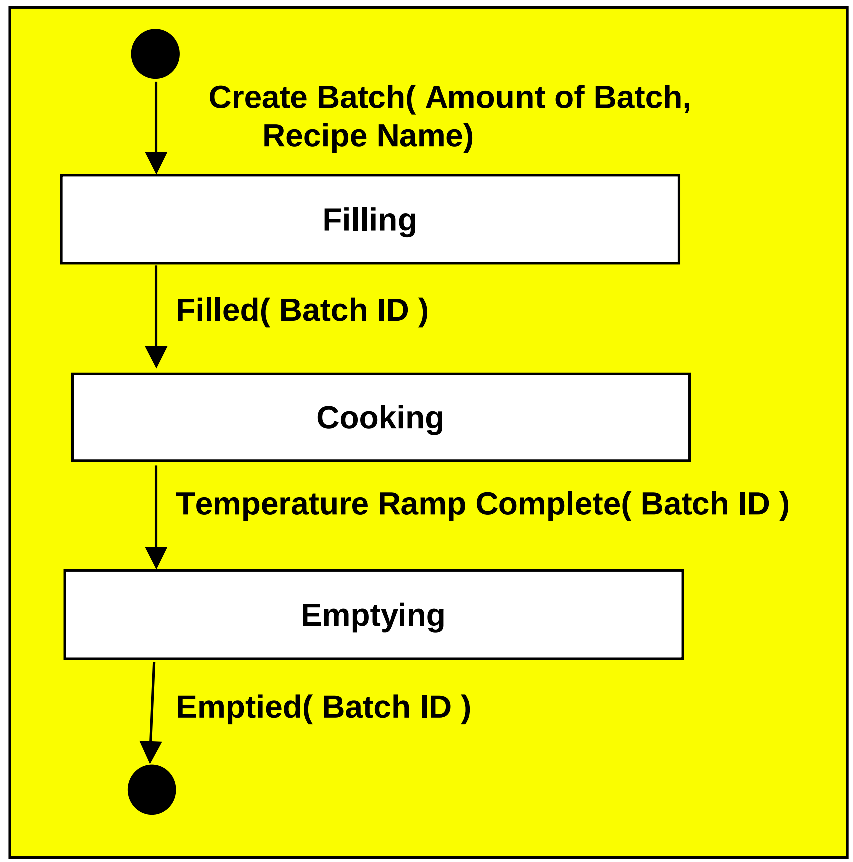
Instance Sets

To select many instances:

.select many *stateS* related to instances of *Class*-> StateChart ->State where (isFinal==False)

StateS =

Filling
Cooking
Emptying



Iteration

To iterate over instances…

```
.select many stateS related to instances of
Class->StateChart -> State where (isFinal == False)
.for each state in stateS
        ${state.Name} ,
.endfor
```

```
Filling,
Cooking,
Emptying,
```

Putting It Together

We may combine these techniques….

.select many *stateS* related to instances of
**   *class*->StateChart-> 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 ,
       Filling ,
       Cooking ,
       NUM_STATES = Emptying
    };
```

Application Semantics

An archetype language gives access to

- ☞ the semantics of the application
- ☞ as stored in the repository.

We may use the archetype language to generate **code**.

A
Direct
Translation

Application Classes

Each application class becomes an implementation class.

```
.select many classES from instances of class
.for each class in classES
class ${class.Name} : public ActiveInstance {
  .invoke addPDMDecl( inst_ref class)
  ...
};
.endfor
```

Application Attributes

Each attribute becomes a private data member:

```
.function addPDMDecl( inst_ref class )
private:
  .select many attrS related to class->Attribute
  .for each attr in attrS
 ${attr.Type} {attr.Name} ;
  .endfor
.end function
```

State Chart Declaration

To declare a state chart: **(i.e. all the actions in the state chart)**

```
.function addProtectedActions( inst_ref class )
.select one statechart related by class->StateChart
protected:
// state action member functions
    .select many stateS related by statechart->State
    .for each state in stateS
        .invoke addActionFunctionDecl( inst_ref state )
    .endfor
 .end function
```

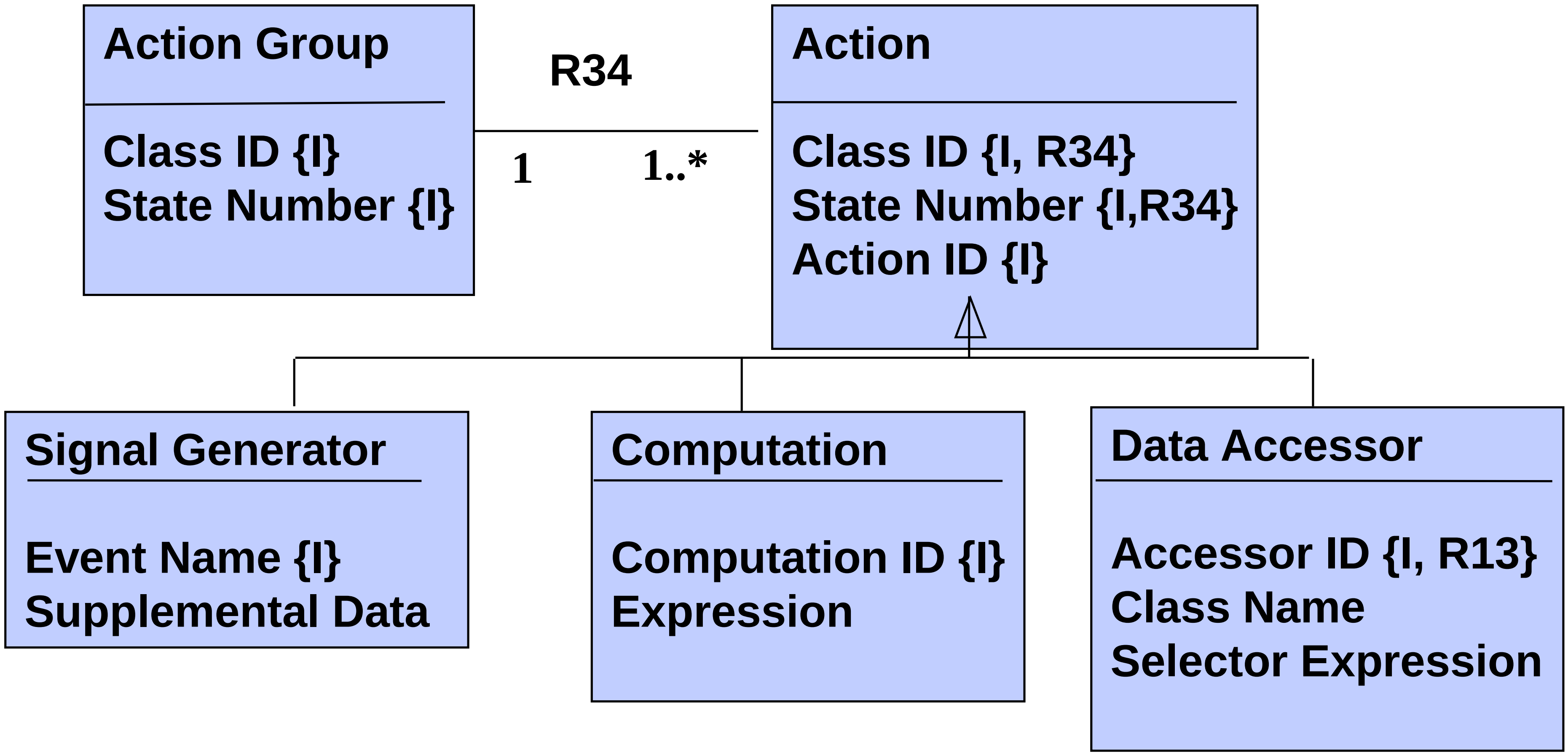
State Action Declaration

To generate the state action declaration:

```
.function addActionFunctionDecl( inst_ref state )
// State action: ${state.Name}
static void sAsyncAction${state.Name}(
                stda_eventMsg_c *eventPtr, int nextState);
           void ${state.Name}(stda_eventMsg_c *p_evt );
void asyncAction${state.Name }( );
.endfor
```

State Action Definition

To *define* the state action function….



…traverse the repository in the same manner.

An Indirect Architecture

Where Have All The Cycles Gone?

Consider 500 Temperature Ramp instances all in the Controlling state, controlling temperature 100 times a second.

Using the direct architecture this means:

- 500 * 100 (= 50,000) state transitions per second, or
- 20 microseconds per transition including the actions

The direct architecture won’t work on a single (cheap) processor.

Where Have All The Cycles Gone?

We could buy several cheaper processors, each controlling different temperature ramps……

Or we could:

- build a separate task that handles all periodic activity at a single rate
- execute the periodic task 100 times/second using a timer
- leave the remaining “event-driven” logic in another task

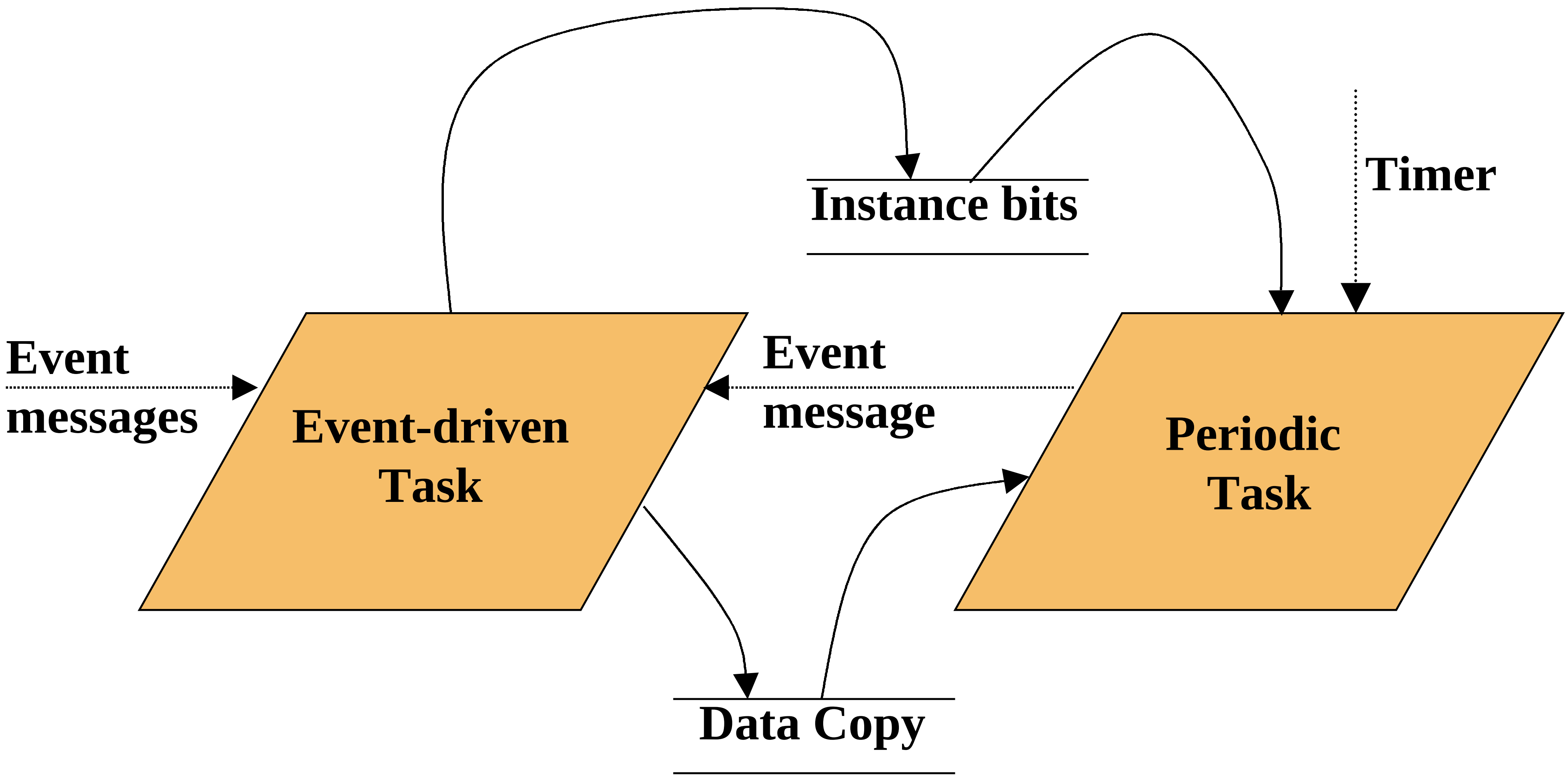
Description of Architecture

This introduces new issues:

- how to indicate transition into the periodic state (use one bit per instance), and
- how to indicate transition out of the periodic state

- the periodic task has to be able to execute when "it's time," so it needs higher priority----
- which raises issues of inconsistent data, so we should duplicate data needed for the control loop----
- and copy it over by the periodic task when required

Description of Architecture



Application Mapping

Recipe

Recipe Name {I}
Cooking Time
Cooking Temperature
Heating Rate

Batch

Batch ID {I}
Amount of Batch
Recipe Name {R2}
Status

Temperature Ramp

Ramp ID {I}
Batch ID {R4}
Start Temperature
Start Time
End Temperature
End Time
Status

Event Driven Task

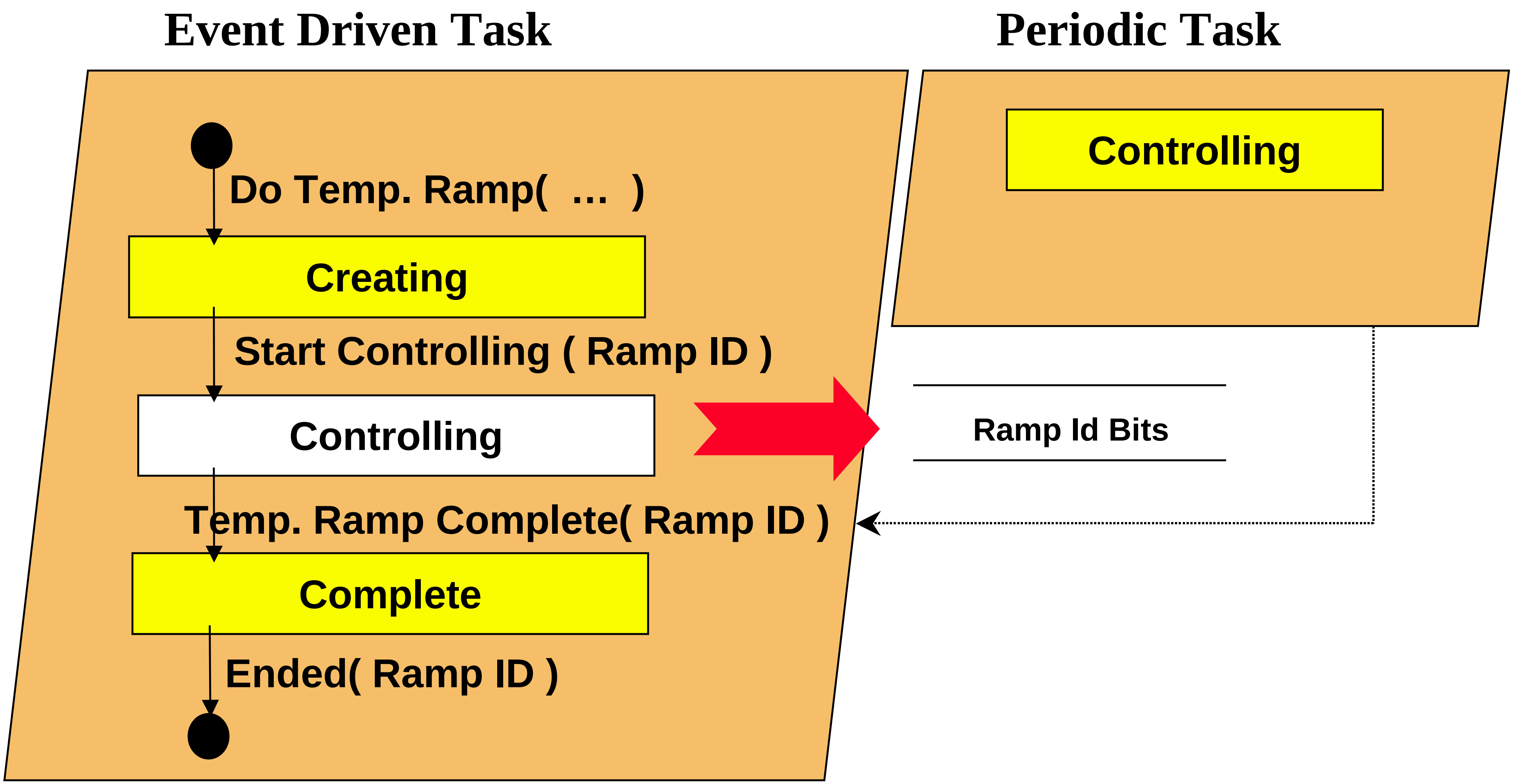
Periodic Task

Temperature Ramp
Ramp ID {I}
Start Temperature
Start Time
End Temperature
End Time

Start Temperature
Start Time
End Temperature
End Time

Ramp Id Bits

Application Mapping



Extended Properties

To make certain distinctions, we need to tag elements of the meta-model.

```
.function addPeriodicStateAction
...
RampIDbits[insNumber].activateActions();
```

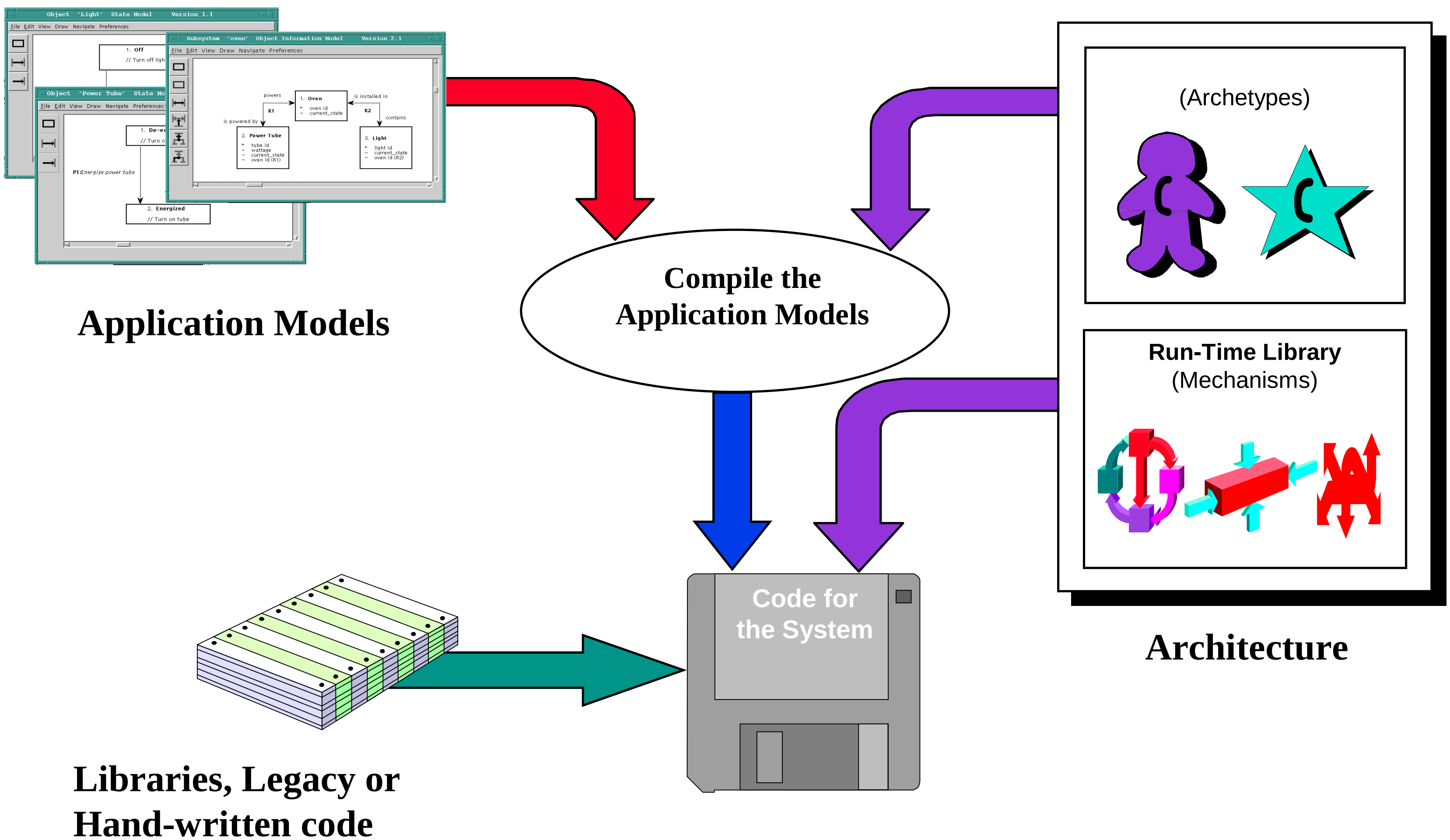
State

Class ID {I, R14}
State Number {I}
Name
isFinal
isPeriodic

System Construction

Production Code

Compile the source code and include initialization data files (if any) to generate the deliverable production code.

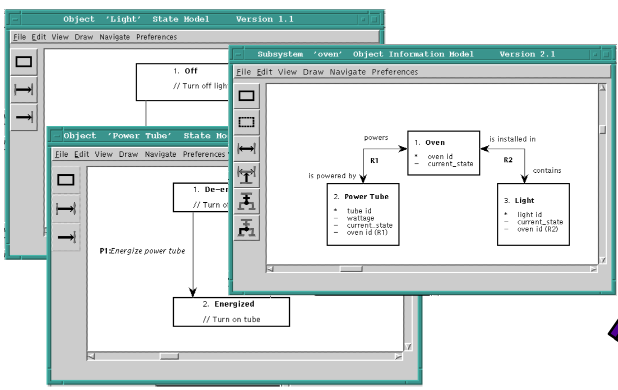


Model-Based Maintenance

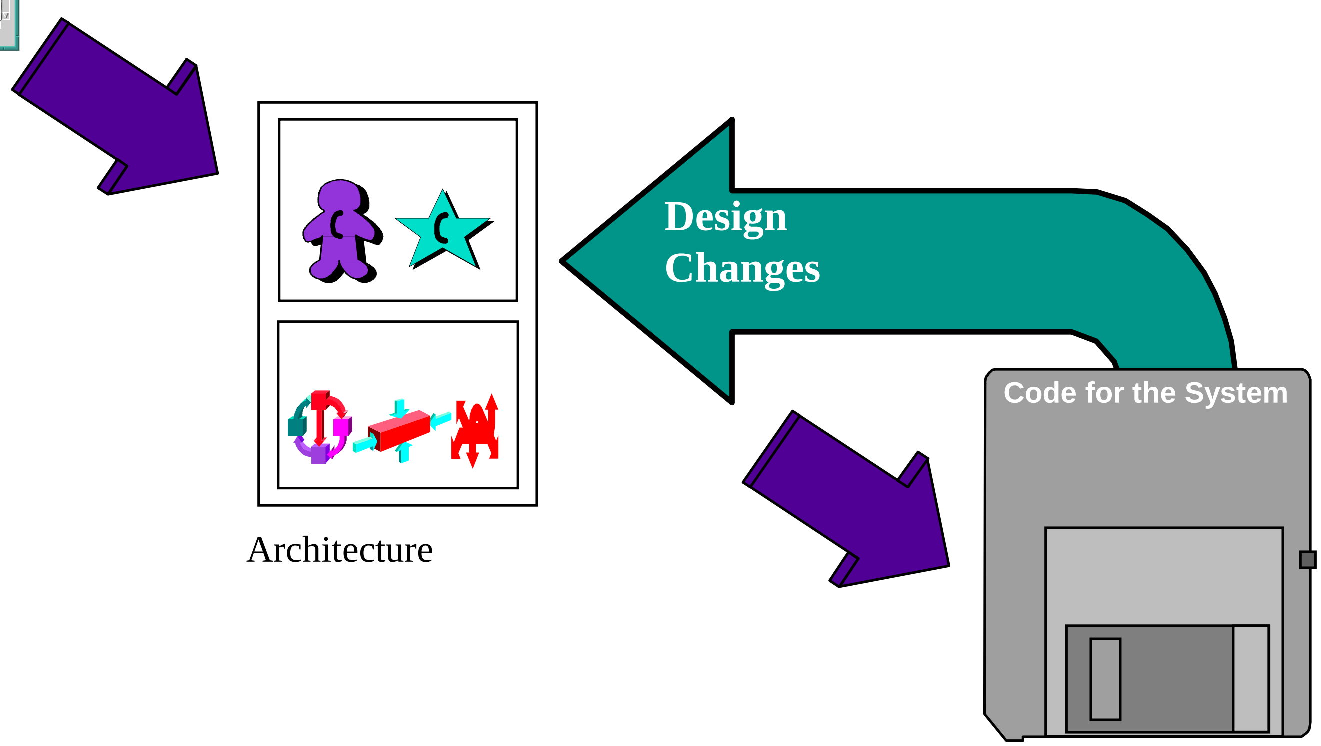
To address performance-based issues:

- modify the architecture models, and
- and regenerate the system.

- Do not modify the generated code directly.



Application Models

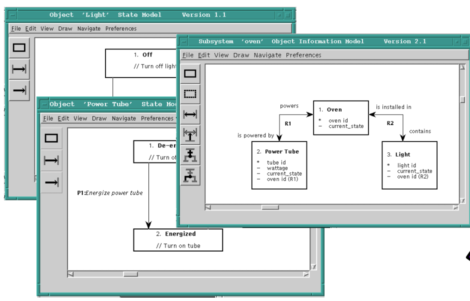


Architecture

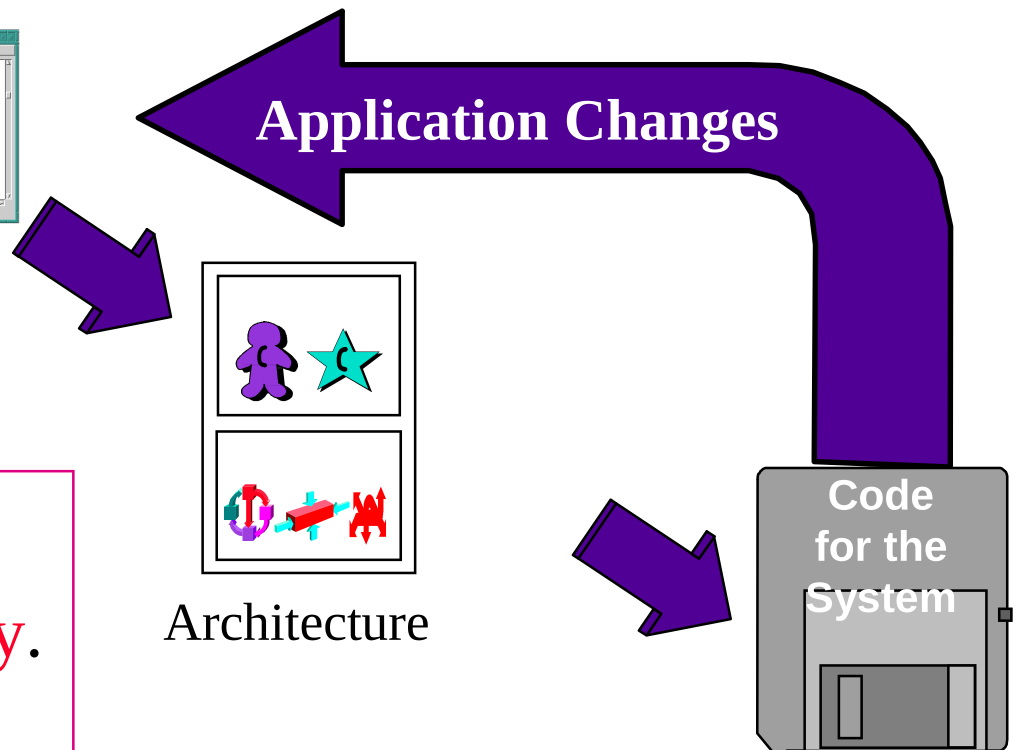
Model-Based Maintenance

To address application behavior issues,

- modify the relevant application model, and
- regenerate the system.



Application Models

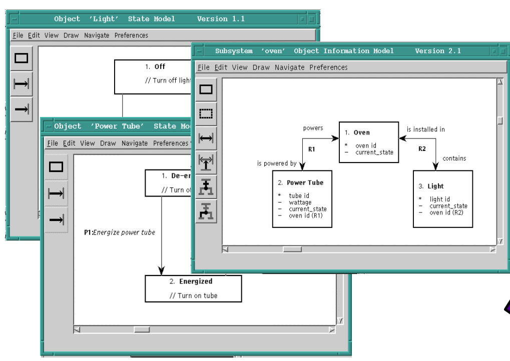


- Do not modify the generated code directly.

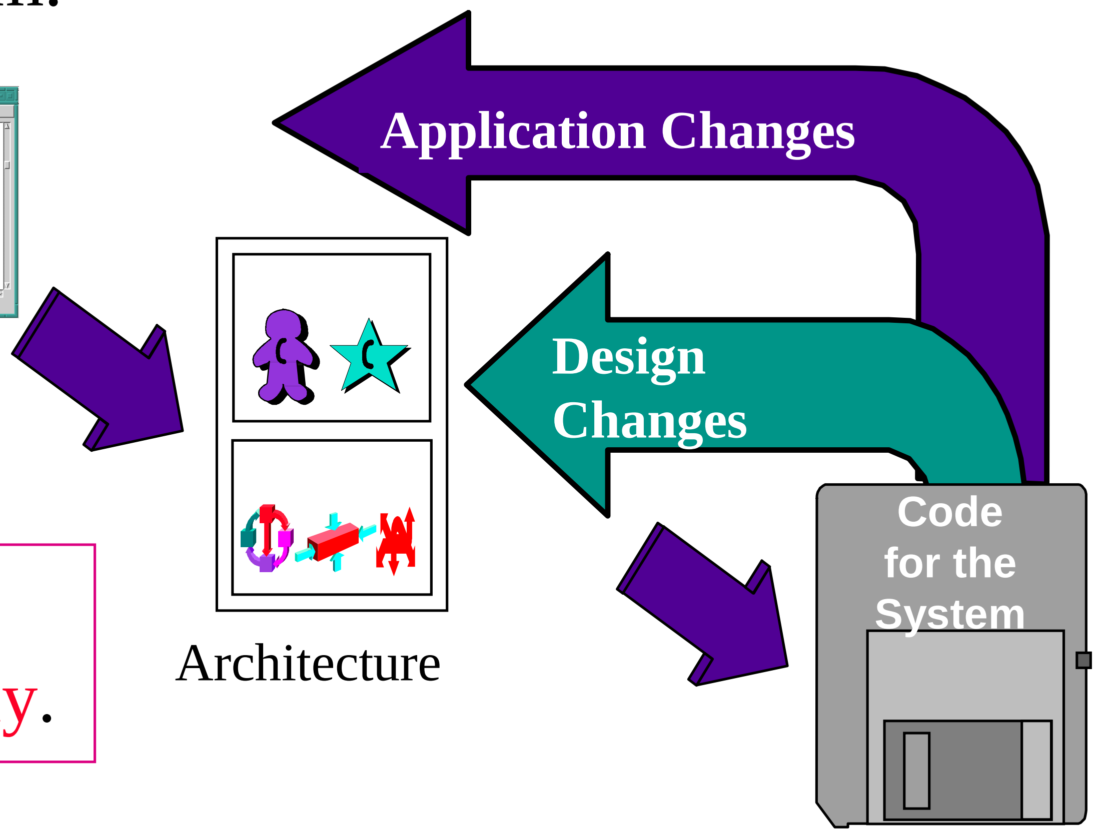
Model-Based Maintenance

For subsequent product enhancements,

- modify or replace the domain in question, and
- regenerate the system.



Application Models



Architecture

> ✦ Do not modify the generated code directly.

Model Compiler

An architecture is an *Executable UML model compiler*.

It translates a system specified in X-UML into the target programming language incorporating decisions made by the architect about:

- data,
- control,
- structures, and
- time.

Architectures, like programming language compilers, can be bought.

System Design

Executable UML

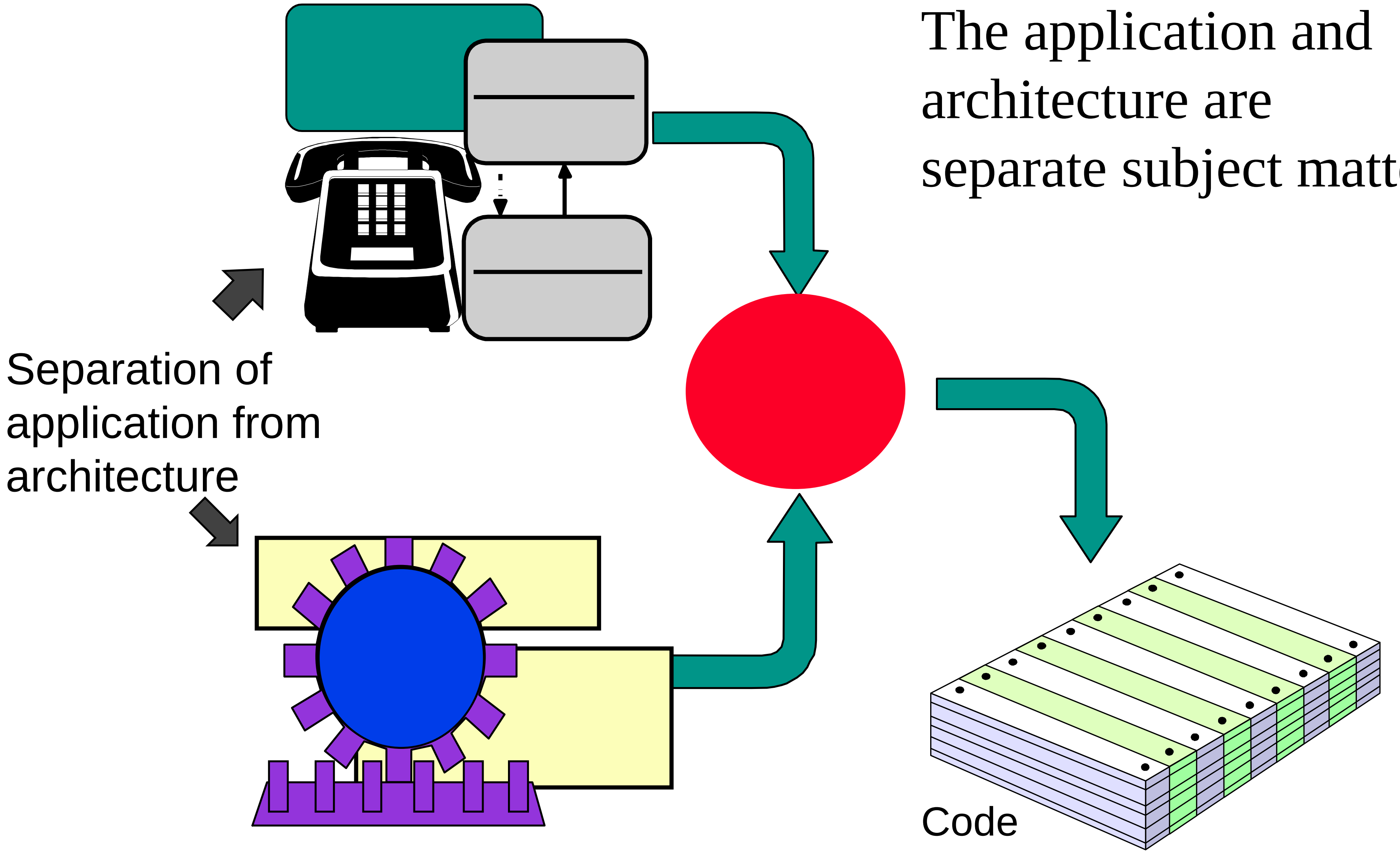
Executable UML relies on:

- separating systems into subject matters (domains)
- specifying each domain with an executable model
- translating the models

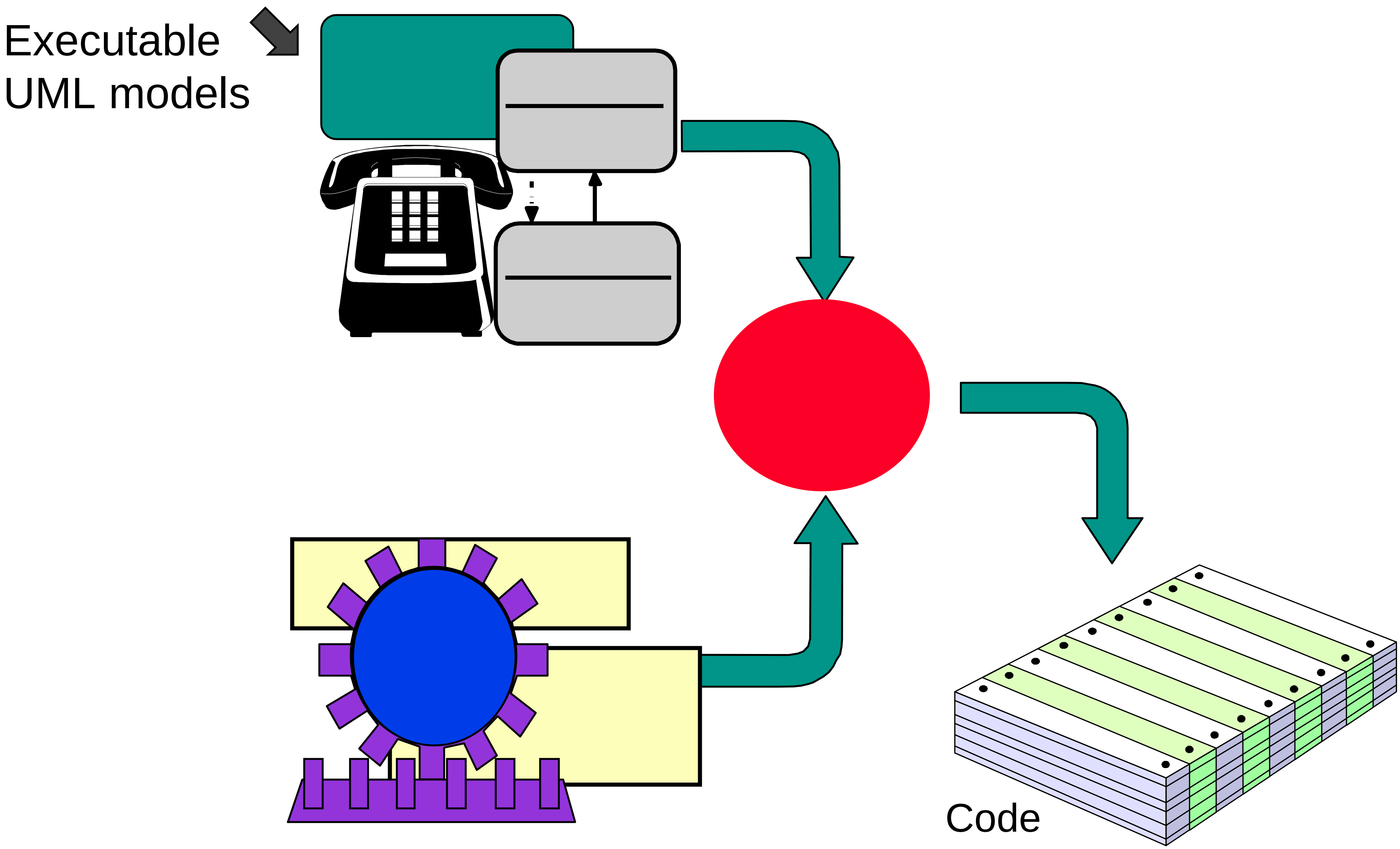
The method may employ UML or any other notation that has a defined semantics.

Subject Matter Separation

The application and architecture are separate subject matters.

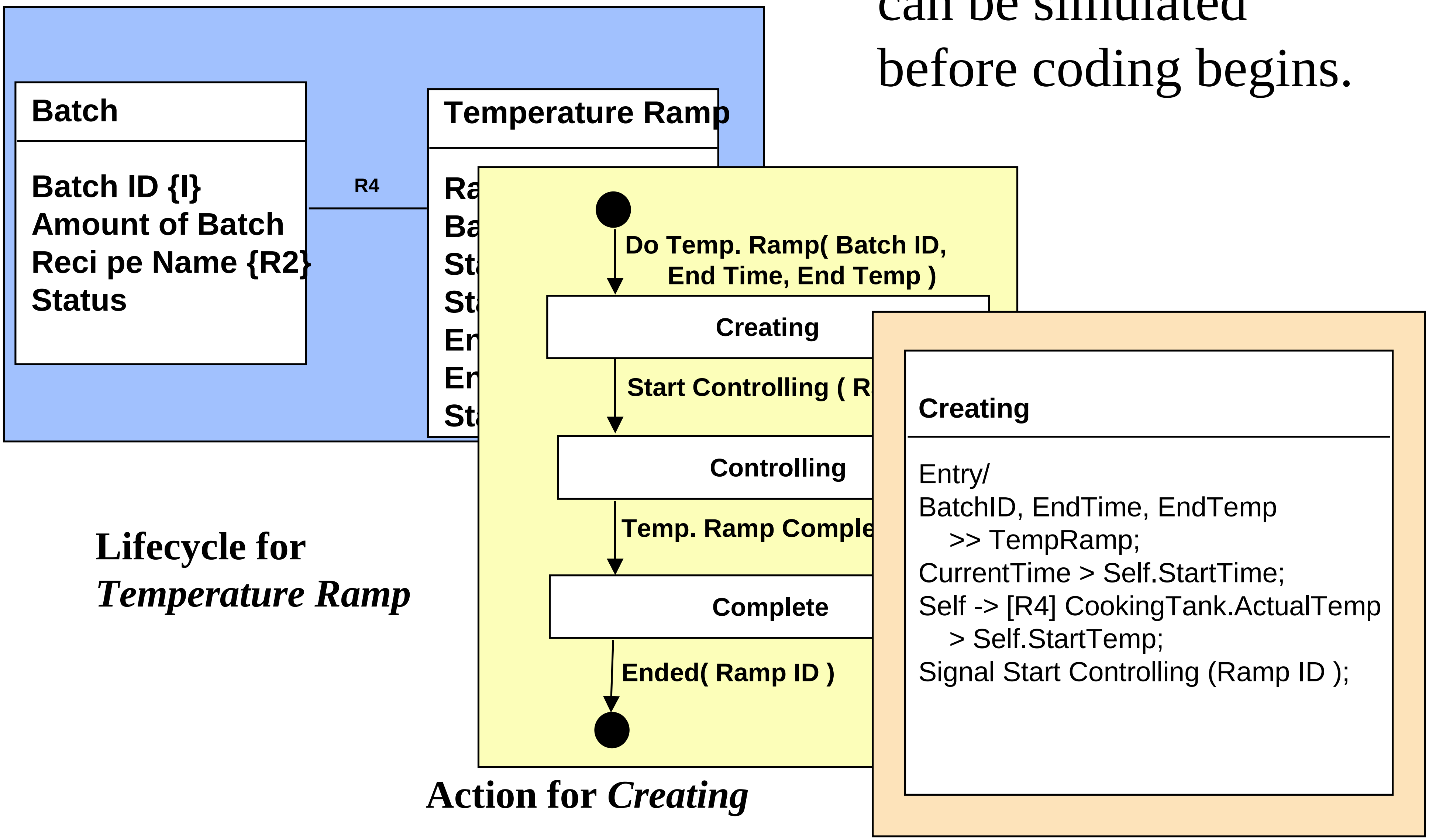


Executable UML Models



Executable UML Models

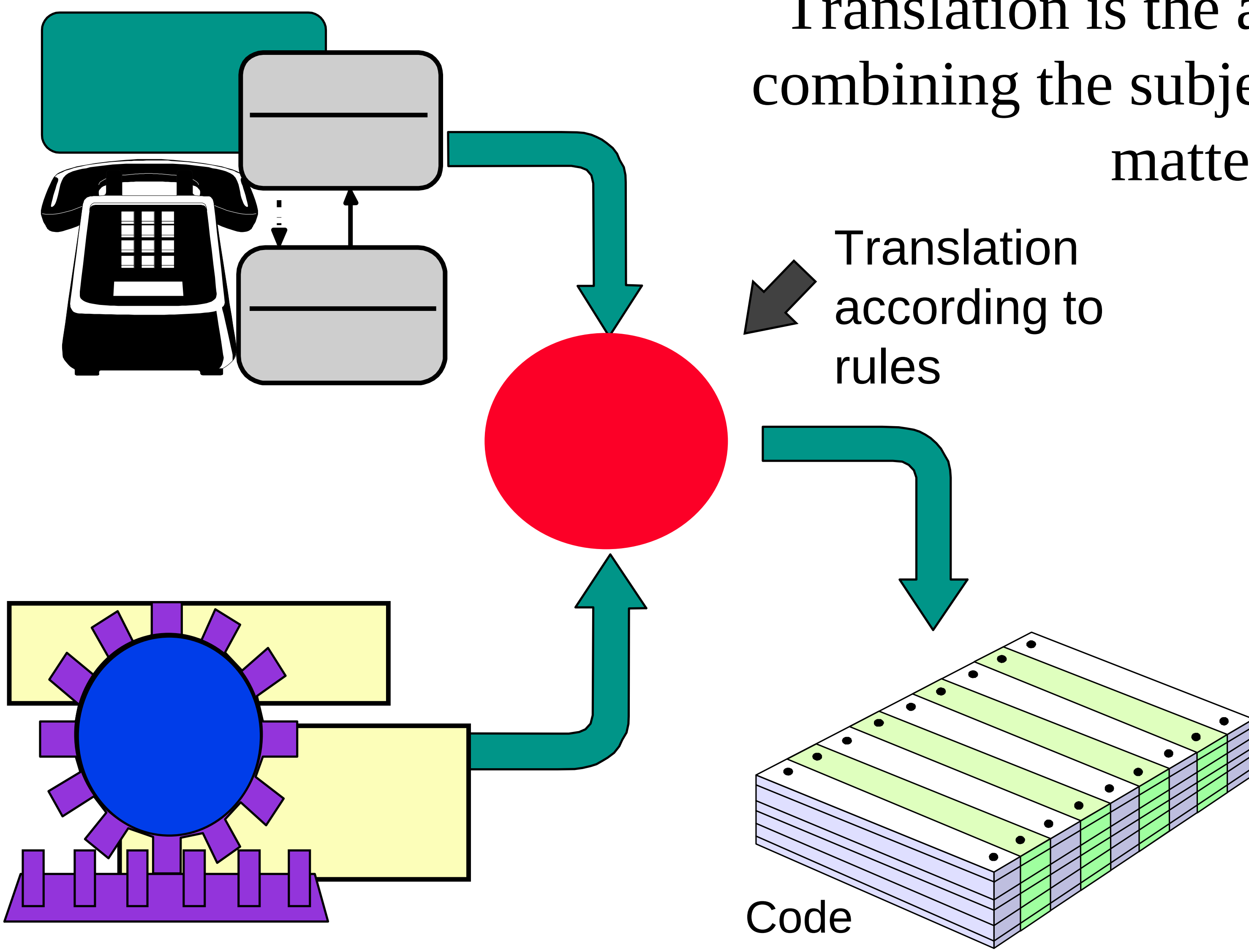
Executable models can be simulated before coding begins.



Lifecycle for *Temperature Ramp*

Action for *Creating*

Translation



Translation is the act of combining the subject matters.

Translation

Translating the application domain models generates:

- highly systematic
- uniform
- reproducible
- *understandable application code*

and minimizes:

- *coding and code inspection effort*
- *coding errors*
- component integration issues

Executable UML

Executable UML meets the challenges of software development by:

- localizing critical software design issues to the software architecture domain
- ensuring that the design decisions are incorporated uniformly and systematically
- providing a framework to modify system performance without affecting system behavior

Brought to you by…

PROJECT TECHNOLOGY INC.

Translating Ideas Into Products.

Makers of BridgePoint ® and DesignPoint ® Modeling Tools

Stephen J. Mellor
Project Technology, Inc.
http://www.projtech.com