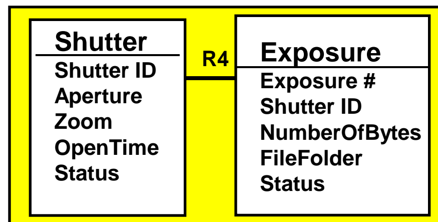
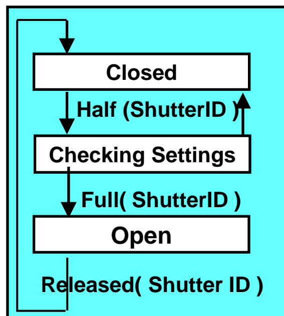
The Gap between Specification and Synthesis

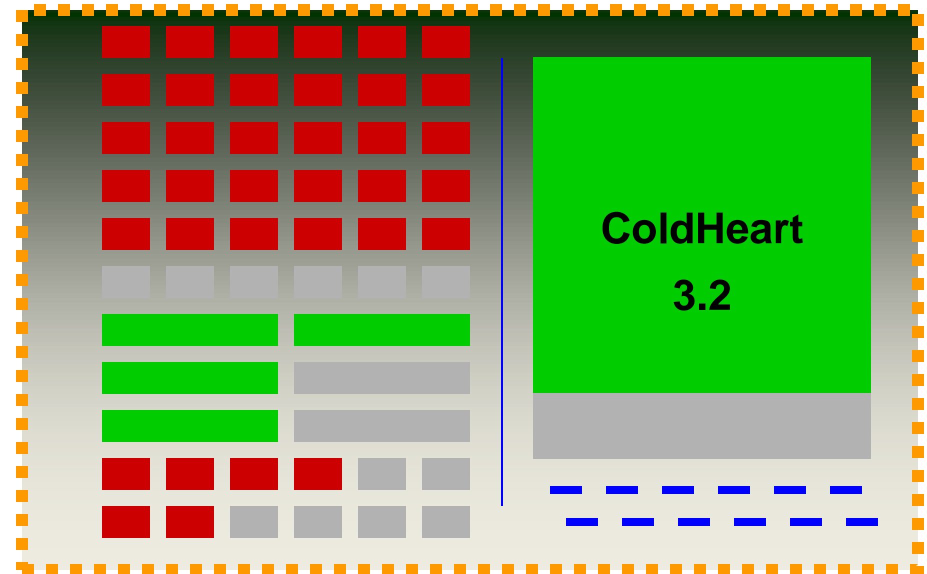
Stephen J Mellor

Too Much Flexibility

The gap between specification and synthesis is large.

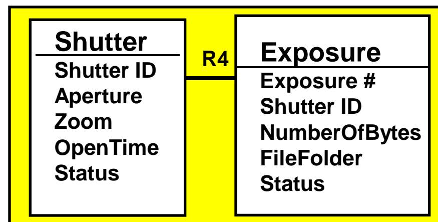


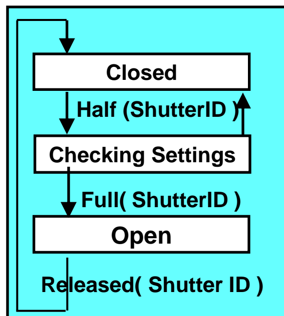


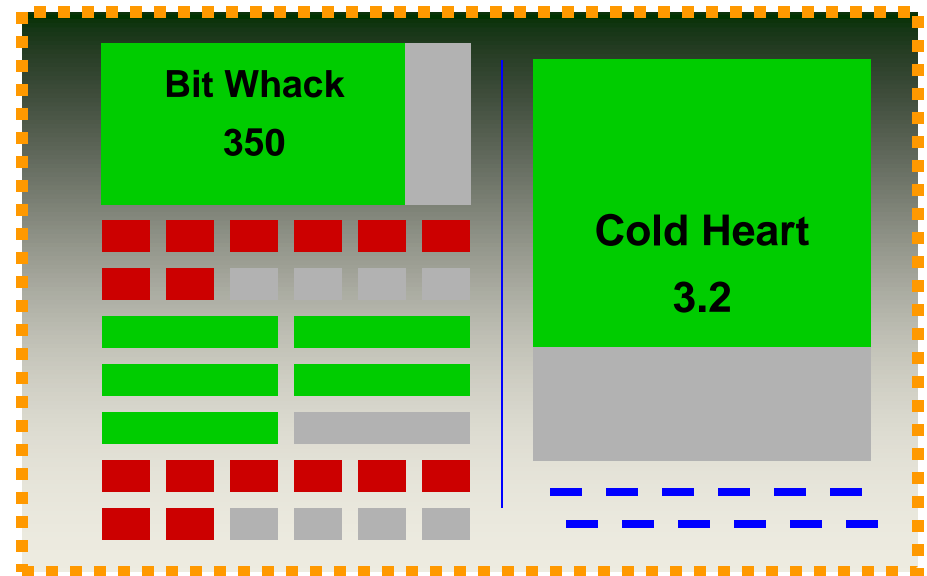


Lots of Options to Explore

Feed the software guys enough caffeine and you'll need at least two cores…



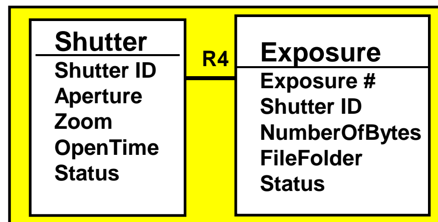


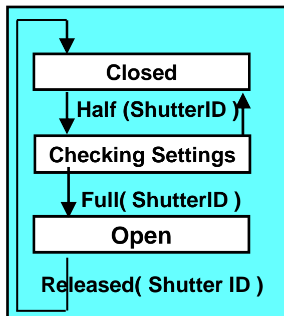


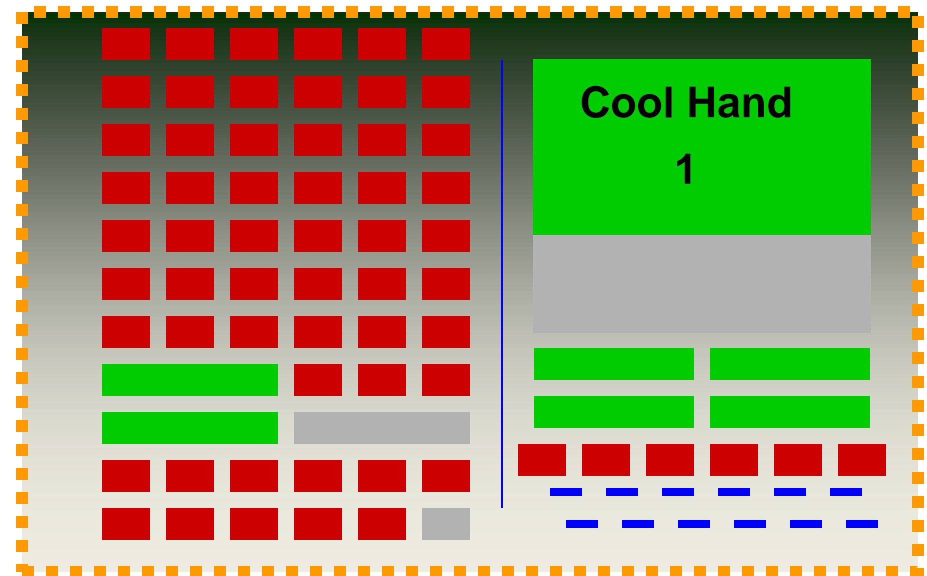
Explore Design Space

Let them go snowboarding, and the logic designers have to do all the work…

…but at least you can use a cheap core.







Process

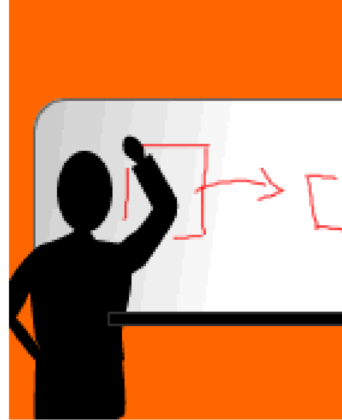
What's required before we can start implementation?

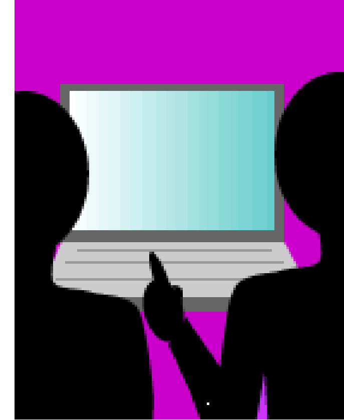
•Hardware •Algorithm •Micro-architecture •Implementation: – C, C++, HDL, RTL	•Software •Algorithm •Software architecture •Implementation: – C, C++, Java, Plex

- Huge investment required before implementation begins
- Testing is delayed until something is running
- Design inertia resists architectural changes

An Idea

Create


Formalize

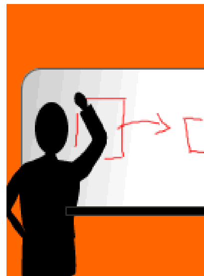

Verify

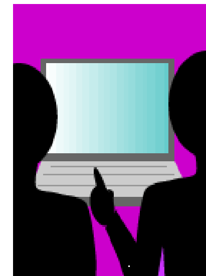
- Since both teams are doing similar things:
  - Creating abstractions,
  - Formalizing them, and then
  - Verifying behavior
- …it seems reasonable to use the same approach
  for specifying and verifying behavior.

Towards a Solution

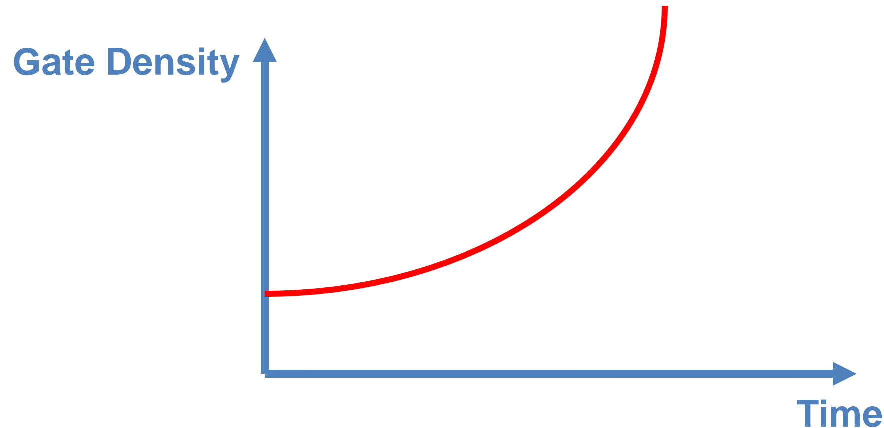
Jointly:

- Build a single Application Model
- Build an Executable Application Model
- Don‘t Model Implementation Structure
- Map the Application Model to Implementation





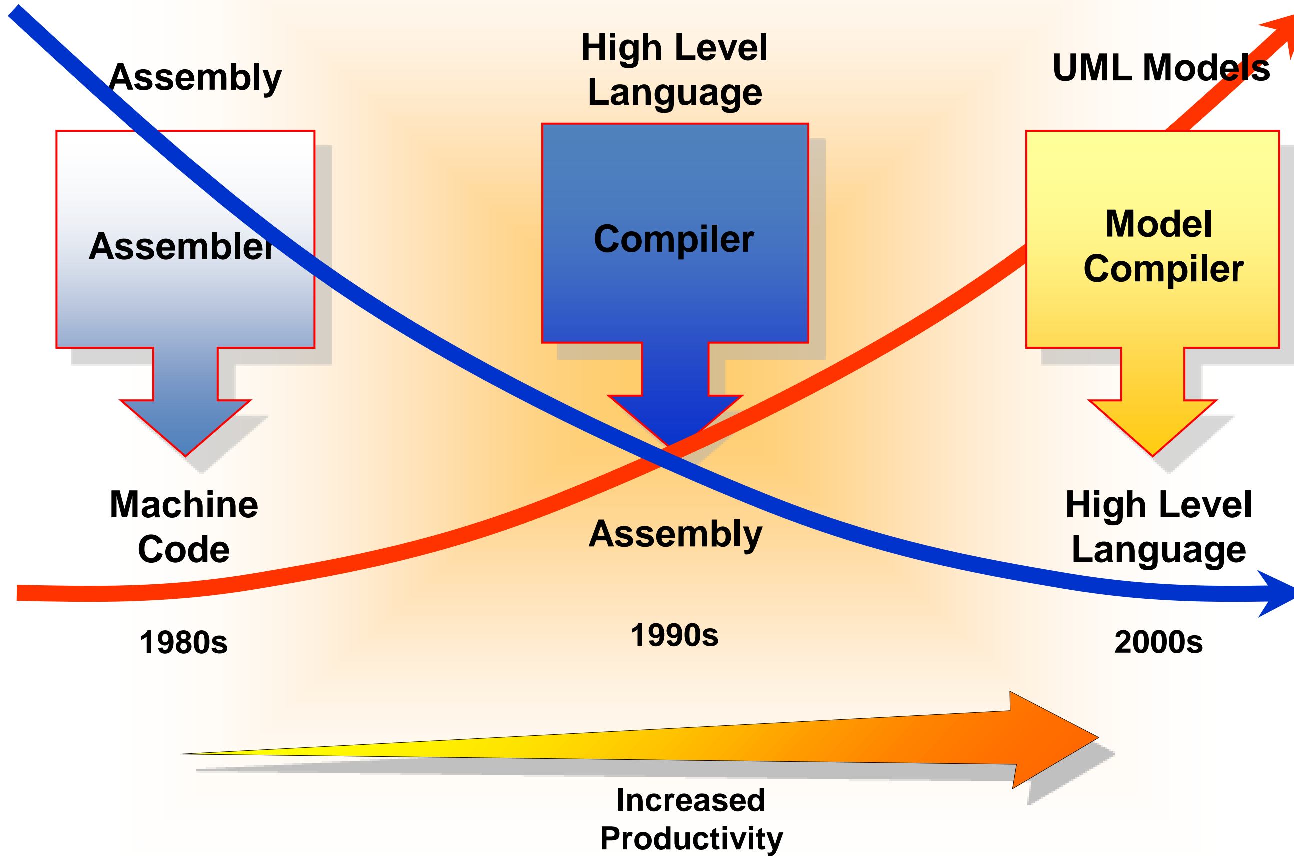
Abstraction in Hardware



- Gate density is increasing exponentially
- Complexity is increasing along with gate density
- We need a way to manage this complexity
- We need to move to a higher level of abstraction

The answer is C!

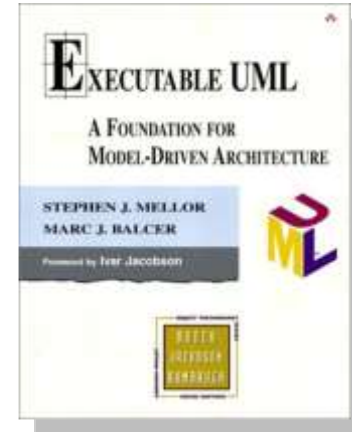
Abstraction in Software



UML – A Big Language

UML is the industry standard

- It has notations for everything you could possibly do in software
- Can we add notations for everything we can possibly do in hardware?

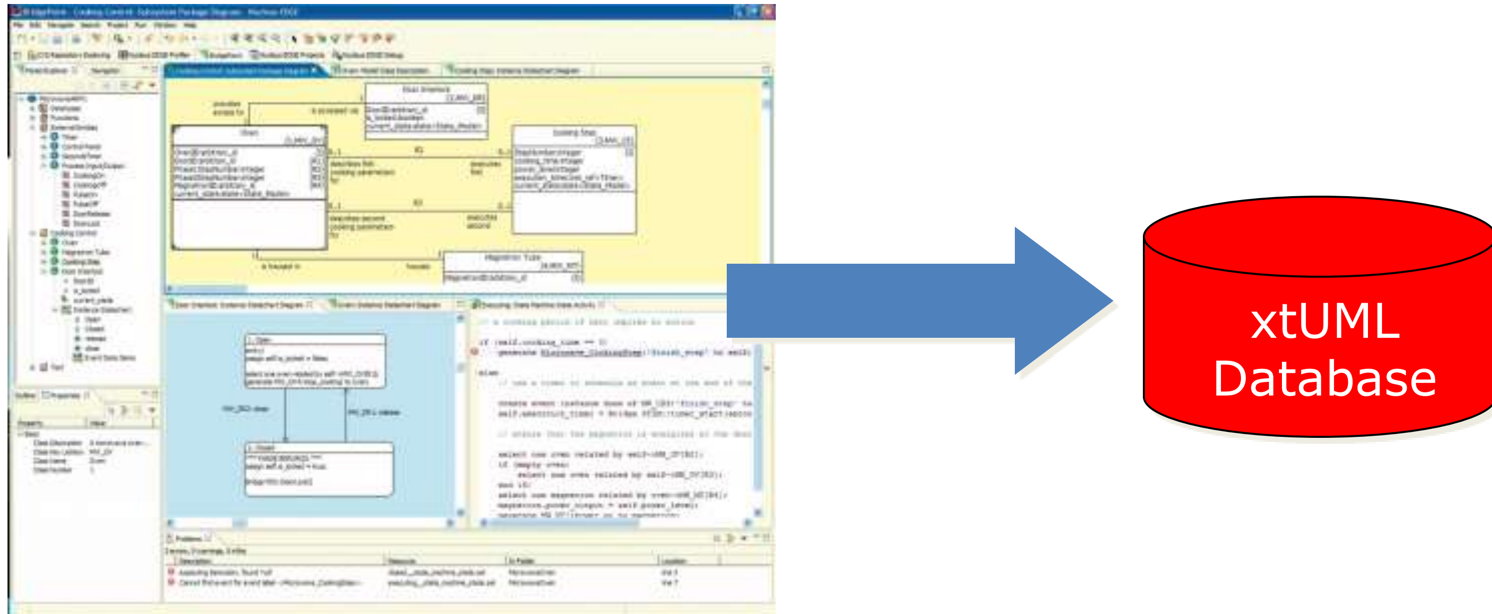


Executable UML is a defined:

- Streamlined
- Tractable
- Subset of UML

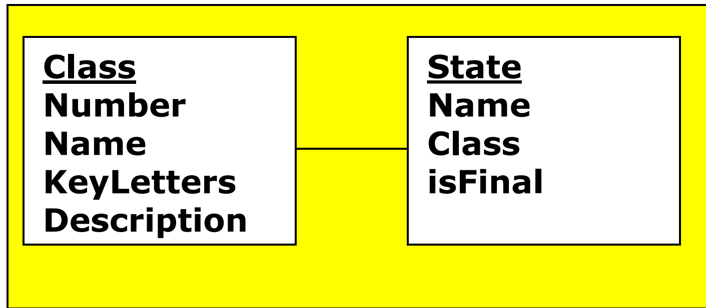
Achieved by having defined execution rules

Building Models

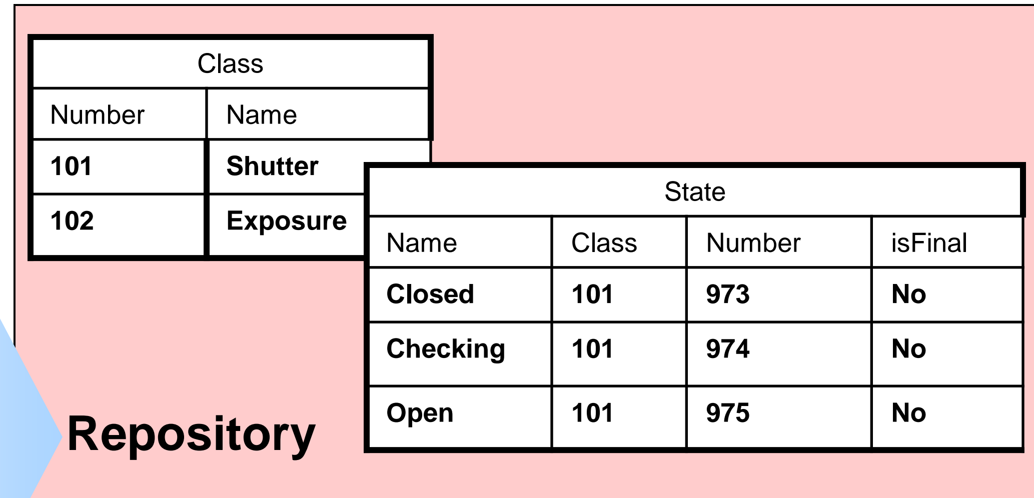


- Models capture the behavior of the entire system
- Including an Object Action Language (OAL)

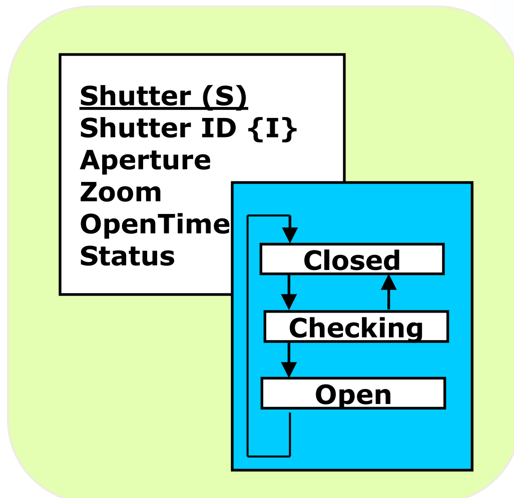
Model Capture



Metamodel



Repository



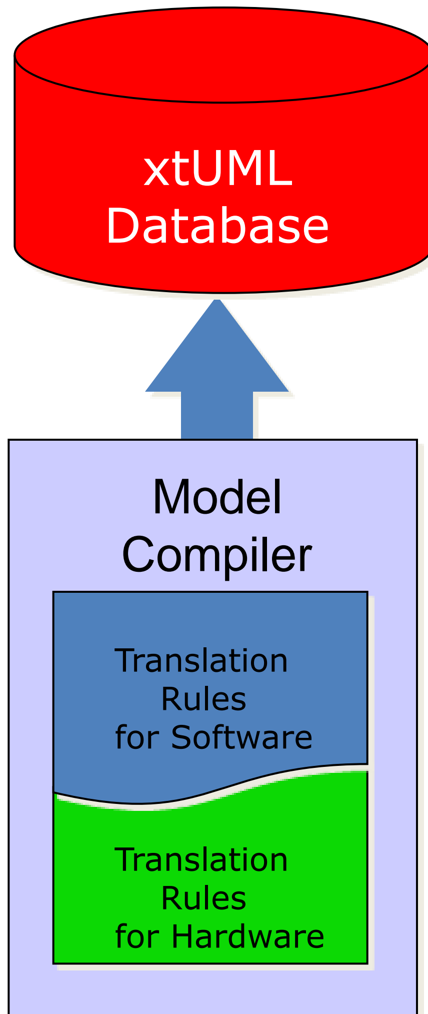
Application

Intelligent model capture verifies models:

- **Syntactically**
- **Semantically**

For an executable model *with actions*

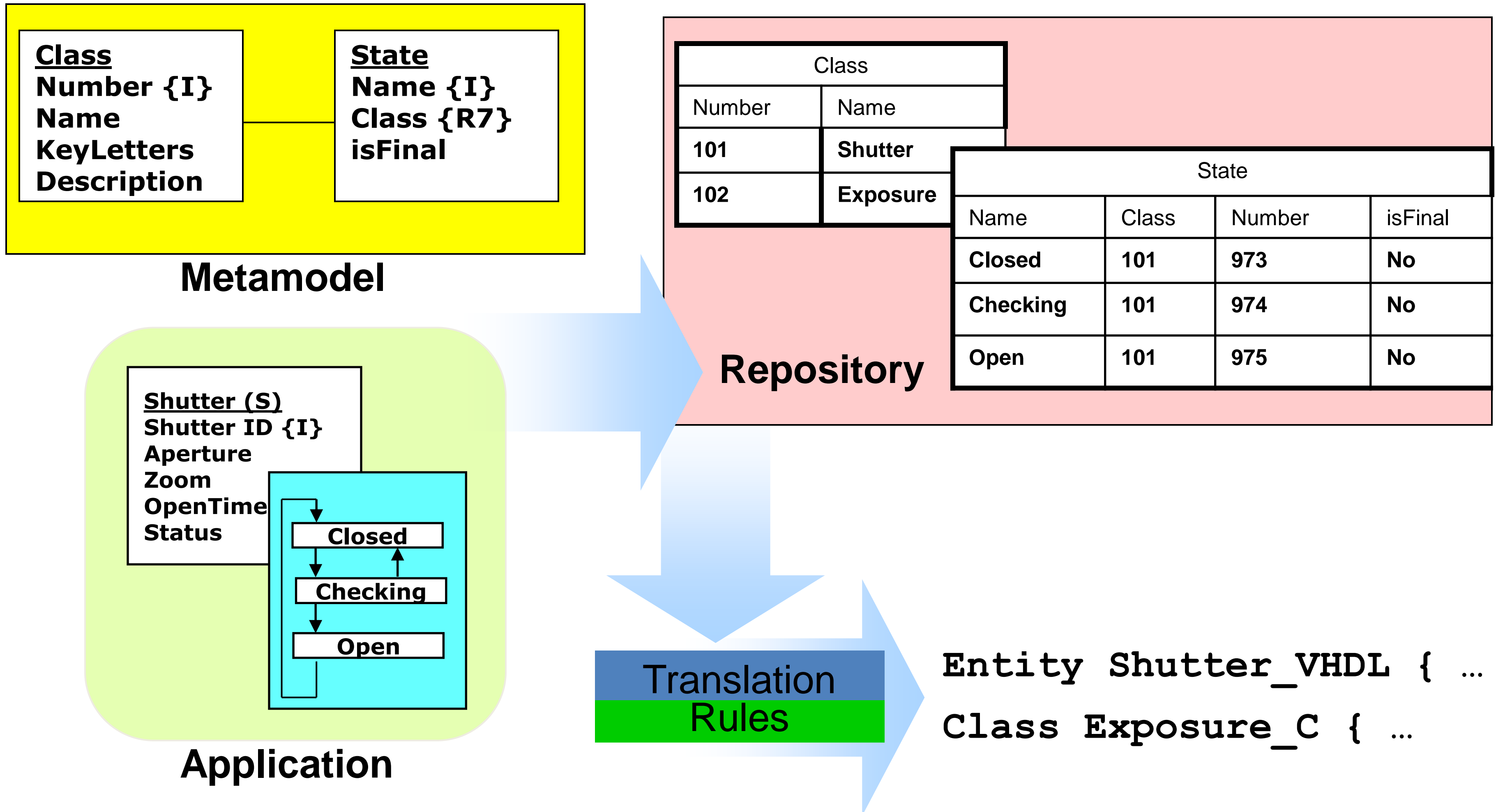
Translation Rules



Translation rules:

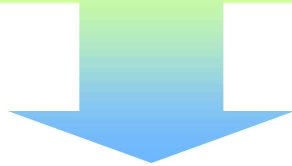
- **Read the database to produce *text***
- **Text can be a language for *software* or *hardware***
- **Build a complete system from models *consistently***

Translation Rules Generate Text



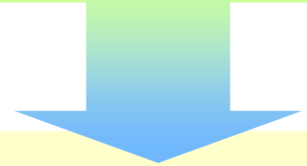
Two Translation Rule Sets

```
.select many stateS related to instances of
      class->State where (isFinal == False)
TYPE t_${class.Name}State IS (
.for each state in stateS
 ${class.Name}_${state.Name}\
      .if ( not last stateS )
         ${state.Name } ,
      .else
         ${state.Name }
      .end if
.end for
) ;
```



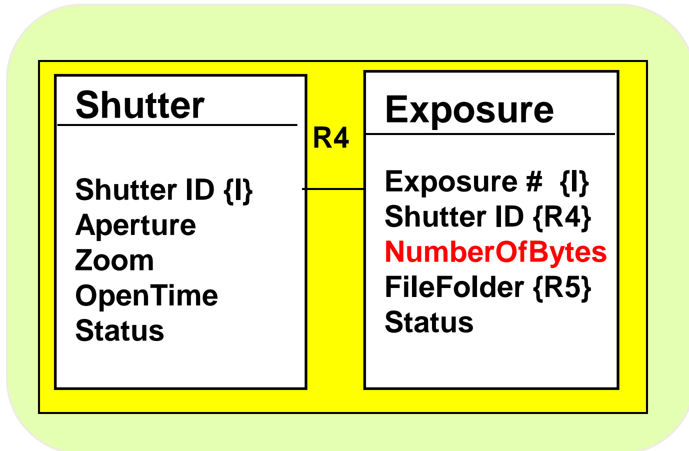
```
TYPE t_CameraState IS (
        CAMERA_OPEN ,
        CAMERA_CHECKING ,
        CAMERA_CLOSED
    ) ;
```

```
.select many stateS related to instances of
      class->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}
      .end if
.end for
};
```

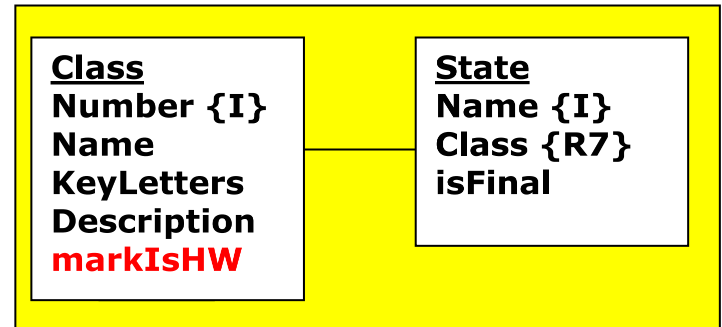


```
public:
  enum states_e
    {  NO_STATE = 0 ,
       CAMERA_OPEN ,
       CAMERA_CHECKING ,
       NUM_STATES = CAMERA_CLOSED
    };
```

Marks



Application



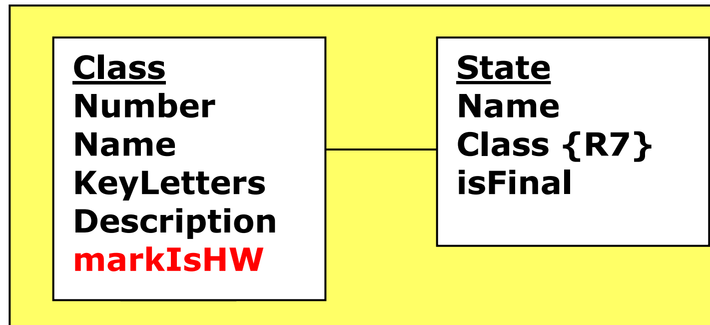
Metamodel

Marks are extended properties of the metamodel that allow different rules to be applied

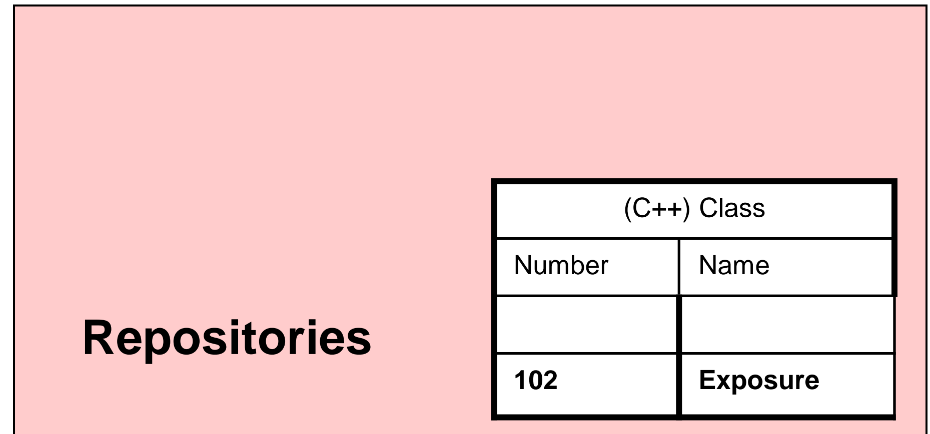
```
.select many classes where markIsHW == TRUE;
// generate VHDL

.select many classes where markIsHW == FALSE;
// generate logic for a C++ class
```

Marks



Metamodel

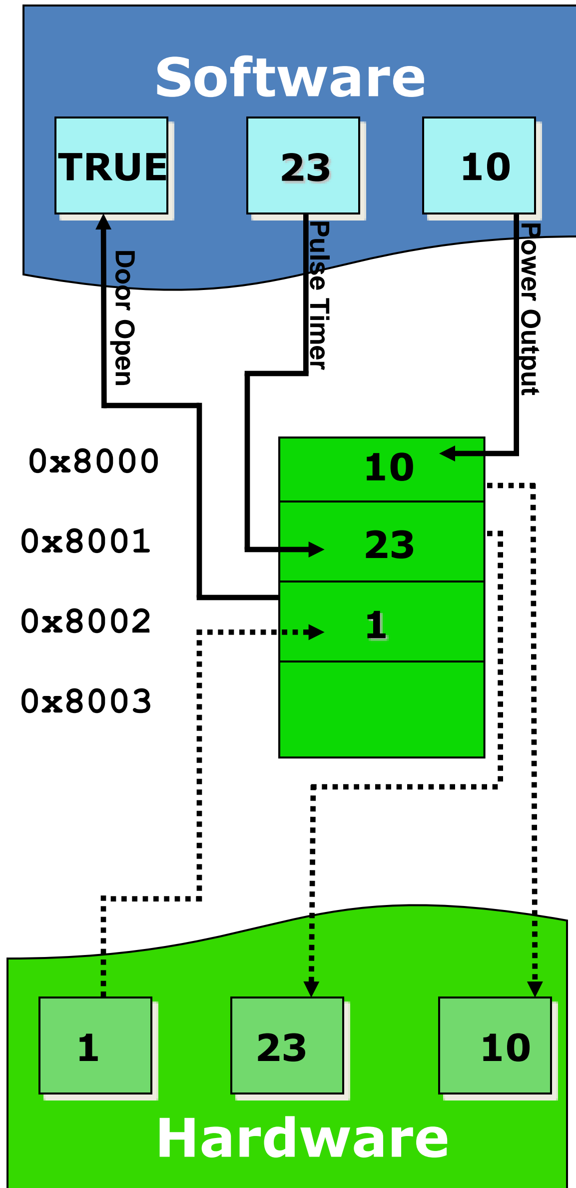


Write rules to create instances in the repositories

```
.select many classes where markIsHW == TRUE;
// populate the VHDL repository

.select many classes where markIsHW == FALSE;
// populate the C++ repository
```

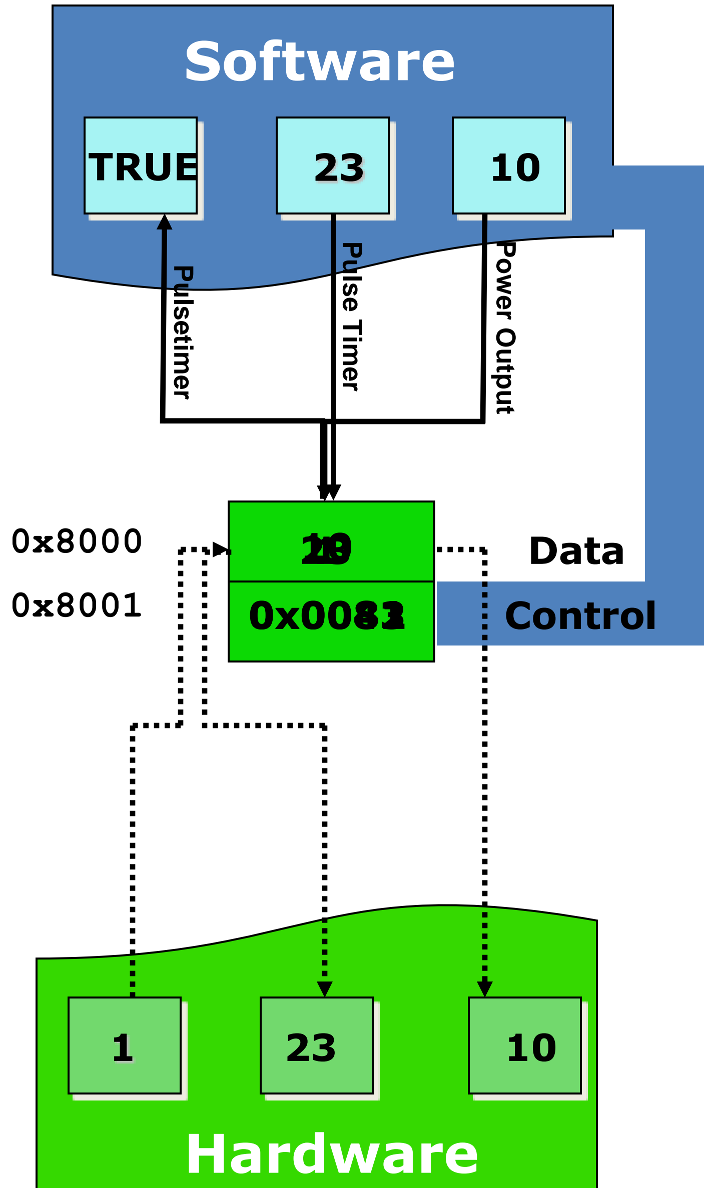
Write clean rules from each repository



Interfaces…

- Tight coupling
- Easy to misunderstand
- Distributed throughout implementation

Memory-mapped I/O



Interfaces…

Changes ripple through both:

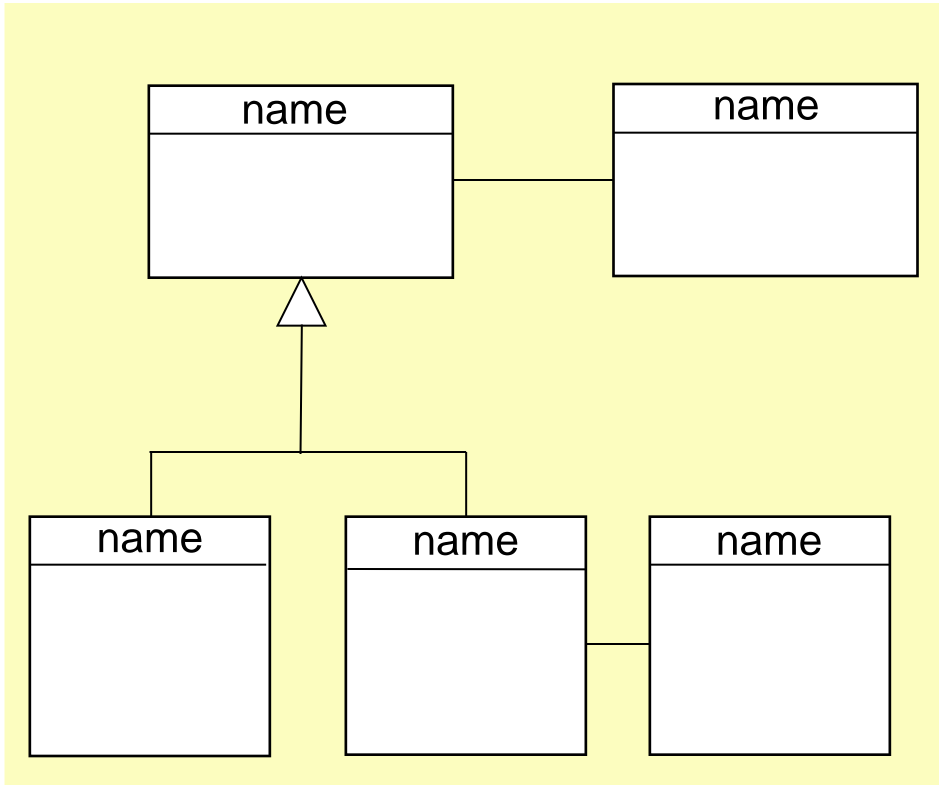
- **Software and**
- **Hardware**

Manual maintenance is

- **Expensive**
- **Error-prone**

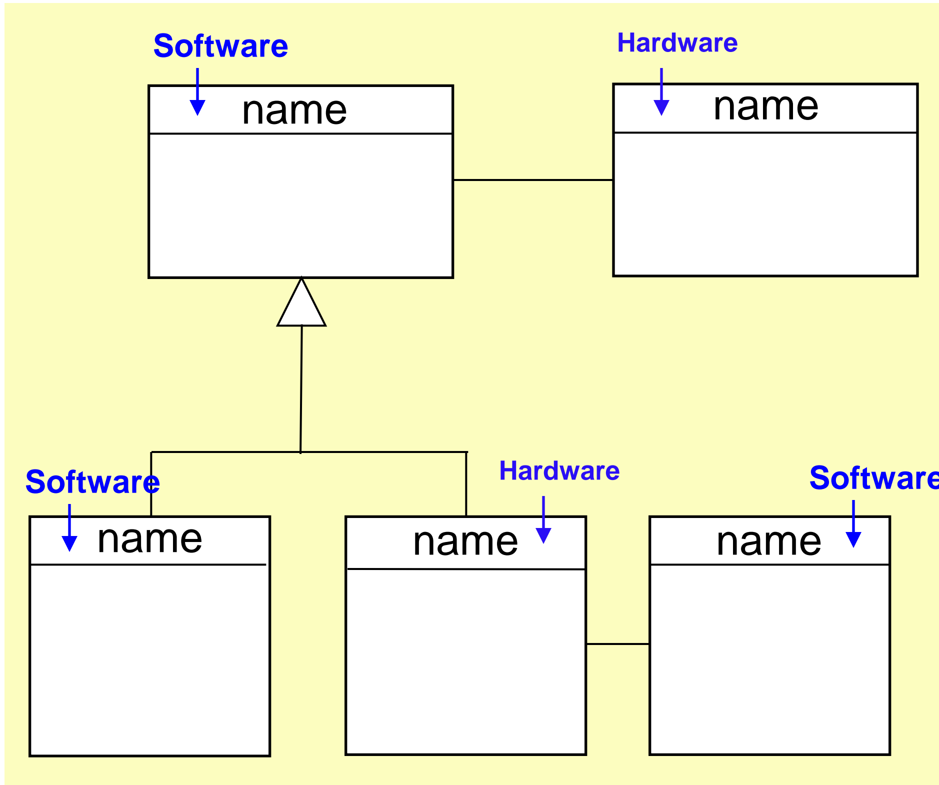
Register I/O

Model it!



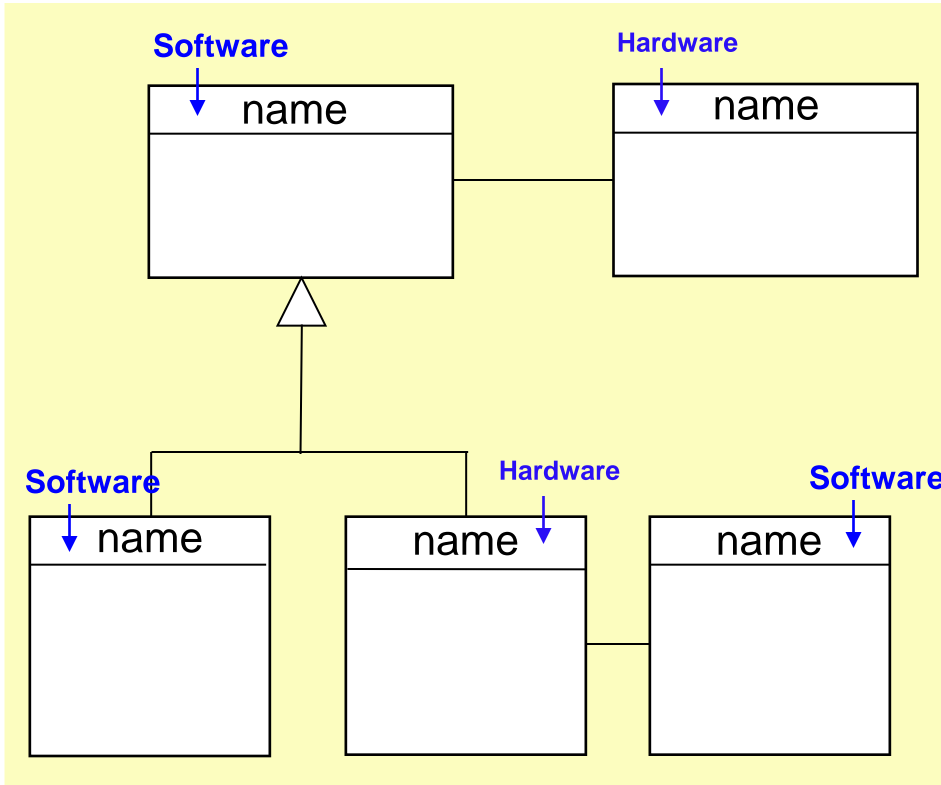
Model the Application

Mark it!

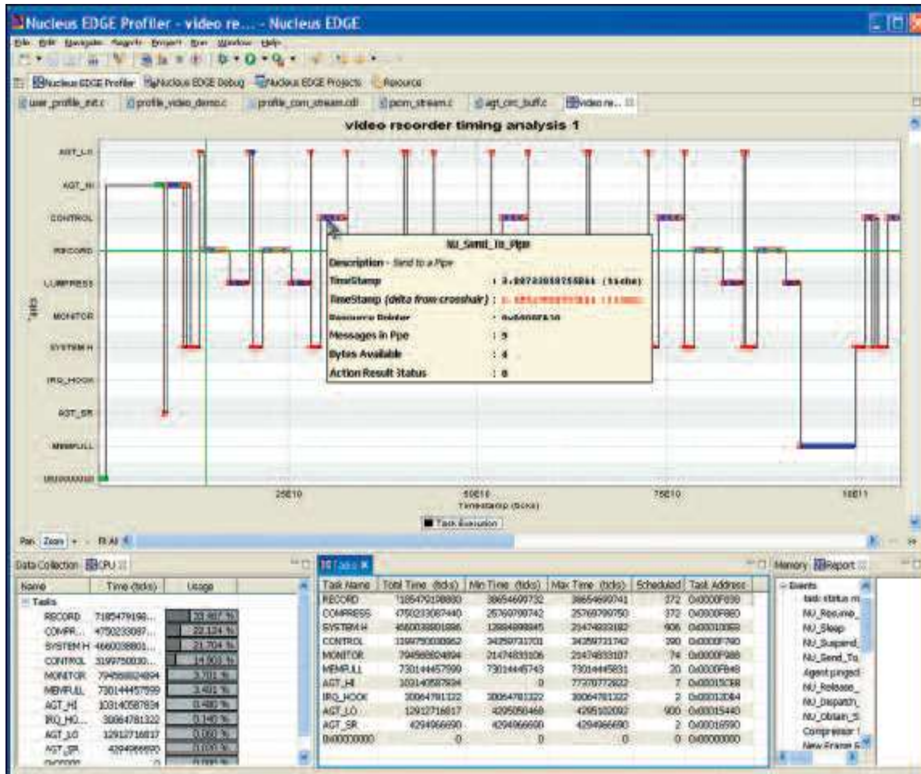


Assign classes to hardware or software

Make it!

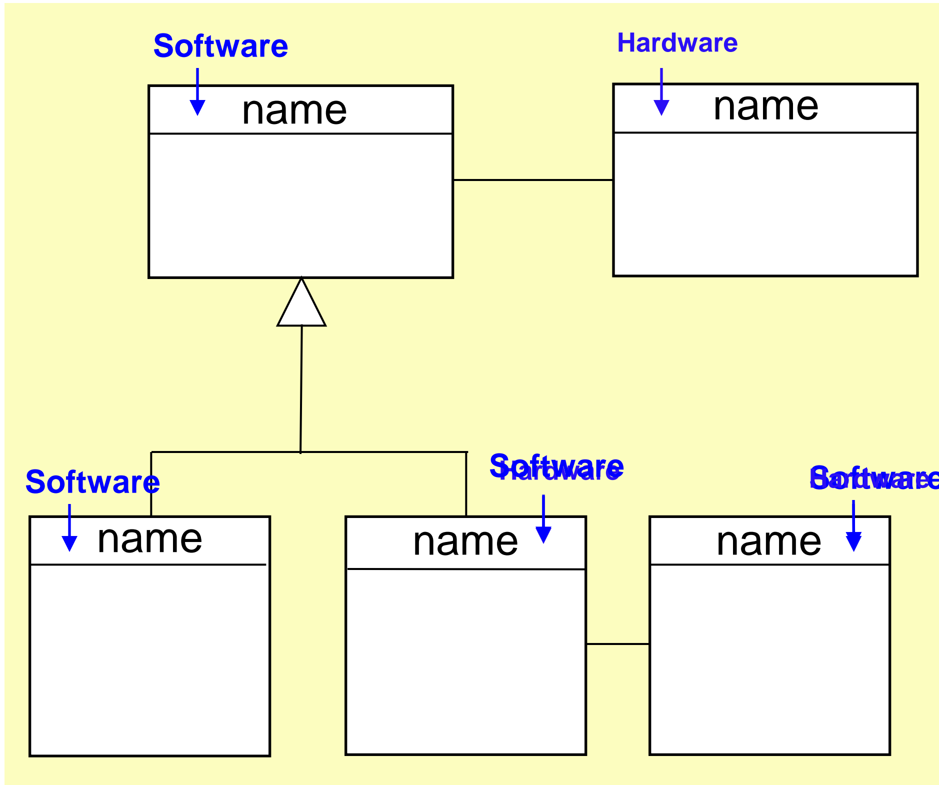


Profile it!



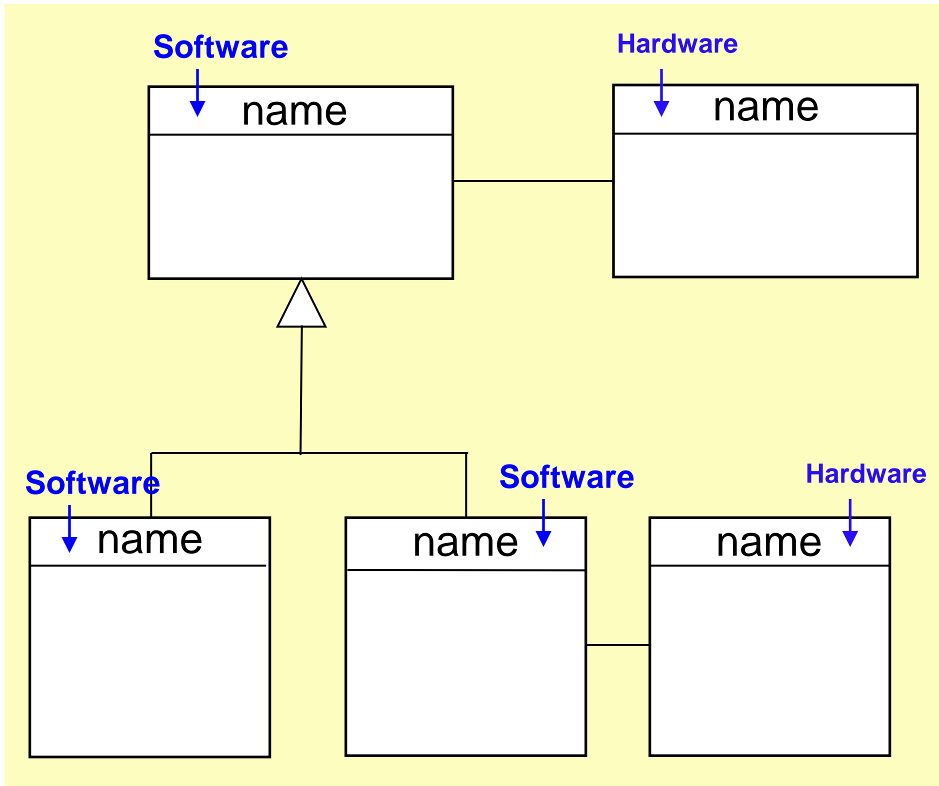
- Run Profiler
- Analyze the performance
- Not as good as you expected?

Re-Mark it!



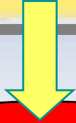
Reassign Classes to Different Tasks

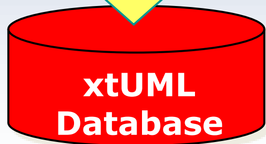
Make it Again!

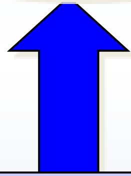


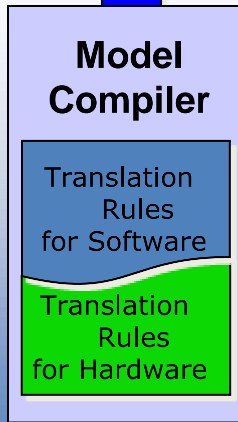
Application

Model Builder









Architecture

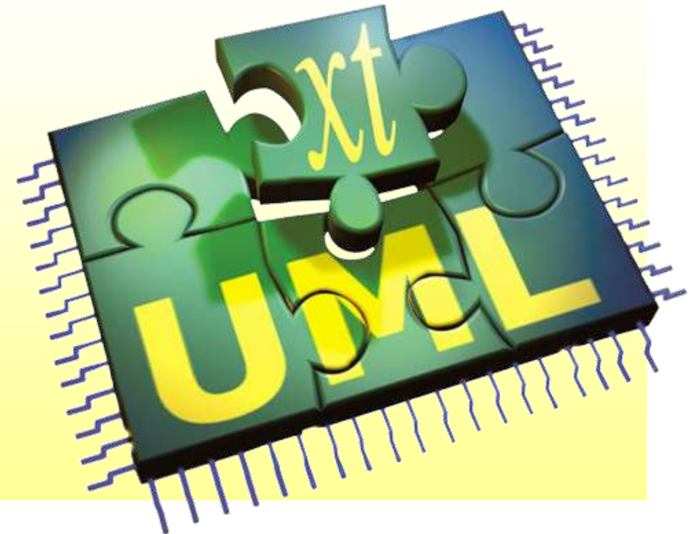
xtUML is a streamlined subset of the UML industry standard that:

(X) Executes models

- **Allows for early verification**
- **Pre-code interpretive execution**
- **Integration of legacy code**

(T) Translates models

- **Complete code generation from models**
- **Customizable compilation rules**
- **Optimized code**



Questions and Discussion

Create Reusable IP Assets

- Application Model
  - Completely captures *application* subject-matter expertise
  - Executable and therefore can be verified
  - Implementation-independent
- Model Compiler
  - Completely captures *design* and *implementation* expertise
  - Executable and therefore can be verified
  - Application-independent

Modeling, what does it buy us?

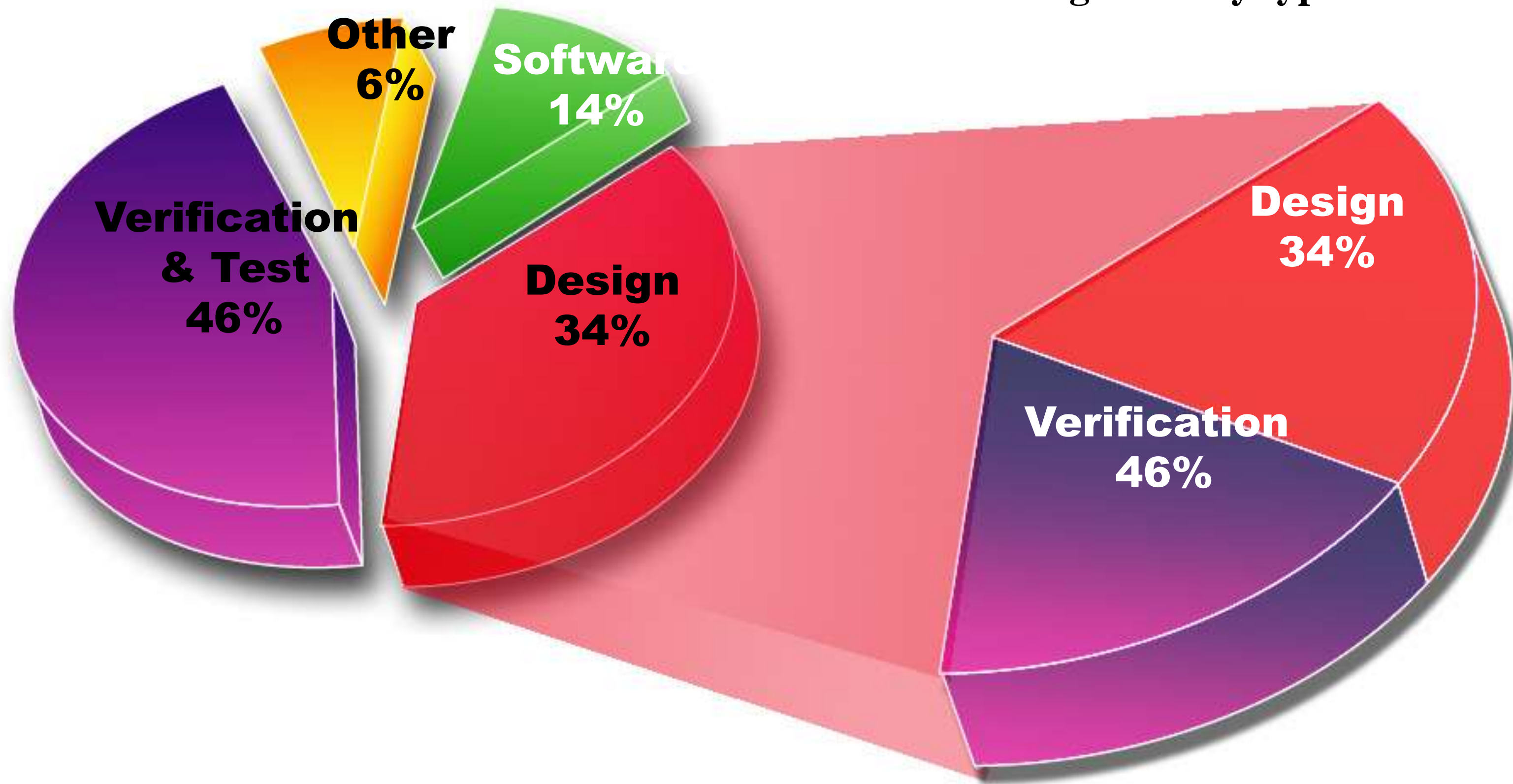
- Abstraction => Increased Productivity
- Communication => Fewer Defects
- Early Verification => Fewer Defects
- Correct Interfaces => Fewer Defects
- Delay Technology Decisions => Lower Cost
- Skill Specialization => Increased Productivity
- Reusable IP => Increased Productivity, Lower Costs

Late-night Blues to Blue-sky

- Exploring a big design space for SoCs
  - Analyze the application models:
    - Statically
    - Dynamically
  - Determine:
    - Degree of concurrency
    - Throughput and response requirements
  - Define, allocate, and generate:
    - Custom cores and logic
    - Optimized software for the cores

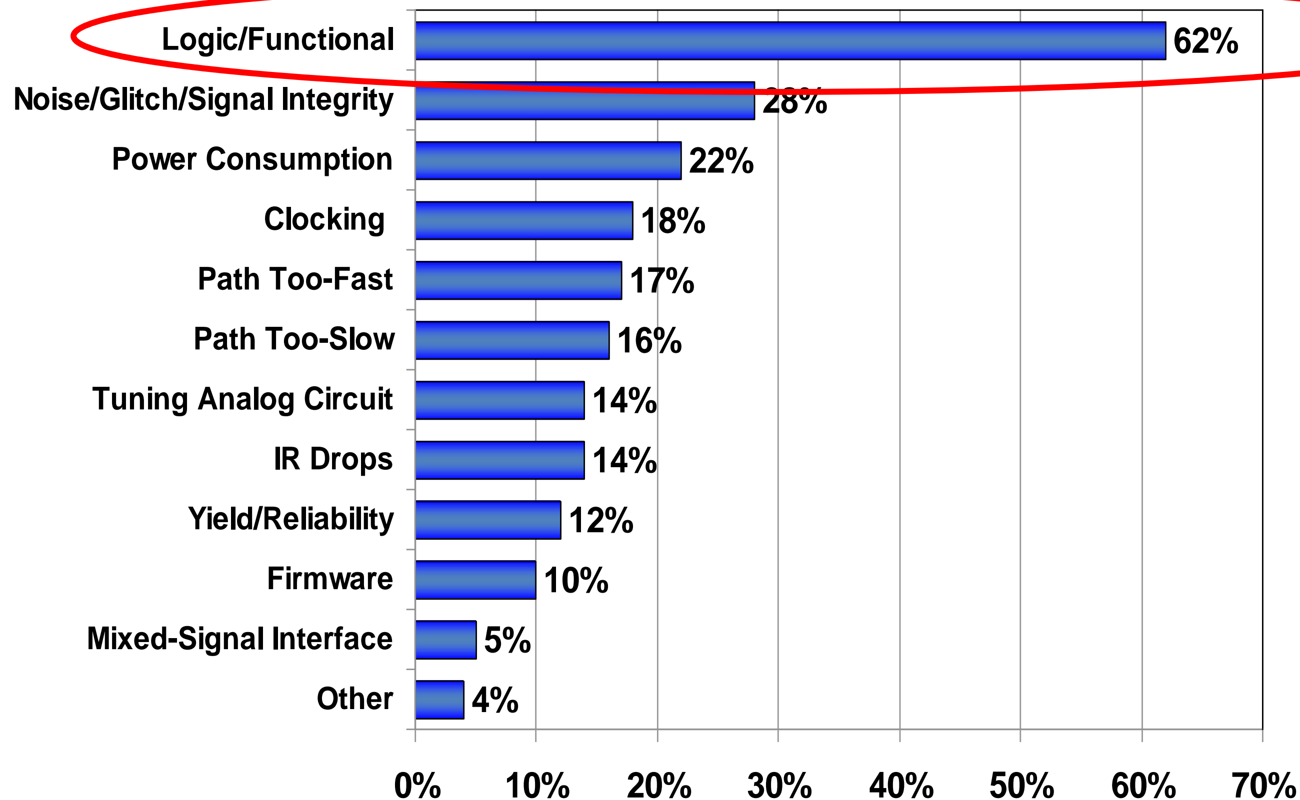
Verification is Over 60% of the IC/ASIC Design Effort…

Effort Allocation of IC/ASIC Design Engineers by type of Activity



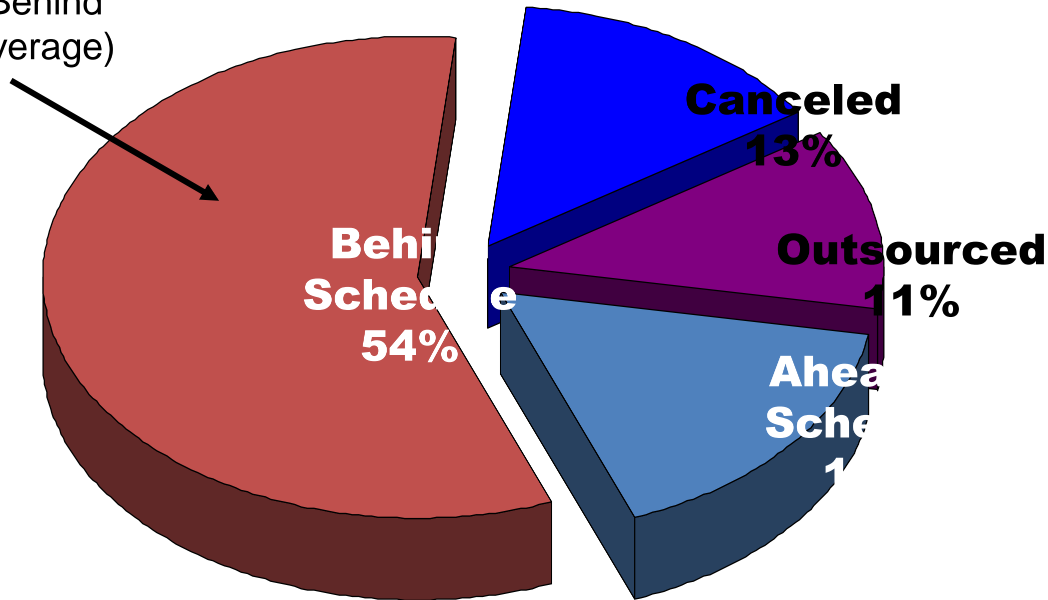
Source: 2002 IC/ASIC Functional Verification Study, Collett International Research

…Yet Flaws Still Make It To Silicon



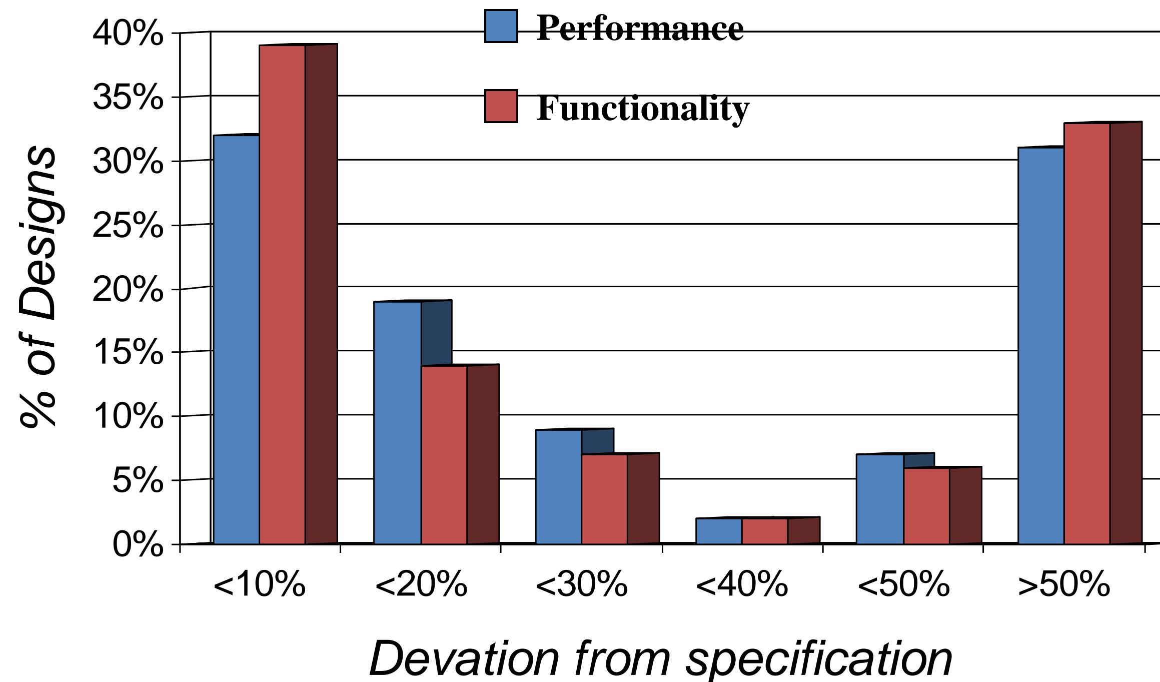
Source: 2002 IC/ASIC Functional Verification Study, Collett International Research

…Putting Half of Embedded Designs Behind Schedule



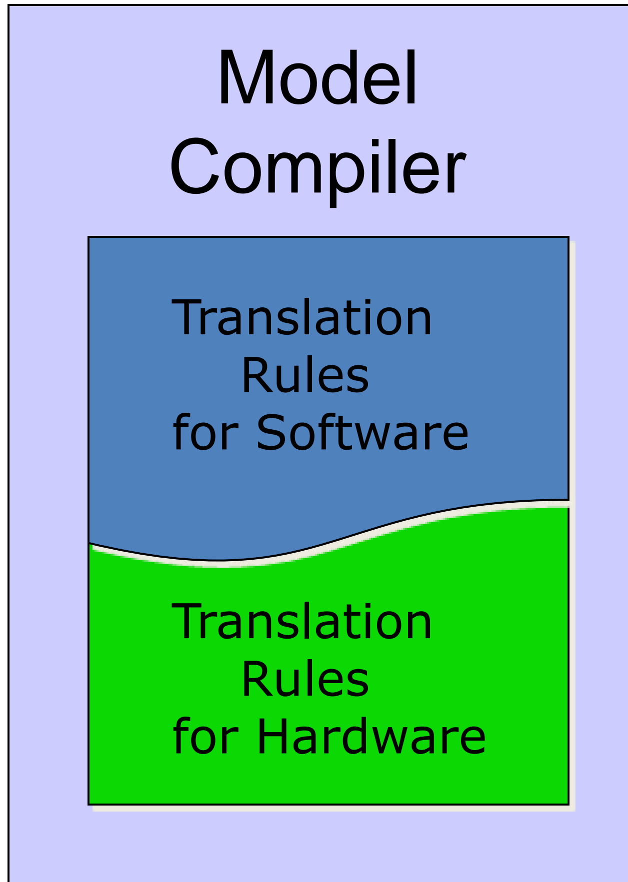
Source: Electronic Market Forecasters, June 2003

…With Half Missing Performance and Functionality Goals



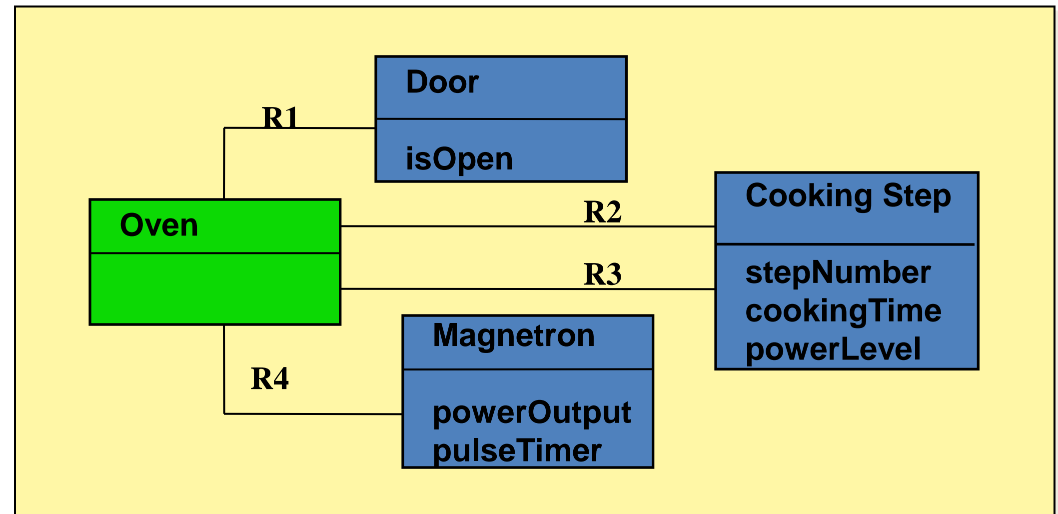
Source: Electronic Market Forecasters, June 2003

Model Compiler



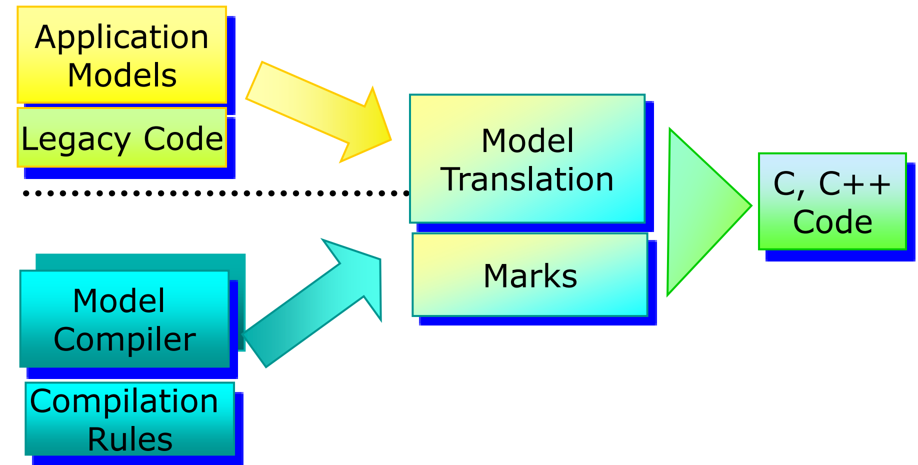
A model compiler

- **is a collection of *translation rules***
- **uses *marks* to decide which rule to apply to which model element**



xtUML Model Translation

- A model compiler
- Consistently builds complete system from models
- Translates into best language for system
- Uses rules and marks for optimization of resulting code



This approach….

- leverages expertise of best architects,
- captures that expertise.