Process

The gap between specification and synthesis is large, so we often specify in terms of implementation. Consequently, we begin with the hardware/software partition. This:

- defines interfaces early
- tests out misunderstandings late
- inhibits changing the partition
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
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

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

Since both teams are doing similar things:
- Creating abstractions,
- Formalizing them, and
- Verifying behavior

Use the same approach to specify and verify behavior.
Towards a Solution

Jointly:

- Build a *single* application model
- Execute the application model for verification
- *Don’t* model implementation structure
- Map the application model to implementation

Create  Formalize  Verify
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!
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:

- Streamlined
- Tractable
- Subset of UML

With defined semantics and execution rules
Building Models

- Models capture the behavior of the entire system
- Including an Object Action Language
- Captured in a database
Model Capture

Intelligent model capture verifies models:

- Syntactically
- Semantically

For an executable model with actions
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

Class
- Number {I}
- Name
- KeyLetters
- Description

State
- Name {I}

Class {R7} isFinal

Metamodel

Entity Shutter_VHDL

Class Exposure_C

Repository

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Class</th>
<th>Number</th>
<th>isFinal</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Shutter</td>
<td>973</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>Exposure</td>
<td>974</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Open</td>
<td>975</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

Application

Translation Rules

Entity Shutter_VHDL { ... 
Class Exposure_C { ... 

Copyright © 2013 Stephen J. Mellor. All rights reserved.
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
);

.public:
enum states_e
{ NO_STATE = 0 ,
 .for each state in stateS
  .if ( not last stateS )
   ${state.Name} ,
  .else
  NUMSTATES = ${state.Name}
  .end if
 .end for
};

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

.public:
enum states_e
{ NO_STATE = 0 ,
  CAMERA_OPEN ,
  CAMERA_CHECKING ,
  NUMSTATES = CAMERA_CLOSED
};
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
```
Write rules to create instances in the repositories

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

(X) Executes models
- Allows for early verification
- Integration of legacy code

(T) Translates models
- Complete code generation from models
- Customizable compilation rules
- Optimized code
Want to learn more?

**Executable UML: A Foundation for Model-Driven Architecture**, Stephen J. Mellor, Marc Balcer

- Comprehensive language introduction and reference

**fuml.modeldriven.org**

**UML for SoC Design**

**Why SoC needs more UML like a Hole in the Head**

Mellor, Wolfe, McCausland

**Mastering the Requirements Process**

Robertson & Robertson

**Model Driven Solutions**

**Co-engineering course**

www.volere.co.uk/reco.html
Thank you