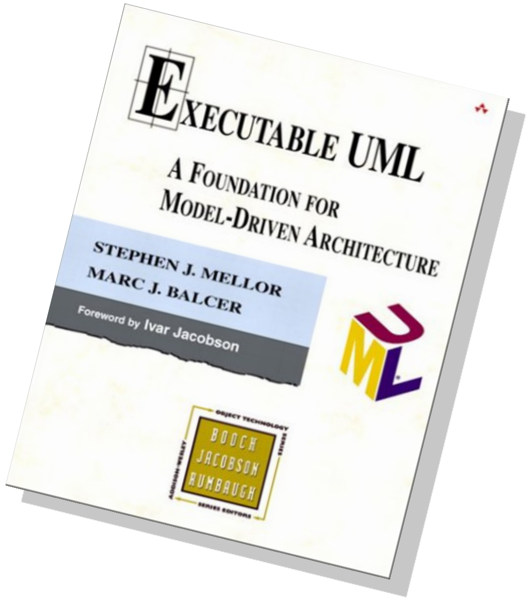
Overview

- **xtUML Modeling**
  - **Method**
  - **Executable model hierarchy**
  - **Relationship between model elements**
  - **Analysis models**
  - **Packages**

xtUML – Executable and Translatable UML

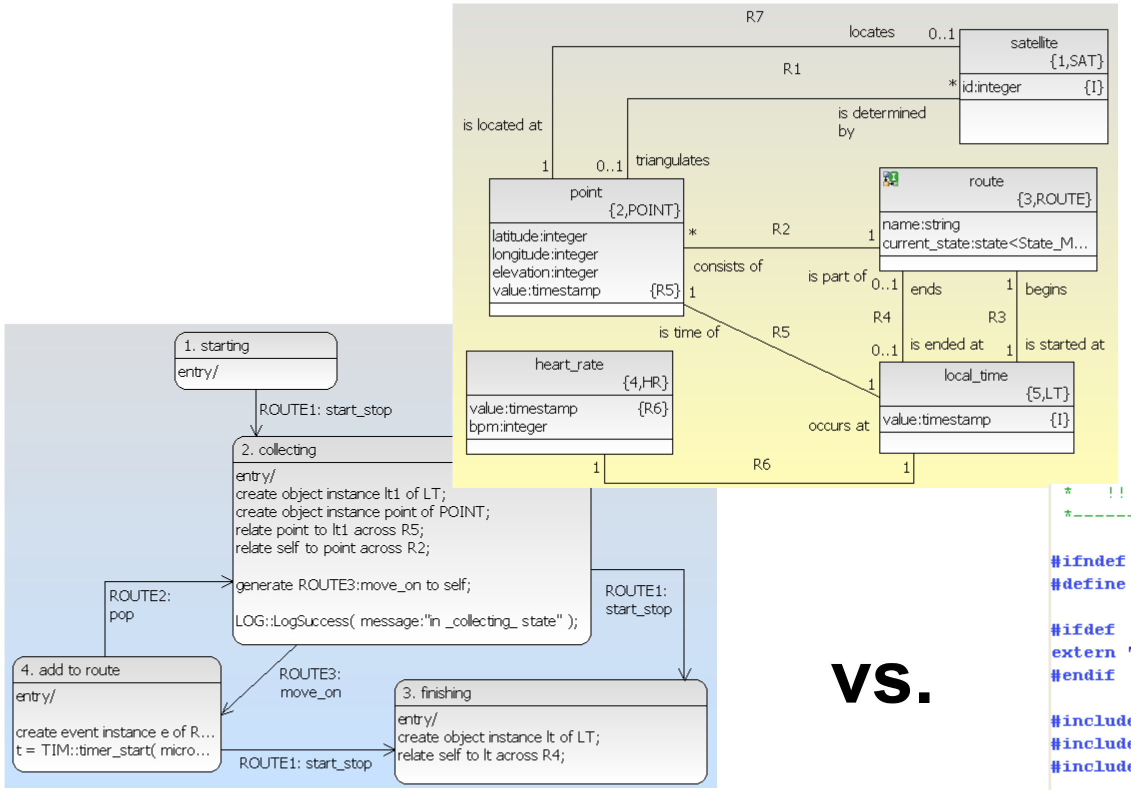
- Defines a method, including:
  - Semantics of diagrams
  - Relationship between diagrams
  - Action language
  - Execution rules
  - Order of construction
  - Path to implementation



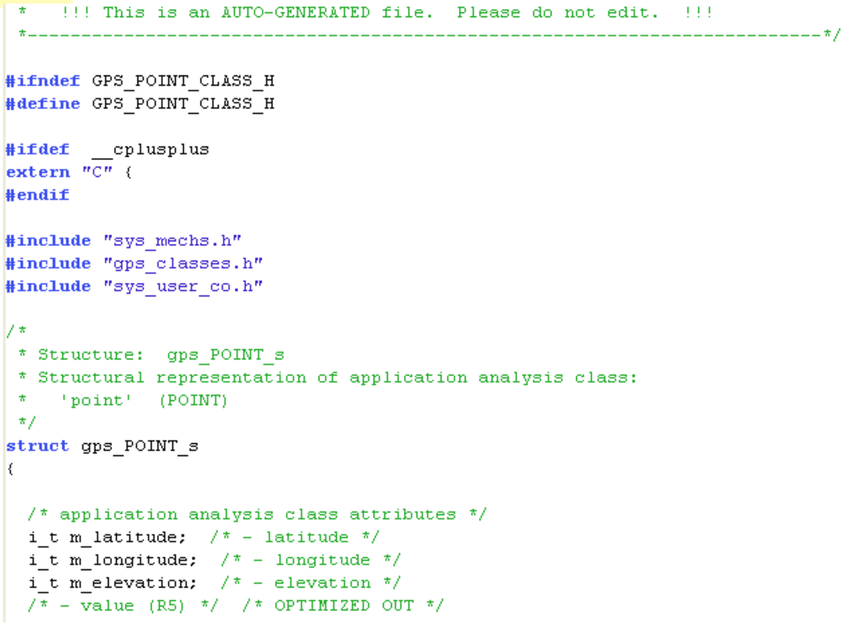
EXECUTABLE UML
A FOUNDATION FOR MODEL-DRIVEN ARCHITECTURE
STEPHEN J. MELLOR
MARC J. BALCER
Foreword by Ivar Jacobson
BOOCH JACOBSON RUMBAUGH

400+ pages

Graphical Models Increase Understanding



vs.



```
 *   !!! This is an AUTO-GENERATED file.  Please do not edit.  !!!
 *--------------------------------------------------------------------------*/

#ifndef GPS_POINT_CLASS_H
#define GPS_POINT_CLASS_H

#ifdef   __cplusplus
extern "C" {
#endif

#include "sys_mechs.h"
#include "gps_classes.h"
#include "sys_user_co.h"

/*
 * Structure:  gps_POINT_s
 * Structural representation of application analysis class:
 *   'point'  (POINT)
 */
struct gps_POINT_s
{

  /* application analysis class attributes */
  i_t m_latitude;  /* - latitude */
  i_t m_longitude;  /* - longitude */
  i_t m_elevation;  /* - elevation */
  /* - value (R5) */  /* OPTIMIZED OUT */
```

- **Diagrams**
  - **Expose Abstractions**
  - **Allow us to reason about correctness**

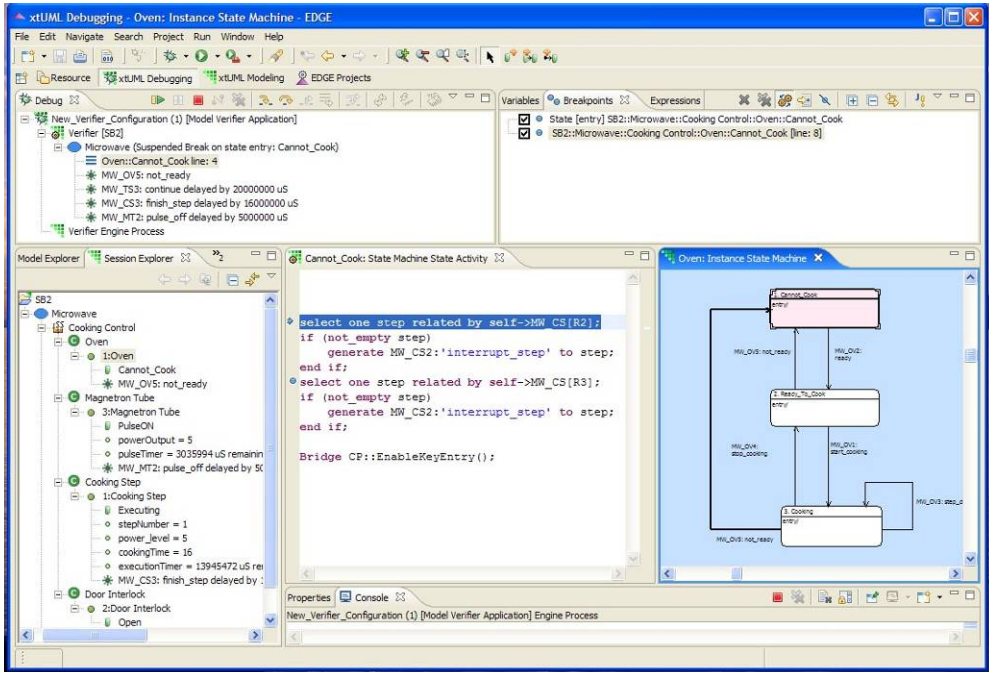
Execution: My piece runs, how about yours?

We find some defects through inspection, but…

…we find the rest by executing the code.

What if we could execute the application before choosing the:

- processor
- language
- OS?


xtUML Debugging - Oven: Instance State Machine - EDGE
File Edit Navigate Search Project Run Window Help
Resource
xtUML Debugging
xtUML Modeling
EDGE Projects
Debug
Variables
Breakpoints
Expressions
Model Explorer
Session Explorer
Cannot_Cook: State Machine State Activity
Oven: Instance State Machine
Properties
Console

```
New_Verifier_Configuration (1) [Model Verifier Application]
  Verifier [SB2]
    Microwave (Suspended Break on state entry: Cannot_Cook)
      Oven::Cannot_Cook line: 4
      MW_OV5: not_ready
      MW_TS3: continue delayed by 20000000 uS
      MW_CS3: finish_step delayed by 16000000 uS
      MW_MT2: pulse_off delayed by 5000000 uS
  Verifier Engine Process
```

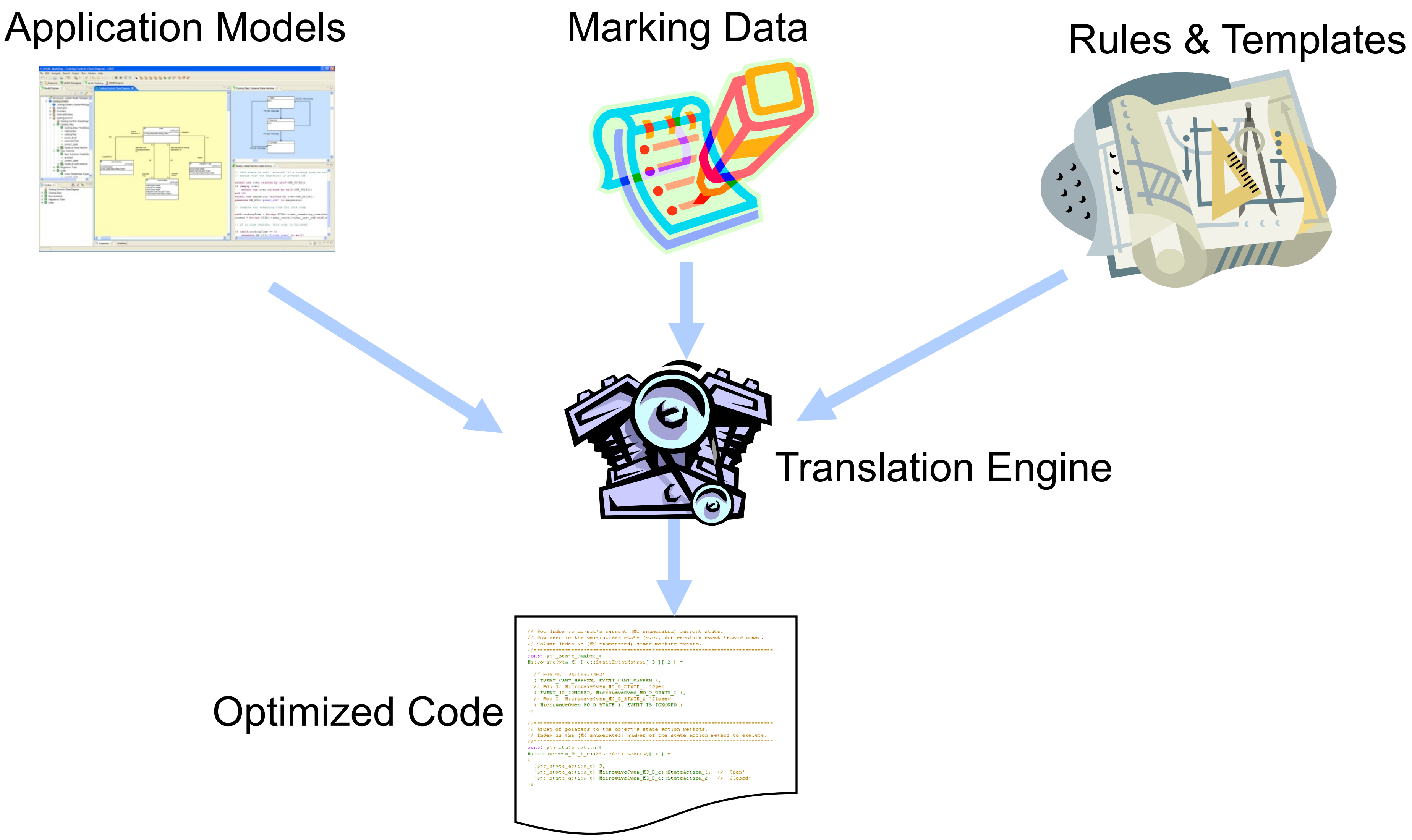
```
State [entry] SB2::Microwave::Cooking Control::Oven::Cannot_Cook
SB2::Microwave::Cooking Control::Oven::Cannot_Cook [line: 8]
```

```
SB2
  Microwave
    Cooking Control
      Oven
        1:Oven
          Cannot_Cook
          MW_OV5: not_ready
      Magnetron Tube
        3:Magnetron Tube
          PulseON
          powerOutput = 5
          pulseTimer = 3035994 uS remainin
          MW_MT2: pulse_off delayed by 5C
      Cooking Step
        1:Cooking Step
          Executing
          stepNumber = 1
          power_level = 5
          cookingTime = 16
          executionTimer = 13945472 uS rei
          MW_CS3: finish_step delayed by :
      Door Interlock
        2:Door Interlock
          Open
```

```
select one step related by self->MW_CS[R2];
if (not_empty step)
    generate MW_CS2:'interrupt_step' to step;
end if;
select one step related by self->MW_CS[R3];
if (not_empty step)
    generate MW_CS2:'interrupt_step' to step;
end if;

Bridge CP::EnableKeyEntry();
```

```
New_Verifier_Configuration (1) [Model Verifier Application] Engine Process
```

Compiling Models



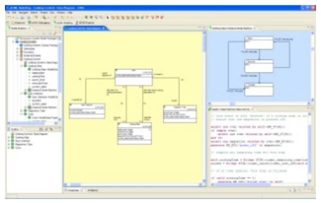
```
// Row index is object's current (MC enumerated) current state.
// Row zero is the uninitialized state (e.g., for creation event transitions).
// Column index is (MC enumerated) state machine events.
//==========================================================================
const ptr_state_number_t
MicrowaveOven_MO_D_c::StateEventMatrix[ 3 ][ 2 ] =
{
  // Row 0: 'Uninitialized'
  { EVENT_CANT_HAPPEN, EVENT_CANT_HAPPEN },
  // Row 1: MicrowaveOven_MO_D_STATE_1 'Open'
  { EVENT_IS_IGNORED, MicrowaveOven_MO_D_STATE_2 },
  // Row 2: MicrowaveOven_MO_D_STATE_2 'Closed'
  { MicrowaveOven_MO_D_STATE_1, EVENT_IS_IGNORED }
};

//==========================================================================
// Array of pointers to the object's state action methods.
// Index is the (MC enumerated) number of the state action method to execute.
//--------------------------------------------------------------------------
const ptr_state_action_t
MicrowaveOven_MO_D_c::[illegible]Array[ 3 ] =
{
  (ptr_state_action_t) 0,
  (ptr_state_action_t) MicrowaveOven_MO_D_c::StateAction_1,  //  'Open'
  (ptr_state_action_t) MicrowaveOven_MO_D_c::StateAction_2   //  'Closed'
};
```

Separate Application from Implementation

- Subject-matter experts focus on application
  - Features and capabilities
  - Intricacies of the application

- Implementation experts focus on optimization
  - Faster, smaller
  - Less power
  - Lower cost

Application Models

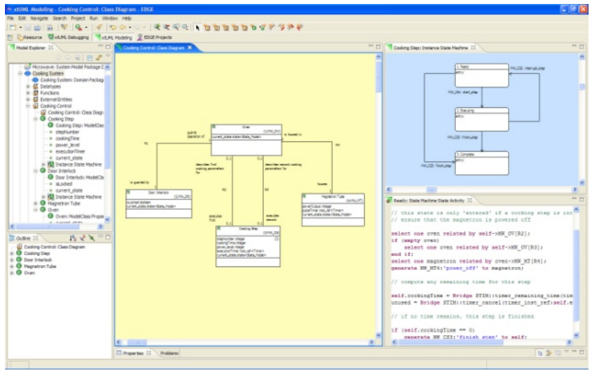


Model Compiler

Two Types of Reusable IP

Reusable IP: Application Models

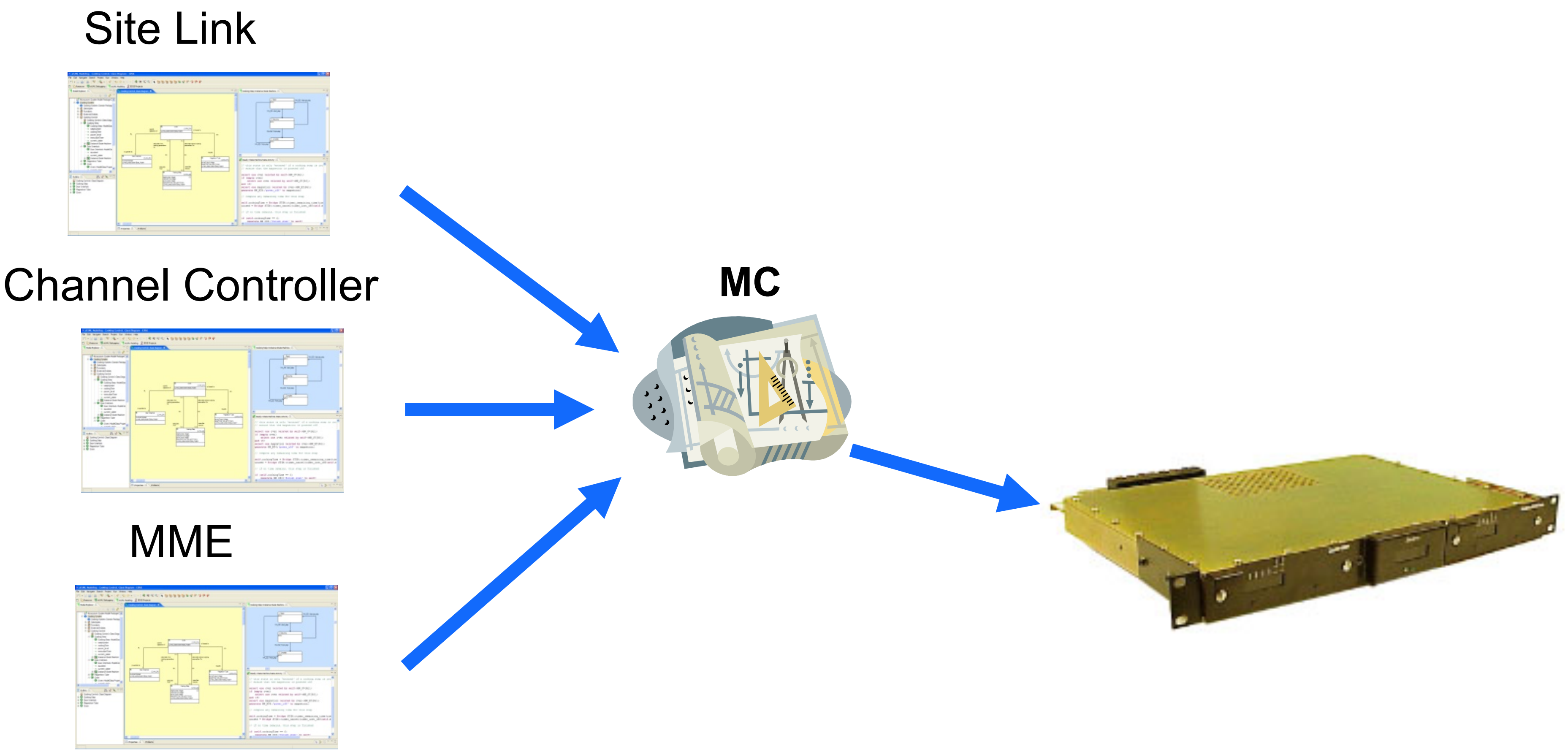
- ♦ **Platform-independent Application Models**
  - ● **Reuse application models across platforms and product variants.**



```
// this state is only 'entered' if a cooking step is int
// ensure that the magnetron is powered off

select one oven related by self->MW_OV[R2];
if (empty oven)
    select one oven related by self->MW_OV[R3];
end if;
select one magnetron related by oven->MW_MT[R4];
generate MW_MT4:'power_off' to magnetron;

// compute any remaining time for this step

self.cookingTime = Bridge STIM::timer_remaining_time(tim
unused = Bridge STIM::timer_cancel(timer_inst_ref:self.e

// if no time remains, this step is finished

if (self.cookingTime == 0)
    generate MW_CS3:'finish step' to self;
```

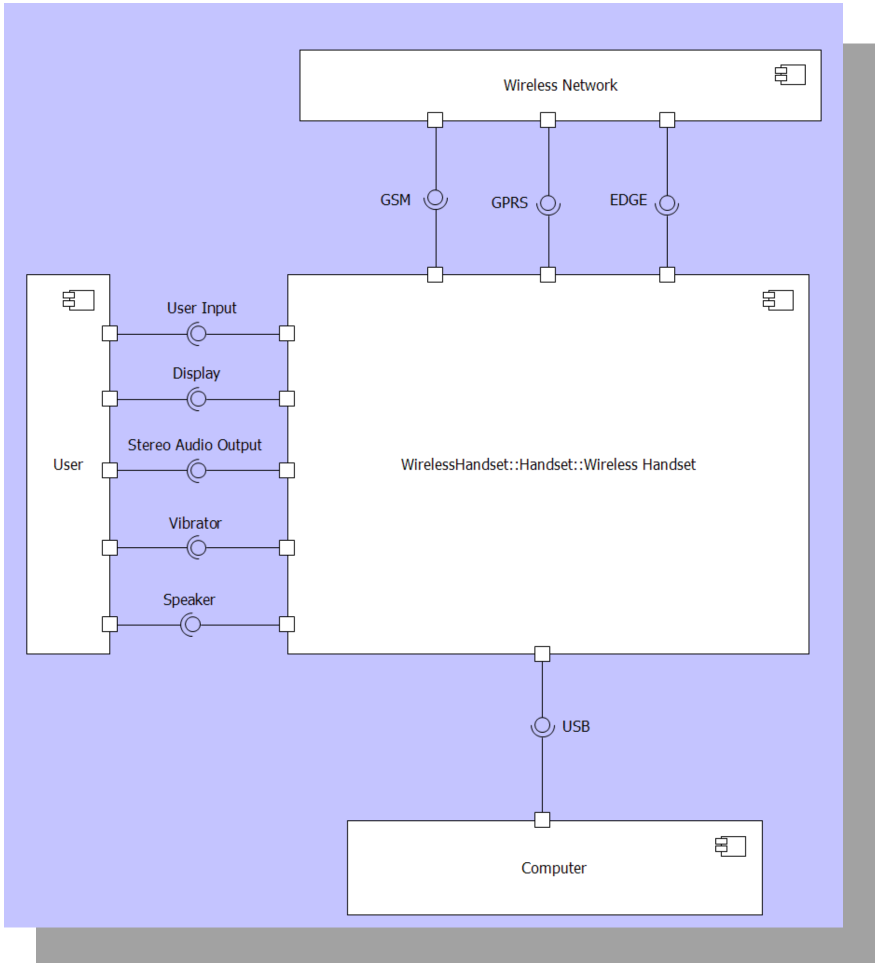
Reusable IP: Model Compilers

- **Application-independent Model Compile**
  - **Reuse one model compiler across many applications.**



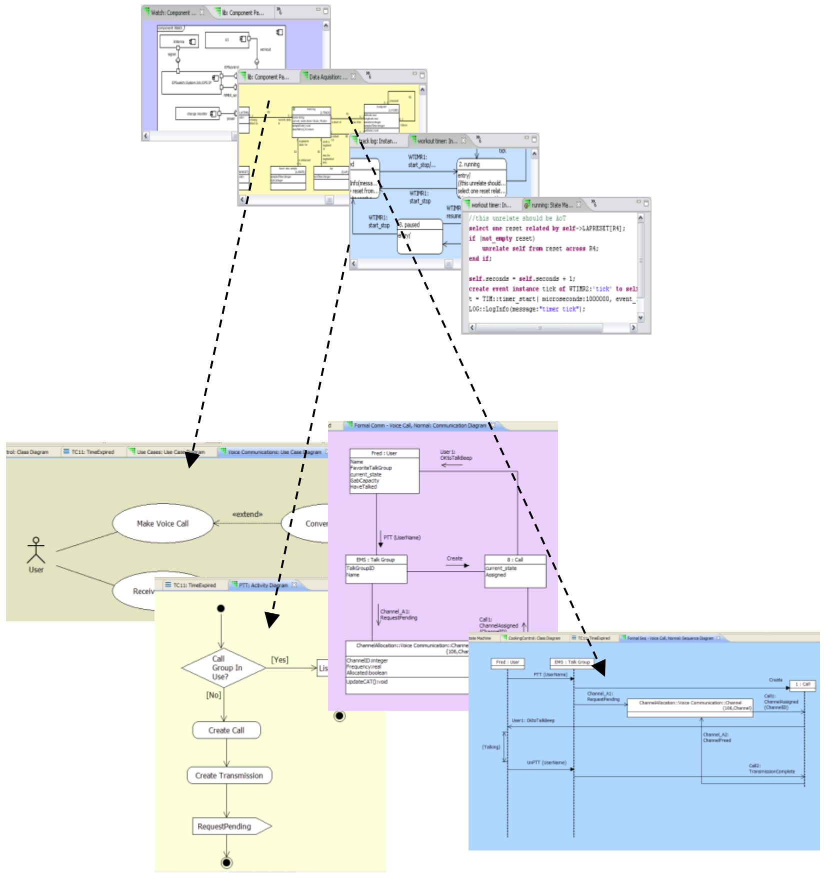
Getting Started

- Analyze application using descriptive modeling
- Divide and conquer
  - Any boundary
  - Hierarchically nesting
- Define interfaces
  - Operations and signals
- Connect components

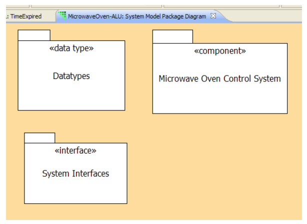


Relationship between Model Elements

- ♦ **Analysis, loosely coupled**
  - **Use Case**
  - **Sequence**
  - **Communication**
  - **Activity**
- ♦ **Executable, tightly coupled:**
  - **Component**
  - **Class**
  - **State**
  - **Action**
- ♦ **Package**



```
//this unrelate should be AoT
select one reset related by self->LAPRESET[R4];
if (not_empty reset)
    unrelate self from reset across R4;
end if;

self.seconds = self.seconds + 1;
create event instance tick of WTIMR2:'tick' to sel
t = TIM::timer_start( microseconds:1000000, event_
LOG::LogInfo(message:"timer tick");
```



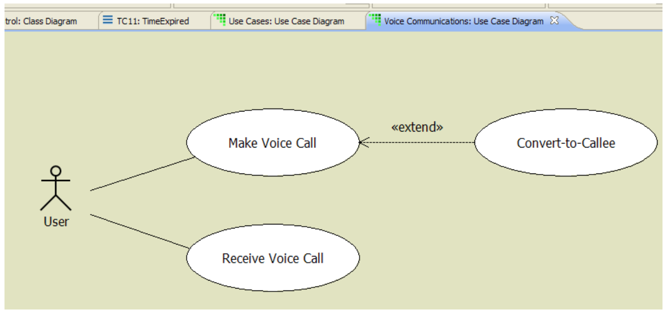
MicrowaveOven-ALU: System Model Package Diagram
«data type»
Datatypes
«component»
Microwave Oven Control System
«interface»
System Interfaces

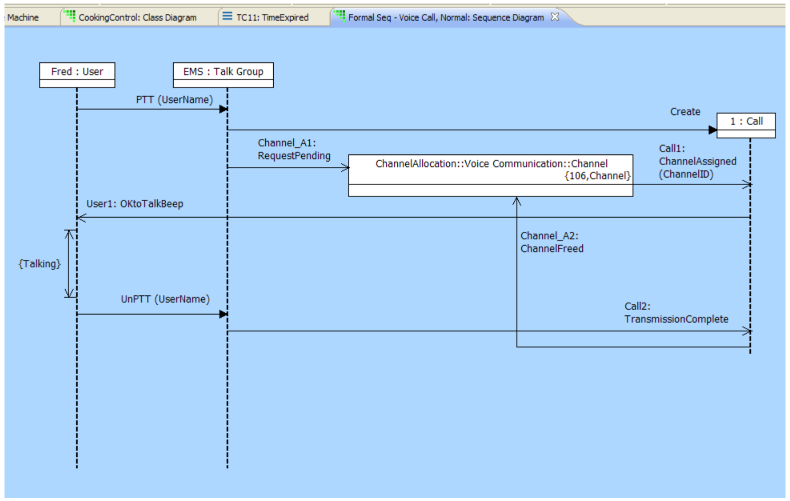
Analysis Models

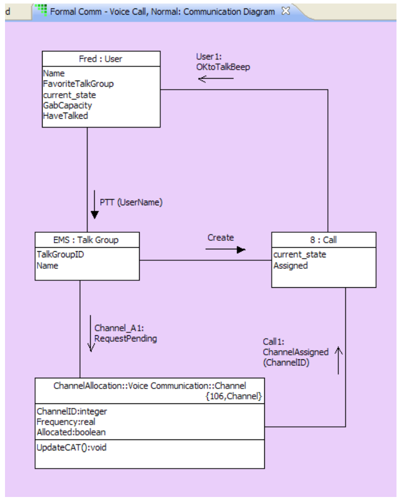
Use as you see fit

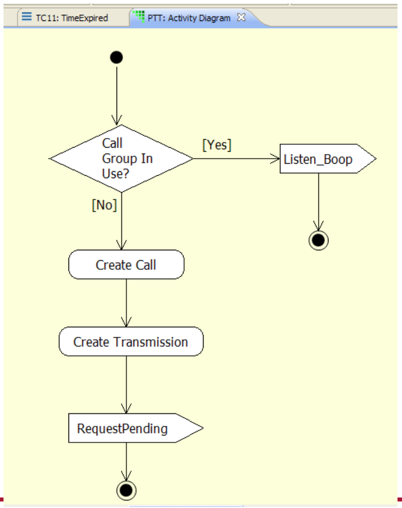
- **Use Case**
- **Sequence ***
- **Communication ***
- **Activity**

* Sequence and Communication can be Formalized




Machine
CookingControl: Class Diagram
TC11: TimeExpired
Formal Seq - Voice Call, Normal: Sequence Diagram
Fred : User
EMS : Talk Group
PTT (UserName)
Create
1 : Call
Channel_A1: RequestPending
ChannelAllocation::Voice Communication::Channel {106,Channel}
Call1: ChannelAssigned (ChannelID)
User1: OKtoTalkBeep
{Talking}
Channel_A2: ChannelFreed
UnPTT (UserName)
Call2: TransmissionComplete





These Diagrams are NOT translated.

Executable Model Hierarchy

High level

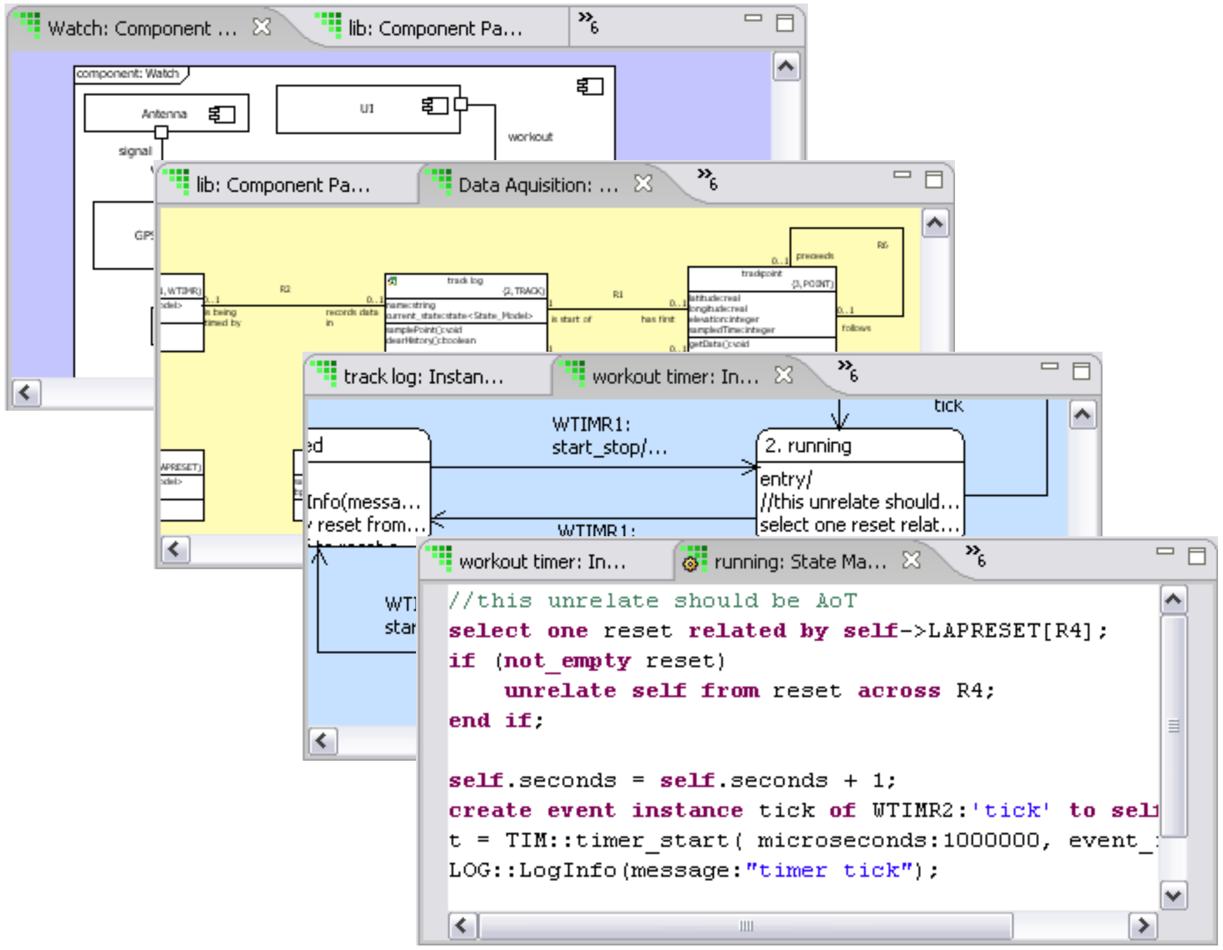
Component Diagram
- **Decompose the application**
- **Define Interfaces**

Class Diagram
- **Abstractions, associations**
- **Operations**

State Diagram
•Functional lifecycle
•Event handling

Action Specification
- **Processing**

Low level

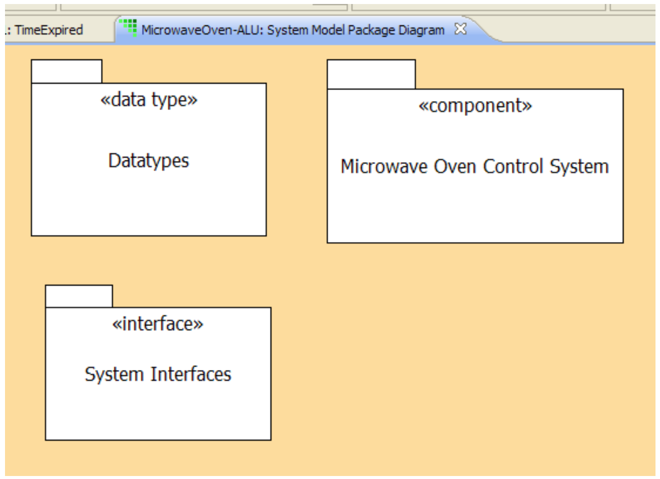


```
//this unrelate should be AoT
select one reset related by self->LAPRESET[R4];
if (not_empty reset)
    unrelate self from reset across R4;
end if;

self.seconds = self.seconds + 1;
create event instance tick of WTIMR2:'tick' to sel
t = TIM::timer_start( microseconds:1000000, event_
LOG::LogInfo(message:"timer tick");
```

These are the Translatable Diagrams.

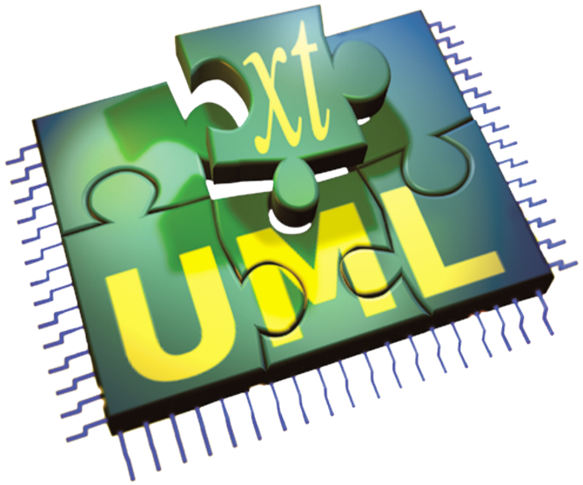
Packages

- Packages can contain: (anything)
  - Package
  - Activitys, Communications, Sequences, Use Cases
  - Components, Interfaces, Data Types, Classes
- Visibility and Namespacing
  - Establish namespace
  - Can limit visibility
- Generic packages, per UML
  - Namespace
  - Visibility controls
  - Separate diagram and package concepts


: TimeExpired
MicrowaveOven-ALU: System Model Package Diagram
«data type»
Datatypes
«component»
Microwave Oven Control System
«interface»
System Interfaces

Summary – Steps in the xtUML Method

- **Analysis** – questioning, thinking, sketching...
  - Descriptive UML diagrams
    - use case, sequence, ...
- **Executable Modeling** – formalizing the analysis:
  - Component Diagrams (partitioning/interfaces)
  - Class Diagrams (data)
  - State Machines (control)
  - Activities (processing)
- **Verification**
  - Interpretive Model Execution
- **Code generation**
  - Template and Rule-Based Translation



BridgePoint™