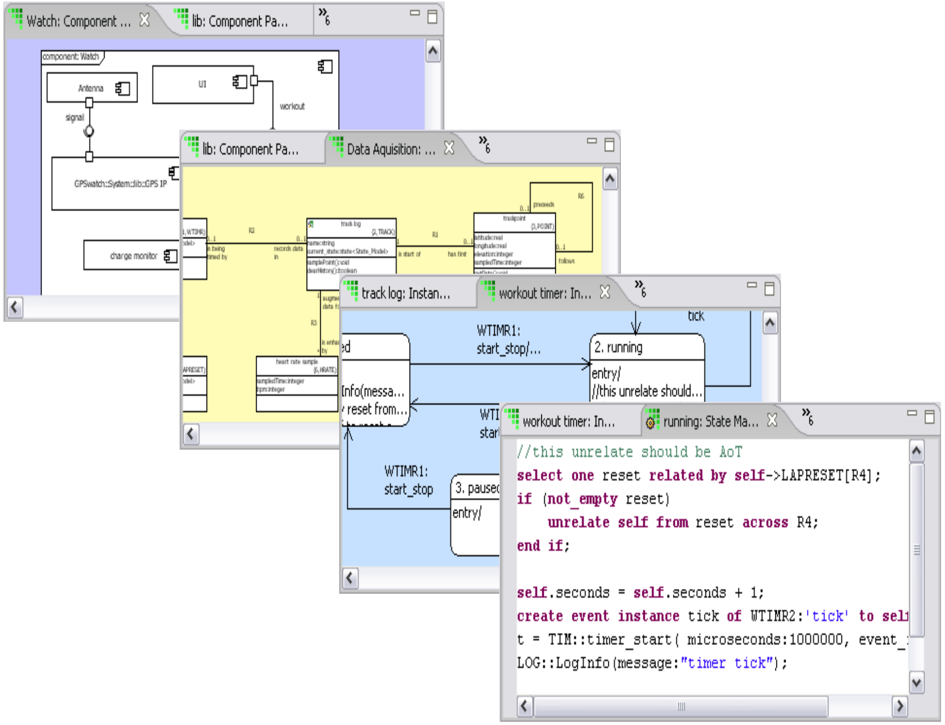
The xtUML method - Verification

- **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

Execution Rules

- **Bottom Up**
  - **Types of actions**
  - **Homes for actions**
  - **State models**
  - **Event delivery**
  - **Event ordering**
  - **Delayed events**
  - **Concurrency**
  - **Interface messages**
  - **Bridges**



```
//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");
```

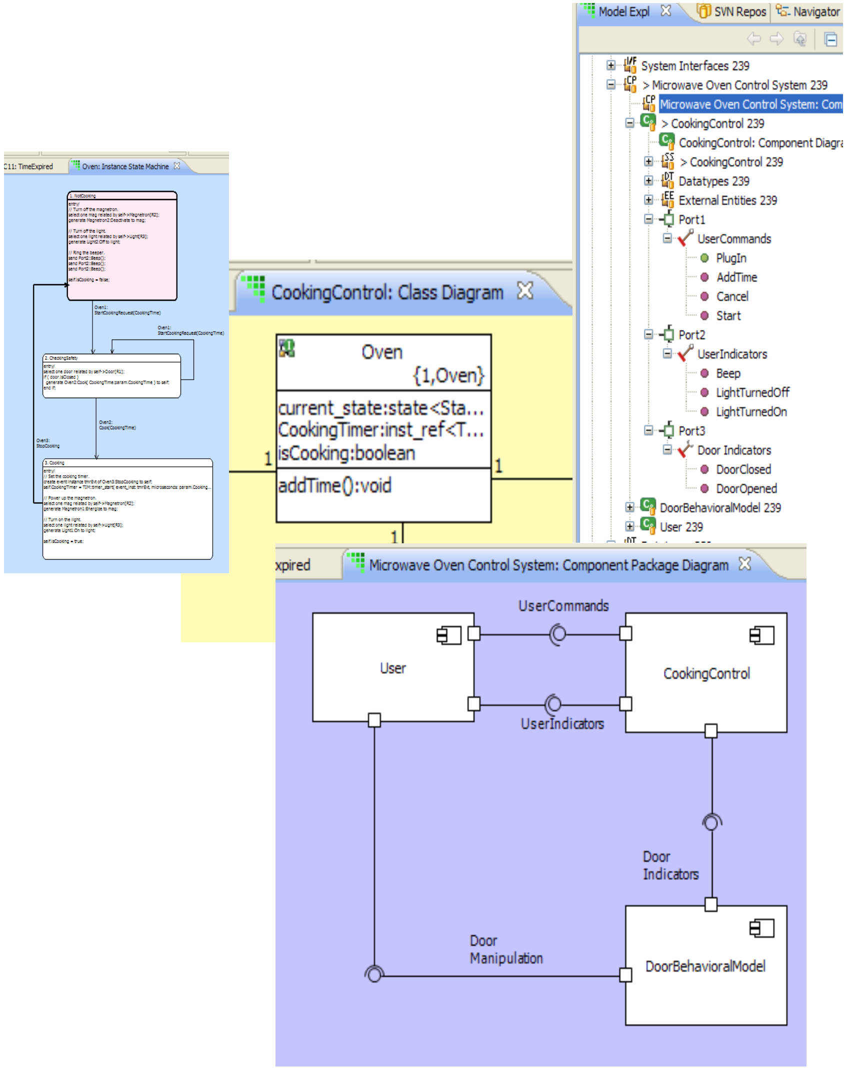
Types of Actions

- create, delete instances
- read, write attributes
- read parameter values
- relate, unrelate instances
- invoke operations, set parameter values
- send events, set parameter values
- find instances
- computation
- create, read, write local variables
- control: iterate, loop, decision

Do we need anything else?

Homes for Actions

- States
- Transitions
- Operations
  - Instance-based
  - Class-based
- Ports
- Mathematically-derived attributes
- Bridge Operations
- Functions



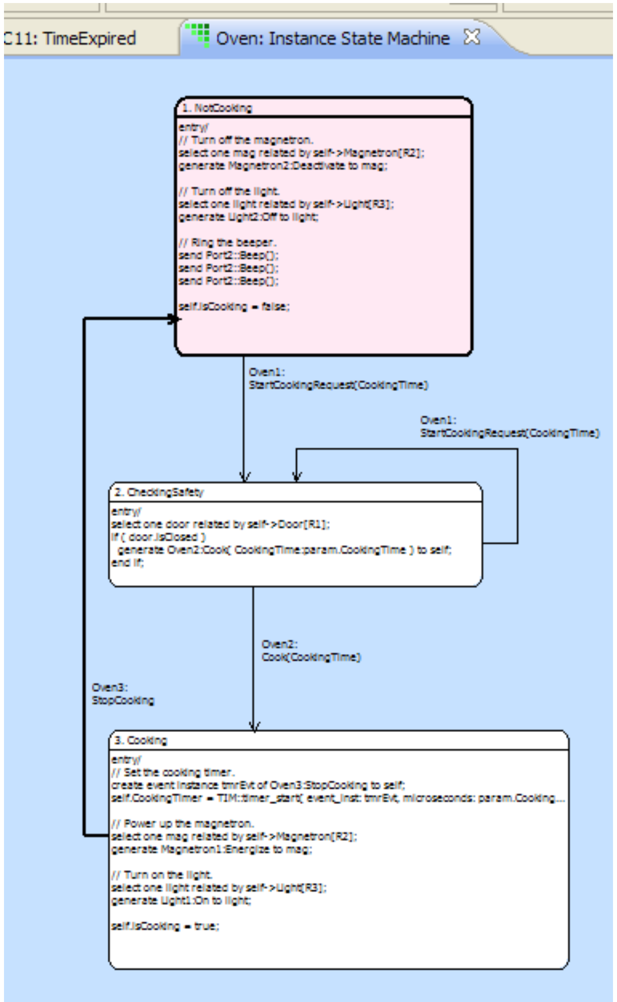
```
entry/
// Turn off the magnetron.
select one mag related by self->Magnetron[R2];
generate Magnetron2:Deactivate to mag;

// Turn off the light.
select one light related by self->Light[R3];
generate Light2:Off to light;

// Ring the beeper.
send Port2::Beep();
send Port2::Beep();
send Port2::Beep();

self.isCooking = false;
```

```
entry/
select one door related by self->Door[R1];
if ( door.isClosed )
  generate Oven2:Cook( CookingTime:param.CookingTime ) to self;
end if;
```

```
entry/
// Set the cooking timer.
create event instance tmrEvt of Oven3:StopCooking to self;
self.CookingTimer = TIM::timer_start( event_inst: tmrEvt, microseconds: param.Cooking...

// Power up the magnetron.
select one mag related by self->Magnetron[R2];
generate Magnetron1:Energize to mag;

// Turn on the light.
select one light related by self->Light[R3];
generate Light1:On to light;

self.isCooking = true;
```

State Models

- Capture lifecycles in state models
  - Instance-based vs. Class-based
- Start by naming the states
- Define the legal transitions
- Associate events with transitions
- Actions take finite time
- Transitions are considered instantaneous
- All state machines execute concurrently
- Synchronous creation
  - No actions executed
  - Lowest numbered state
- Asynchronous creation
  - Action executed
  - Destination state for creation transition
- Final state


C11: TimeExpired
Oven: Instance State Machine
Oven1:
StartCookingRequest(CookingTime)
Oven1:
StartCookingRequest(CookingTime)
Oven2:
Cook(CookingTime)
Oven3:
StopCooking

1. NotCooking

```
entry/
// Turn off the magnetron.
select one mag related by self->Magnetron[R2];
generate Magnetron2:Deactivate to mag;

// Turn off the light.
select one light related by self->Light[R3];
generate Light2:Off to light;

// Ring the beeper.
send Port2::Beep();
send Port2::Beep();
send Port2::Beep();

self.IsCooking = false;
```

2. CheckingSafety

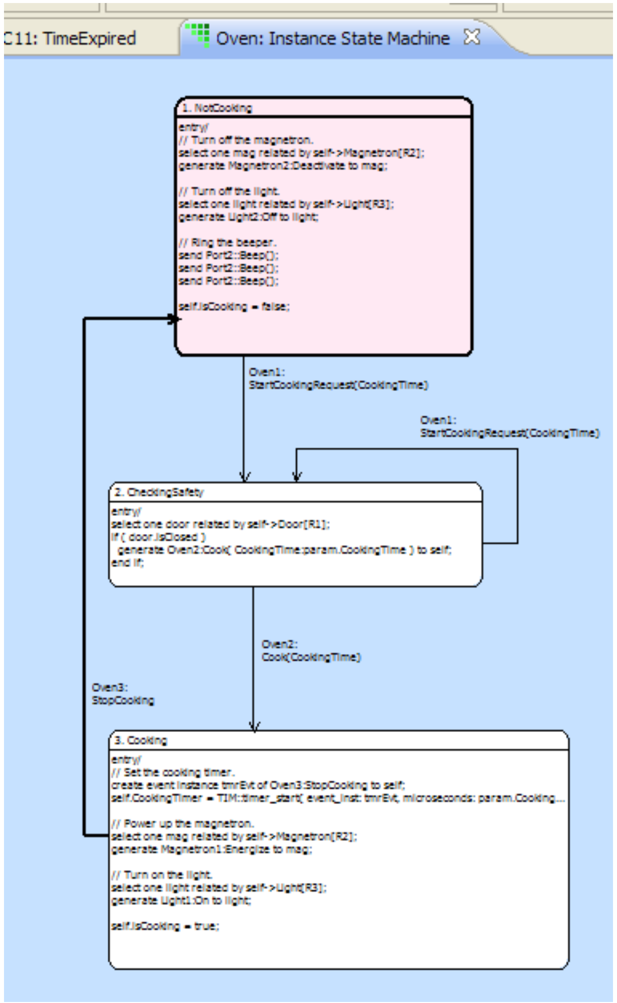
```
entry/
select one door related by self->Door[R1];
if ( door.IsClosed )
  generate Oven2:Cook( CookingTime:param.CookingTime ) to self;
end if;
```

3. Cooking

```
entry/
// Set the cooking timer.
create event instance tmrEvt of Oven3:StopCooking to self;
self.CookingTimer = TIM::timer_start( event_inst: tmrEvt, microseconds: param.Cooking...

// Power up the magnetron.
select one mag related by self->Magnetron[R2];
generate Magnetron1:Energize to mag;

// Turn on the light.
select one light related by self->Light[R3];
generate Light1:On to light;

self.IsCooking = true;
```

State Dispatch

- Event delivery causes one of:
  - Transition
  - Ignore
  - Error (“Can’t Happen”)
- Transition:
  - Execute actions on transition
  - Execute actions within state
  - Change current state
- Ignore:
  - Event is discarded, no state change, no actions
- Error:
  - System-level (as opposed to modeled) recovery invoked


C11: TimeExpired
Oven: Instance State Machine
Oven1:
StartCookingRequest(CookingTime)
Oven1:
StartCookingRequest(CookingTime)
Oven2:
Cook(CookingTime)
Oven3:
StopCooking

1. NotCooking

```
entry/
// Turn off the magnetron.
select one mag related by self->Magnetron[R2];
generate Magnetron2:Deactivate to mag;

// Turn off the light.
select one light related by self->Light[R3];
generate Light2:Off to light;

// Ring the beeper.
send Port2::Beep();
send Port2::Beep();
send Port2::Beep();

self.isCooking = false;
```

2. CheckingSafety

```
entry/
select one door related by self->Door[R1];
if ( door.isClosed )
  generate Oven2:Cook( CookingTime:param.CookingTime ) to self;
end if;
```

3. Cooking

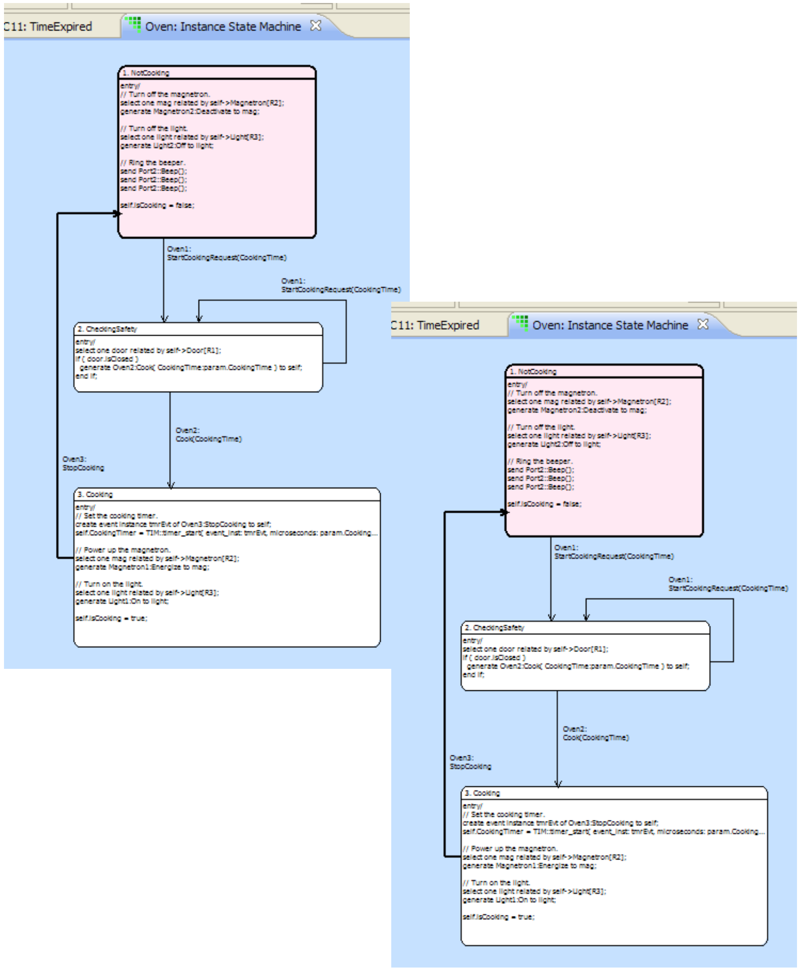
```
entry/
// Set the cooking timer.
create event instance tmrEvt of Oven3:StopCooking to self;
self.CookingTimer = TIM::timer_start( event_inst: tmrEvt, microseconds: param.Cooking...

// Power up the magnetron.
select one mag related by self->Magnetron[R2];
generate Magnetron1:Energize to mag;

// Turn on the light.
select one light related by self->Light[R3];
generate Light1:On to light;

self.isCooking = true;
```

Event Delivery

- Events are reliable
- Events do not interrupt executing actions
- Order is preserved among sender/receiver pair
- Self-directed events are delivered before others
- Delayed events specify *minimum* delay
  - Time EE provides timer operations and real-time clock
- Currently:
  - No guards
  - No re-queuing
  - No peeking or selecting among multiple events

Concurrency

- All state machines execute concurrently
- Models of concurrency may vary
- Full concurrency
  - Actions on transitions and within states may preempt others
  - Models must ensure data integrity
- Interleaved
  - Preemption occurs only on state boundaries
  - Models may assume state atomicity
- Consistent Data Access Set
  - Like full concurrency
  - Models may assume consistent data access set


C11: TimeExpired
Oven: Instance State Machine
Oven1: StartCookingRequest(CookingTime)
Oven1: StartCookingRequest(CookingTime)
Oven2: Cook(CookingTime)
Oven3: StopCooking

1. NotCooking

```
entry/
// Turn off the magnetron.
select one mag related by self->Magnetron[R2];
generate Magnetron2:Deactivate to mag;

// Turn off the light.
select one light related by self->Light[R3];
generate Light2:Off to light;

// Ring the beeper.
send Port2::Beep();
send Port2::Beep();
send Port2::Beep();

self.isCooking = false;
```

2. CheckingSafety

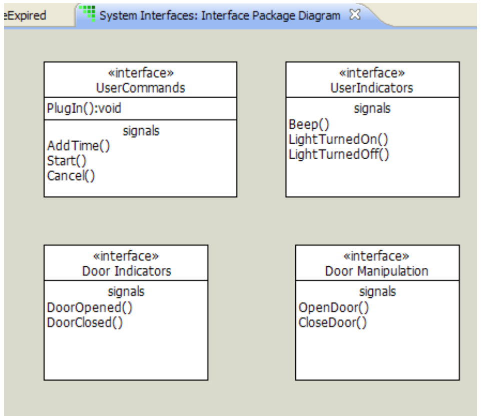
```
entry/
select one door related by self->Door[R1];
if ( door.isClosed )
  generate Oven2:Cook( CookingTime:param.CookingTime ) to self;
end if;
```

3. Cooking

```
entry/
// Set the cooking timer.
create event instance tmrEvt of Oven3:StopCooking to self;
self.CookingTimer = TIM::timer_start( event_inst: tmrEvt, microseconds: param.Cooking...

// Power up the magnetron.
select one mag related by self->Magnetron[R2];
generate Magnetron1:Energize to mag;

// Turn on the light.
select one light related by self->Light[R3];
generate Light1:On to light;

self.isCooking = true;
```

Interface Messages

- Provide inter-component communication
- Carry parameters
- Asynchronous Signals
  - May be mapped to class-based events
- Synchronous Operations
  - *Future*: May be mapped to class-based operations
- Use actions in ports to define behavior



eExpired

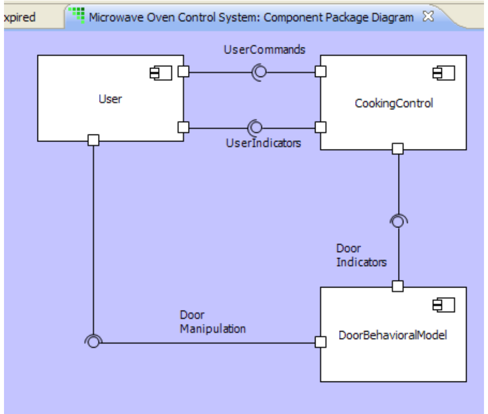
System Interfaces: Interface Package Diagram

«interface»
UserCommands
PlugIn():void
signals
AddTime()
Start()
Cancel()

«interface»
UserIndicators
signals
Beep()
LightTurnedOn()
LightTurnedOff()

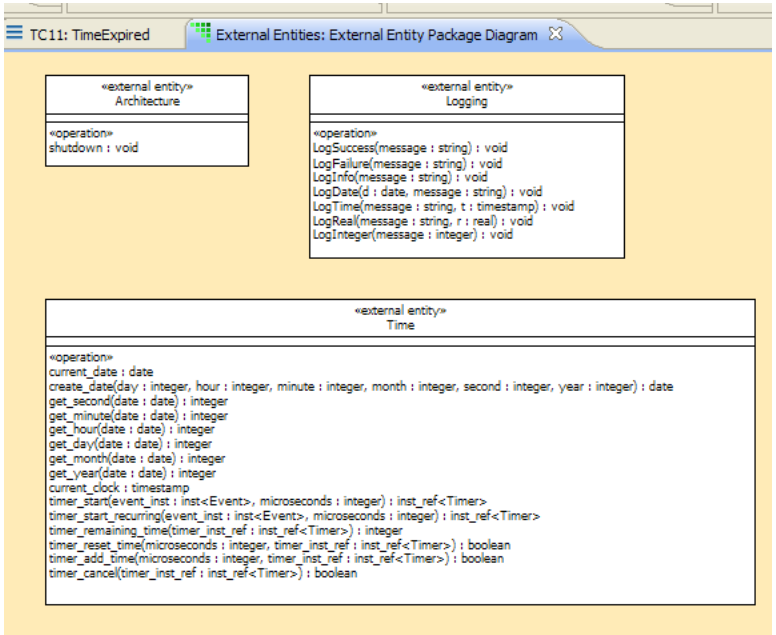
«interface»
Door Indicators
signals
DoorOpened()
DoorClosed()

«interface»
Door Manipulation
signals
OpenDoor()
CloseDoor()



Bridges

- Another form of synchronous operation
  - Takes parameters
  - Can be wired to external code or defined with OAL
- Use for library functions
  - Time
  - Logging
  - Math
- Use for scaffolding
  - OAL or Java for Verifier
  - Hand-written code for target



```
«external entity»
Architecture

«operation»
shutdown : void
```

```
«external entity»
Logging

«operation»
LogSuccess(message : string) : void
LogFailure(message : string) : void
LogInfo(message : string) : void
LogDate(d : date, message : string) : void
LogTime(message : string, t : timestamp) : void
LogReal(message : string, r : real) : void
LogInteger(message : integer) : void
```

```
«external entity»
Time

«operation»
current_date : date
create_date(day : integer, hour : integer, minute : integer, month : integer, second : integer, year : integer) : date
get_second(date : date) : integer
get_minute(date : date) : integer
get_hour(date : date) : integer
get_day(date : date) : integer
get_month(date : date) : integer
get_year(date : date) : integer
current_clock : timestamp
timer_start(event_inst : inst<Event>, microseconds : integer) : inst_ref<Timer>
timer_start_recurring(event_inst : inst<Event>, microseconds : integer) : inst_ref<Timer>
timer_remaining_time(timer_inst_ref : inst_ref<Timer>) : integer
timer_reset_time(microseconds : integer, timer_inst_ref : inst_ref<Timer>) : boolean
timer_add_time(microseconds : integer, timer_inst_ref : inst_ref<Timer>) : boolean
timer_cancel(timer_inst_ref : inst_ref<Timer>) : boolean
```

Summary

- **These rules represent a 'contract' between the world of analysis and the world of implementation**

- **The architecture undertakes to implement dynamic behavior according to the agreed semantics.**