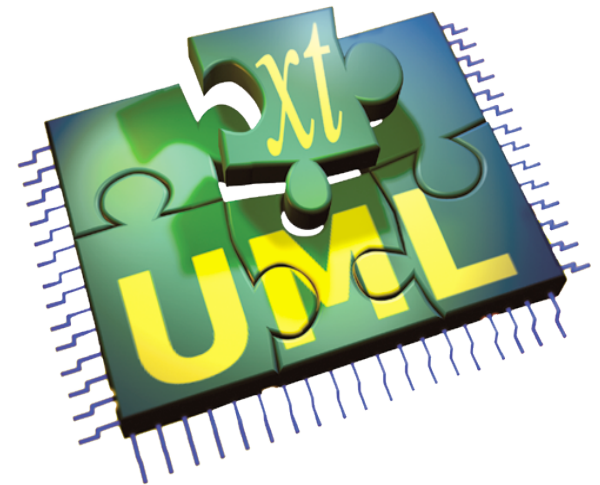


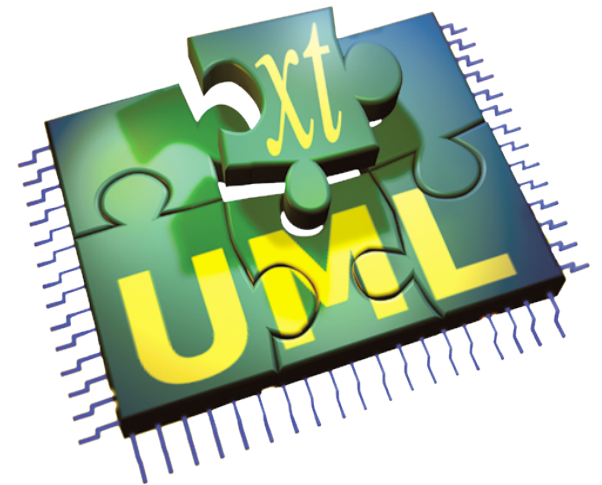
Verification– Step 3 in the xtUML Method

- ◆ **Analysis** – questioning, thinking, sketching...
 - Informal UML diagrams
 - use case, sequence, ...
- ◆ **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



xtUML Model Driven Test

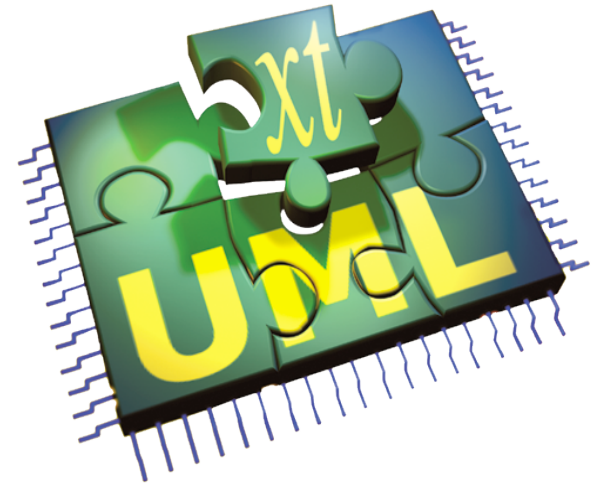
Model Driven Test delivers verified models to the next level



xtUML Model Driven Test

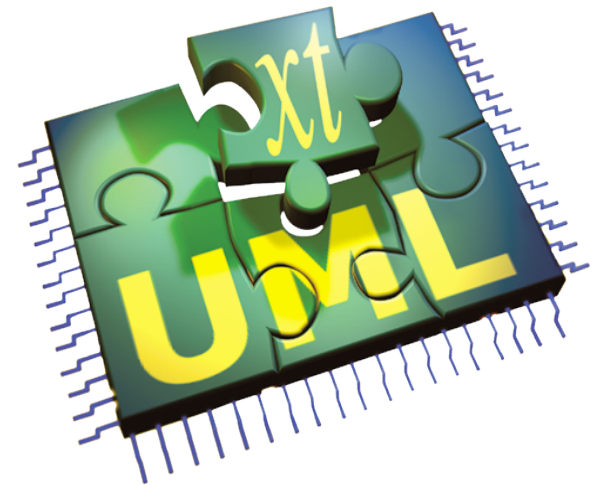
Abstraction accelerates verification

- Models have lower complexity
- Fewer requirements to check
- Faster test execution
- Easier to write tests
- Lower Model decay



xtUML Model Driven Test

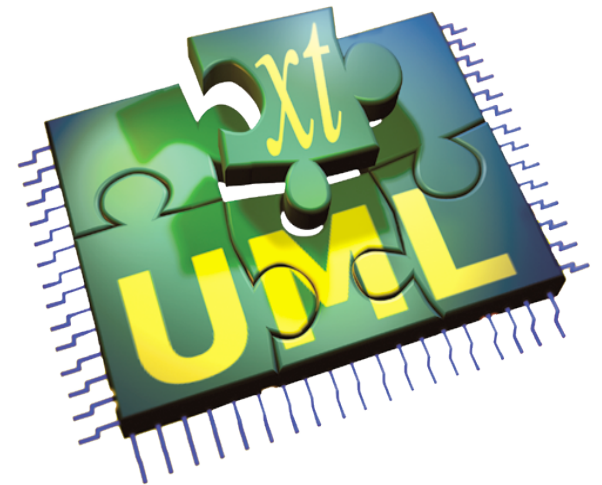
**Translation provides traceable path to
“Golden” Reference for architecture-
specific and technology-specific models**



xtUML Model Driven Test

MDT offers Test Driven Design benefits

- Higher re-use with test component
- Lower system model debris
- Eliminates situation of design waiting for test



Validation vs Verification

Validation

- **Confirming that the system performs intended function under Intended Usage**
- **Often emphasized by Modeling Analysis**

Verification

- **Confirming that the system is validated and does not perform “Bad Things” under Unintended Usage**

Validation vs Verification

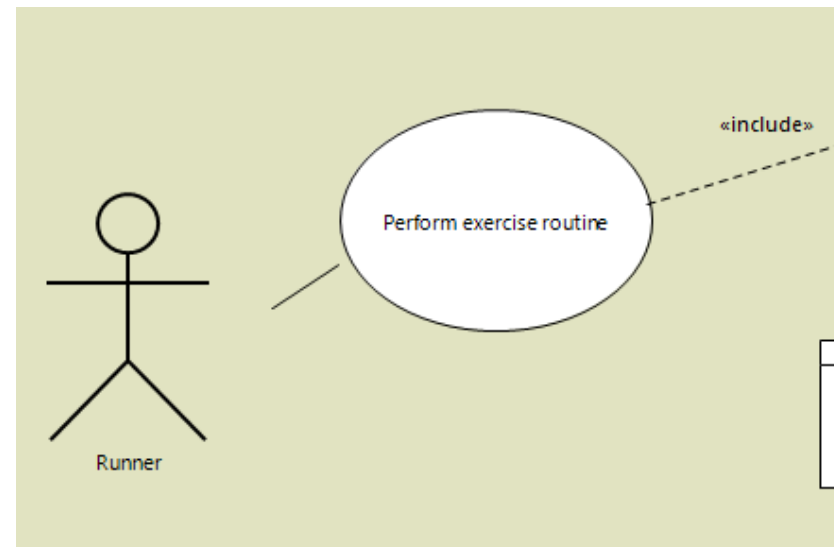
xtMDT requires Verification

xtMDT Verification

xtMDT starts with Analysis

xtMDT Analysis - Use Cases

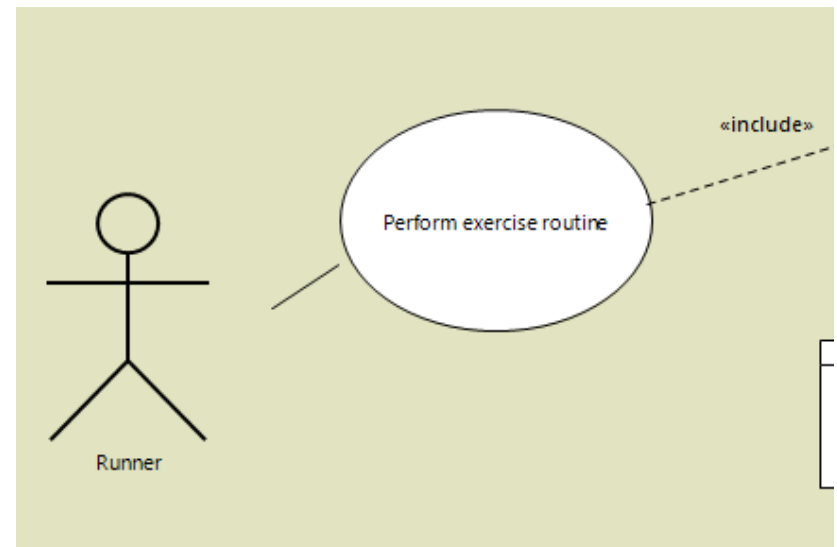
Test Environment models Actors



xtMDT Analysis - Use Cases

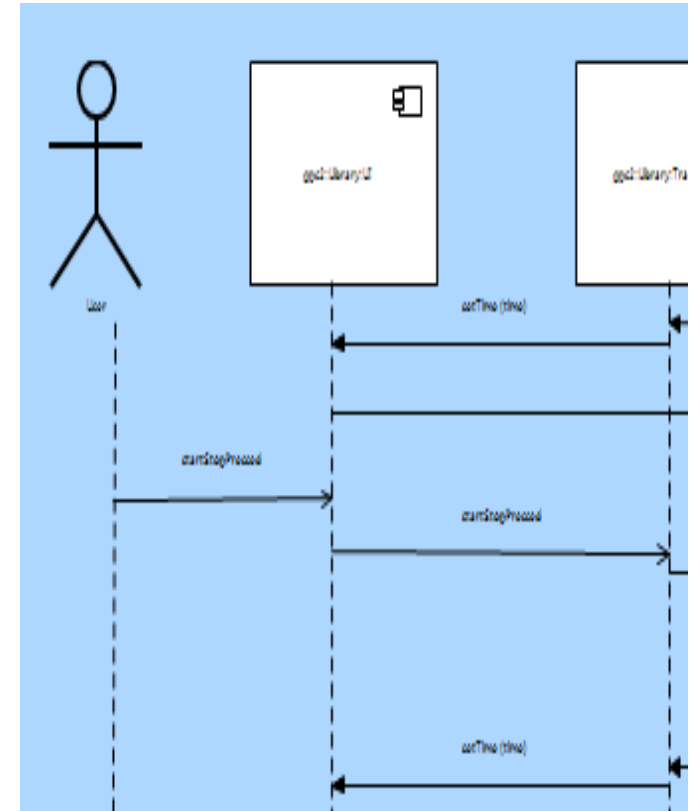
Use Cases must

- Capture desired functionality
- Constrain undesired functionality



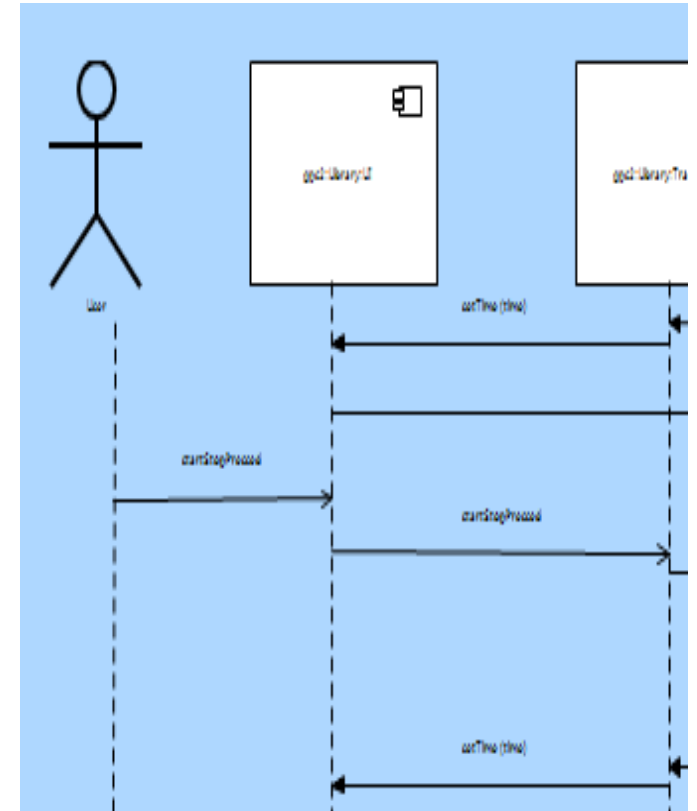
xtMDT Analysis - Sequence Diagrams

- ◆ Interactions between External Actors and the system model are good test candidates



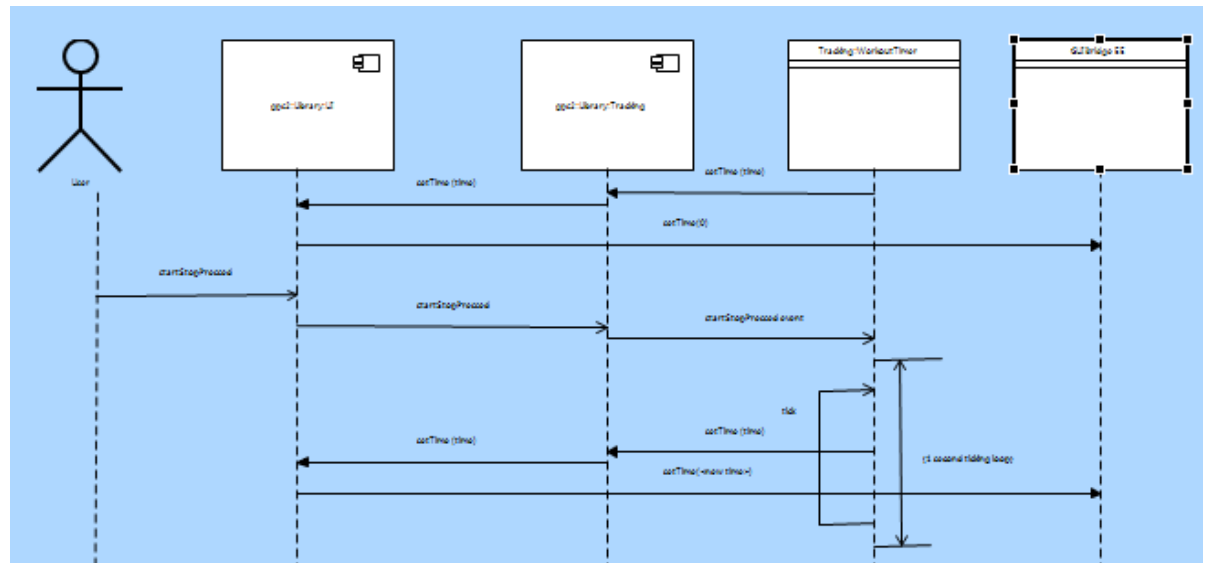
xtMDT Analysis - Sequence Diagrams

- ◆ Analysis should include both intended and unexpected (ie. valid and invalid) messages
- ◆ Pre-Conditions define constraints
- ◆ Post-Conditions define “Golden” comparison



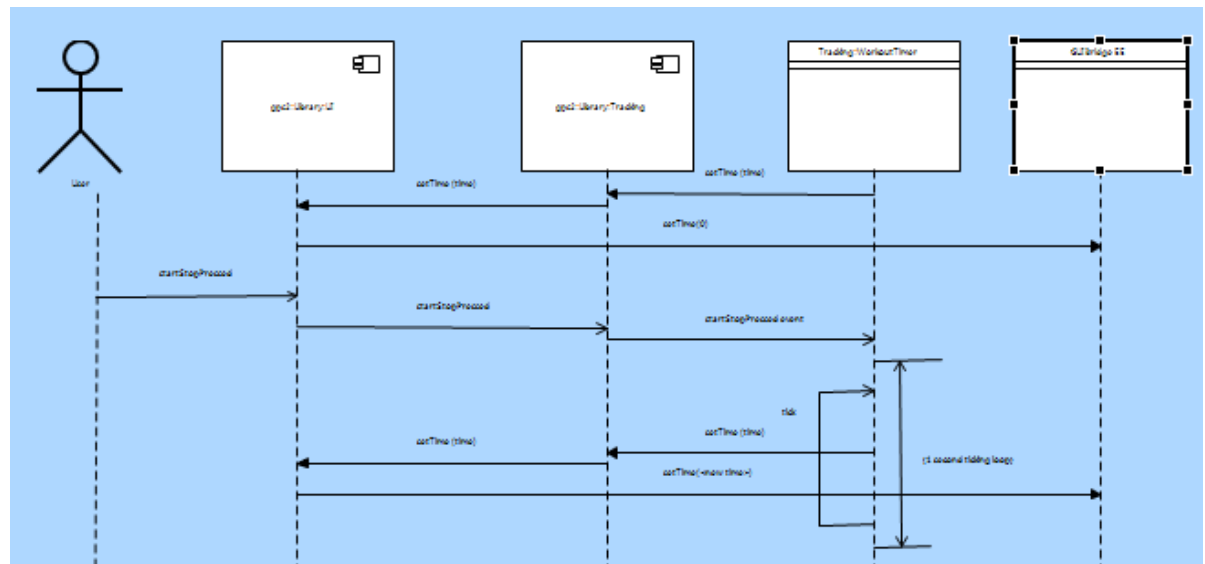
xtMDT Analysis - Scenarios

- ◆ **Scenarios are Sequence Diagrams with messages from the Actors constrained to give only a unique set of messages from the system.**



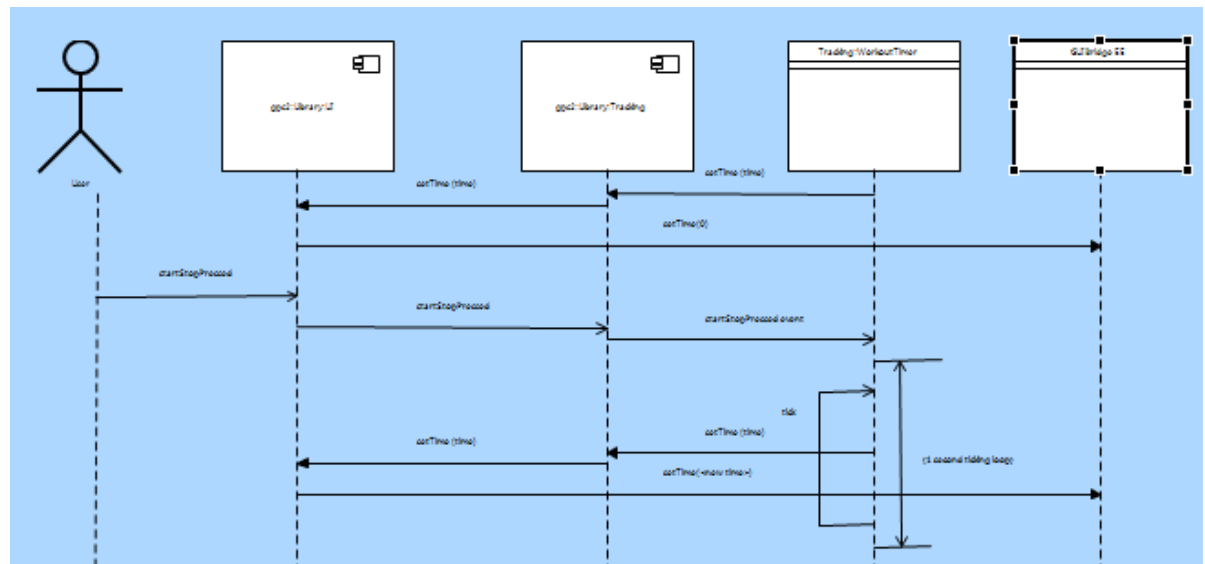
xtMDT Analysis - Scenarios

- ◆ Scenarios provide “Golden Data” to compare model generated messages against.



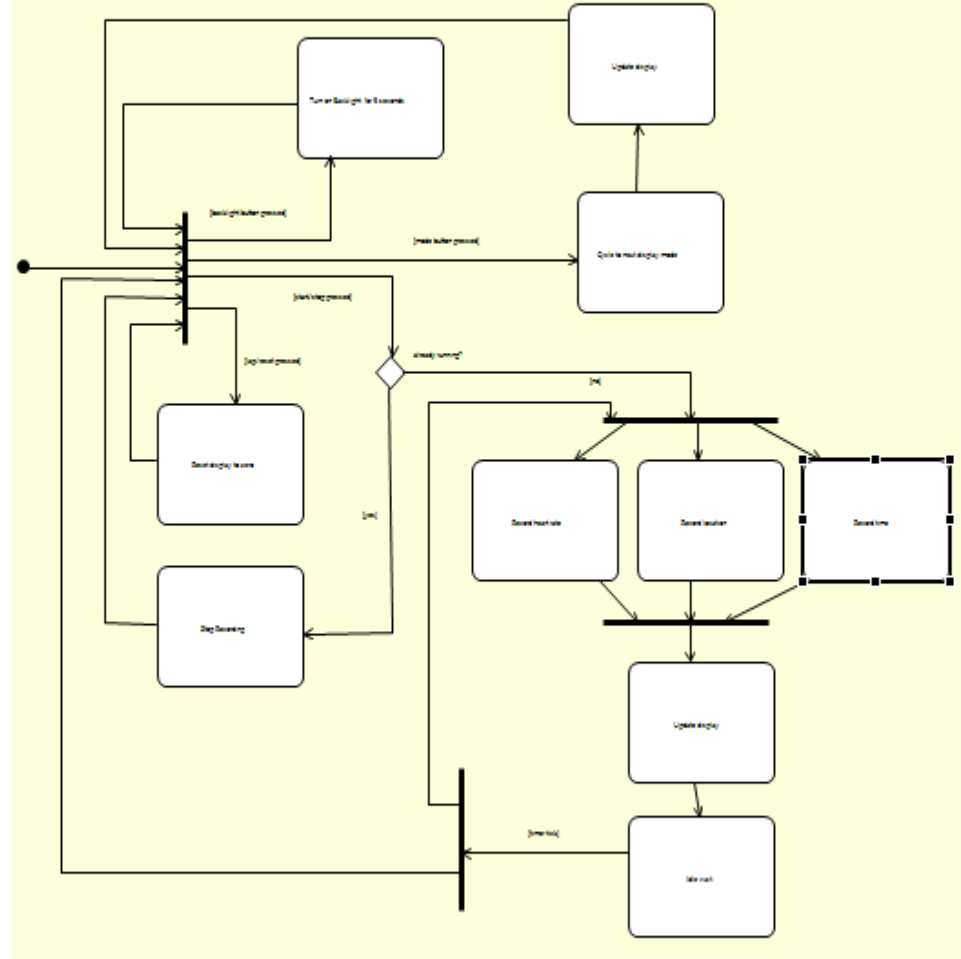
xtMDT Analysis - Scenarios

- ◆ Scenario = initial state + sequence trajectory
- ◆ Caution – emphasizing Scenarios in analysis can give false sense of completeness.



xtMDT Analysis - Activity Diagrams

- ◆ Activity Diagrams show the flow of processing interactions
- ◆ Capture “Behavior / Habits” of an Actor



Mapping Analysis to Tests

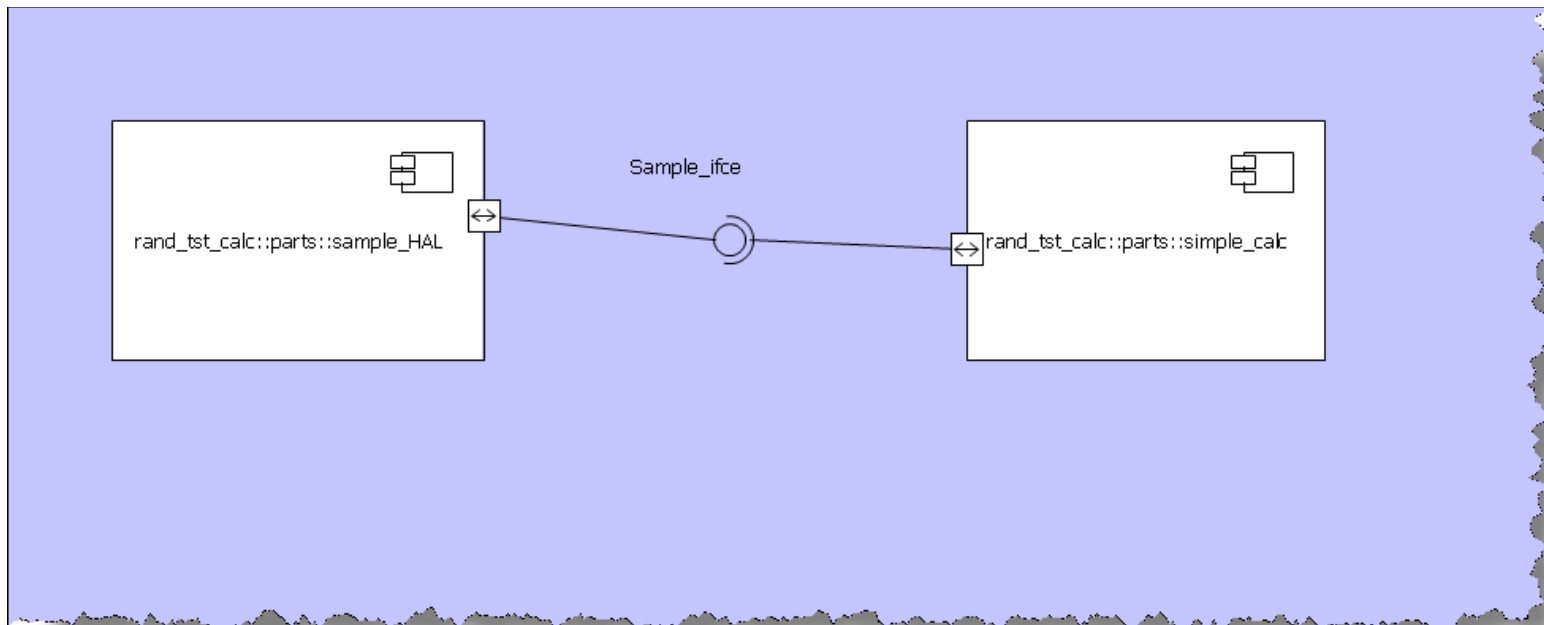
- ◆ **Use Case Diagrams**
 - **Identify components and message categories for test system**
- ◆ **Sequence and Scenario Diagrams**
 - **Identify order of key behavior**
 - **Identify boundary constraints and correctness**
- ◆ **Activity Diagrams**
 - **Identify classes and messages related to common “Behavior”**

Mapping Analysis to Tests

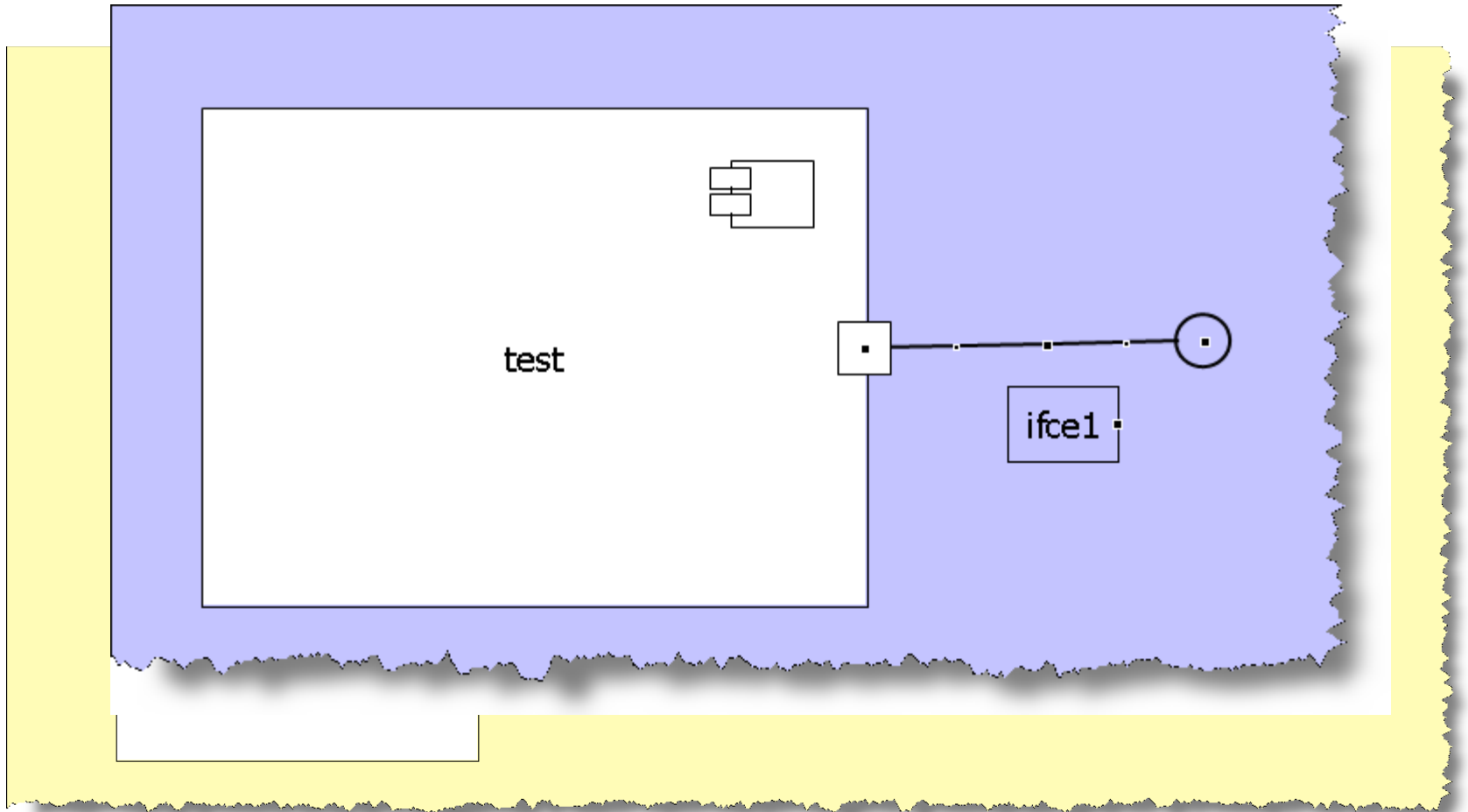
Scenarios + Activity Diagrams provide a prototype of a test component.

Suggested System Structure

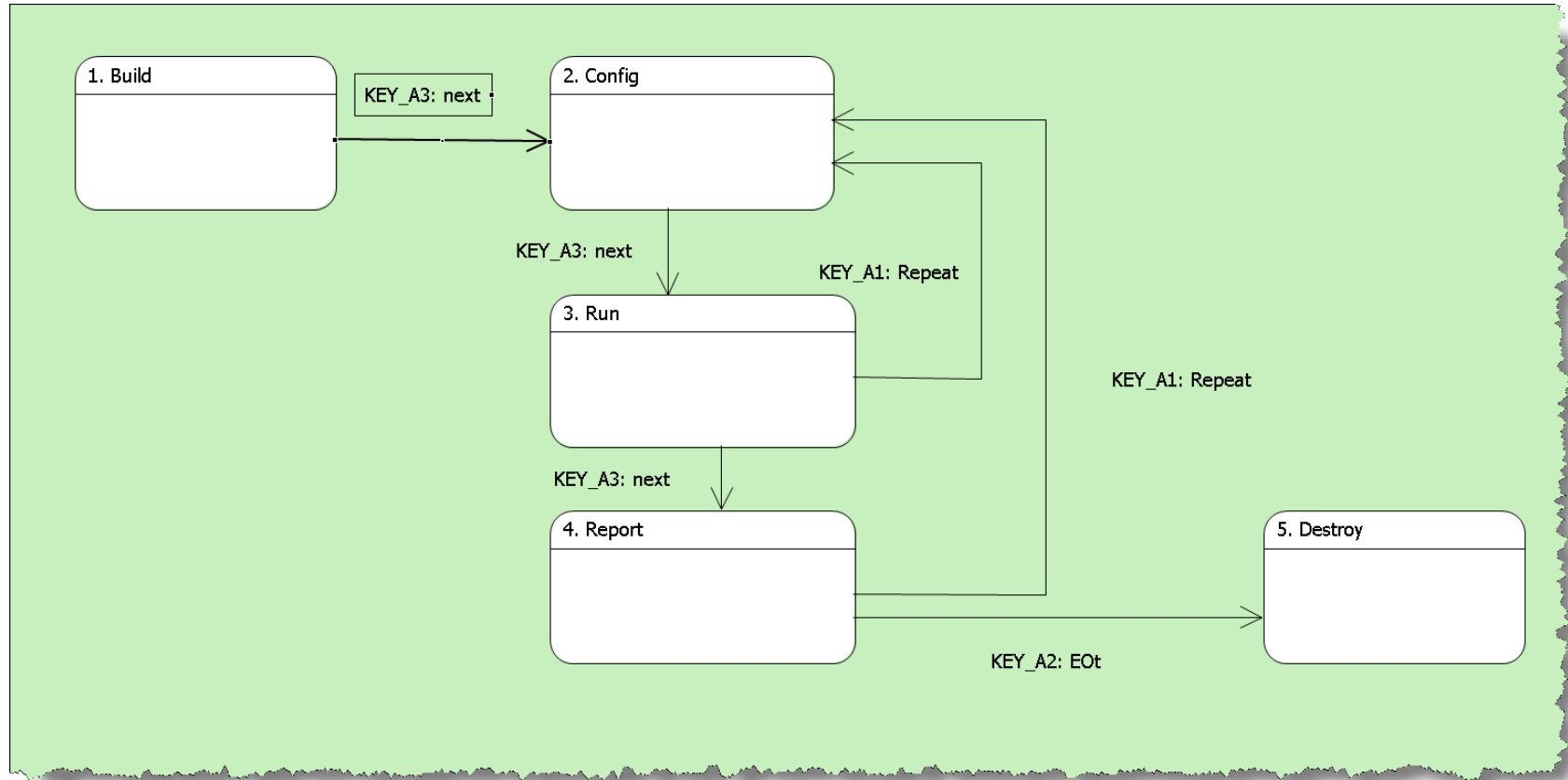
**Most Commonly split into
Test Component and System Component**



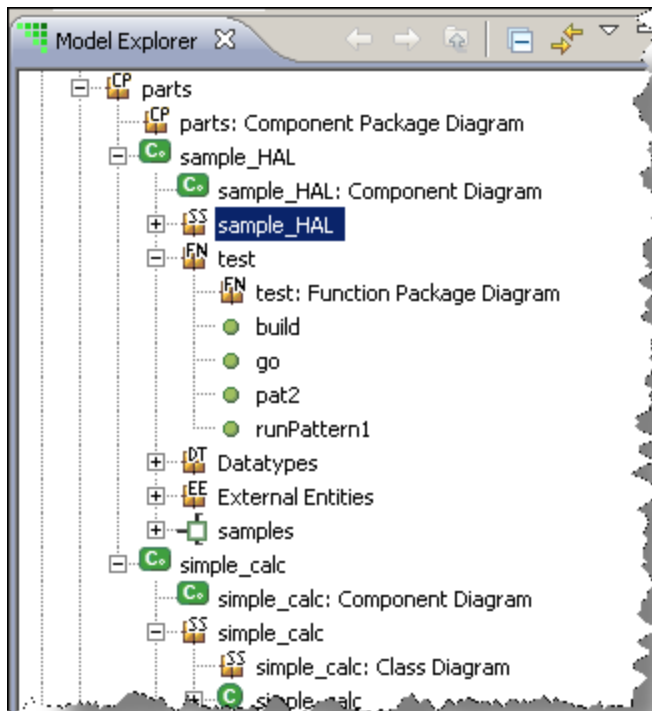
Structure of a Test



xtMDT– Test Component Phases

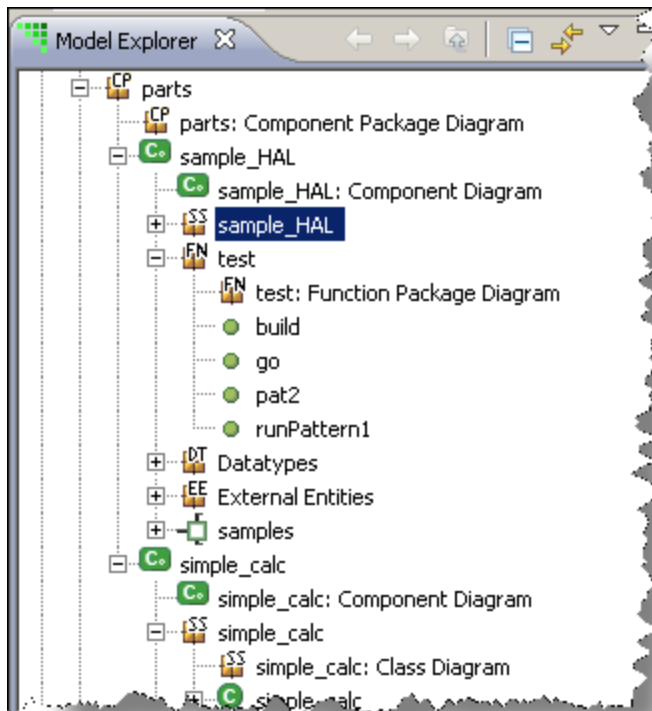


xtMDT– Test Component Phases



```
pat2: Function Activity X
::build();
select any hal from instances of HAL;
hal.lc1_step = 3;
hal.test_cnt = 2;
::go();
```

xtMDT– Test Component Phases



```
xtUML Modeling xtUML Debugging EDGE Projects
build: Function Acti EOT: State Machine S shutdown: Bridge Act

// Test Banner
LOG::LogInfo (message:"Test - init for OAL HAL");

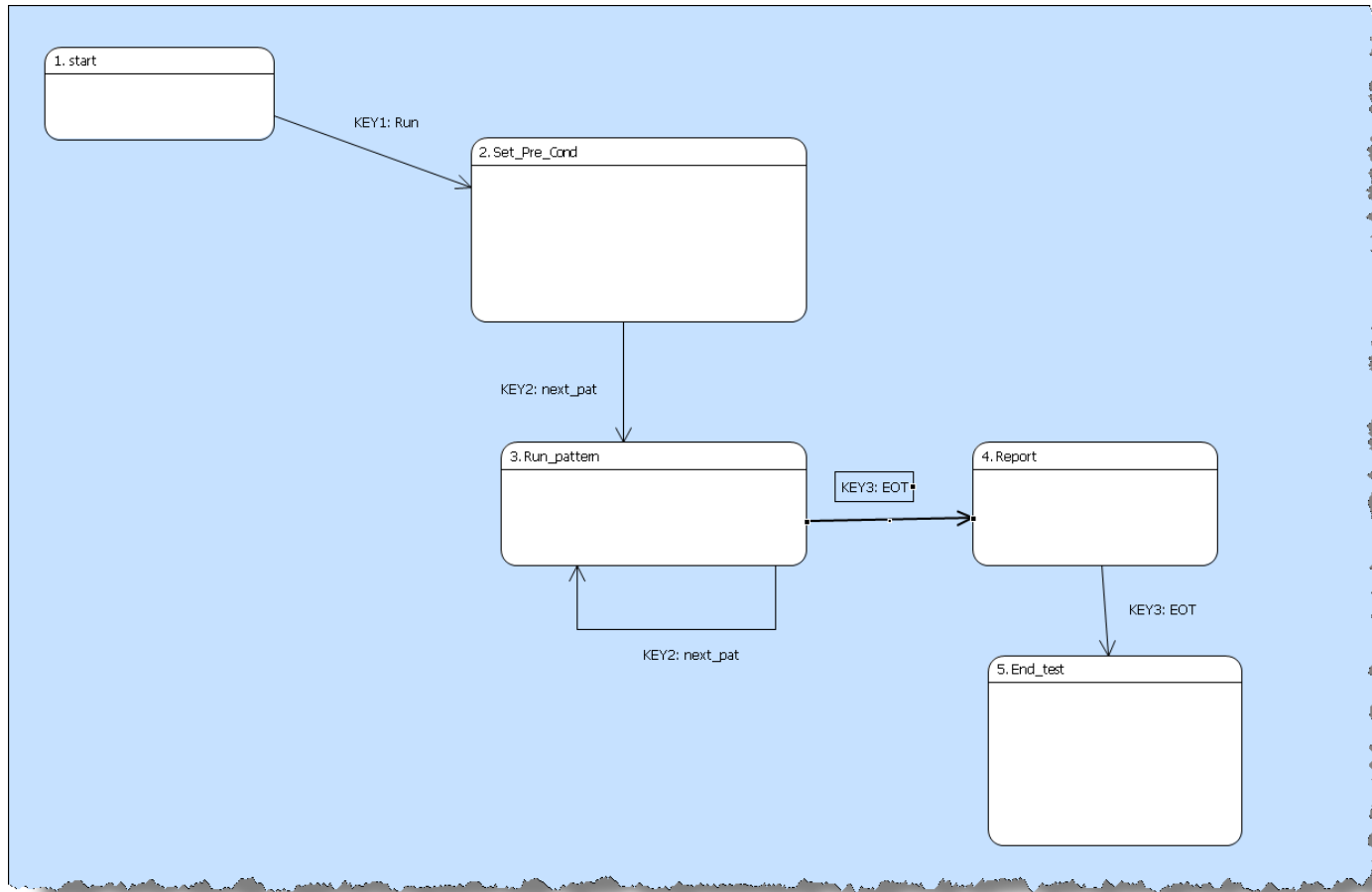
select any hal from instances of HAL;
if (empty hal)
    create object instance hal of HAL;
end if;

send samples::init();
```

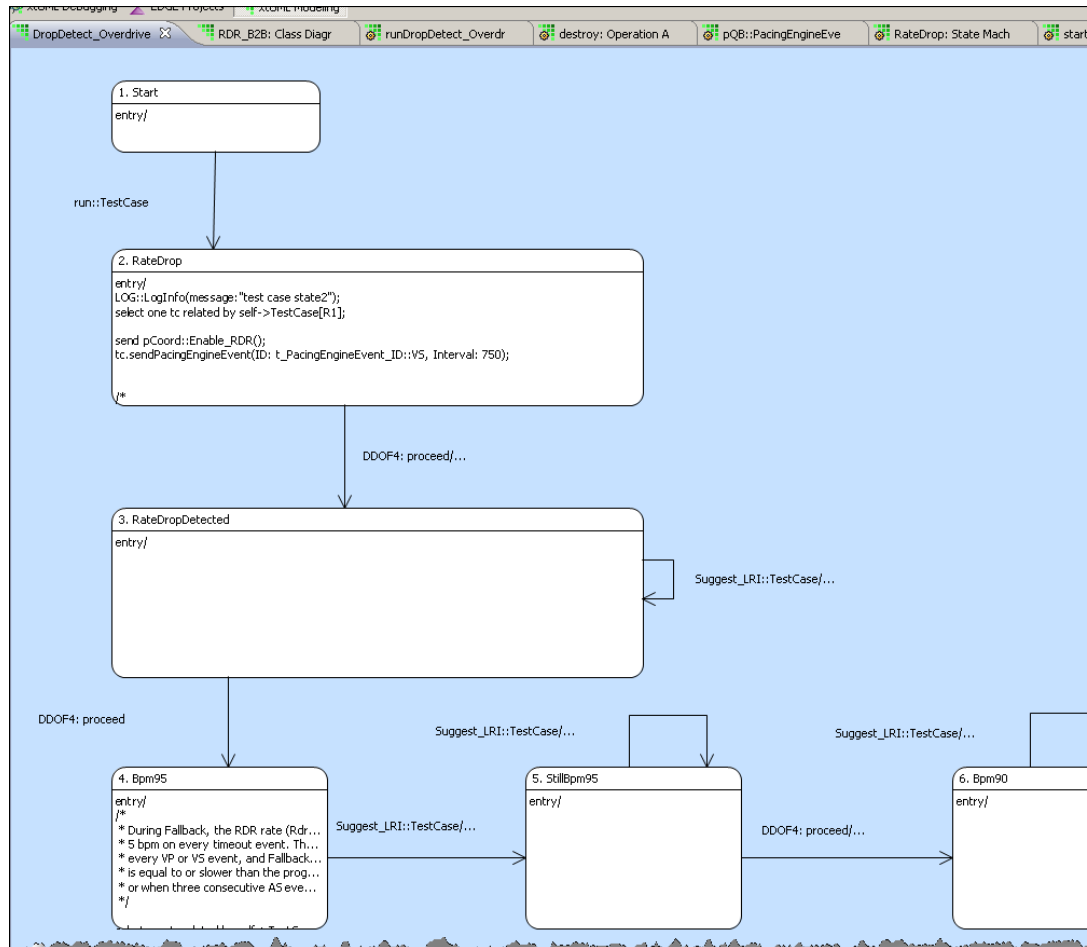
```
pat2: Function Activity build: Function Activity init: Required

select any calc from instances of SIMPLE_CALC;
if (empty calc)
    create object instance calc of SIMPLE_CALC;
end if ;
```

xtMDT– Test Phases



xtMDT Example from Application Model

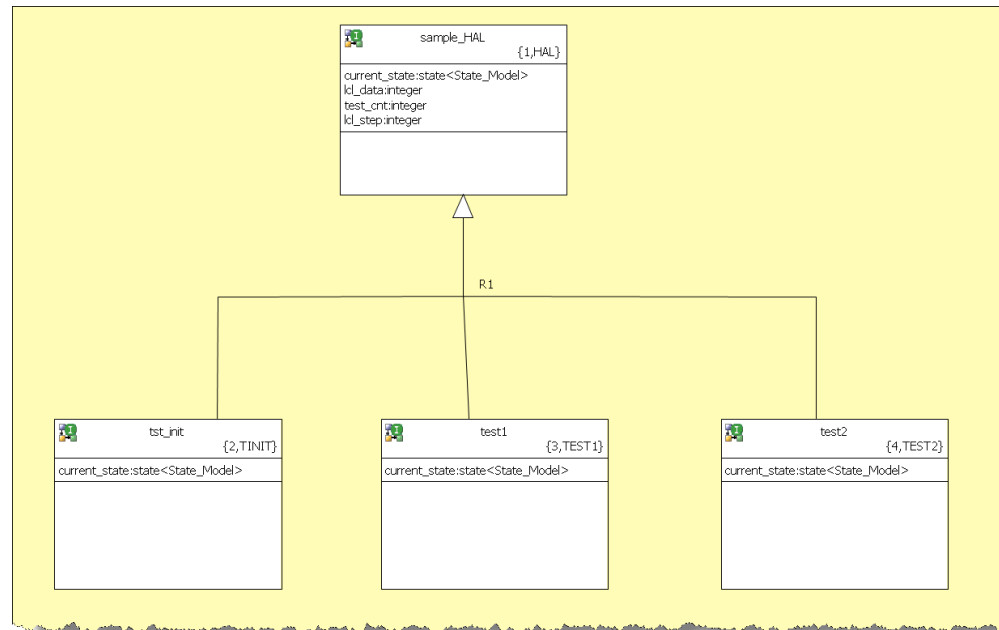


xtMDT Summary

- ◆ **Benefits of Platform-Independence applies to test**
- ◆ **Abstraction reduces complexity and simplifies test writing**
- ◆ **Start with xtUML Analysis artifacts**
- ◆ **Structure model and micro-sprints around component phases**
- ◆ **xtMDT eases transition to Advanced Verification**
- ◆ **Code generation enables easier path**

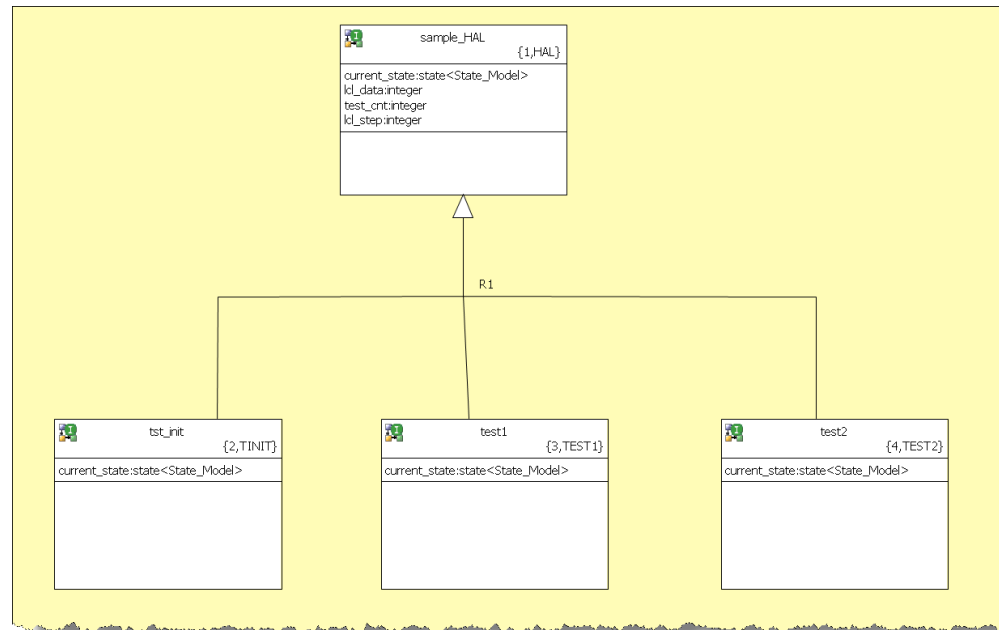
xtMDL – Advanced test Class

Use super-class and sub-class structure to generalize the test structure



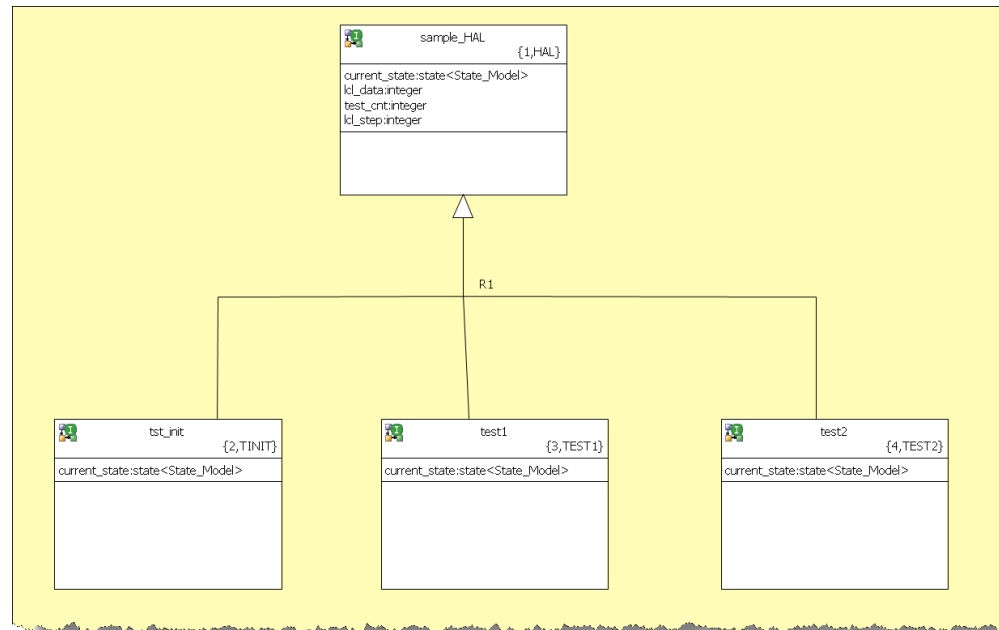
xtMDL – Advanced test Class

Super-class contains OAL to implement collection of parameterized sequences



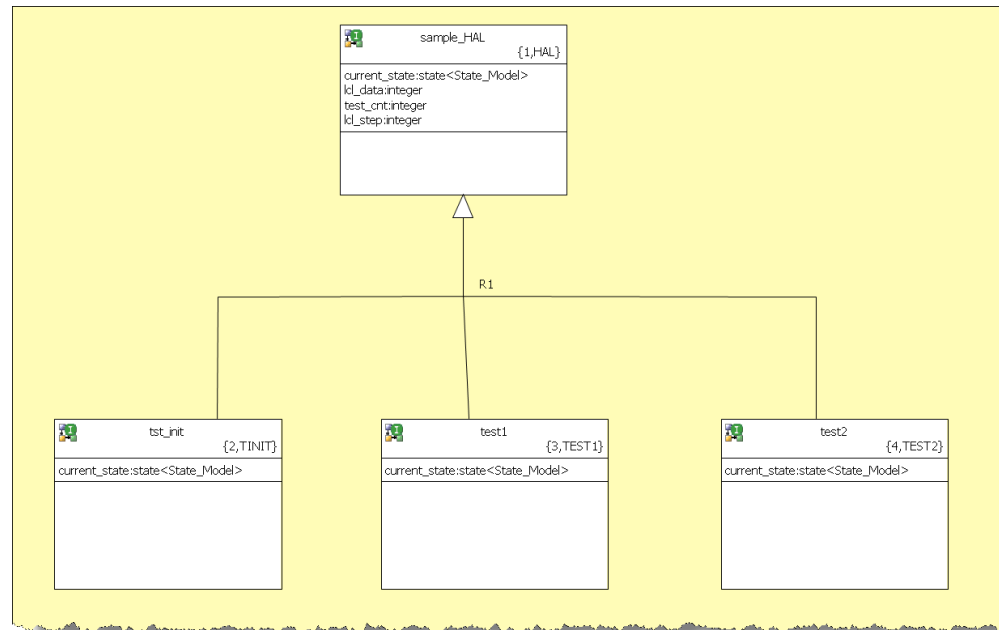
xtMDL – Advanced test Class

Sub-class contains configuration details and orders the sequences



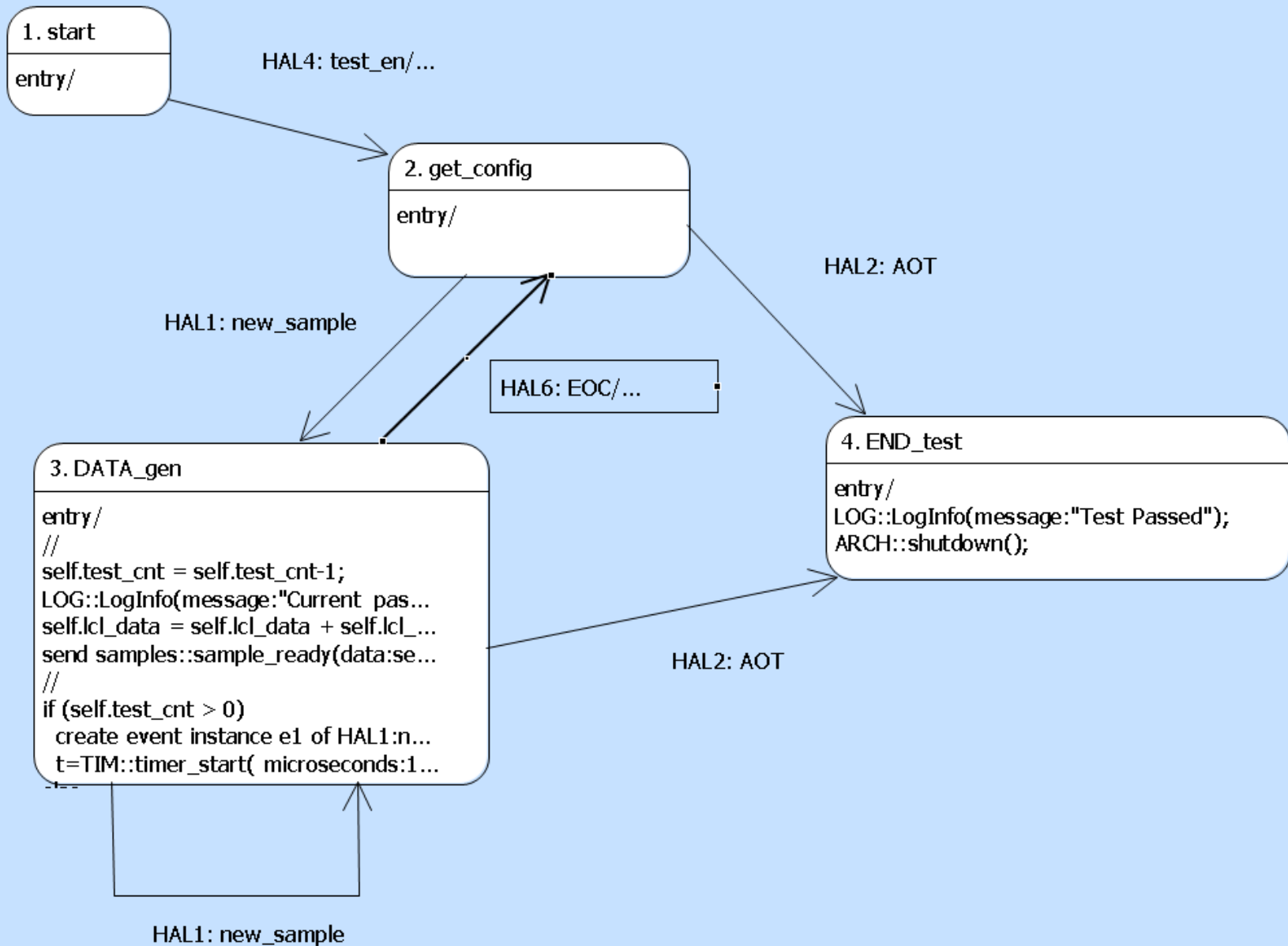
xtMDL – Advanced test Class

Randomization + Sequences = Compact Test Suite Description

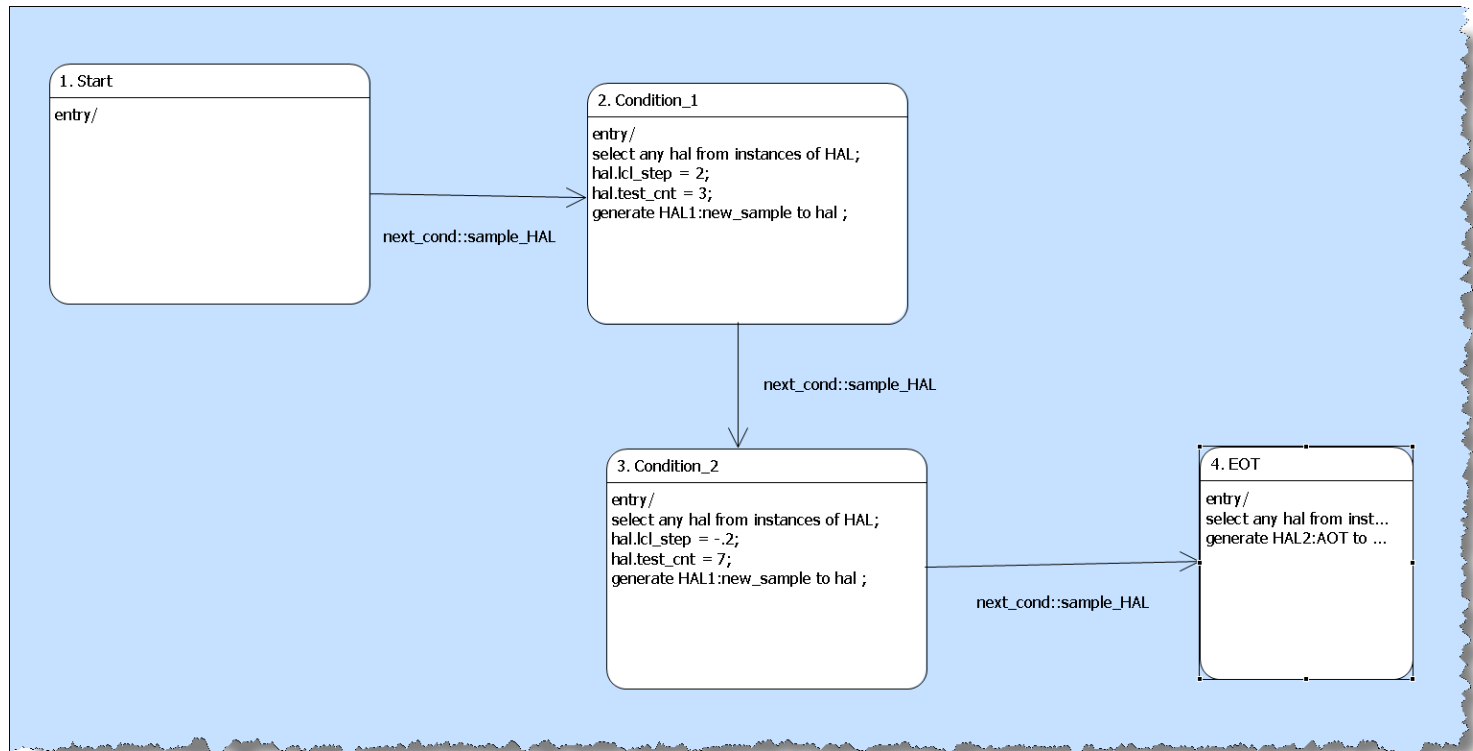


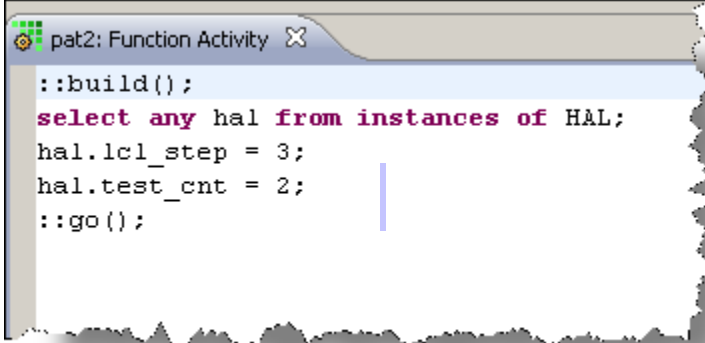
**Mentor
Graphics®**

www.mentor.com

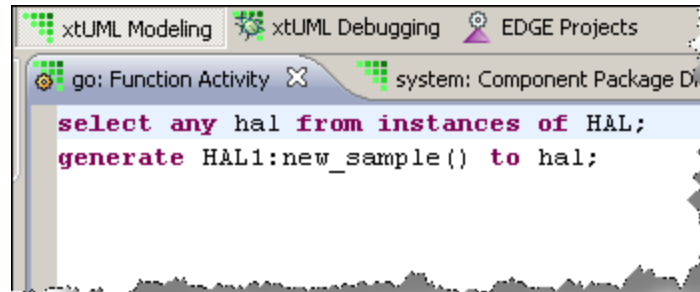


xtMDL - Test Classes





```
pat2: Function Activity X  
::build();  
select any hal from instances of HAL;  
hal.lcl_step = 3;  
hal.test_cnt = 2;  
::go();
```



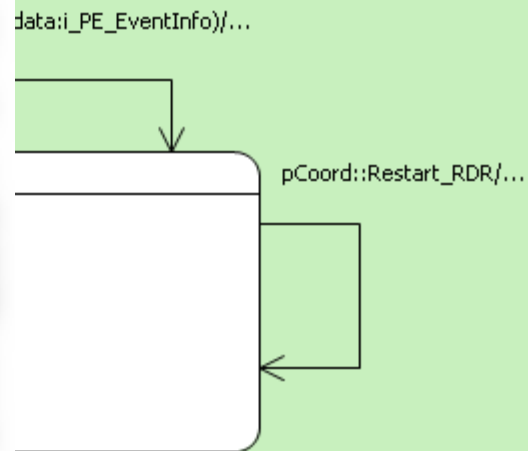
The image shows a screenshot of an IDE window with a tab titled "go: Function Activity". The window contains the following code snippet:

```
select any hal from instances of HAL;  
generate HAL1:new_sample() to hal;
```

```
xtUML Debugging | EDGE Projects | xtUML Modeling
pCoord::Enable_RDR i | RDR_B2B: Class Diagram | TestCases: Cla
select any rdr from instances of RdrB2B;
if (empty rdr)
  create object instance rdr of RdrB2B;
end if;

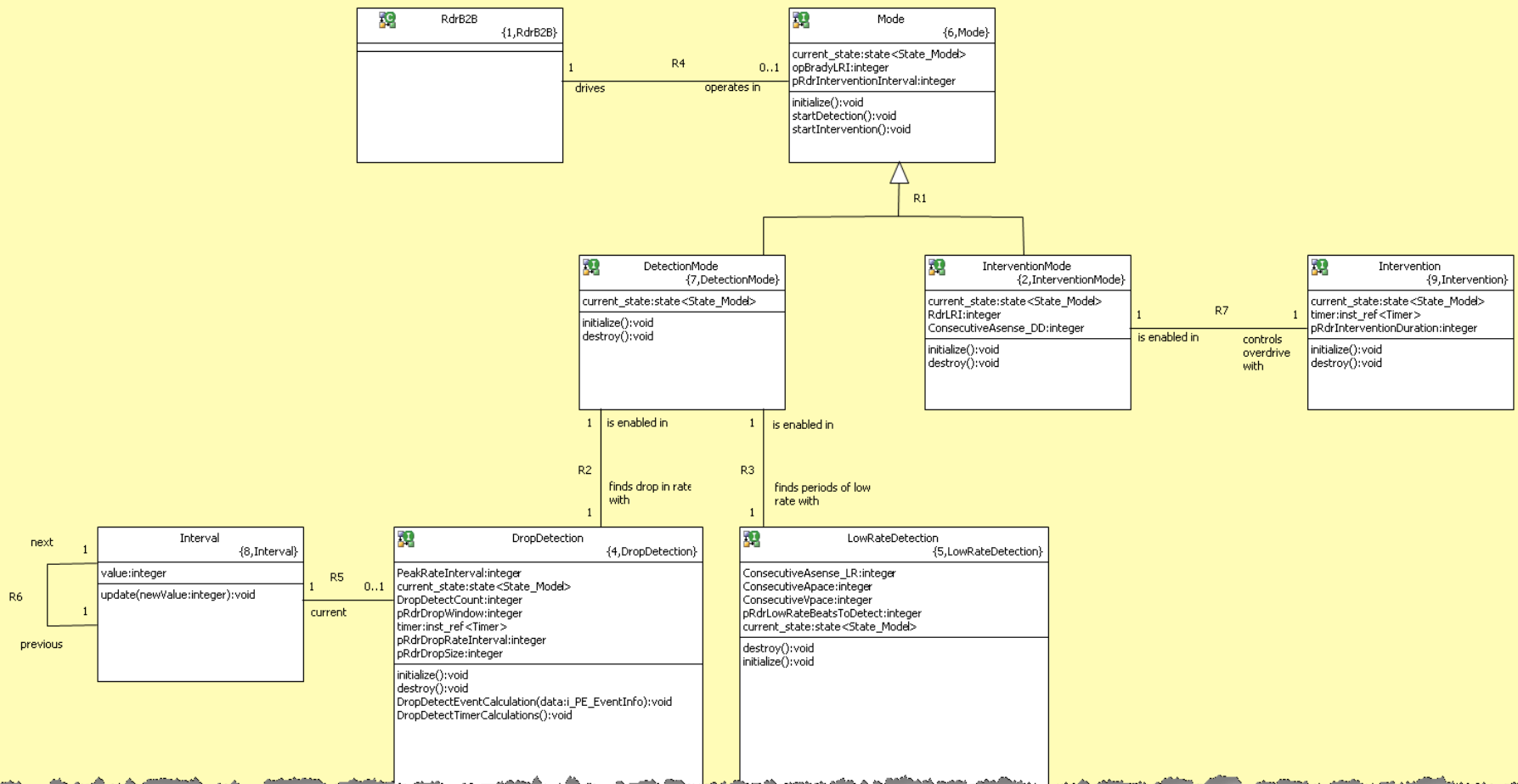
1
er
//
create object instance mode of Mode;
relate rdr to mode across R4;
mode.initialize();

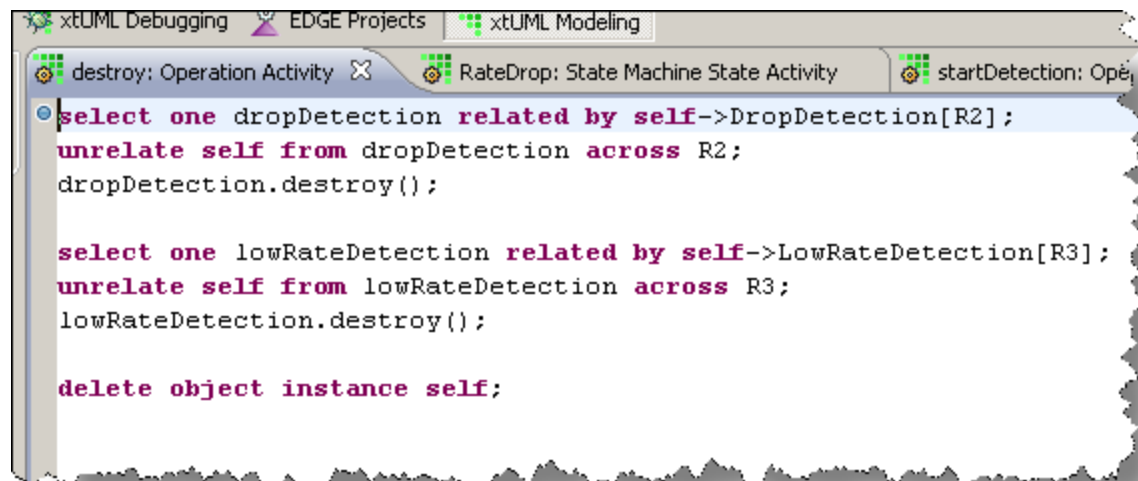
send pDM::get_RDR_Params();
```



```
xtUML Debugging | EDGE Projects | xtUML Modeling
initialize: Operatio | RDR_B2B: Class Diagram | TestCases: Class Dia | runDropDetect
create object instance dropDetection of DropDetection;
relate self to dropDetection across R2;
dropDetection.initialize();

create object instance lowRateDetection of LowRateDetection;
relate self to lowRateDetection across R3;
lowRateDetection.initialize();
```





The screenshot shows the xtUML Modeling application interface. The title bar includes 'xtUML Debugging', 'EDGE Projects', and 'xtUML Modeling'. Below the title bar, there are three tabs: 'destroy: Operation Activity', 'RateDrop: State Machine State Activity', and 'startDetection: Opé'. The main editor area displays the following code:

```
select one dropDetection related by self->DropDetection[R2];  
unrelate self from dropDetection across R2;  
dropDetection.destroy();  
  
select one lowRateDetection related by self->LowRateDetection[R3];  
unrelate self from lowRateDetection across R3;  
lowRateDetection.destroy();  
  
delete object instance self;
```

