SD03

Model Driven Development using

Rhapsody for Embedded Systems

Mark Richardson

Lead Application Engineer,

Telelogic an IBM Company

mark.richardson@uk.ibm.com

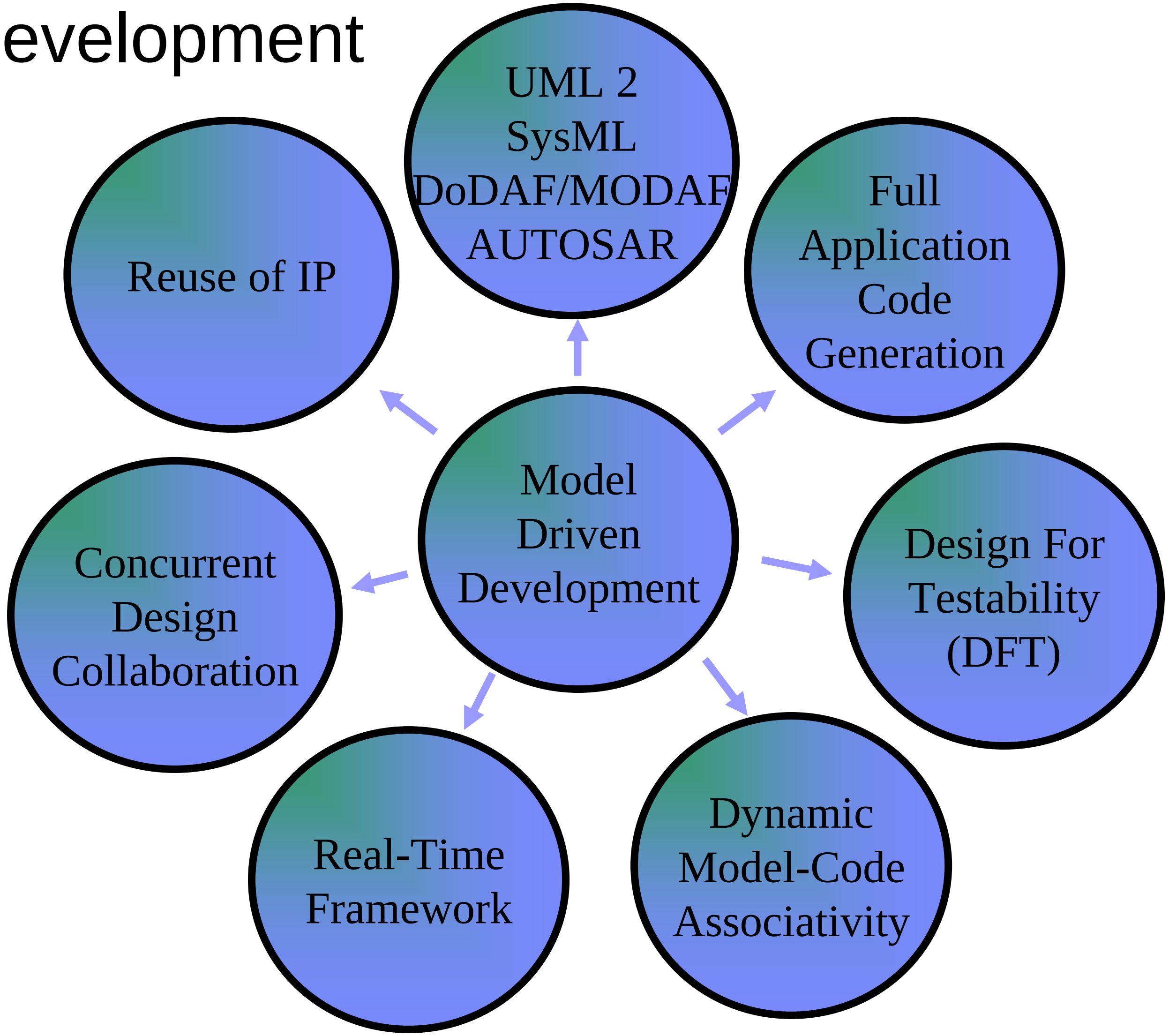
IBM Rational Software Development Conference 2008

WHERE TEAMS ARE R-HEROES

IBM

Rational. software

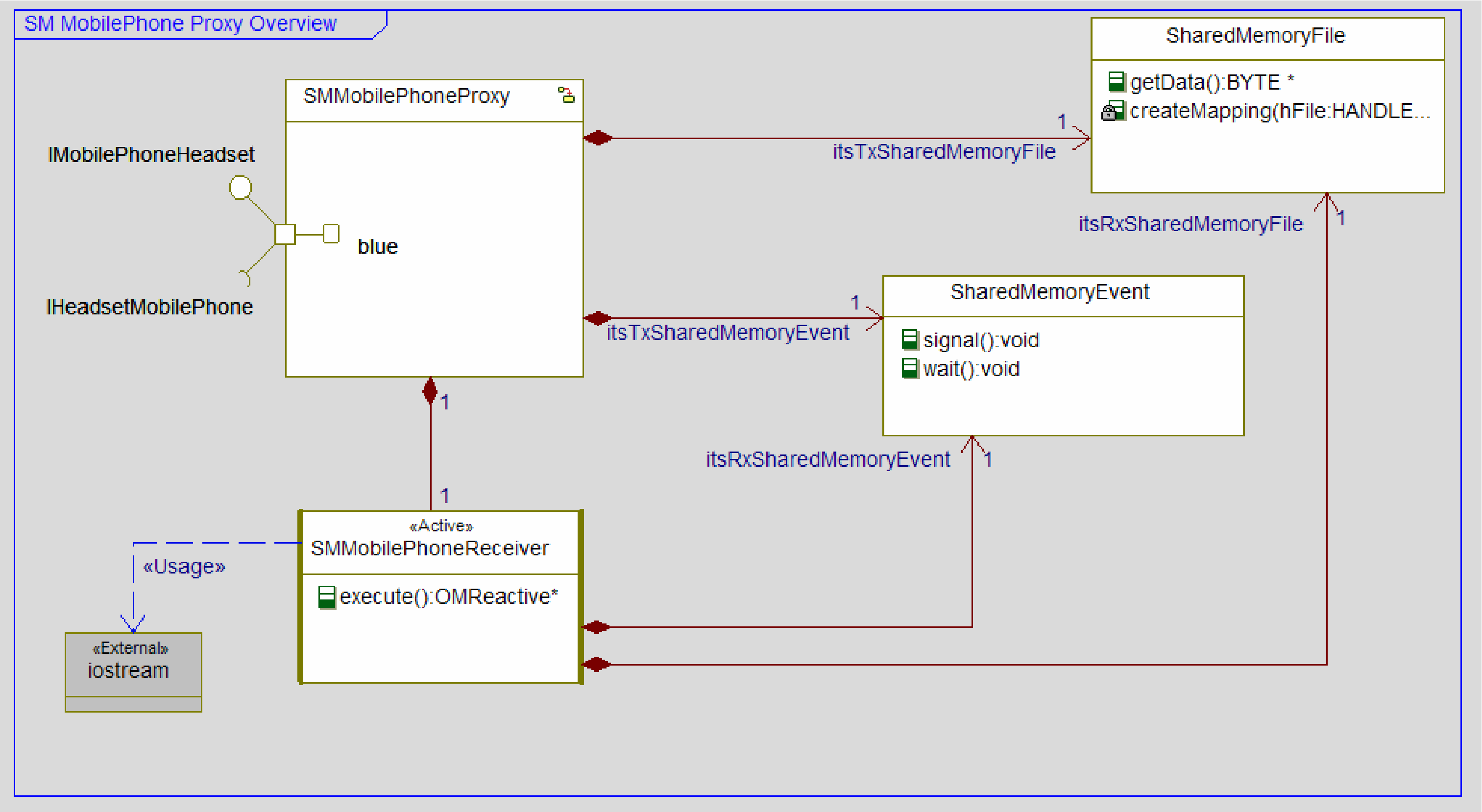
Model Driven Development



UML 2
SysML
DoDAF/MODAF
AUTOSAR
Reuse of IP
Full Application Code Generation
Model Driven Development
Concurrent Design Collaboration
Design For Testability (DFT)
Real-Time Framework
Dynamic Model-Code Associativity

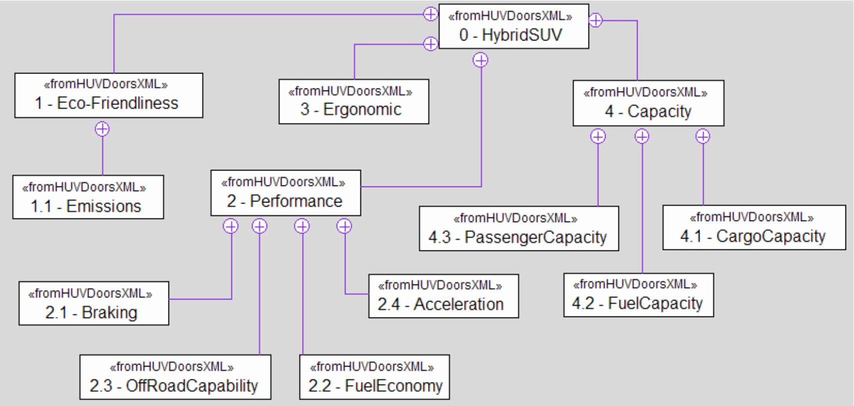
UML2

Rhapsody is the leading UML 2 compliant solution for embedded systems



SysML

- SysML is a domain customization of UML 2 for systems engineers
  - Supports the standard proposal in its latest form (V1.0)
- Support for SysML views
  - Requirements: Requirements diagram; Use case diagram
  - Structure: Block Definition diagram; Internal Block diagram
  - Behavior: Statechart; Activity diagram; Sequence diagram
  - Constraints: Parametric diagram
- *Uniquely Integrated Requirements and Design modeling environment*
- More than just modeling…
  - Simulation of SysML models
  - System testing for SysML



«fromHUVDoorsXML» 0 - HybridSUV

«fromHUVDoorsXML» 1 - Eco-Friendliness

«fromHUVDoorsXML» 3 - Ergonomic

«fromHUVDoorsXML» 4 - Capacity

«fromHUVDoorsXML» 1.1 - Emissions

«fromHUVDoorsXML» 2 - Performance

«fromHUVDoorsXML» 4.3 - PassengerCapacity

«fromHUVDoorsXML» 4.1 - CargoCapacity

«fromHUVDoorsXML» 2.1 - Braking

«fromHUVDoorsXML» 2.4 - Acceleration

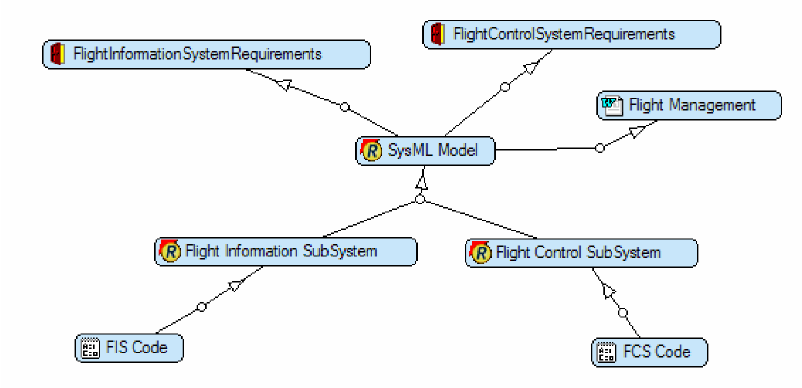
«fromHUVDoorsXML» 4.2 - FuelCapacity

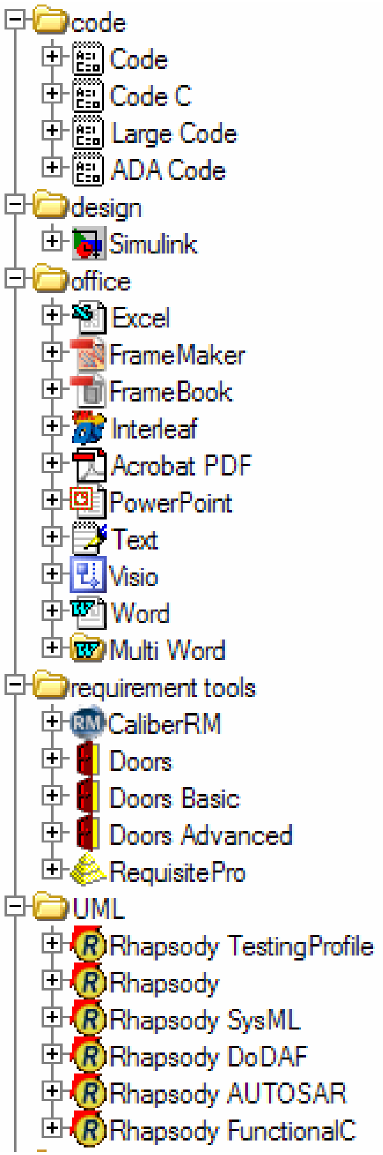
«fromHUVDoorsXML» 2.3 - OffRoadCapability

«fromHUVDoorsXML» 2.2 - FuelEconomy

Requirements Modelling

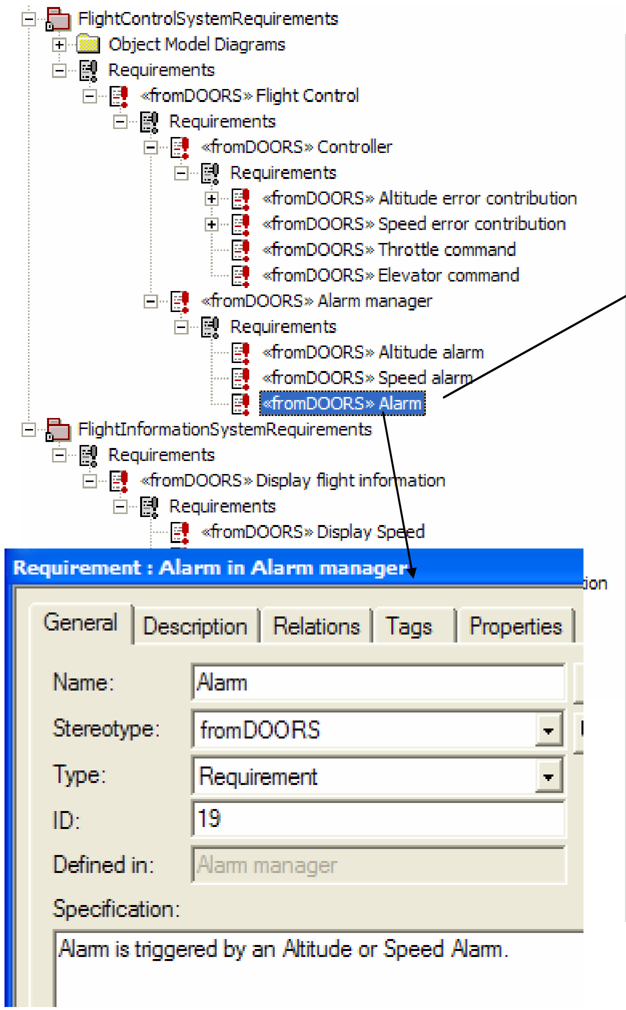
- Requirements Capture
- Requirements Traceability
  - Create traceability links from model to requirements
  - Automatic traceability documentation
- Requirements Analysis
  - Requirement Coverage Analysis
  - Change Impact analysis
  - Automatic report generation


FlightInformationSystemRequirements
FlightControlSystemRequirements
Flight Management
SysML Model
Flight Information SubSystem
Flight Control SubSystem
FIS Code
FCS Code



- code
  - Code
  - Code C
  - Large Code
  - ADA Code
- design
  - Simulink
- office
  - Excel
  - FrameMaker
  - FrameBook
  - Interleaf
  - Acrobat PDF
  - PowerPoint
  - Text
  - Visio
  - Word
  - Multi Word
- requirement tools
  - CaliberRM
  - Doors
  - Doors Basic
  - Doors Advanced
  - RequisitePro
- UML
  - Rhapsody TestingProfile
  - Rhapsody
  - Rhapsody SysML
  - Rhapsody DoDAF
  - Rhapsody AUTOSAR
  - Rhapsody FunctionalC

Requirements Capture and Trace

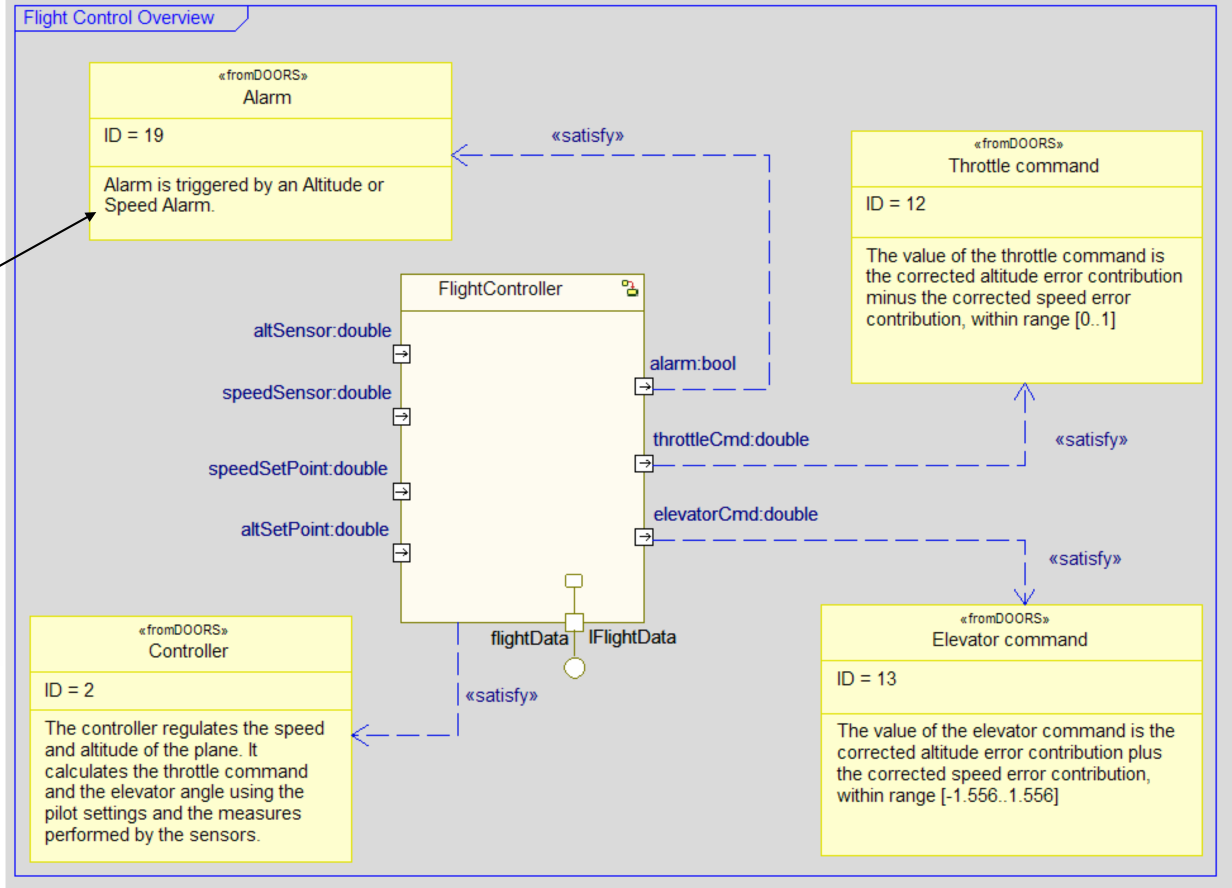

FlightControlSystemRequirements
Object Model Diagrams
Requirements
«fromDOORS» Flight Control
Requirements
«fromDOORS» Controller
Requirements
«fromDOORS» Altitude error contribution
«fromDOORS» Speed error contribution
«fromDOORS» Throttle command
«fromDOORS» Elevator command
«fromDOORS» Alarm manager
Requirements
«fromDOORS» Altitude alarm
«fromDOORS» Speed alarm
«fromDOORS» Alarm
FlightInformationSystemRequirements
Requirements
«fromDOORS» Display flight information
Requirements
«fromDOORS» Display Speed
ion
General
Description
Relations
Tags
Properties

Requirement : Alarm in Alarm manager

Name:	Alarm
Stereotype:	fromDOORS
Type:	Requirement
ID:	19
Defined in:	Alarm manager

Specification:

Alarm is triggered by an Altitude or Speed Alarm.



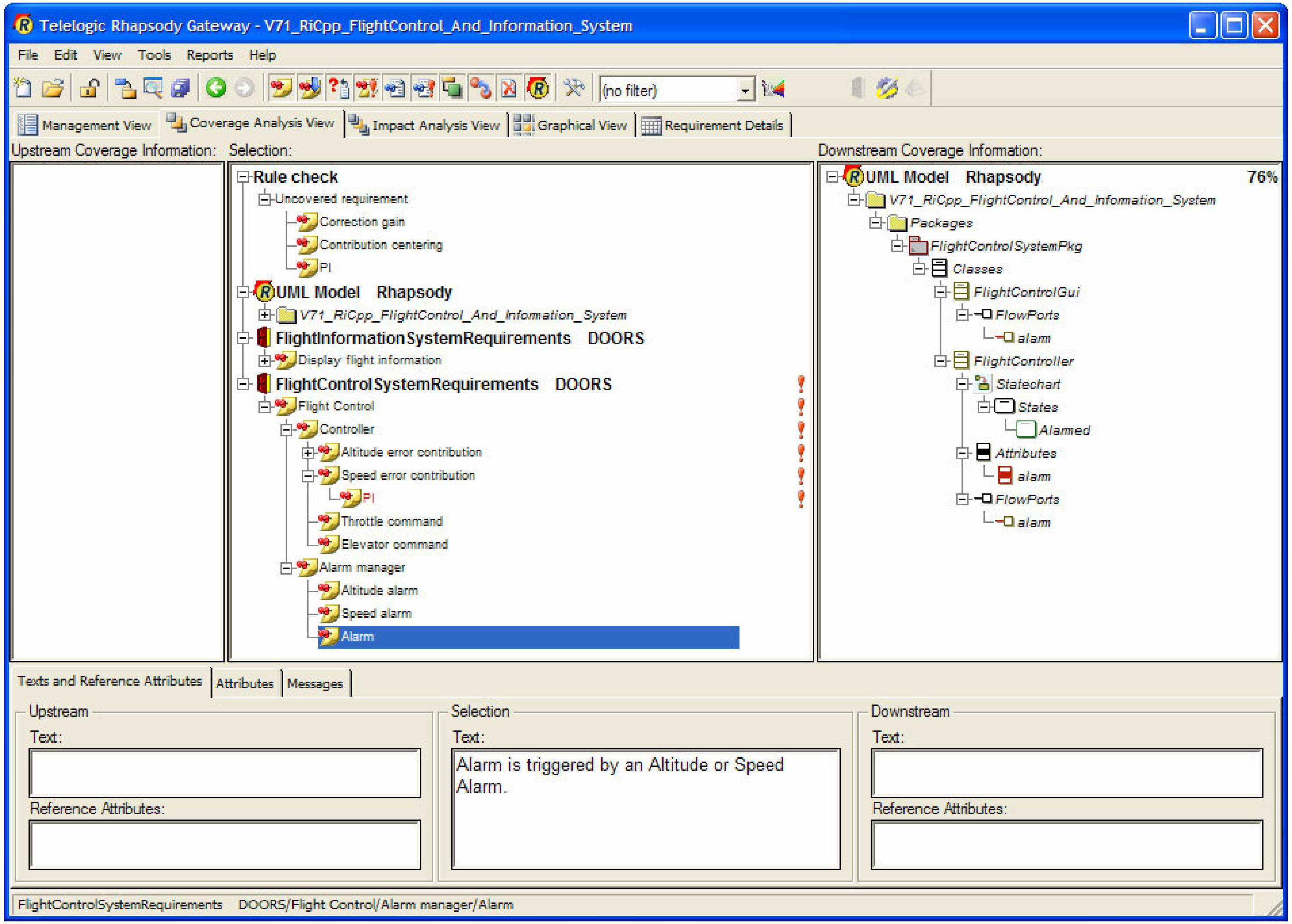
«fromDOORS»
Alarm
ID = 19
Alarm is triggered by an Altitude or Speed Alarm.

«fromDOORS»
Throttle command
ID = 12
The value of the throttle command is the corrected altitude error contribution minus the corrected speed error contribution, within range [0..1]

«fromDOORS»
Controller
ID = 2
The controller regulates the speed and altitude of the plane. It calculates the throttle command and the elevator angle using the pilot settings and the measures performed by the sensors.

«fromDOORS»
Elevator command
ID = 13
The value of the elevator command is the corrected altitude error contribution plus the corrected speed error contribution, within range [-1.556..1.556]

Requirements Coverage Analysis



Selection:

```
Rule check
  Uncovered requirement
    Correction gain
    Contribution centering
    PI
UML Model   Rhapsody
  V71_RiCpp_FlightControl_And_Information_System
FlightInformationSystemRequirements   DOORS
  Display flight information
FlightControlSystemRequirements   DOORS
  Flight Control
    Controller
      Altitude error contribution
      Speed error contribution
        PI
      Throttle command
      Elevator command
    Alarm manager
      Altitude alarm
      Speed alarm
      Alarm
```

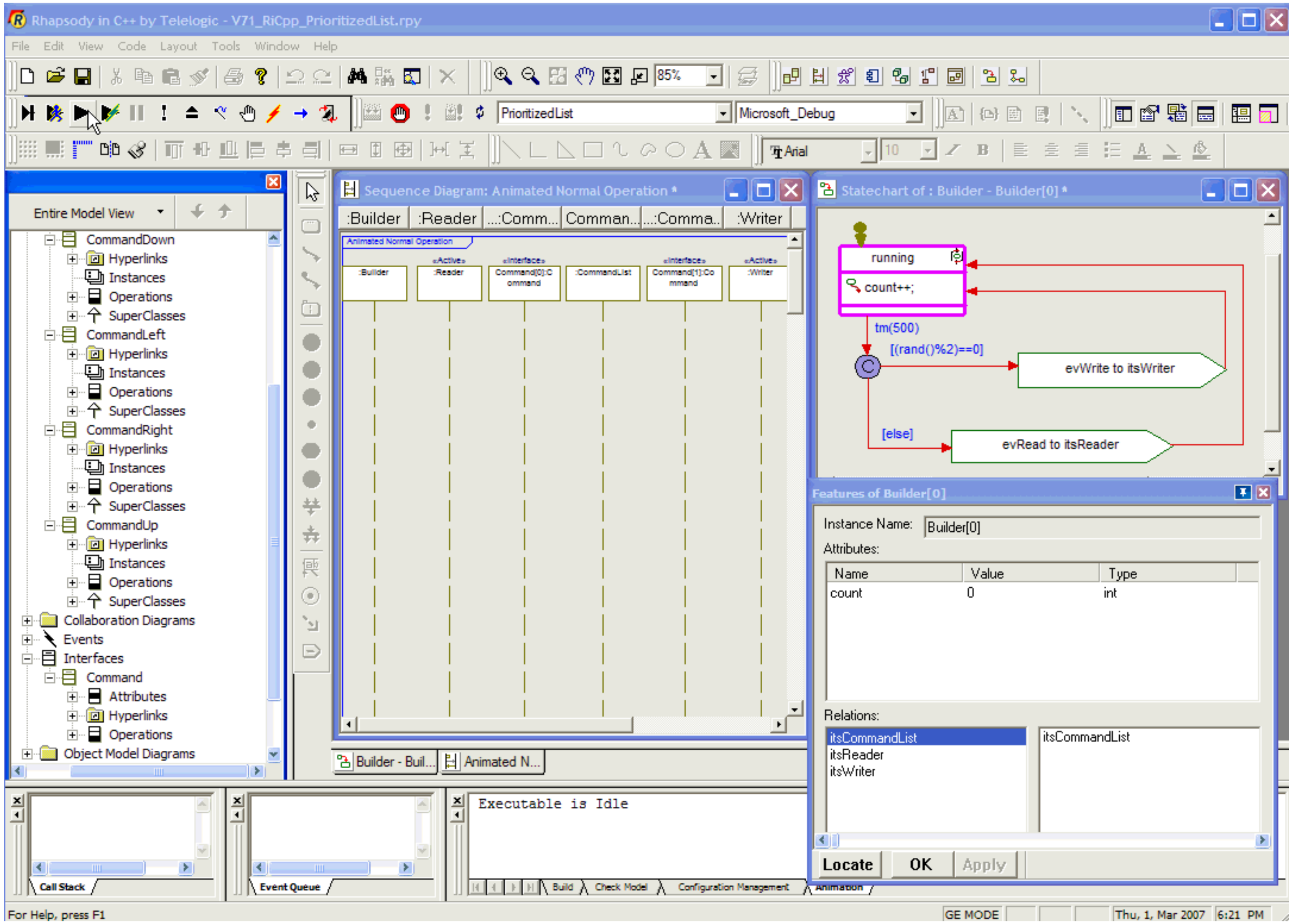
Downstream Coverage Information:

```
UML Model   Rhapsody                                   76%
  V71_RiCpp_FlightControl_And_Information_System
    Packages
      FlightControlSystemPkg
        Classes
          FlightControlGui
            FlowPorts
              alarm
          FlightController
            Statechart
              States
                Alarmed
            Attributes
              alarm
            FlowPorts
              alarm
```

Selection Text: Alarm is triggered by an Altitude or Speed Alarm.

FlightControlSystemRequirements DOORS/Flight Control/Alarm manager/Alarm

DFT : Executable Models on Host & Target



```
Executable is Idle
```

Features of Builder[0]

Instance Name: Builder[0]

Attributes:

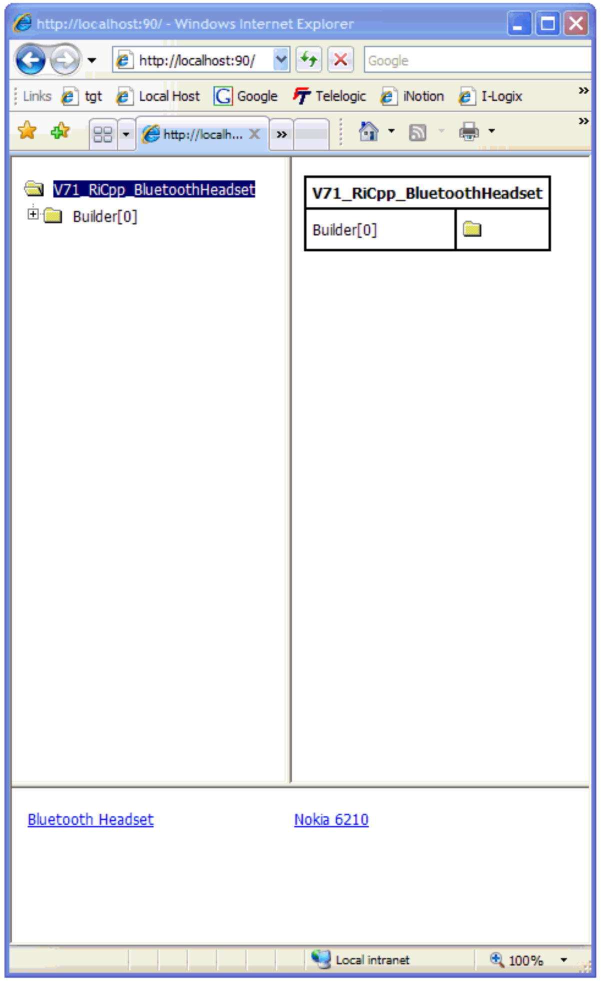
Name	Value	Type
count	0	int

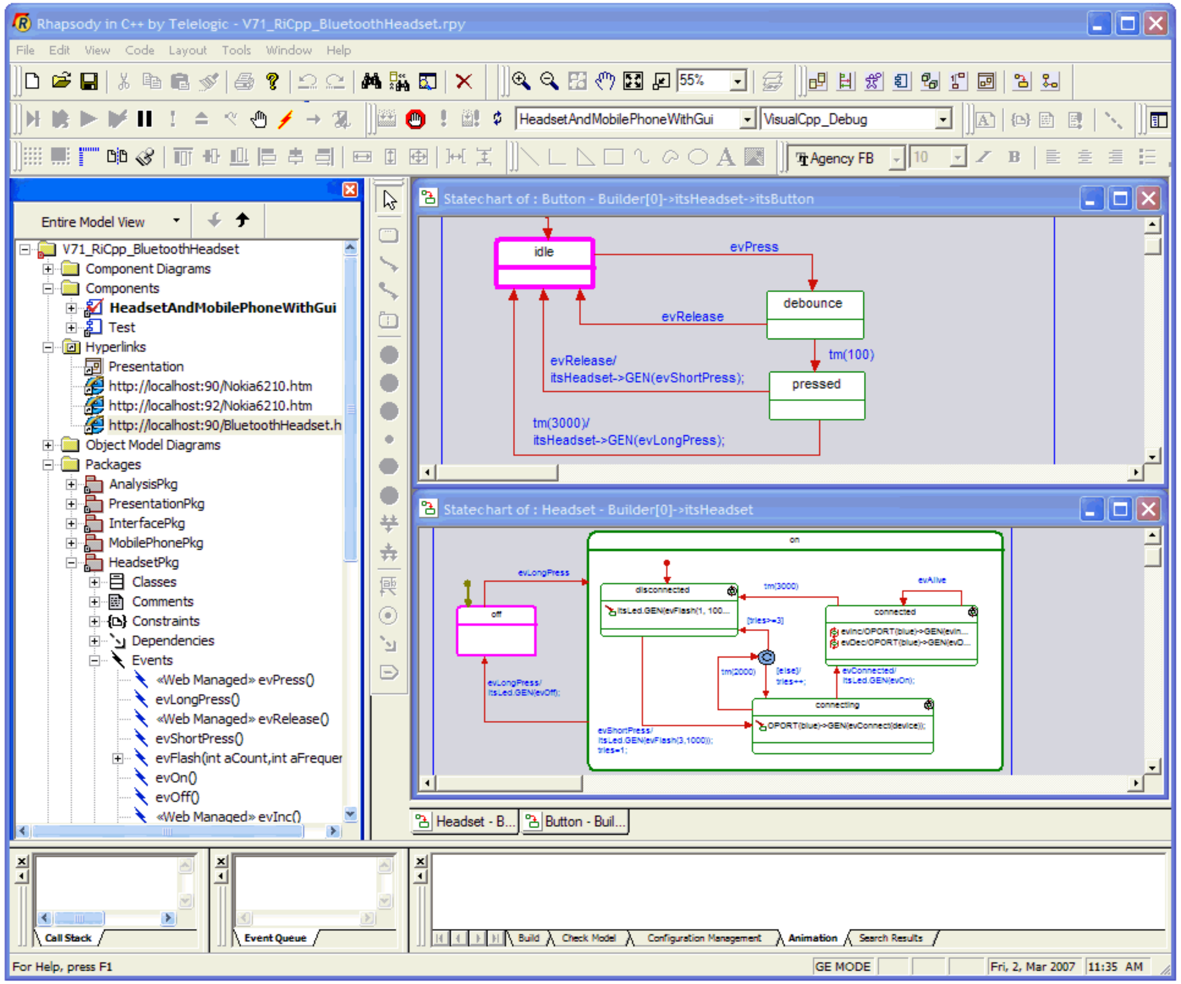
Relations:

itsCommandList
itsReader
itsWriter

itsCommandList

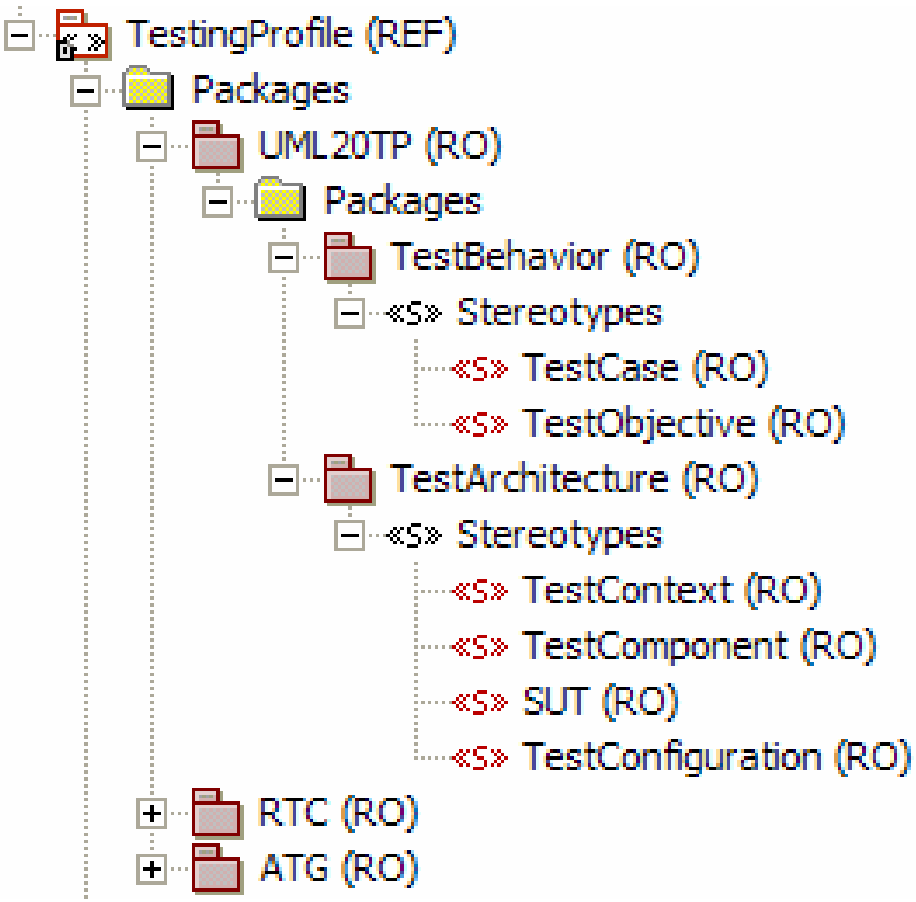
DFT : Rapid HTML Gui’s


http://localhost:90/ - Windows Internet Explorer
http://localhost:90/
Google
Links tgt Local Host Google Telelogic iNotion I-Logix
http://localh...
V71_RiCpp_BluetoothHeadset
Builder[0]
V71_RiCpp_BluetoothHeadset
Builder[0]
Bluetooth Headset
Nokia 6210
Local intranet
100%



DFT : Model Based Testing

UML 2 Testing Profile



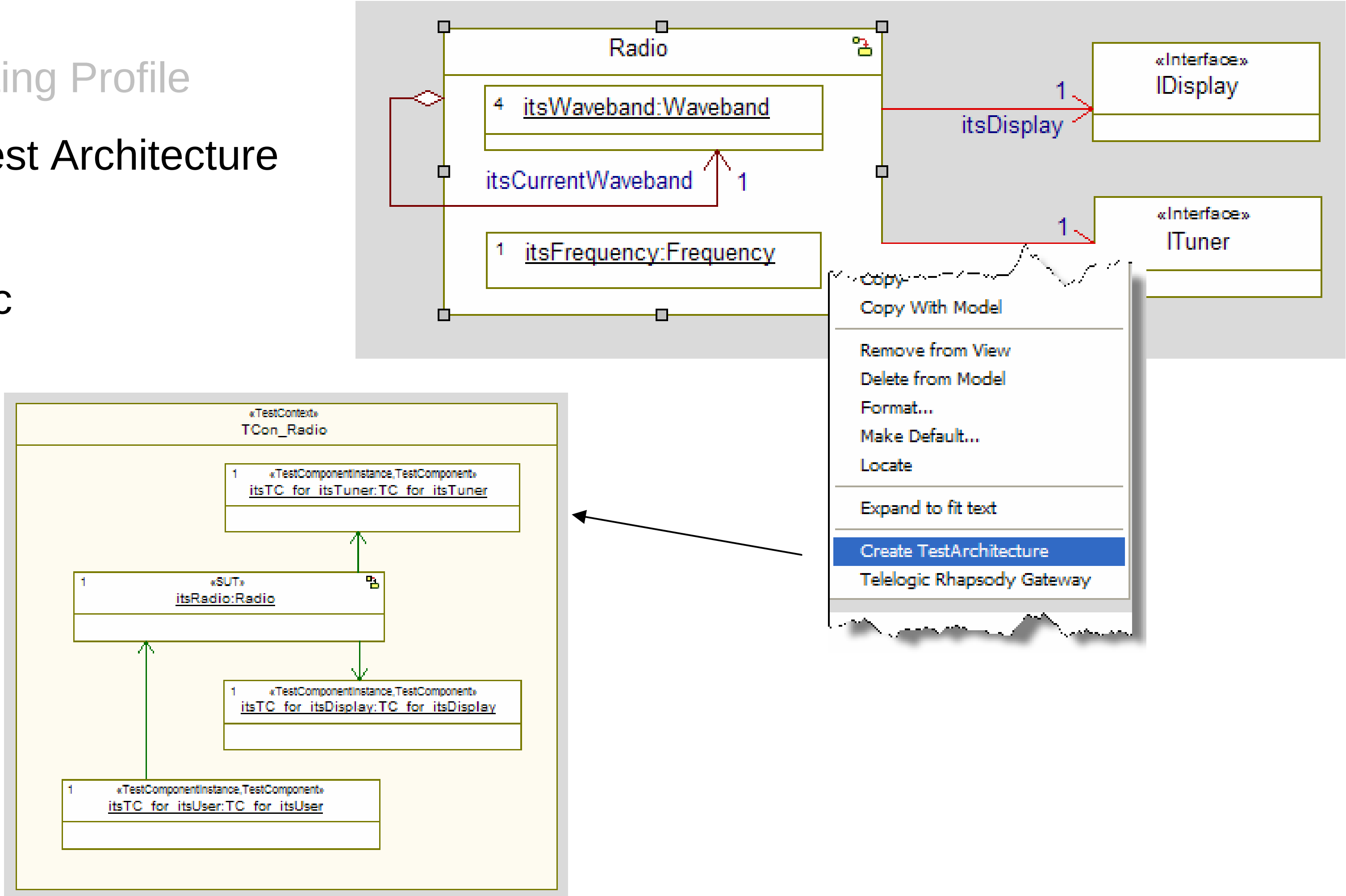
- TestingProfile (REF)
  - Packages
    - UML20TP (RO)
      - Packages
        - TestBehavior (RO)
          - «S» Stereotypes
            - «S» TestCase (RO)
            - «S» TestObjective (RO)
        - TestArchitecture (RO)
          - «S» Stereotypes
            - «S» TestContext (RO)
            - «S» TestComponent (RO)
            - «S» SUT (RO)
            - «S» TestConfiguration (RO)
    - RTC (RO)
    - ATG (RO)

DFT : Model Based Testing

UML 2 Testing Profile

Create a Test Architecture

- Manually
- Automatic



DFT : Model Based Testing

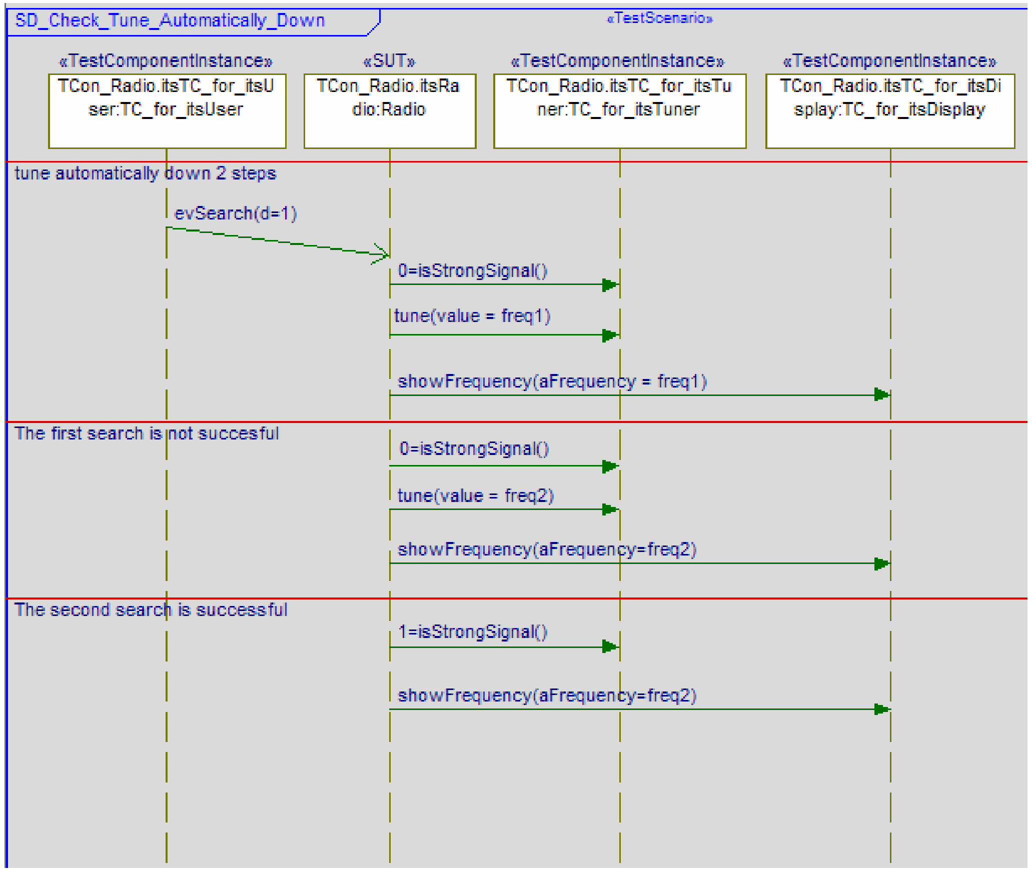
UML 2 Testing Profile

Create a Test Architecture

- Manually
- Automatic

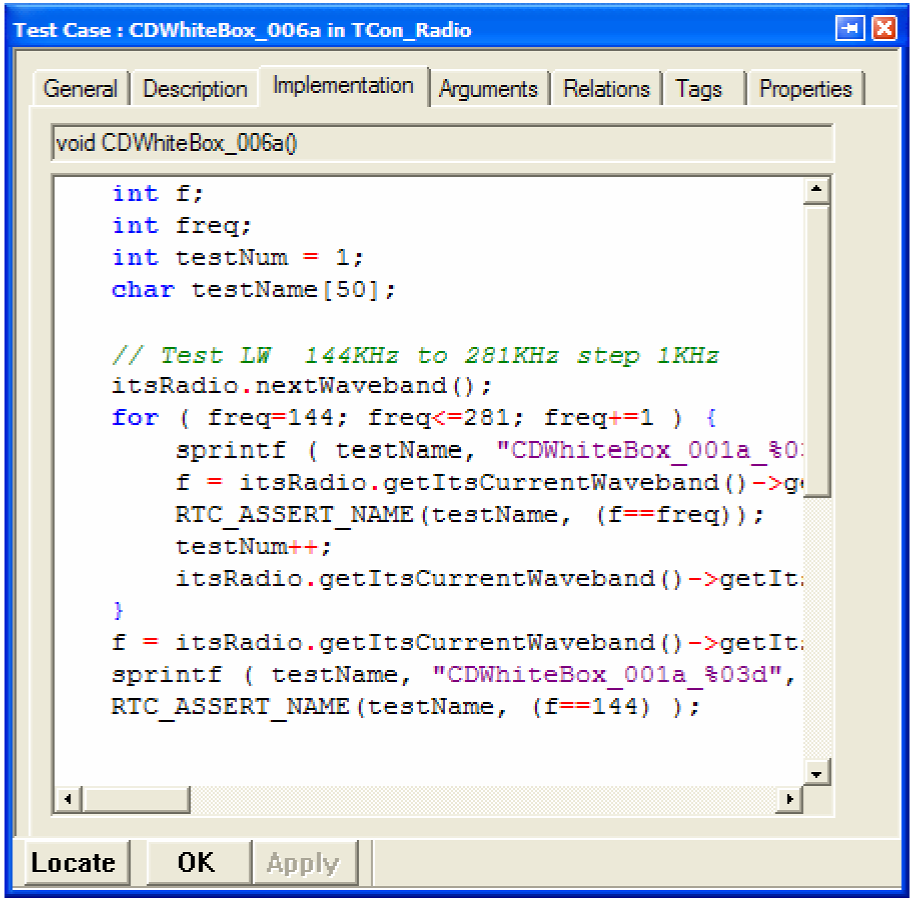
Create Test Cases

- Test Cases can be written:
  - Via Sequence Diagrams


SD_Check_Tune_Automatically_Down
«TestScenario»
«TestComponentInstance» TCon_Radio.itsTC_for_itsUser:TC_for_itsUser
«SUT» TCon_Radio.itsRadio:Radio
«TestComponentInstance» TCon_Radio.itsTC_for_itsTuner:TC_for_itsTuner
«TestComponentInstance» TCon_Radio.itsTC_for_itsDisplay:TC_for_itsDisplay
tune automatically down 2 steps
evSearch(d=1)
0=isStrongSignal()
tune(value = freq1)
showFrequency(aFrequency = freq1)
The first search is not succesful
0=isStrongSignal()
tune(value = freq2)
showFrequency(aFrequency=freq2)
The second search is successful
1=isStrongSignal()
showFrequency(aFrequency=freq2)

DFT : Model Based Testing

- UML 2 Testing Profile
- Create a Test Architecture
  - Manually
  - Automatic
- Create Test Cases
  - Test Cases can be written:
    - Via Sequence Diagrams
    - Manually via code

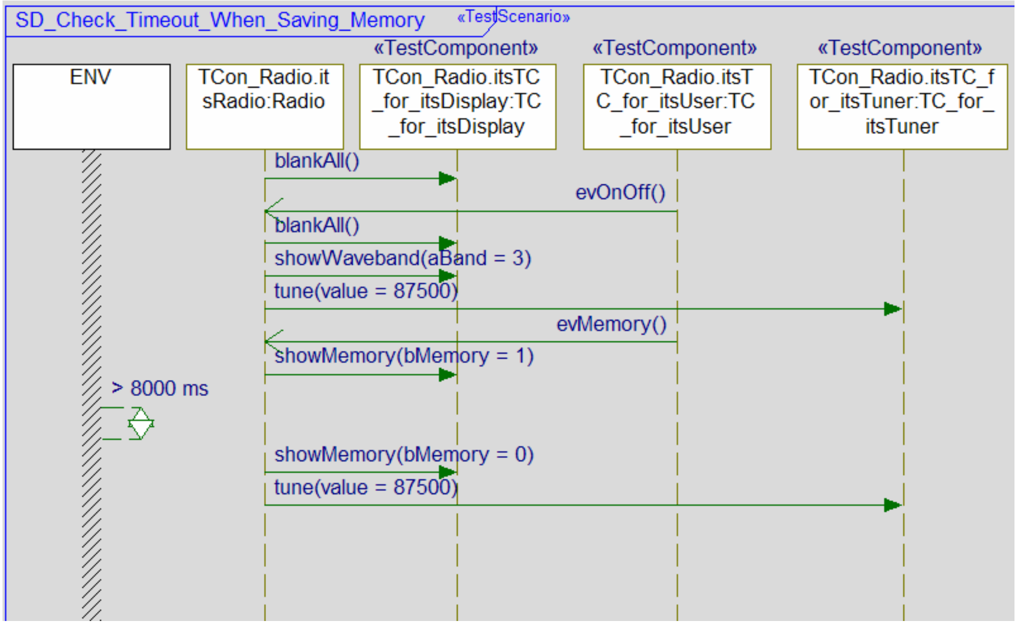


void CDWhiteBox_006a()

```
    int f;
    int freq;
    int testNum = 1;
    char testName[50];

    // Test LW  144KHz to 281KHz step 1KHz
    itsRadio.nextWaveband();
    for ( freq=144; freq<=281; freq+=1 ) {
        sprintf ( testName, "CDWhiteBox_001a_%0
        f = itsRadio.getItsCurrentWaveband()->g
        RTC_ASSERT_NAME(testName, (f==freq));
        testNum++;
        itsRadio.getItsCurrentWaveband()->getIt
    }
    f = itsRadio.getItsCurrentWaveband()->getIt
    sprintf ( testName, "CDWhiteBox_001a_%03d",
    RTC_ASSERT_NAME(testName, (f==144) );
```

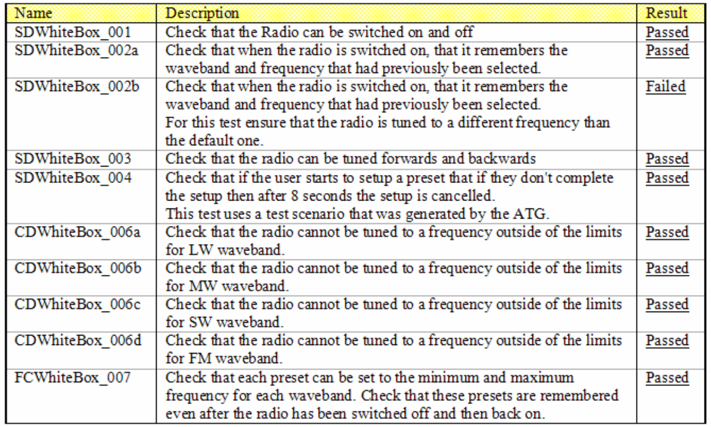
DFT : Model Based Testing

- UTML 2 Testing Profile
- Create a Test Architecture
  - Manually
  - Automatic
- Create Test Cases
  - Test Cases can be written:
    - Via Sequence Diagrams
    - Manually via code
    - Automatically with the ATG (Automatic Test Generator)

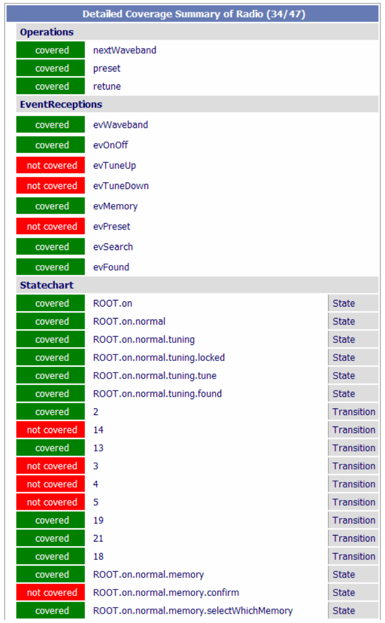


DFT : Model Based Testing

- UML 2 Testing Profile
- Create a Test Architecture
  - Manually
  - Automatic
- Create Test Cases
  - Test Cases can be written:
    - Via Sequence Diagrams
    - Manually via code
    - Automatically with the ATG (Automatic Test Generator)
- Execute Test Cases
  - The Test Cases can be executed automatically

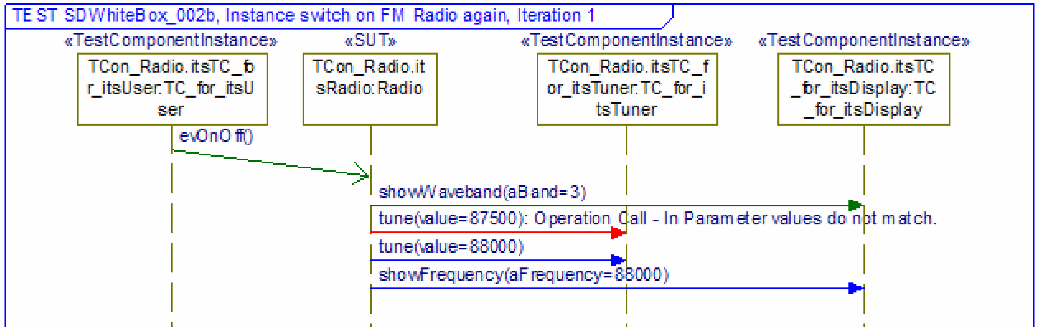


Name	Description	Result
SDWhiteBox_001	Check that the Radio can be switched on and off	Passed
SDWhiteBox_002a	Check that when the radio is switched on, that it remembers the waveband and frequency that had previously been selected.	Passed
SDWhiteBox_002b	Check that when the radio is switched on, that it remembers the waveband and frequency that had previously been selected. For this test ensure that the radio is tuned to a different frequency than the default one.	Failed
SDWhiteBox_003	Check that the radio can be tuned forwards and backwards	Passed
SDWhiteBox_004	Check that if the user starts to setup a preset that if they don't complete the setup then after 8 seconds the setup is cancelled. This test uses a test scenario that was generated by the ATG.	Passed
CDWhiteBox_006a	Check that the radio cannot be tuned to a frequency outside of the limits for LW waveband.	Passed
CDWhiteBox_006b	Check that the radio cannot be tuned to a frequency outside of the limits for MW waveband.	Passed
CDWhiteBox_006c	Check that the radio cannot be tuned to a frequency outside of the limits for SW waveband.	Passed
CDWhiteBox_006d	Check that the radio cannot be tuned to a frequency outside of the limits for FM waveband.	Passed
FCWhiteBox_007	Check that each preset can be set to the minimum and maximum frequency for each waveband. Check that these presets are remembered even after the radio has been switched off and then back on.	Passed



Detailed Coverage Summary of Radio (34/47)

Operations		
covered	nextWaveband	
covered	preset	
covered	retune	
EventReceptions		
covered	evWaveband	
covered	evOnOff	
not covered	evTuneUp	
not covered	evTuneDown	
covered	evMemory	
not covered	evPreset	
covered	evSearch	
covered	evFound	
Statechart		
covered	ROOT.on	State
covered	ROOT.on.normal	State
covered	ROOT.on.normal.tuning	State
covered	ROOT.on.normal.tuning.locked	State
covered	ROOT.on.normal.tuning.tune	State
covered	ROOT.on.normal.tuning.found	State
covered	2	Transition
not covered	14	Transition
covered	13	Transition
not covered	3	Transition
not covered	4	Transition
not covered	5	Transition
covered	19	Transition
covered	21	Transition
covered	18	Transition
covered	ROOT.on.normal.memory	State
not covered	ROOT.on.normal.memory.confirm	State
covered	ROOT.on.normal.memory.selectWhichMemory	State



TEST SDWhiteBox_002b, Instance switch on FM Radio again, Iteration 1

«TestComponentInstance»
TCon_Radio.itsTC_for_itsUser:TC_for_itsUser
«SUT»
TCon_Radio.itsRadio:Radio
«TestComponentInstance»
TCon_Radio.itsTC_for_itsTuner:TC_for_itsTuner
«TestComponentInstance»
TCon_Radio.itsTC_for_itsDisplay:TC_for_itsDisplay

evOnOff()

showWaveband(aBand=3)

tune(value=87500): Operation Call - In Parameter values do not match.

tune(value=88000)

showFrequency(aFrequency=88000)

Model Coverage

The ATG (Automatic Test Generator) is an add-on product to Rhapsody that automatically generates test cases with high coverage of the design:

- Model element coverage: Covers states, transitions, operation calls, event generation
- Code coverage: Generates all relevant combinations of inputs for MC/DC (Modified Condition / Decision Coverage)
- Can be used for Unit, Integration, Regression Tests and Target based testing

Auto-generated test cases should be linked to requirements

Special test cases still need to be captured manually…

Full Application Code Generation

- Rhapsody leverages *all* structural and behavioral model views to produce an executable application
  - Structure models
  - State charts: event driven behavior
  - Activity graphs: algorithms and process flows
  - Components and artifacts

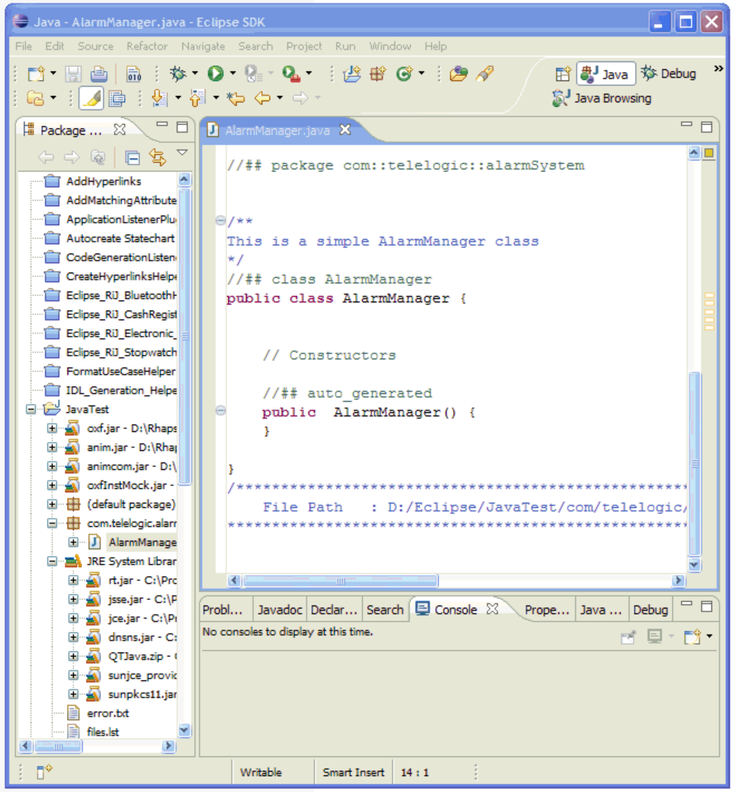
- Rhapsody generates very clean, readable code, easily debugged through any commercial IDE
  - Integrated “white-box” Code (C, C++, Java, Ada, IDL) generation
  - MISRA C compliant code generation
  - High productivity; low cost of maintenance

- Comprehensive code generation technologies
  - OO based and / or functional based, Stereotype based
  - Rules based : Rules Composer / Rules Player

Dynamic Model Code Associativity

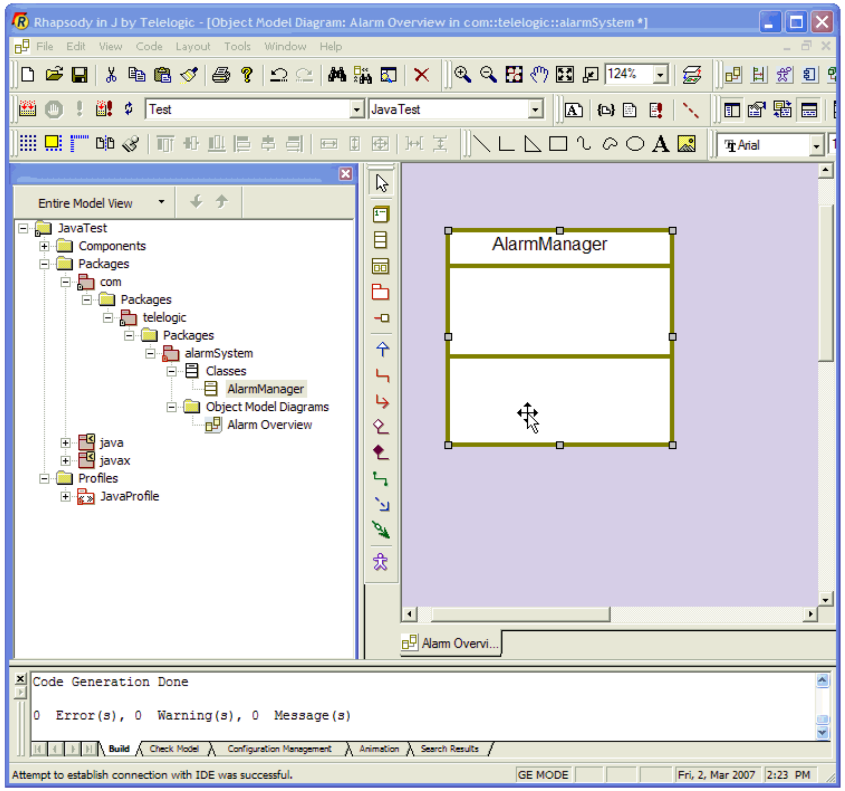
Change one view, the others ***change automatically***

Code and Model always in sync



```
//## package com::telelogic::alarmSystem


/**
This is a simple AlarmManager class
*/
//## class AlarmManager
public class AlarmManager {


    // Constructors

    //## auto_generated
    public  AlarmManager() {
    }

}
/*****************************************************
    File Path   : D:/Eclipse/JavaTest/com/telelogic/
*****************************************************
```

No consoles to display at this time.

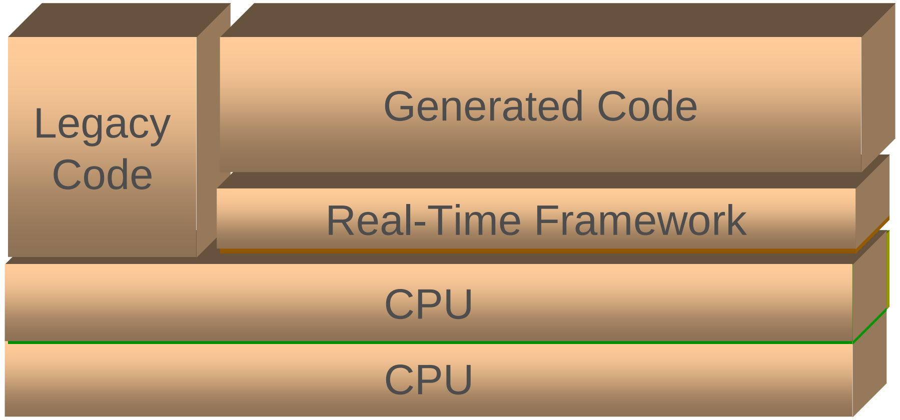


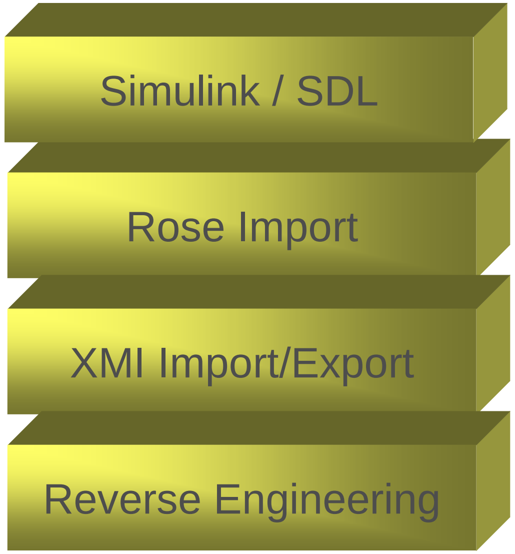
```
Code Generation Done

0 Error(s), 0 Warning(s), 0 Message(s)
```

Attempt to establish connection with IDE was successful.

Real-Time Framework

- Rhapsody provides an executable real-time framework
  - Framework is delivered as a Rhapsody model
    - Provides a clear understanding of structure and functionality, enabling fine tuning
    - Model includes all requirements and design rationale so it’s easily understood
    - Validation suite can be made available through Professional Services
  - Facilitates scaling down for smaller footprint applications
    - Pick and choose only the necessary components
    - Customize components
  - Facilitates certification of the framework ex: DO-178B.





Extended C Framework

Time Triggered *and/or* Event Triggered

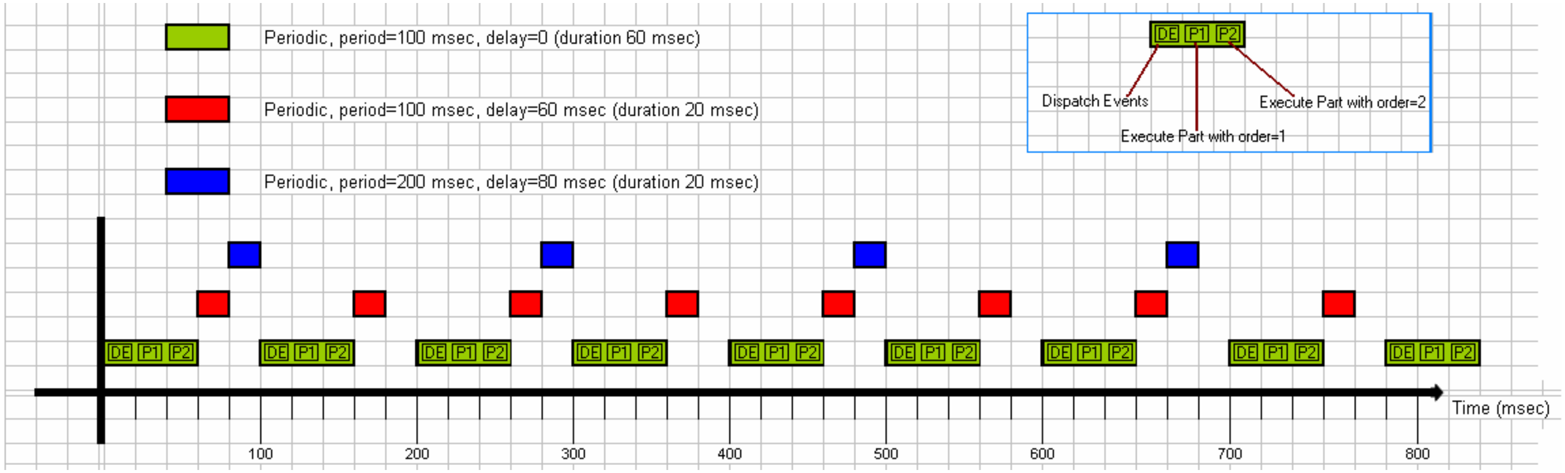
MISRA C

OSEK / Main Loop / Simple OS

All data allocated at compile time

Initialisation done at compile time

Network Ports (for mapping to CAN, LIN, MOST, FlexRAY, ...)

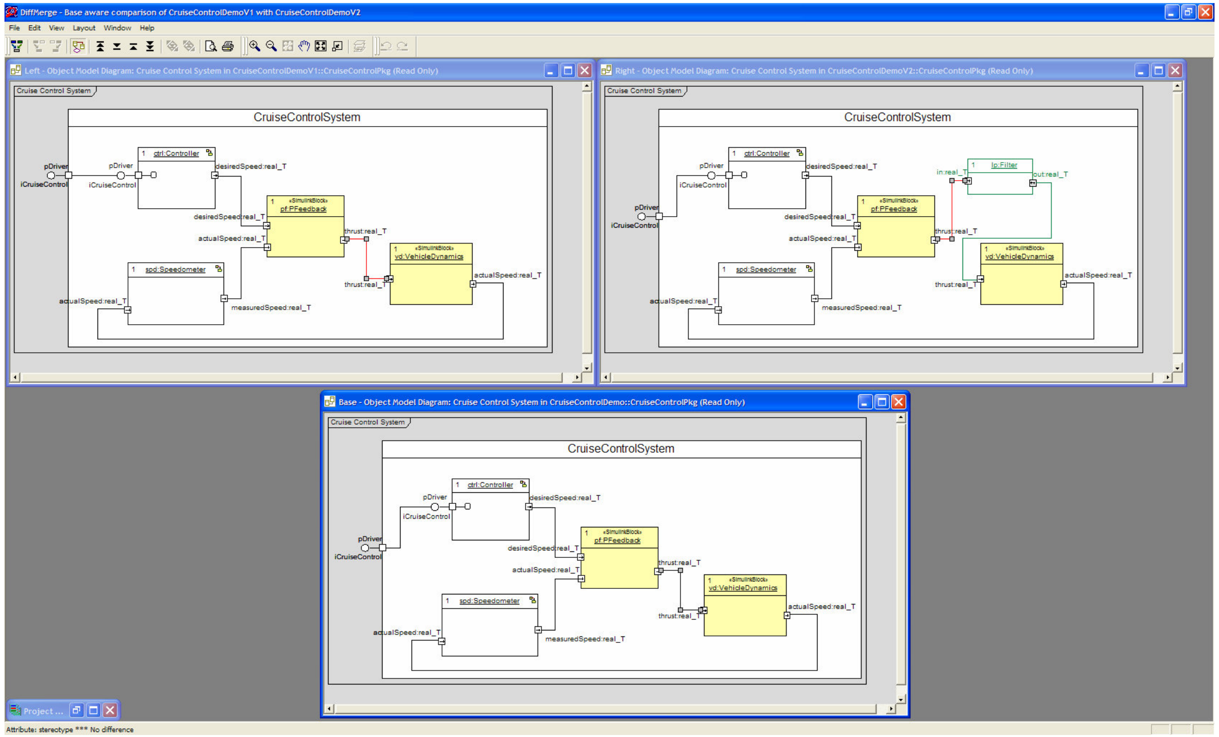


Concurrent Design Collaboration

Small and Large Scale Development

Tight integration with configuration management

Three way Visual Differencing and Merging



DiffMerge - Base aware comparison of CruiseControlDemoV1 with CruiseControlDemoV2

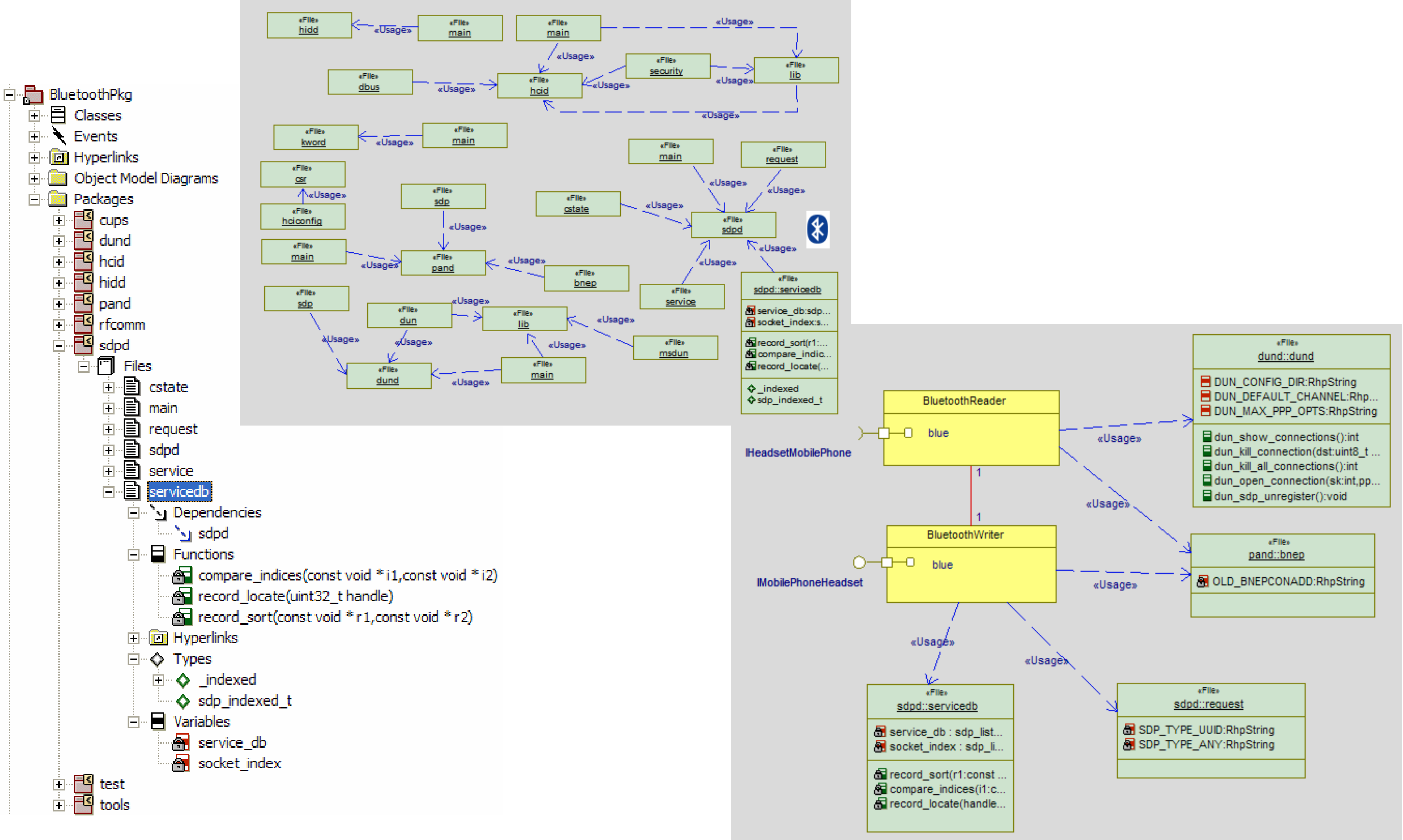
Left - Object Model Diagram: Cruise Control System in CruiseControlDemoV1::CruiseControlPkg (Read Only)

Right - Object Model Diagram: Cruise Control System in CruiseControlDemoV2::CruiseControlPkg (Read Only)

Base - Object Model Diagram: Cruise Control System in CruiseControlDemo::CruiseControlPkg (Read Only)

Attribute: stereotype *** No difference

Reuse of IP : Import Legacy “C” code



- BluetoothPkg
  - Classes
  - Events
  - Hyperlinks
  - Object Model Diagrams
  - Packages
    - cups
    - dund
    - hcid
    - hidd
    - pand
    - rfcomm
    - sdpd
      - Files
        - cstate
        - main
        - request
        - sdpd
        - service
        - servicedb
          - Dependencies
            - sdpd
          - Functions
            - compare_indices(const void * i1,const void * i2)
            - record_locate(uint32_t handle)
            - record_sort(const void * r1,const void * r2)
          - Hyperlinks
          - Types
            - _indexed
            - sdp_indexed_t
          - Variables
            - service_db
            - socket_index
    - test
    - tools

Documentation: Rhapsody ReporterPLUS™

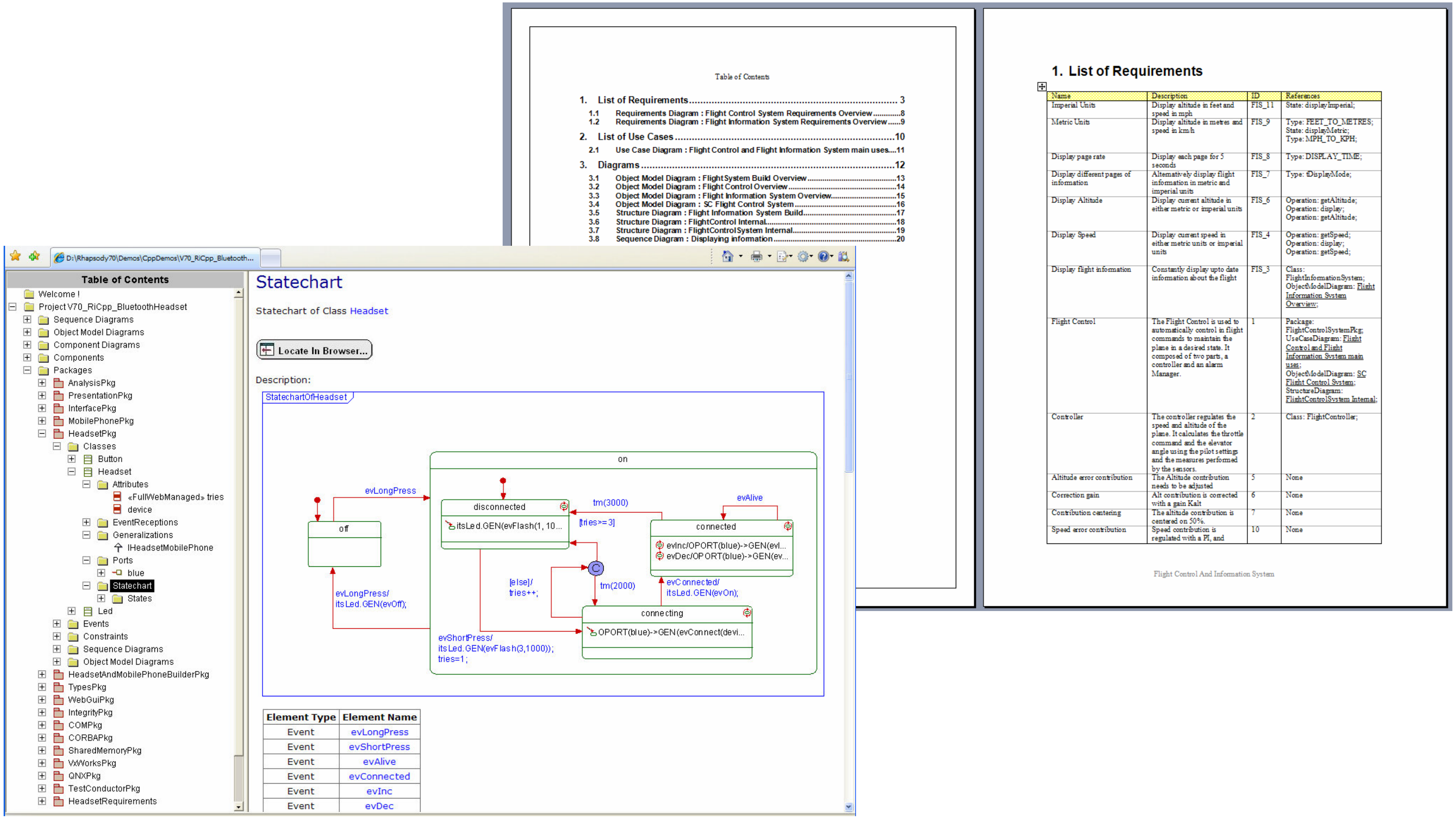


Table of Contents

1. List of Requirements

Name	Description	ID	References
Imperial Units	Display altitude in feet and speed in mph	FIS_11	State: displayImperial;

Metric Units	Display altitude in metres and speed in km/h	FIS_9	Type: FEET_TO_METRES;
			State: displayMetric;
			Type: MPH_TO_KPH;

Display page rate	Display each page for 5 seconds	FIS_8	Type: DISPLAY_TIME;
Display different pages of infomation	Alternatively display flight infomation in metric and imperial units	FIS_7	Type: tDisplayMode;

Display Altitude	Display current altitude in either metric or imperial units	FIS_6	Operation: getAltitude;
			Operation: display;
			Operation: getAltitude;

Display Speed	Display current speed in either metric units or imperial units	FIS_4	Operation: getSpeed;
			Operation: display;
			Operation: getSpeed;

Display flight infomation	Constantly display upto date infomation about the flight	FIS_3	Class: FlightInformationSystem;
			ObjectModelDiagram: Flight Information System Overview;

Flight Control	The Flight Control is used to automatically control in flight commands to maintain the plane in a desired state. It composed of two parts, a controller and an alarm Manager.	1	Package: FlightControlSystemPkg;
			UseCaseDiagram: Flight Control and Flight Information System main uses;
			ObjectModelDiagram: SC Flight Control System;
			StructureDiagram: FlightControlSystem Internal;

Controller	The controller regulates the speed and altitude of the plane. It calculates the throttle command and the elevator angle using the pilot settings and the measures performed by the sensors.	2	Class: FlightController;
Altitude error contribution	The Altitude contribution needs to be adjusted	5	None
Correction gain	Alt contribution is corrected with a gain Kalt	6	None
Contribution centering	The altitude contribution is centered on 50%.	7	None
Speed error contribution	Speed contribution is regulated with a PI, and	10	None

Flight Control And Information System

D:\Rhapsody70\Demos\CppDemos\V70_RiCpp_Bluetooth...

Table of Contents

Statechart

Statechart of Class Headset

Locate In Browser...

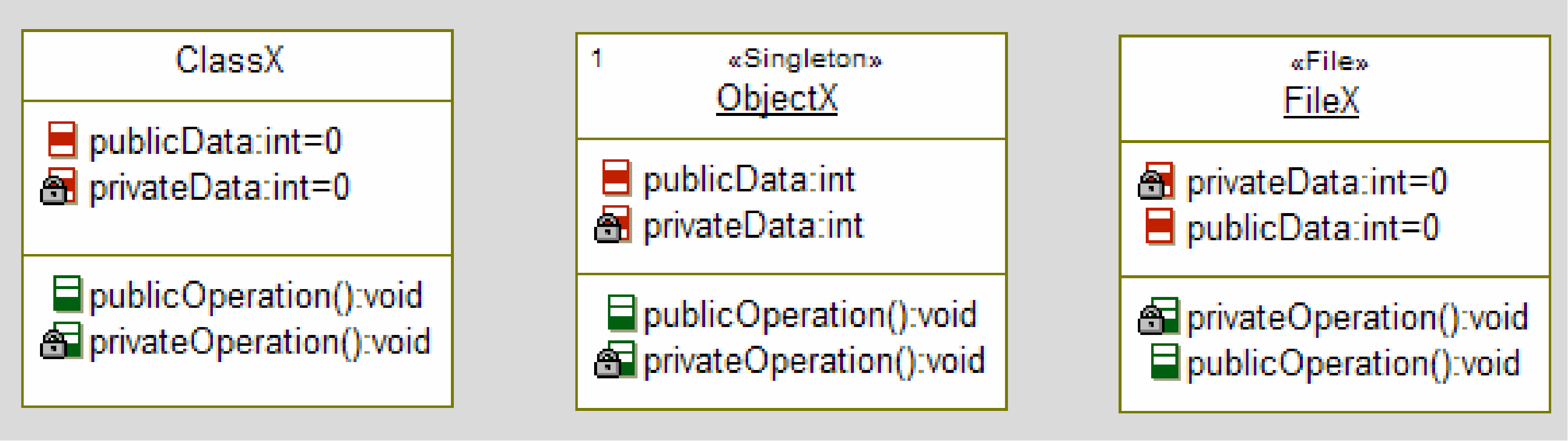
Description:

Element Type	Element Name
Event	evLongPress
Event	evShortPress
Event	evAlive
Event	evConnected
Event	evInc
Event	evDec

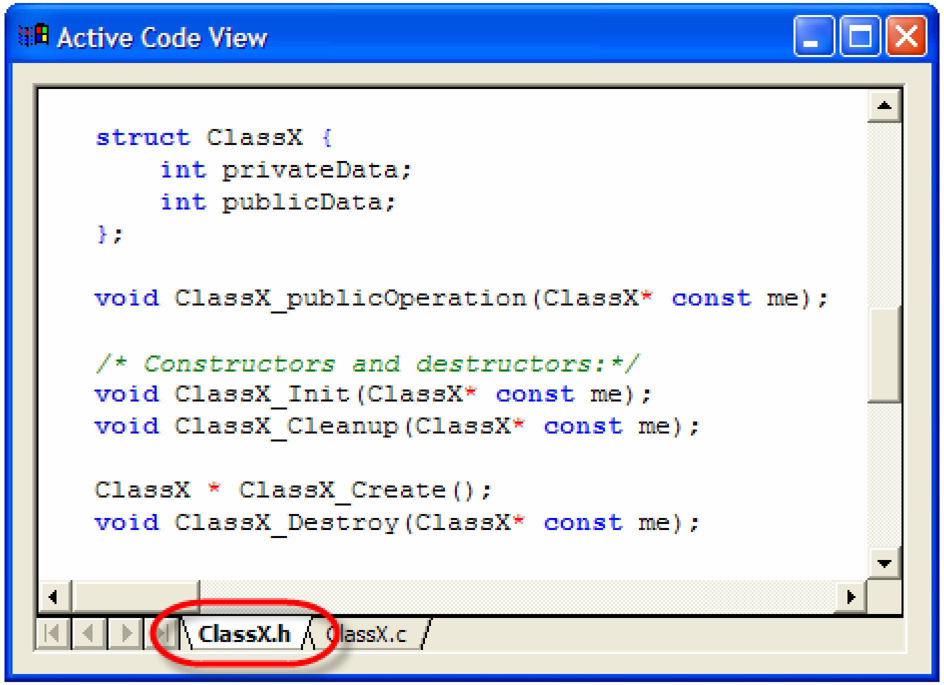
Generating “C” code from UML

When we want to create a class, we have the choice of being able to create:

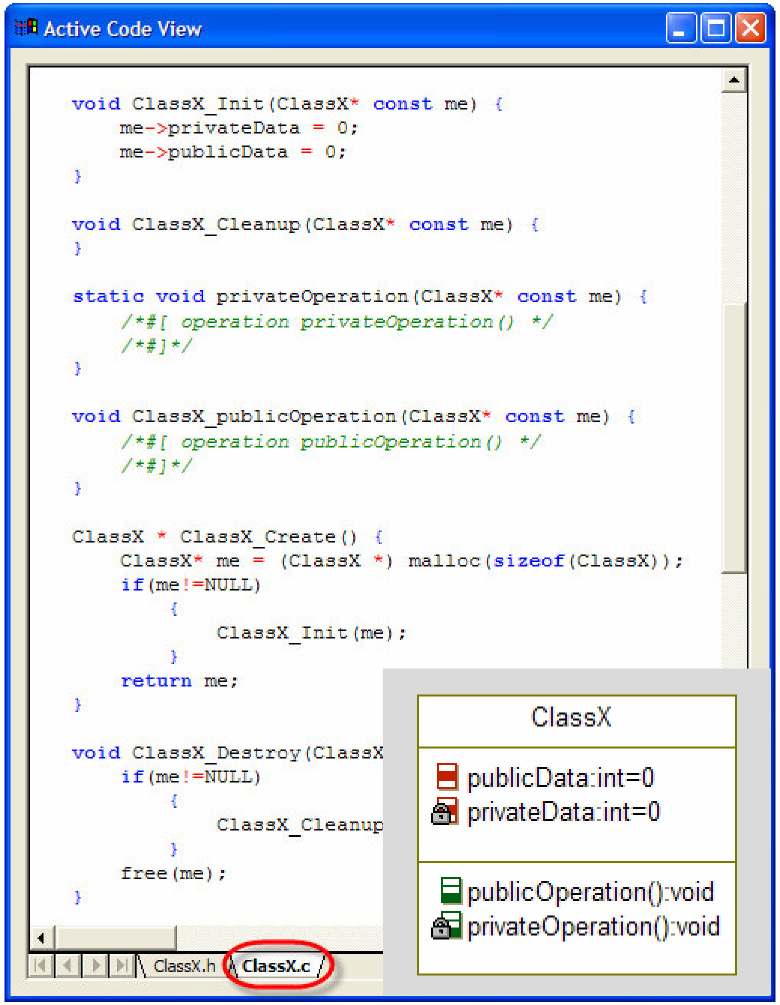
- A Class (like in C++ and java) that we can create and delete dynamically.
- A Singleton Object
- A File



“C” Code generation for a Class



```
struct ClassX {
    int privateData;
    int publicData;
};

void ClassX_publicOperation(ClassX* const me);

/* Constructors and destructors:*/
void ClassX_Init(ClassX* const me);
void ClassX_Cleanup(ClassX* const me);

ClassX * ClassX_Create();
void ClassX_Destroy(ClassX* const me);
```



```
void ClassX_Init(ClassX* const me) {
    me->privateData = 0;
    me->publicData = 0;
}

void ClassX_Cleanup(ClassX* const me) {
}

static void privateOperation(ClassX* const me) {
    /*#[ operation privateOperation() */
    /*#]*/
}

void ClassX_publicOperation(ClassX* const me) {
    /*#[ operation publicOperation() */
    /*#]*/
}

ClassX * ClassX_Create() {
    ClassX* me = (ClassX *) malloc(sizeof(ClassX));
    if(me!=NULL)
        {
            ClassX_Init(me);
        }
    return me;
}

void ClassX_Destroy(ClassX
    if(me!=NULL)
        {
            ClassX_Cleanup
        }
    free(me);
}
```

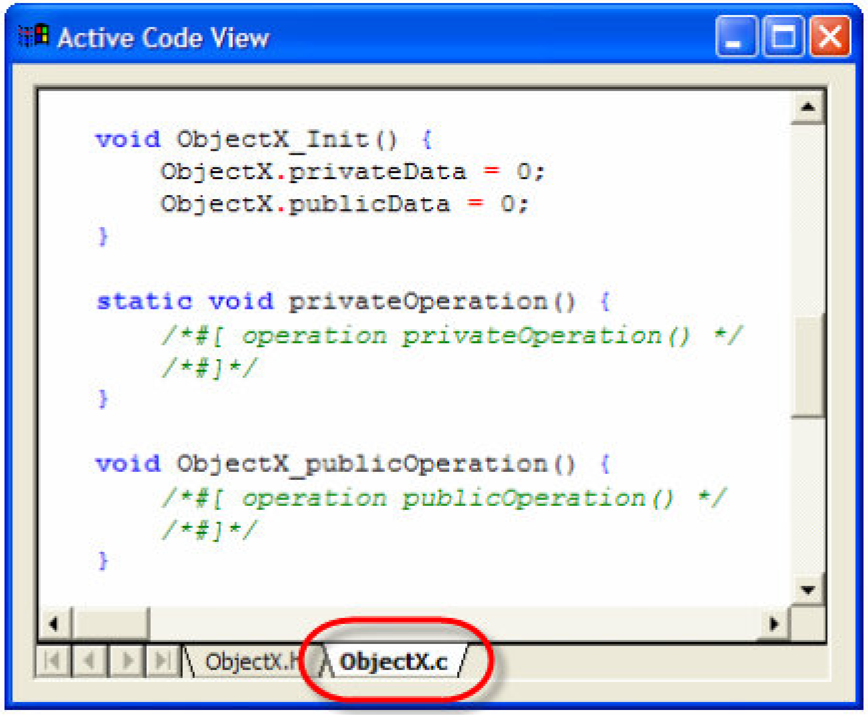
ClassX

publicData:int=0
privateData:int=0

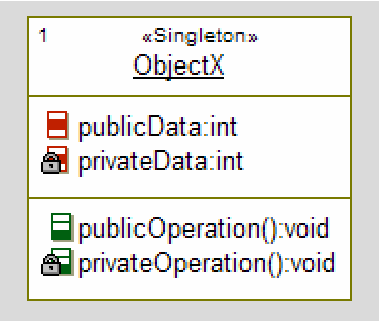
publicOperation():void
privateOperation():void

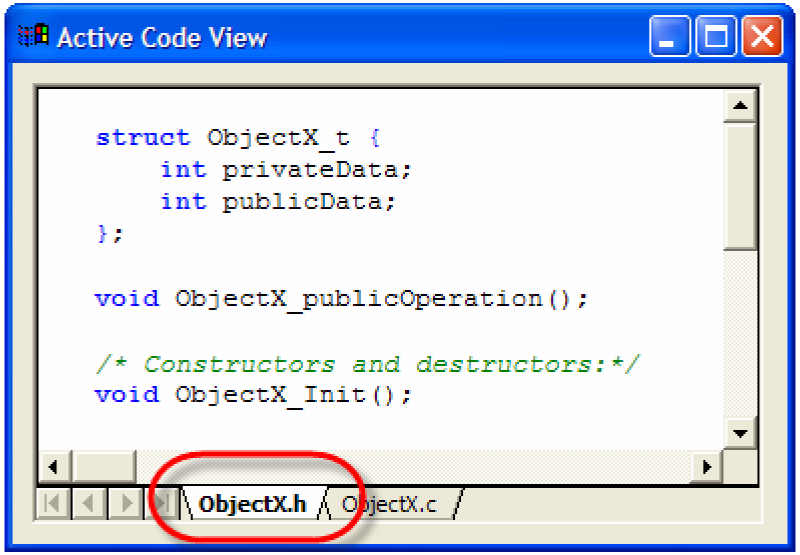
We can set properties to disable the generation of the _Create, _Destroy and Cleanup operations

“C” Code generation for a «Singleton» Object



```
void ObjectX_Init() {
    ObjectX.privateData = 0;
    ObjectX.publicData = 0;
}

static void privateOperation() {
    /*#[ operation privateOperation() */
    /*#]*/
}

void ObjectX_publicOperation() {
    /*#[ operation publicOperation() */
    /*#]*/
}
```

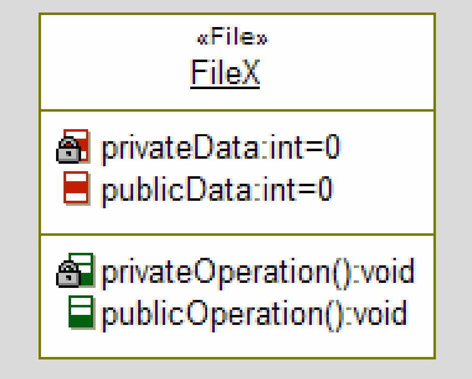




```
struct ObjectX_t {
    int privateData;
    int publicData;
};

void ObjectX_publicOperation();

/* Constructors and destructors:*/
void ObjectX_Init();
```

With a Singleton object we still get a structure containing our attributes, but no “me” pointer.

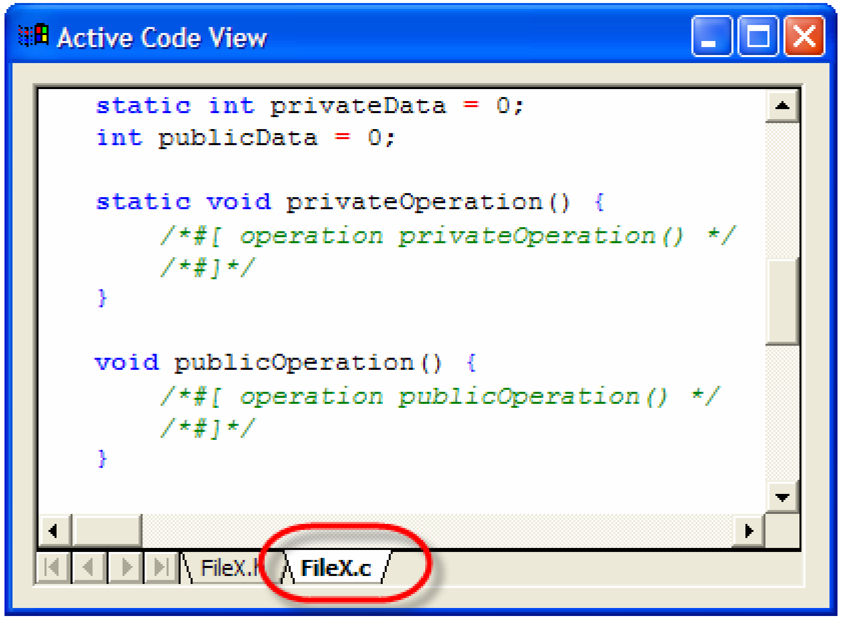
"C" Code generation for a File



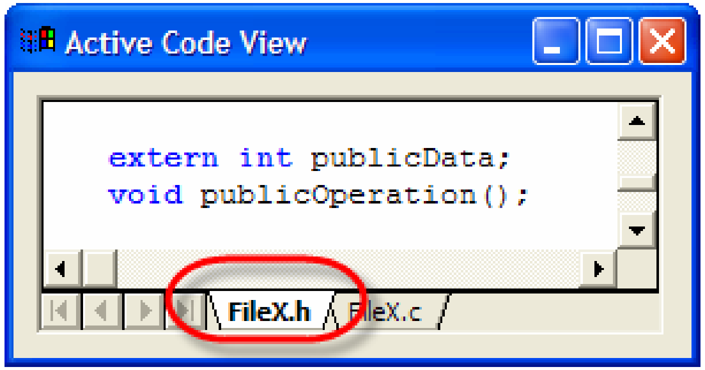
«File»
FileX

privateData:int=0
publicData:int=0

privateOperation():void
publicOperation():void



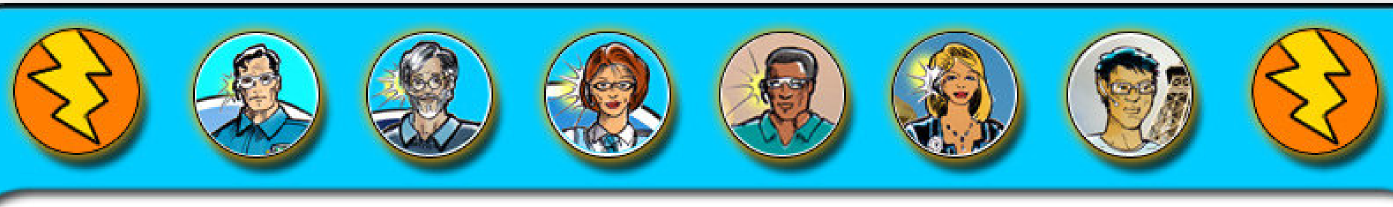
```
static int privateData = 0;
int publicData = 0;

static void privateOperation() {
    /*#[ operation privateOperation() */
    /*#]*/
}

void publicOperation() {
    /*#[ operation publicOperation() */
    /*#]*/
}
```



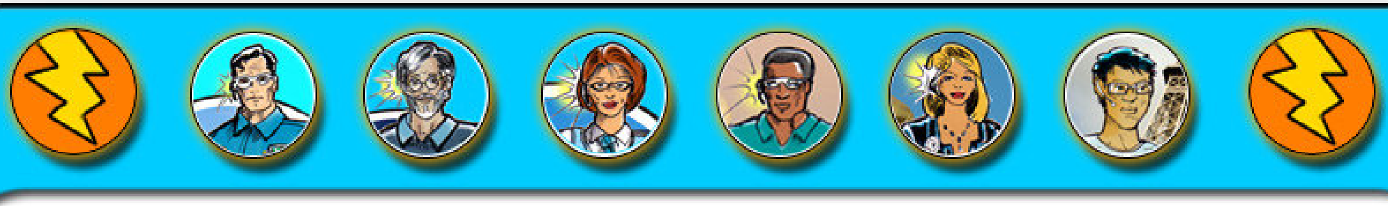
```
extern int publicData;
void publicOperation();
```

The code generation for a file is very simple to understand and use.

Rhapsody Demo



QUESTIONS



THANK YOU

Learn more at:

IBM Rational software

IBM Rational Software Delivery Platform

Process and portfolio management

Change and release management

Quality management

Architecture management

Rational trial downloads

Leading Innovation Web site

developerWorks Rational

IBM Rational TV

IBM Rational Business Partners