Embedded Development with UML 2 and Xholon

Ken Webb

Primordion

presented at OCLUG

December 5, 2006

Abstract

Many embedded developers view the Unified Modeling Language (UML) with disdain as just a bunch of *bubble diagrams*. In fact, with the recent introduction of version 2, UML now incorporates the key concepts used by former local firm ObjecTime in their product for real-time and embedded software development. It is now possible to develop and maintain complete applications, with all the code, using UML 2, and a runtime environment such as provided by Xholon.

Introduction

Some initial comments

- Xholon is currently largely a research project.
- Nothing in Xholon is specific to Linux. It runs anywhere that Java is available.
- Initial version of Xholon is Java, but concepts I will discuss are applicable to C, C++, etc.
- Xholon is intended partly for the development of embedded systems, but an embedded version is not yet available.
  - It uses the same concepts as high-end UML embedded development tools. This presentation is mostly about these concepts, and uses open-source Xholon to demonstrate them.

UML Apps on 8-bit microcontroller

- Interesting paper I recently saw:
  - Samek, M. (2006). UML Statecharts at $10.99. Dr. Dobb's Journal, May 24, 2006. http://www.ddj.com/dept/embedded/188101799
- “describes a method and software for implementing UML statecharts in C, small enough to fit a low-end 8-bit microcontroller. More specifically, it presents a nontrivial UML statechart example, implemented with QP-nano, that runs on the USB Toolstick from Silicon Laboratories with room to spare (8051-derivative, 256 bytes of RAM, 8KB of Flash).”

ObjecTime Developer (OTD)

- A visual modeling tool designed to help software developers design, implement, execute and debug real-time and embedded software.
- Originally Unix and Smalltalk GUI.
- Generated C++ and C code.
- Started as a BNR/Nortel project.
- Taken over by ObjecTime Ltd. In 1993, which was based in Kanata.

Rational Rose RealTime (RRT)

- In 2000 ObjecTime purchased by Rational.
- OTD became Rational Rose RealTime (RRT).
  - NOT the same as Rational Rose.
- Written in C++. Development system runs on Unix, Windows, Linux?.
- Generates C, C++, Java.
- Numerous realtime and embedded targets.

IBM Rational Rose Technical Developer

- Rational was bought by IBM.
- “a robust Model-Driven Development (MDD) solution”
- “Based on the industry-standard Unified Modeling Language™ (UML™), it is designed expressly to meet the challenges of complex systems development.”
- “Delivers runtime model execution, visual model debugging, model-based testing, and advanced modeling constructs”
- “Integrates with ... most popular embedded IDEs and RealTime Operating Systems”
- “Operating systems supported: Linux, UNIX, Windows”
- http://www-306.ibm.com/software/awdtools/developer/technical/

Realtime Object Oriented Modeling (ROOM)

- ROOM is the technical methodology behind OTD, RRT and IBM Technical Developer.
- Many ROOM concepts have been adopted into UML 2.

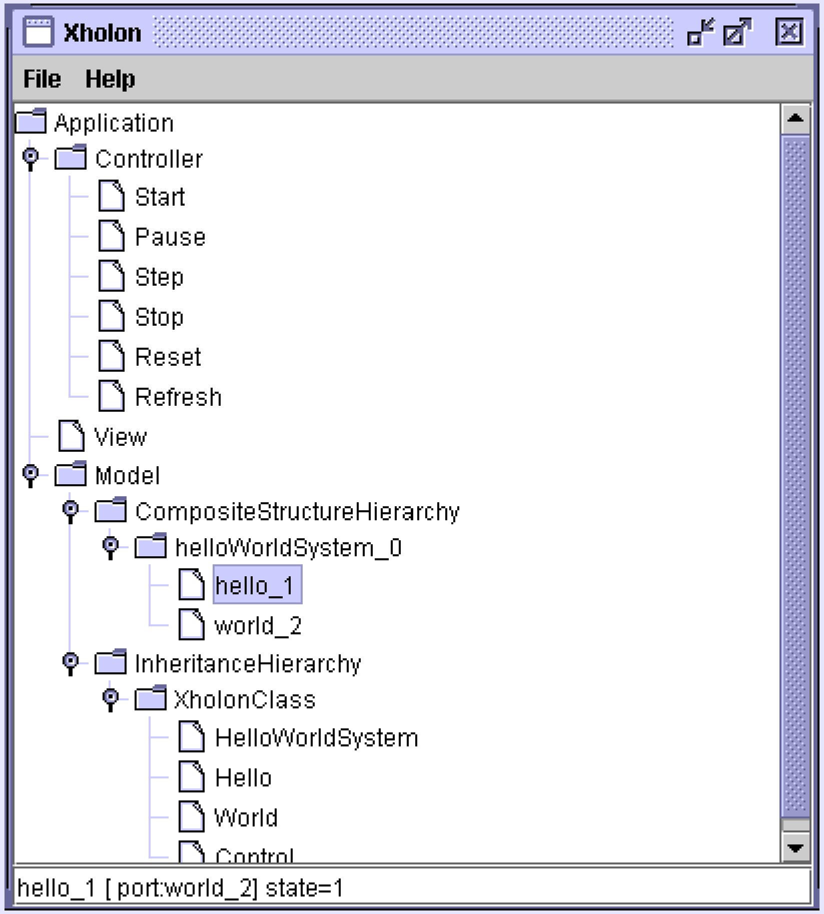
Unified Modeling Language 2 (UML 2)

- Best known as a general-purpose visual modeling language.
- UML 2 offers 13 diagram types and a large number of element types.
  - It offers such a wealth of alternatives that it can be quite confusing.
- UML 2 offers some new constructs that make executable UML more practical, especially for embedded systems.
  - This presentation focuses on a very limited number of constructs that were also available in ROOM.

Xholon & Xholon runtime framework

- Xholon is an open-source tool that I have developed to allow the modeling and execution of event-driven systems.
- Models consist of a set of XML and Java files.
- Xholon offers a simple default GUI at runtime.
- Third-party UML visual design tools can be used to create the model and all the code.
  - XSLT scripts to transform to Xholon format.
- The Xholon runtime framework can execute these models.

Xholon GUI



```
Application
  Controller
    Start
    Pause
    Step
    Stop
    Reset
    Refresh
  View
  Model
    CompositeStructureHierarchy
      helloWorldSystem_0
        hello_1
        world_2
    InheritanceHierarchy
      XholonClass
        HelloWorldSystem
        Hello
        World
        Control
```

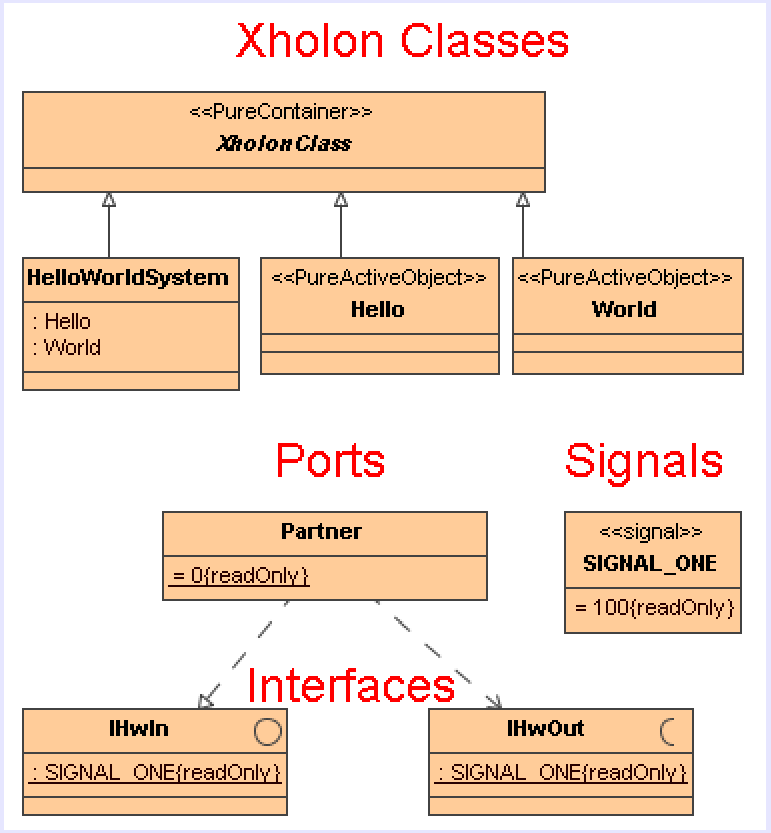
```
hello_1 [ port:world_2] state=1
```

Some UML 2 concepts and how they are used in Xholon

Object-oriented

- Classes.
- Class inheritance hierarchy.
- Object instances are of a certain single class type, but may change their class during execution.
- Objects in Xholon are called *xholons*
  - they are wholes that may contain parts
  - they themselves may be parts of other wholes.
- Xholons should typically all be objects from your application domain.

UML Class Inheritance Tree



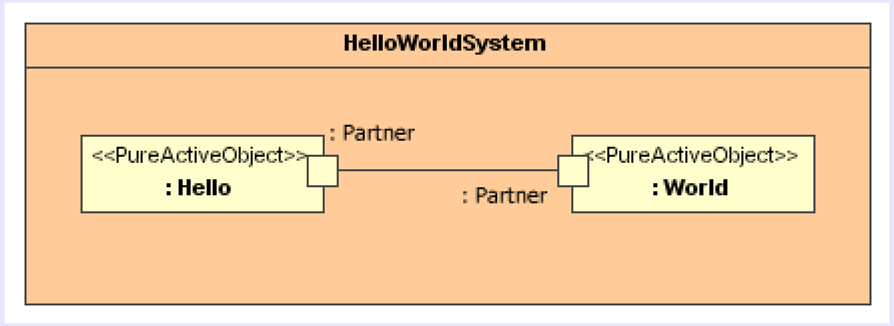
Composite structure

- All xholons (runtime objects) are organized into a single composite structure tree.
- Each xholon node has a parent, and may have one or more children.
- Xholon nodes can be added, removed, or moved around at runtime.
- Composite structure diagrams added in UML 2.

Ports

- Ports are interaction points that allow xholons to connect laterally to other xholons in the tree.
- At runtime, ports allow:
  - Message passing between xholons.
  - Fuction calls from one xholon to another.

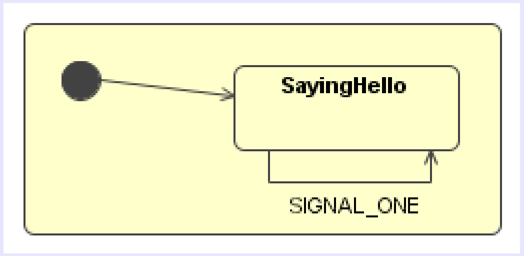
UML Composite Structure with Ports and Connectors



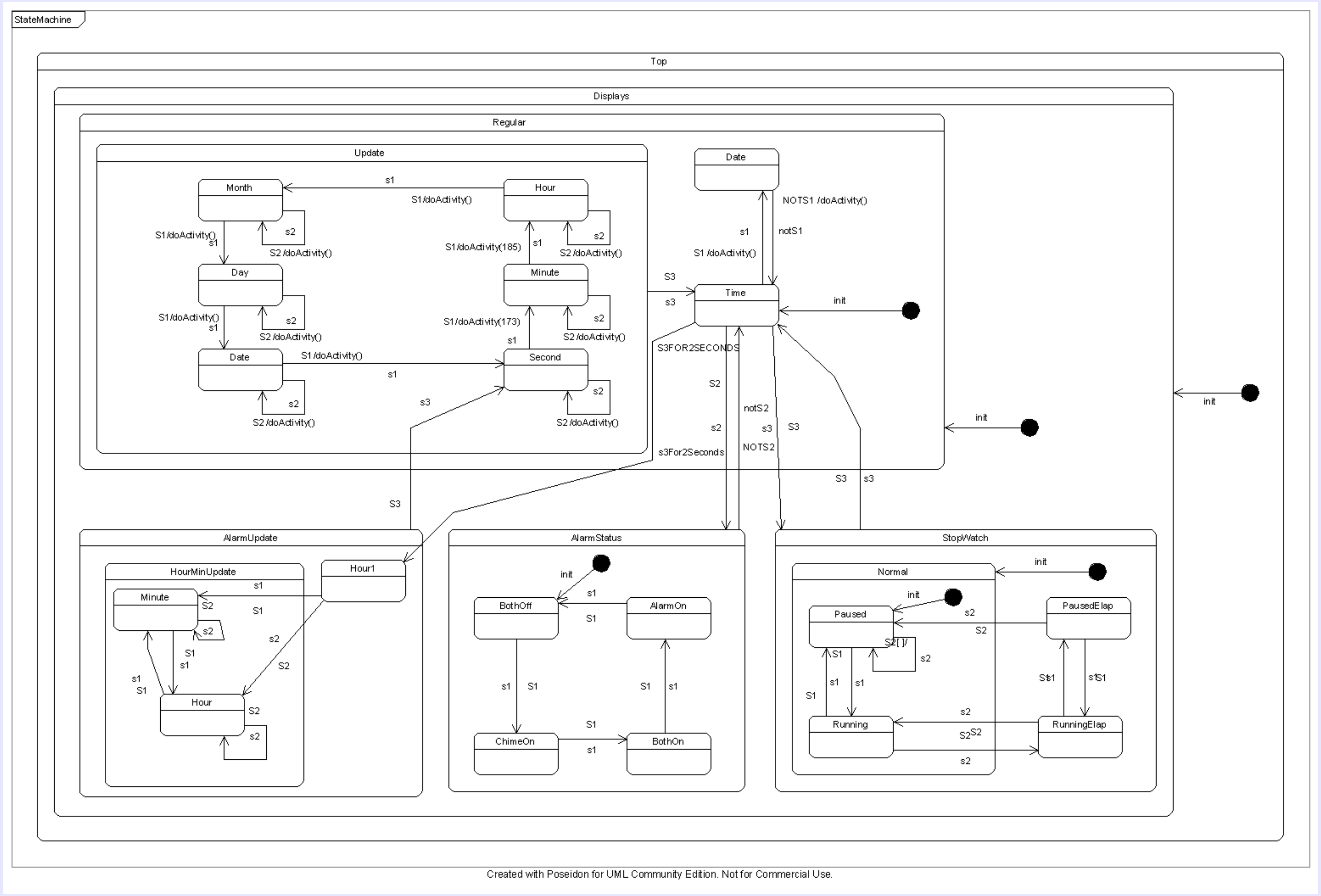
State machines

- Some xholons are active objects with behavior.
- This behavior may be expressed using finite state machines (FSMs).
- UML offers a rich set of FSM constructs, most of which can be executed by Xholon.

UML Finite State Machine (FSM)



More complex FSM



Message passing through ports between state machines

- A Xholon application can be built as a set of objects in a tree that can send messages to each other through ports.
- Each message becomes an event in the receiver's FSM.
- This approach is supported in UML 2.

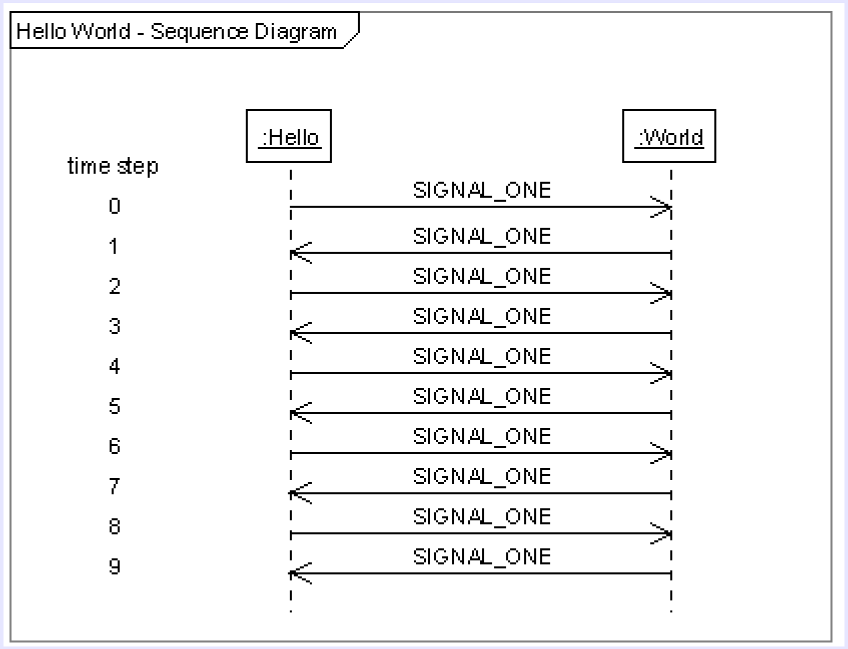
Code as an integral part of the model

- UML models often do not include actual code.
- Xholon requires that all code exist within the model. This can be accommodated in various ways within different UML tools.
- Code typically lives in state transitions, and in act() functions.

Forward engineering with code generation

- Many UML tools do reverse engineering of existing code.
- Xholon only uses forward engineering.
- This is a Model Driven Development (MDD) approach. The model is complete, and can be transformed into an executable system through some transformation process.
  - Xholon uses XSLT scripts to do 100% code generation from UML 2 models.

Sequence Diagrams



Xholon

Xholon

- Background
  - I needed free or inexpensive tools for my research. Commercial tools are all VERY expensive.
- Goals
  - Model and run simulations of complex biological systems.
  - Model and run embedded systems.
  - Be able to model and run other event-driven systems.
  - Support multiple paradigms.

Xholon - Current status

- Beta; a work in progress
- Runs on Linux and Windows hosts
- First implementation is written in Java
- Not yet able to run on an embedded target
    - J2ME for Java embedded environments
- Open source project hosted at sourceforge

Xholon - Aligned with UML 2

- Xholon is roughly aligned with a subset of UML 2.
- It is also aligned with SysML, a subset of UML with extensions to support systems engineering, especially embedded systems.

Xholon
Some technical details

Xholon - Scheduling

- Processing of messages in message Q.
- Execution of act() function every time step for each xholon in the composite structure tree.
  - Also preAct() and postAct() if needed.
- Do activities in finite state machines.
- Timeouts in finite state machines.
- The time step rate is configurable.
- All xholons in Model run on same thread.

Run to completion

- Each bit of code runs to completion before the scheduler allows any other code to run.
  - Exception: timeouts can interrupt other processing to insert a message into the message Q.
- Code snippets, such as in the act() functions or on FSM state transitions, must execute within a small amount of time.
- This allows concurrency without having to have separate threads.

Misc. Technical Details

- Uses XPath to connect ports at runtime.
- Model-View-Controller (MVC)
  - Only the Model is mandatory.
  - Graphical viewers are optional, to make it possible to run on smaller devices.

Xholon Demo

Demo

This is a worked example to be demonstrated on a Linux laptop. We will go through these steps:

(1) View and modify an existing model using third-party commercial UML 2 tool MagicDraw.

- No UML 2 open-source products are yet available that I know of. There are lots of UML 1 products.

(2) Generate code using the Xholon XSLT scripts.

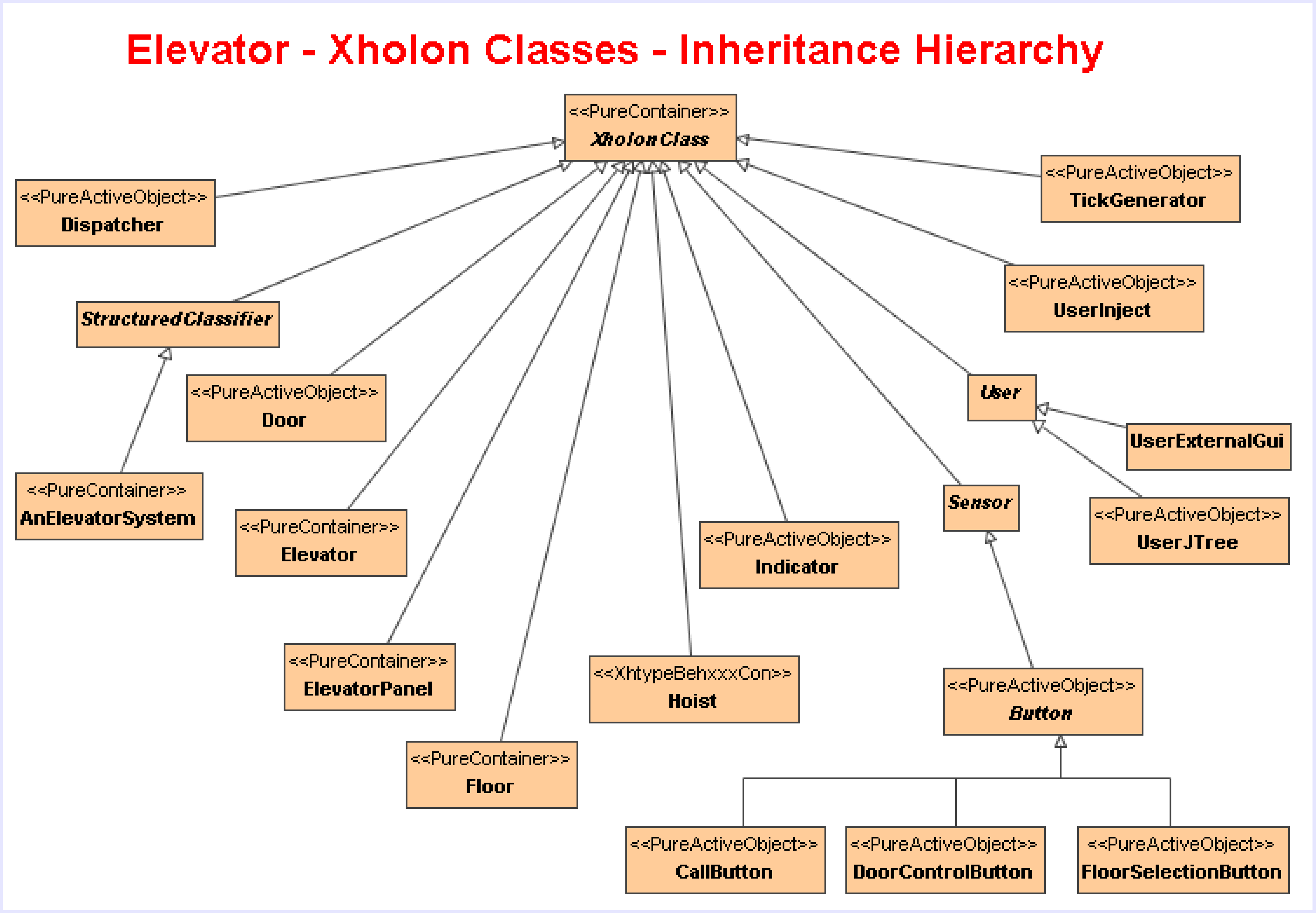
(3) Compile the generated Java code.

(4) Execute and observe the generated application.

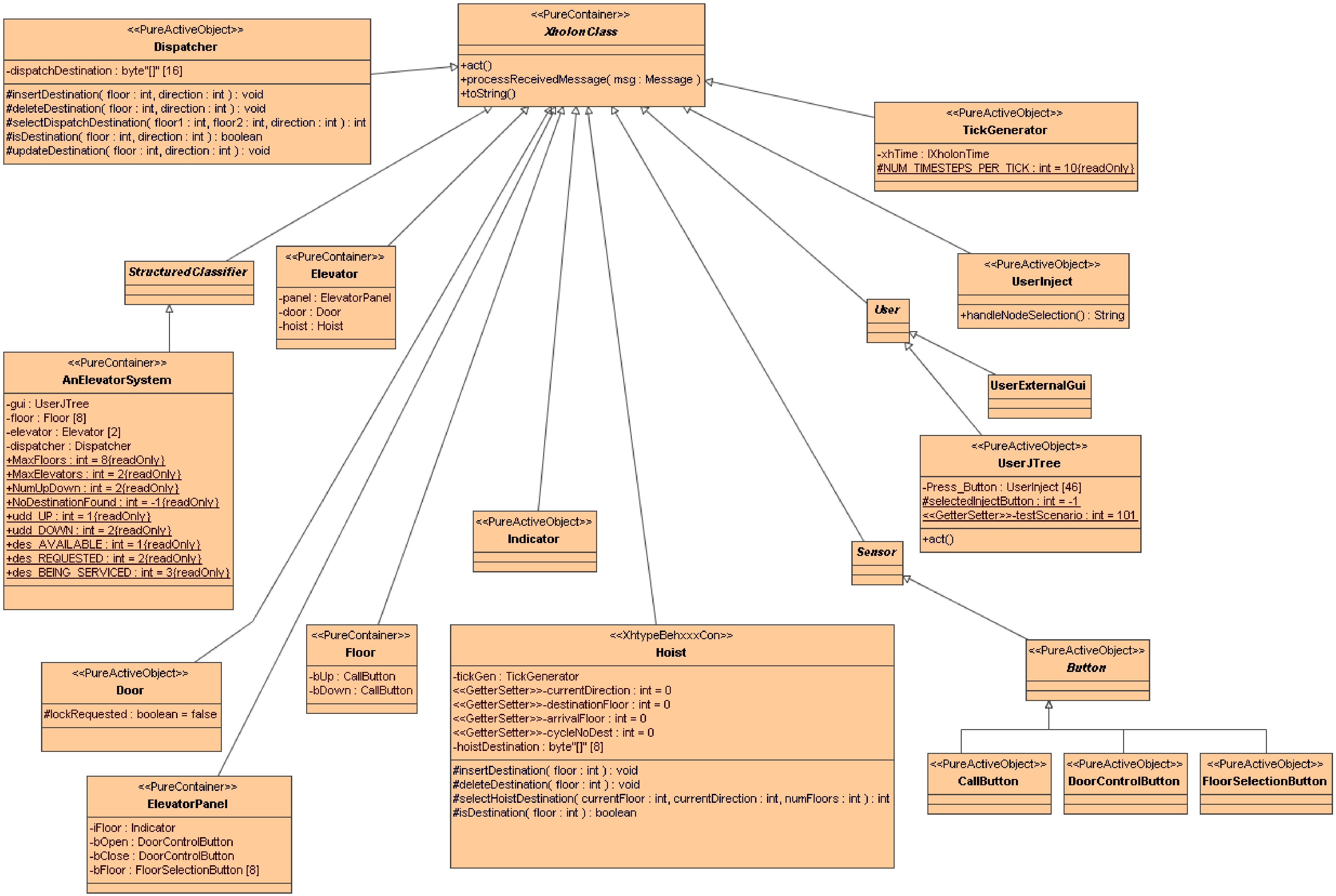
Elevator System

- The demo application is a simulation of a simple elevator system.

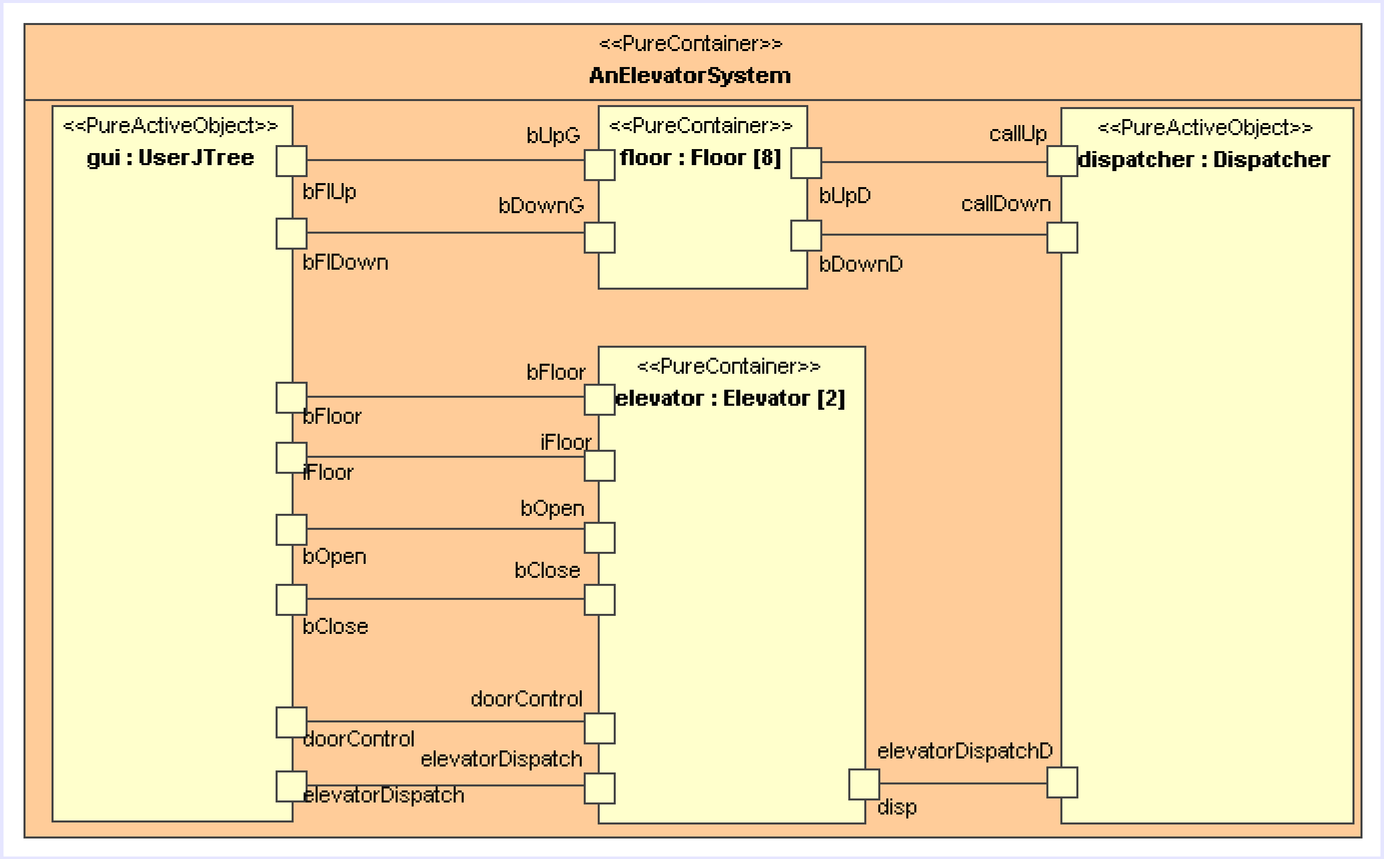
Inheritance Hierarchy


Elevator - Xholon Classes - Inheritance Hierarchy
<<PureContainer>> XholonClass
<<PureActiveObject>> Dispatcher
<<PureActiveObject>> TickGenerator
<<PureActiveObject>> UserInject
StructuredClassifier
<<PureActiveObject>> Door
User
UserExternalGui
<<PureContainer>> AnElevatorSystem
<<PureContainer>> Elevator
Sensor
<<PureActiveObject>> UserJTree
<<PureActiveObject>> Indicator
<<PureContainer>> ElevatorPanel
<<XhtypeBehxxxCon>> Hoist
<<PureActiveObject>> Button
<<PureContainer>> Floor
<<PureActiveObject>> CallButton
<<PureActiveObject>> DoorControlButton
<<PureActiveObject>> FloorSelectionButton

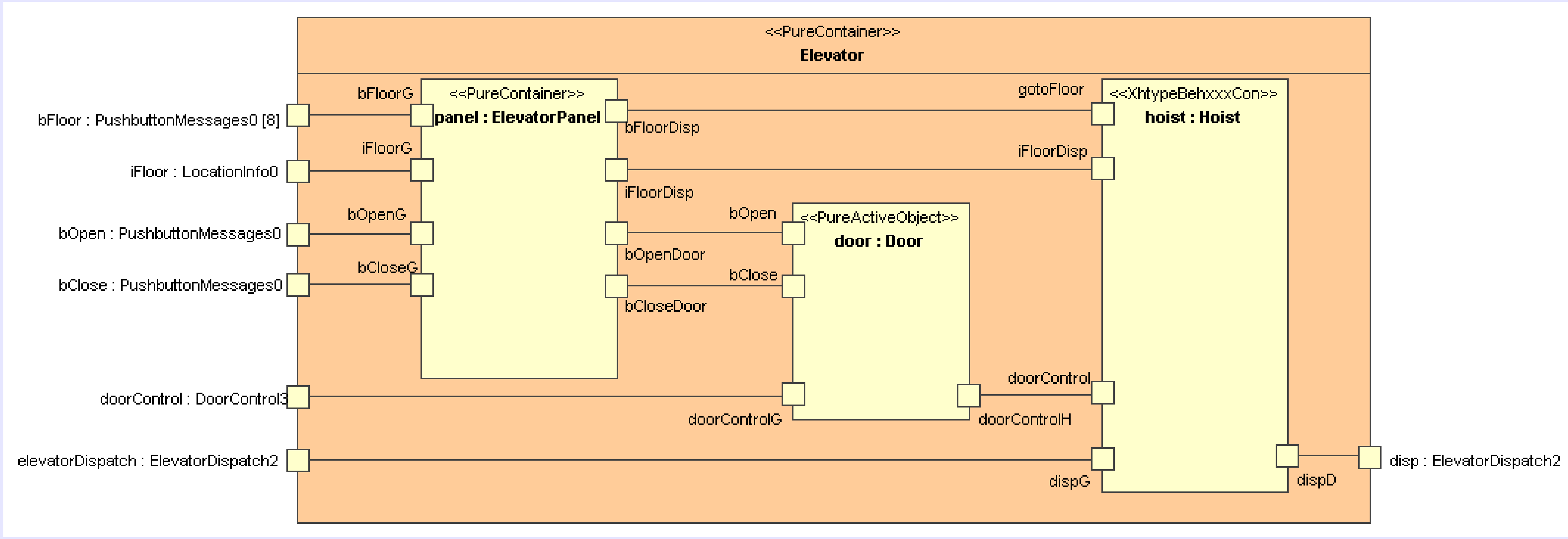
Elevator Xholon Classes


<<PureContainer>>
XholonClass
+act()
+processReceivedMessage(msg : Message)
+toString()
<<PureActiveObject>>
Dispatcher
-dispatchDestination : byte"[]" [16]
#insertDestination(floor : int, direction : int) : void
#deleteDestination(floor : int, direction : int) : void
#selectDispatchDestination(floor1 : int, floor2 : int, direction : int) : int
#isDestination(floor : int, direction : int) : boolean
#updateDestination(floor : int, direction : int) : void
<<PureActiveObject>>
TickGenerator
-xhTime : IXholonTime
#NUM_TIMESTEPS_PER_TICK : int = 10{readOnly}
StructuredClassifier
<<PureContainer>>
Elevator
-panel : ElevatorPanel
-door : Door
-hoist : Hoist
<<PureActiveObject>>
UserInject
+handleNodeSelection() : String
User
UserExternalGui
<<PureContainer>>
AnElevatorSystem
-gui : UserJTree
-floor : Floor [8]
-elevator : Elevator [2]
-dispatcher : Dispatcher
+MaxFloors : int = 8{readOnly}
+MaxElevators : int = 2{readOnly}
+NumUpDown : int = 2{readOnly}
+NoDestinationFound : int = -1{readOnly}
+udd_UP : int = 1{readOnly}
+udd_DOWN : int = 2{readOnly}
+des_AVAILABLE : int = 1{readOnly}
+des_REQUESTED : int = 2{readOnly}
+des_BEING_SERVICED : int = 3{readOnly}
<<PureActiveObject>>
UserJTree
-Press_Button : UserInject [46]
#selectedInjectButton : int = -1
<<GetterSetter>>-testScenario : int = 101
+act()
<<PureActiveObject>>
Indicator
Sensor
<<PureActiveObject>>
Door
#lockRequested : boolean = false
<<PureContainer>>
Floor
-bUp : CallButton
-bDown : CallButton
<<XhtypeBehxxxCon>>
Hoist
-tickGen : TickGenerator
<<GetterSetter>>-currentDirection : int = 0
<<GetterSetter>>-destinationFloor : int = 0
<<GetterSetter>>-arrivalFloor : int = 0
<<GetterSetter>>-cycleNoDest : int = 0
-hoistDestination : byte"[]" [8]
#insertDestination(floor : int) : void
#deleteDestination(floor : int) : void
#selectHoistDestination(currentFloor : int, currentDirection : int, numFloors : int) : int
#isDestination(floor : int) : boolean
<<PureActiveObject>>
Button
<<PureContainer>>
ElevatorPanel
-iFloor : Indicator
-bOpen : DoorControlButton
-bClose : DoorControlButton
-bFloor : FloorSelectionButton [8]
<<PureActiveObject>>
CallButton
<<PureActiveObject>>
DoorControlButton
<<PureActiveObject>>
FloorSelectionButton

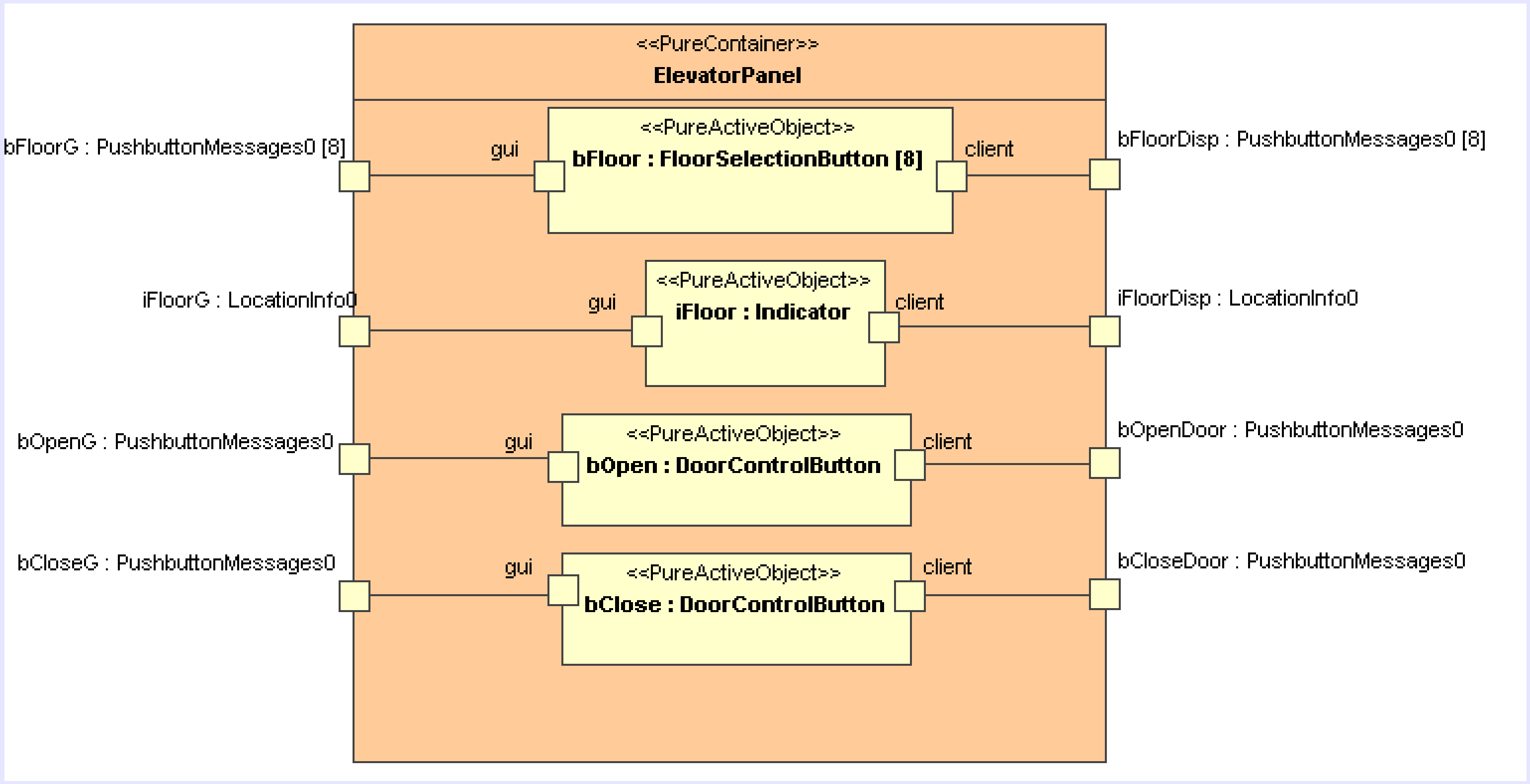
Overall composite structure



Elevator composite structure

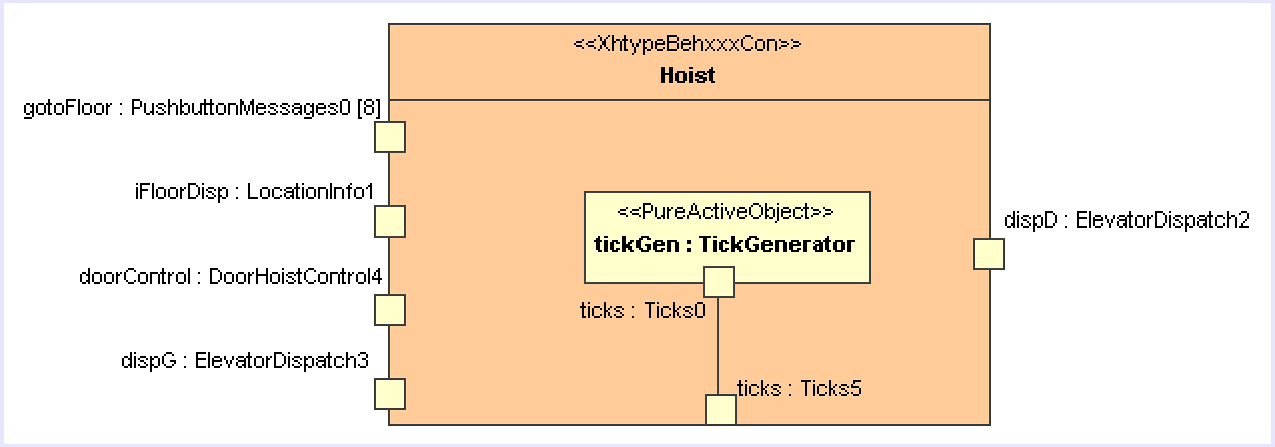


Elevator Panel composite structure



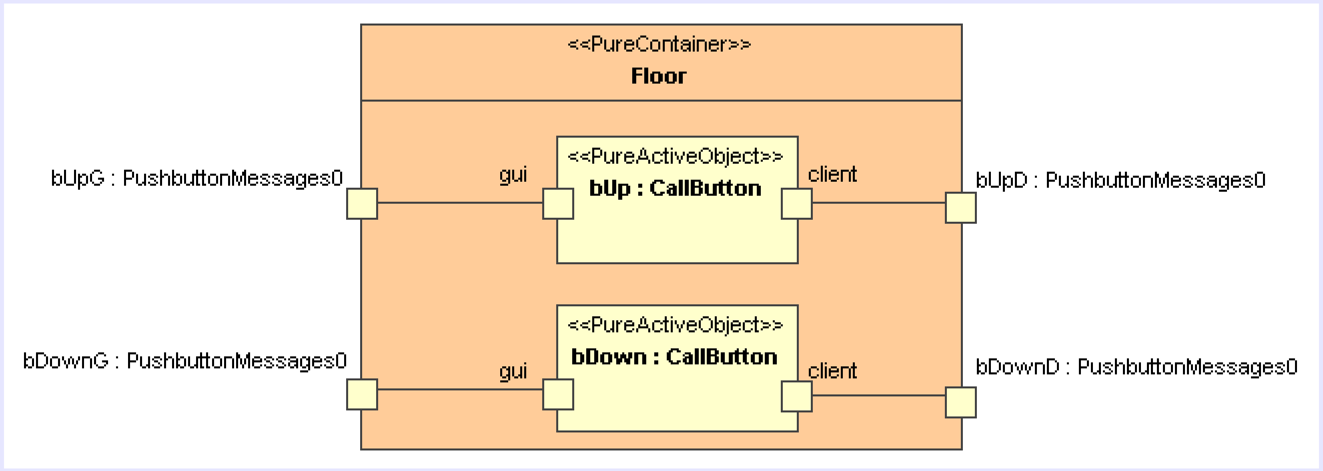
<<PureContainer>>
ElevatorPanel
bFloorG : PushbuttonMessages0 [8]
gui
<<PureActiveObject>>
bFloor : FloorSelectionButton [8]
client
bFloorDisp : PushbuttonMessages0 [8]
iFloorG : LocationInfo0
gui
<<PureActiveObject>>
iFloor : Indicator
client
iFloorDisp : LocationInfo0
bOpenG : PushbuttonMessages0
gui
<<PureActiveObject>>
bOpen : DoorControlButton
client
bOpenDoor : PushbuttonMessages0
bCloseG : PushbuttonMessages0
gui
<<PureActiveObject>>
bClose : DoorControlButton
client
bCloseDoor : PushbuttonMessages0

Hoist composite structure

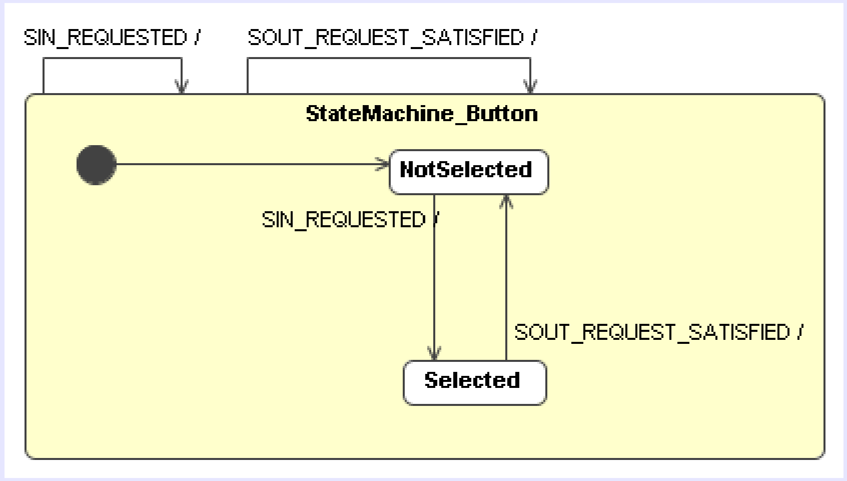


<<XhtypeBehxxxCon>>
Hoist
gotoFloor : PushbuttonMessages0 [8]
iFloorDisp : LocationInfo1
doorControl : DoorHoistControl4
dispG : ElevatorDispatch3
<<PureActiveObject>>
tickGen : TickGenerator
ticks : Ticks0
ticks : Ticks5
dispD : ElevatorDispatch2

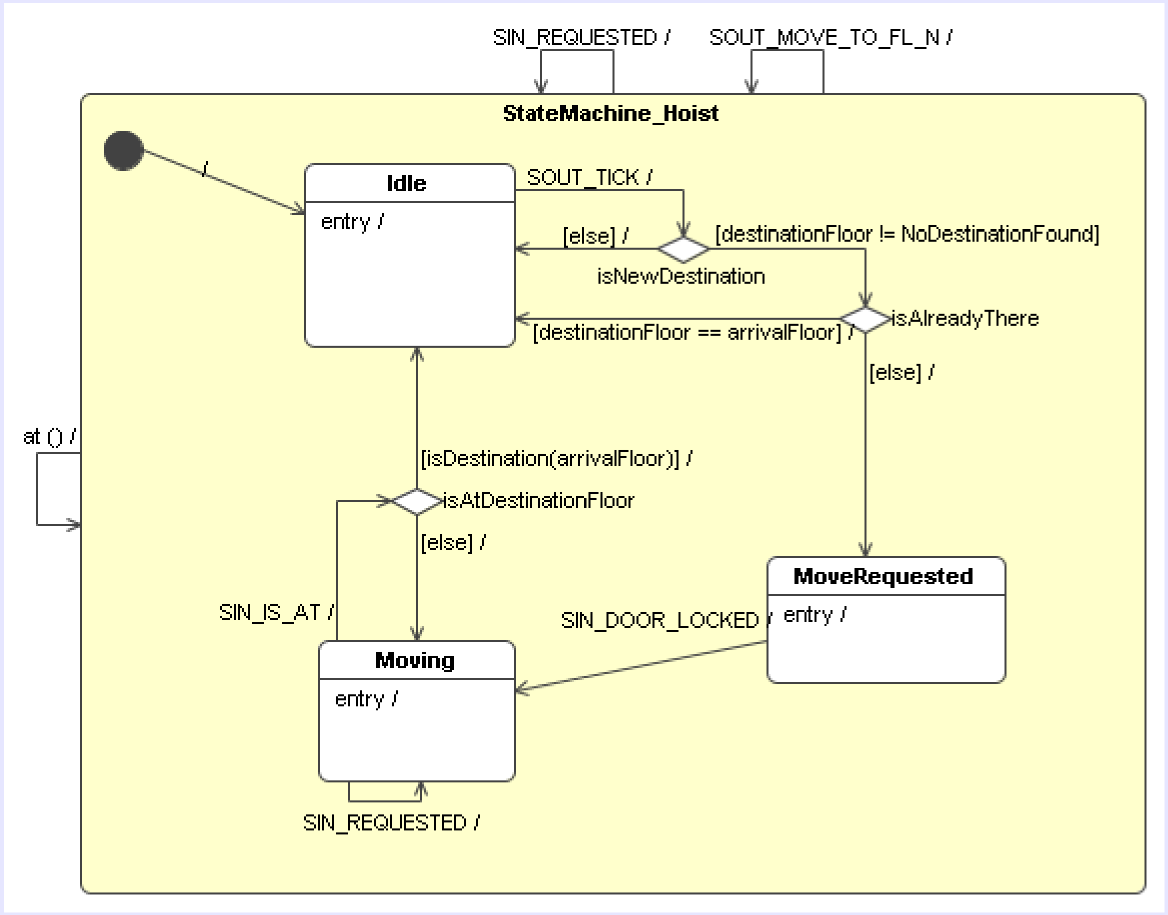
Floor composite structure



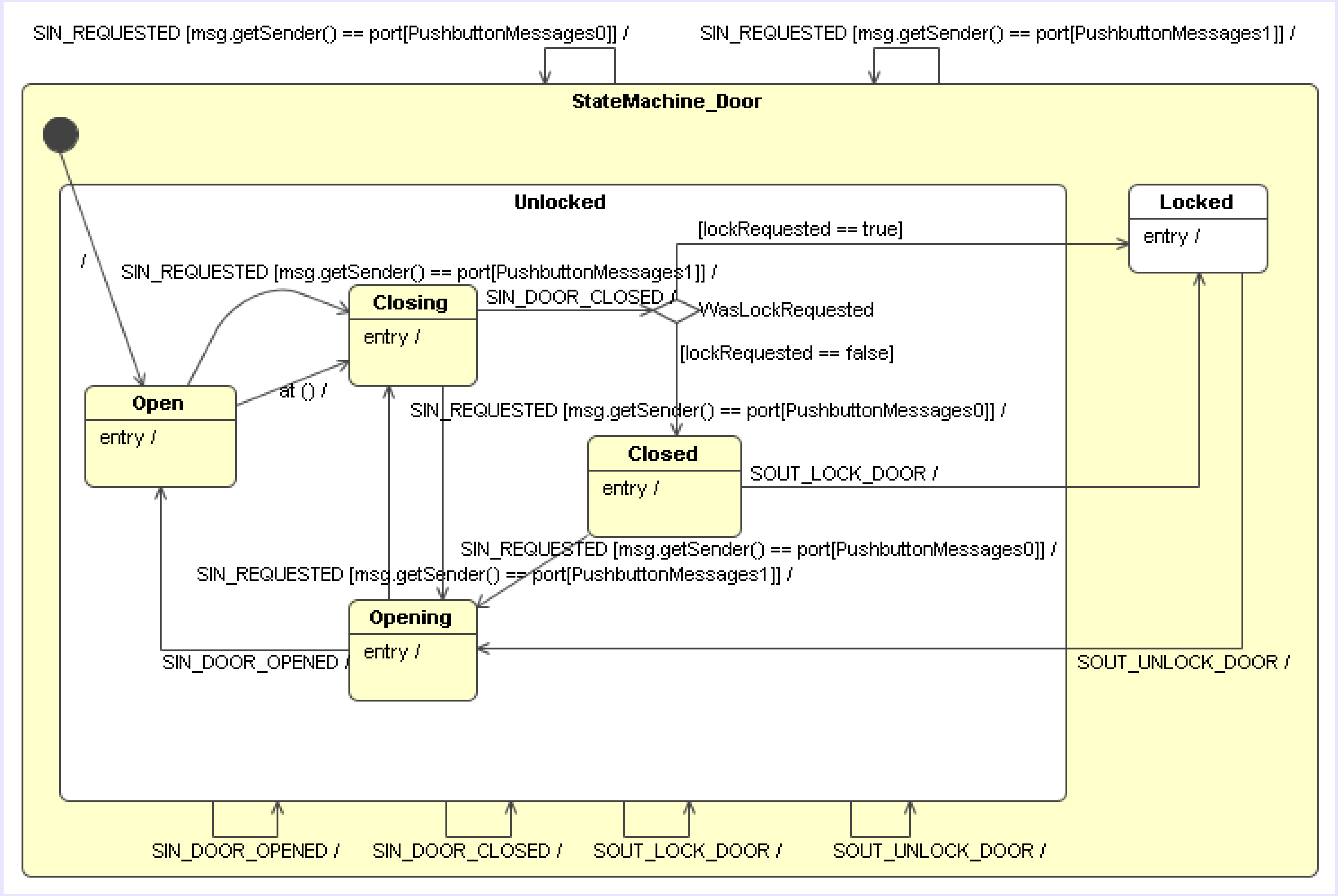
Button FSM


SIN_REQUESTED /
SOUT_REQUEST_SATISFIED /
StateMachine_Button
NotSelected
SIN_REQUESTED /
SOUT_REQUEST_SATISFIED /
Selected

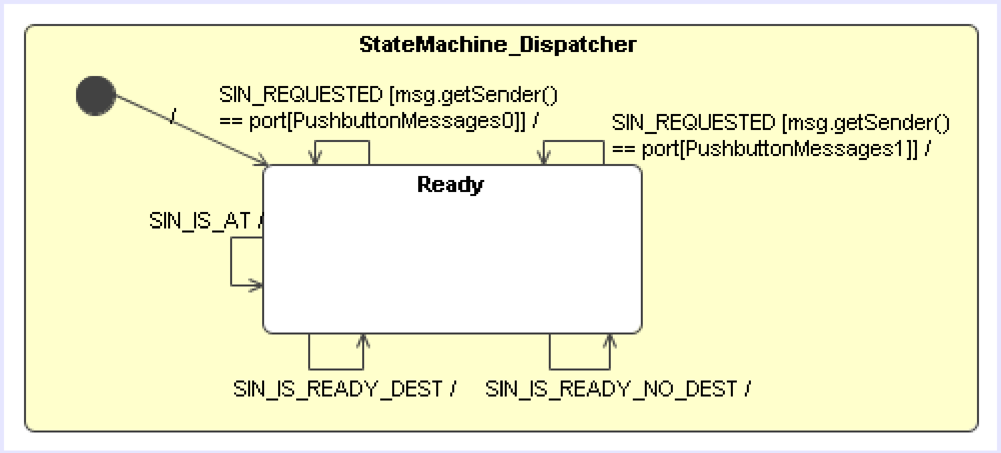
Hoist FSM


SIN_REQUESTED /
SOUT_MOVE_TO_FL_N /
StateMachine_Hoist
/
Idle
entry /
SOUT_TICK /
[else] /
[destinationFloor != NoDestinationFound]
isNewDestination
isAlreadyThere
[destinationFloor == arrivalFloor] /
[else] /
at () /
[isDestination(arrivalFloor)] /
isAtDestinationFloor
[else] /
SIN_IS_AT /
MoveRequested
entry /
SIN_DOOR_LOCKED /
Moving
entry /
SIN_REQUESTED /

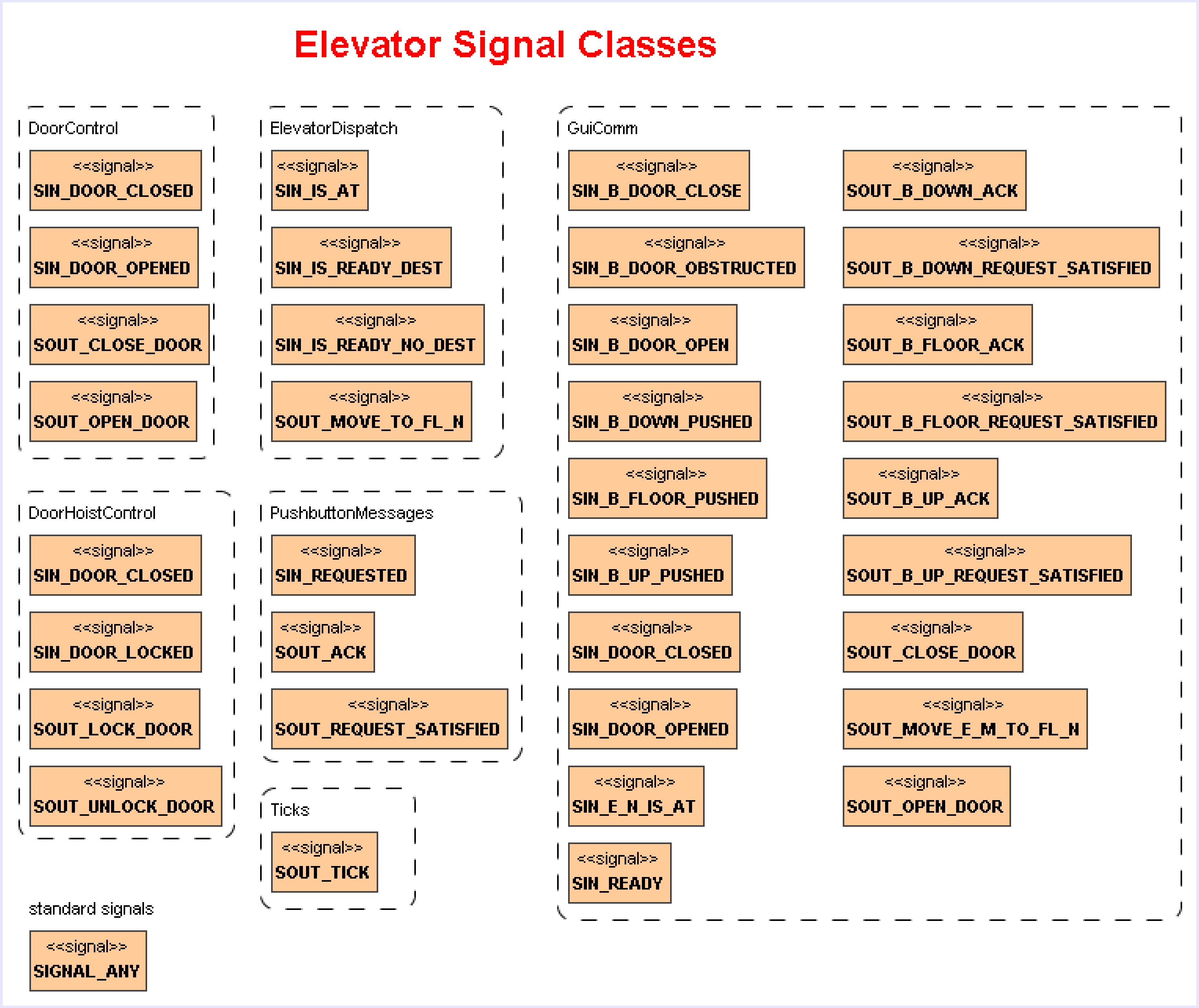
Door FSM


SIN_REQUESTED [msg.getSender() == port[PushbuttonMessages0]] /
SIN_REQUESTED [msg.getSender() == port[PushbuttonMessages1]] /
StateMachine_Door
Unlocked
Locked
entry /
[lockRequested == true]
/
SIN_REQUESTED [msg.getSender() == port[PushbuttonMessages1]] /
Closing
entry /
SIN_DOOR_CLOSED /
WasLockRequested
[lockRequested == false]
Open
entry /
at () /
SIN_REQUESTED [msg.getSender() == port[PushbuttonMessages0]] /
Closed
entry /
SOUT_LOCK_DOOR /
SIN_REQUESTED [msg.getSender() == port[PushbuttonMessages0]] /
SIN_REQUESTED [msg.getSender() == port[PushbuttonMessages1]] /
Opening
entry /
SIN_DOOR_OPENED /
SOUT_UNLOCK_DOOR /
SIN_DOOR_OPENED /
SIN_DOOR_CLOSED /
SOUT_LOCK_DOOR /
SOUT_UNLOCK_DOOR /

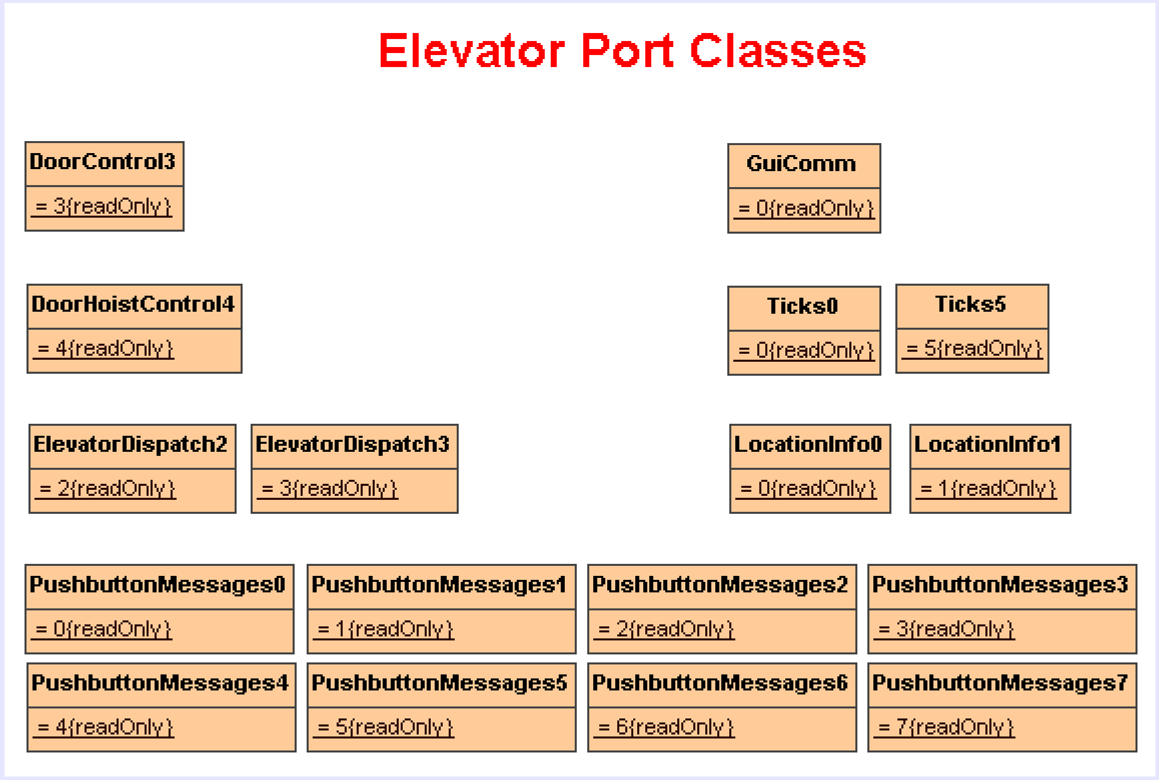
Dispatcher FSM


StateMachine_Dispatcher
/
SIN_REQUESTED [msg.getSender() == port[PushbuttonMessages0]] /
SIN_REQUESTED [msg.getSender() == port[PushbuttonMessages1]] /
Ready
SIN_IS_AT /
SIN_IS_READY_DEST /
SIN_IS_READY_NO_DEST /

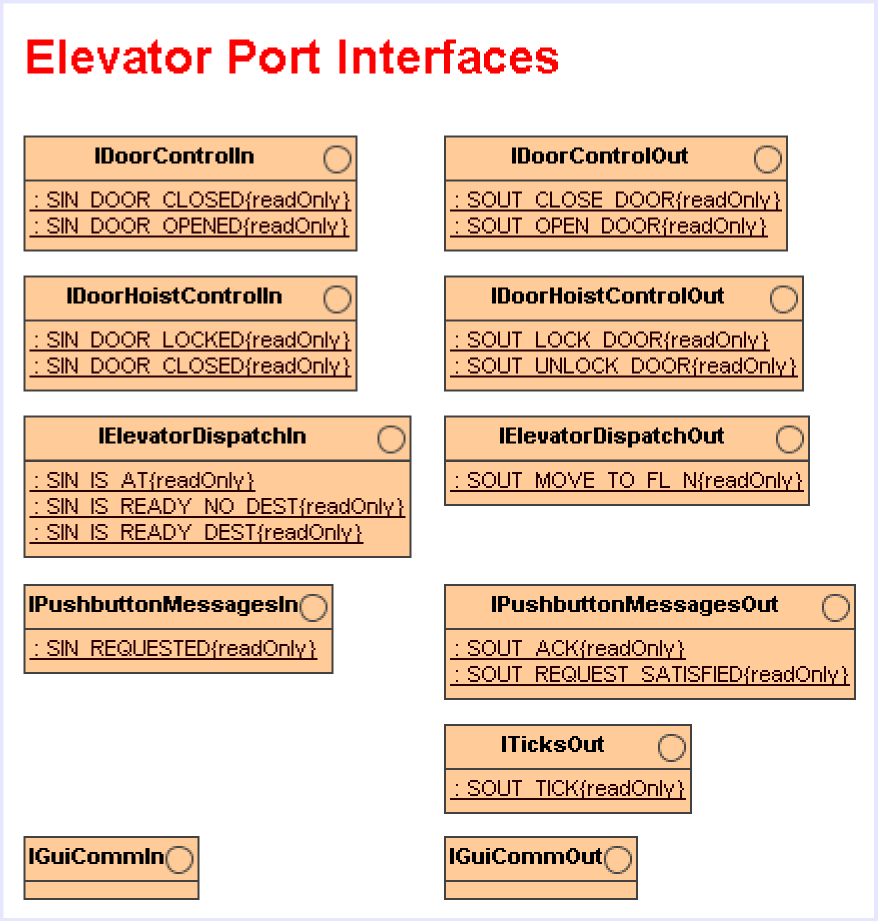
Signal Classes


Elevator Signal Classes
DoorControl
<<signal>> SIN_DOOR_CLOSED
<<signal>> SIN_DOOR_OPENED
<<signal>> SOUT_CLOSE_DOOR
<<signal>> SOUT_OPEN_DOOR
ElevatorDispatch
<<signal>> SIN_IS_AT
<<signal>> SIN_IS_READY_DEST
<<signal>> SIN_IS_READY_NO_DEST
<<signal>> SOUT_MOVE_TO_FL_N
GuiComm
<<signal>> SIN_B_DOOR_CLOSE
<<signal>> SIN_B_DOOR_OBSTRUCTED
<<signal>> SIN_B_DOOR_OPEN
<<signal>> SIN_B_DOWN_PUSHED
<<signal>> SIN_B_FLOOR_PUSHED
<<signal>> SIN_B_UP_PUSHED
<<signal>> SIN_DOOR_CLOSED
<<signal>> SIN_DOOR_OPENED
<<signal>> SIN_E_N_IS_AT
<<signal>> SIN_READY
<<signal>> SOUT_B_DOWN_ACK
<<signal>> SOUT_B_DOWN_REQUEST_SATISFIED
<<signal>> SOUT_B_FLOOR_ACK
<<signal>> SOUT_B_FLOOR_REQUEST_SATISFIED
<<signal>> SOUT_B_UP_ACK
<<signal>> SOUT_B_UP_REQUEST_SATISFIED
<<signal>> SOUT_CLOSE_DOOR
<<signal>> SOUT_MOVE_E_M_TO_FL_N
<<signal>> SOUT_OPEN_DOOR
DoorHoistControl
<<signal>> SIN_DOOR_CLOSED
<<signal>> SIN_DOOR_LOCKED
<<signal>> SOUT_LOCK_DOOR
<<signal>> SOUT_UNLOCK_DOOR
PushbuttonMessages
<<signal>> SIN_REQUESTED
<<signal>> SOUT_ACK
<<signal>> SOUT_REQUEST_SATISFIED
Ticks
<<signal>> SOUT_TICK
standard signals
<<signal>> SIGNAL_ANY

Port Classes

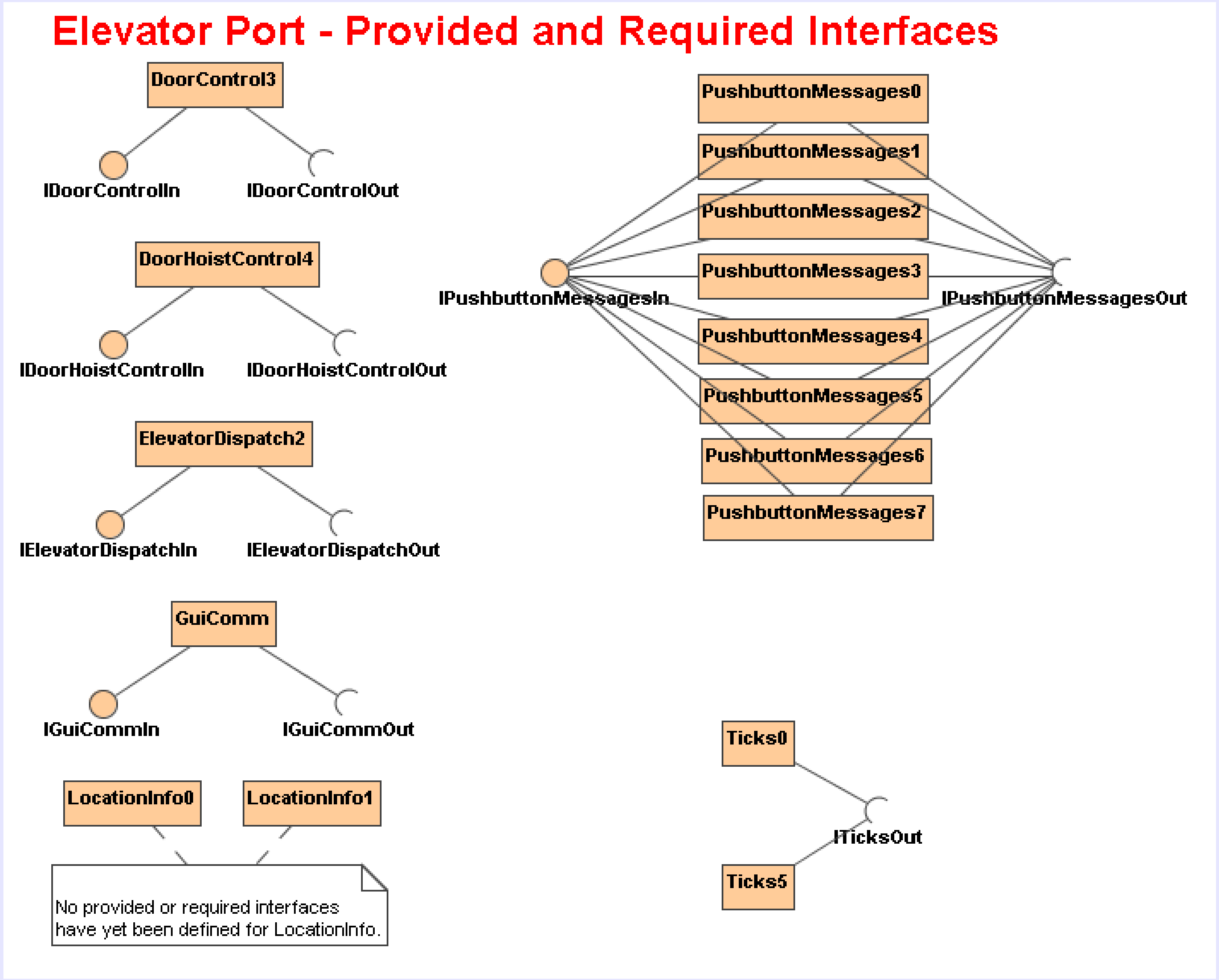


Interfaces

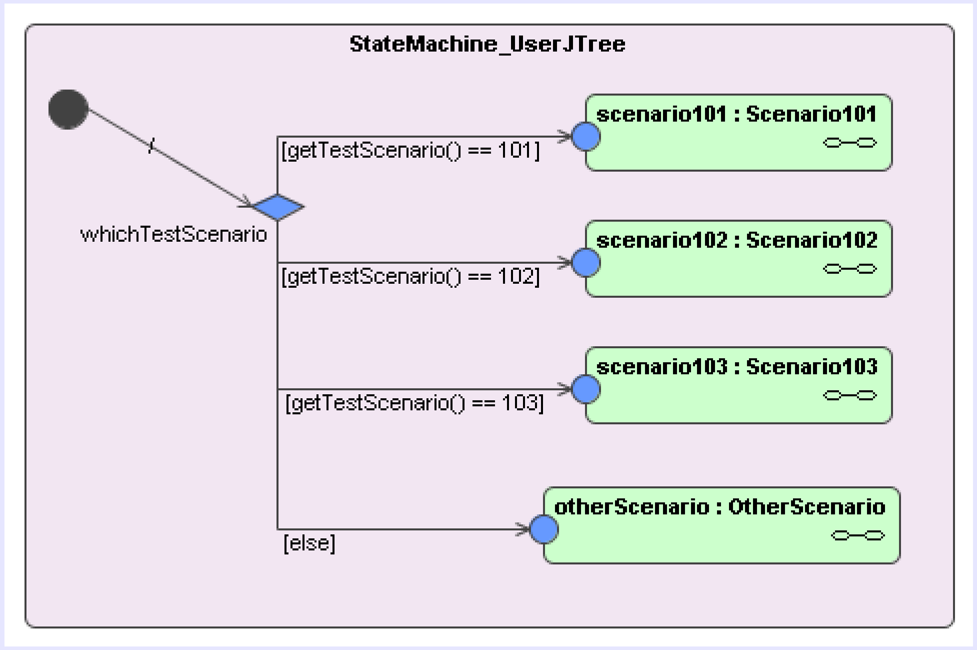


Elevator Port Interfaces
IDoorControlIn
: SIN_DOOR_CLOSED{readOnly}
: SIN_DOOR_OPENED{readOnly}
IDoorControlOut
: SOUT_CLOSE_DOOR{readOnly}
: SOUT_OPEN_DOOR{readOnly}
IDoorHoistControlIn
: SIN_DOOR_LOCKED{readOnly}
: SIN_DOOR_CLOSED{readOnly}
IDoorHoistControlOut
: SOUT_LOCK_DOOR{readOnly}
: SOUT_UNLOCK_DOOR{readOnly}
IElevatorDispatchIn
: SIN_IS_AT{readOnly}
: SIN_IS_READY_NO_DEST{readOnly}
: SIN_IS_READY_DEST{readOnly}
IElevatorDispatchOut
: SOUT_MOVE_TO_FL_N{readOnly}
IPushbuttonMessagesIn
: SIN_REQUESTED{readOnly}
IPushbuttonMessagesOut
: SOUT_ACK{readOnly}
: SOUT_REQUEST_SATISFIED{readOnly}
ITicksOut
: SOUT_TICK{readOnly}
IGuiCommIn
IGuiCommOut

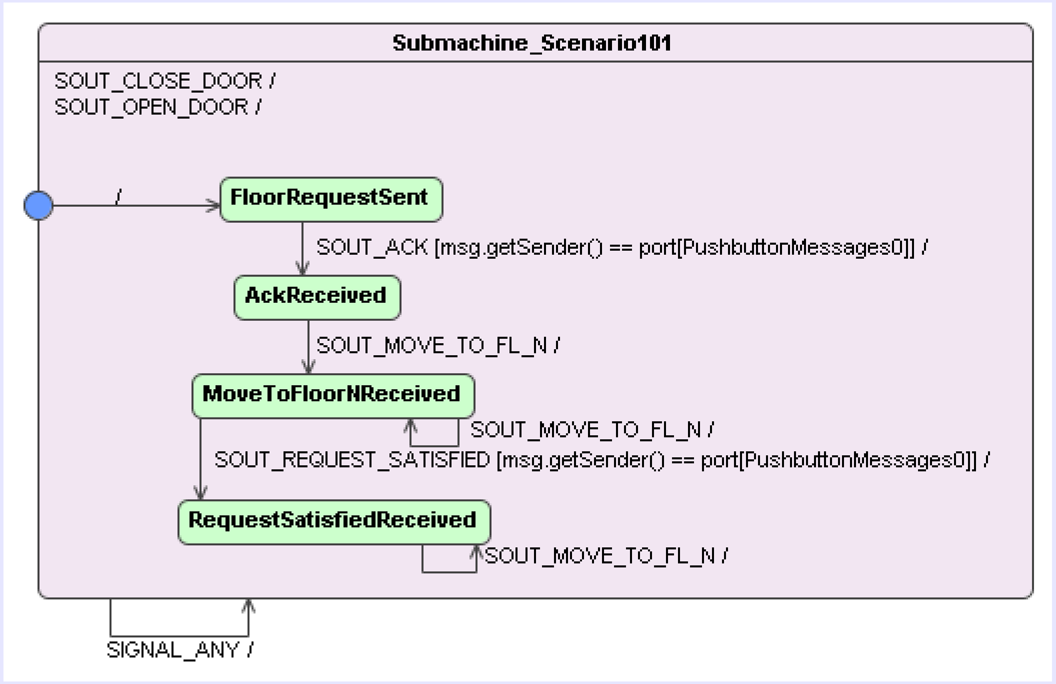
Provided & Required Interfaces



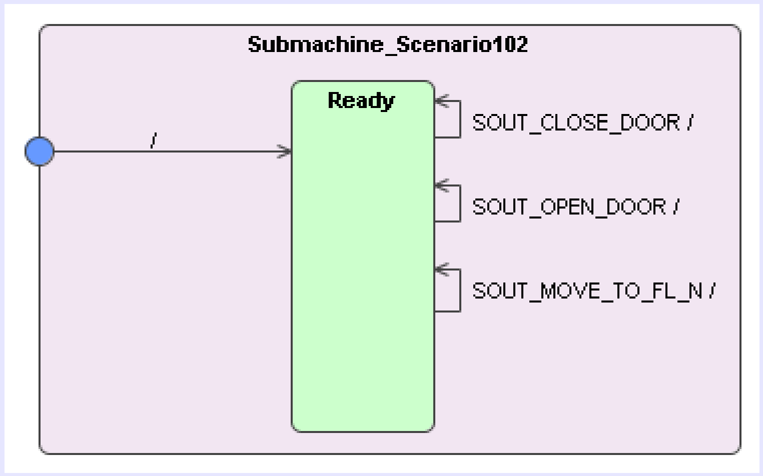
Test Harness FSM

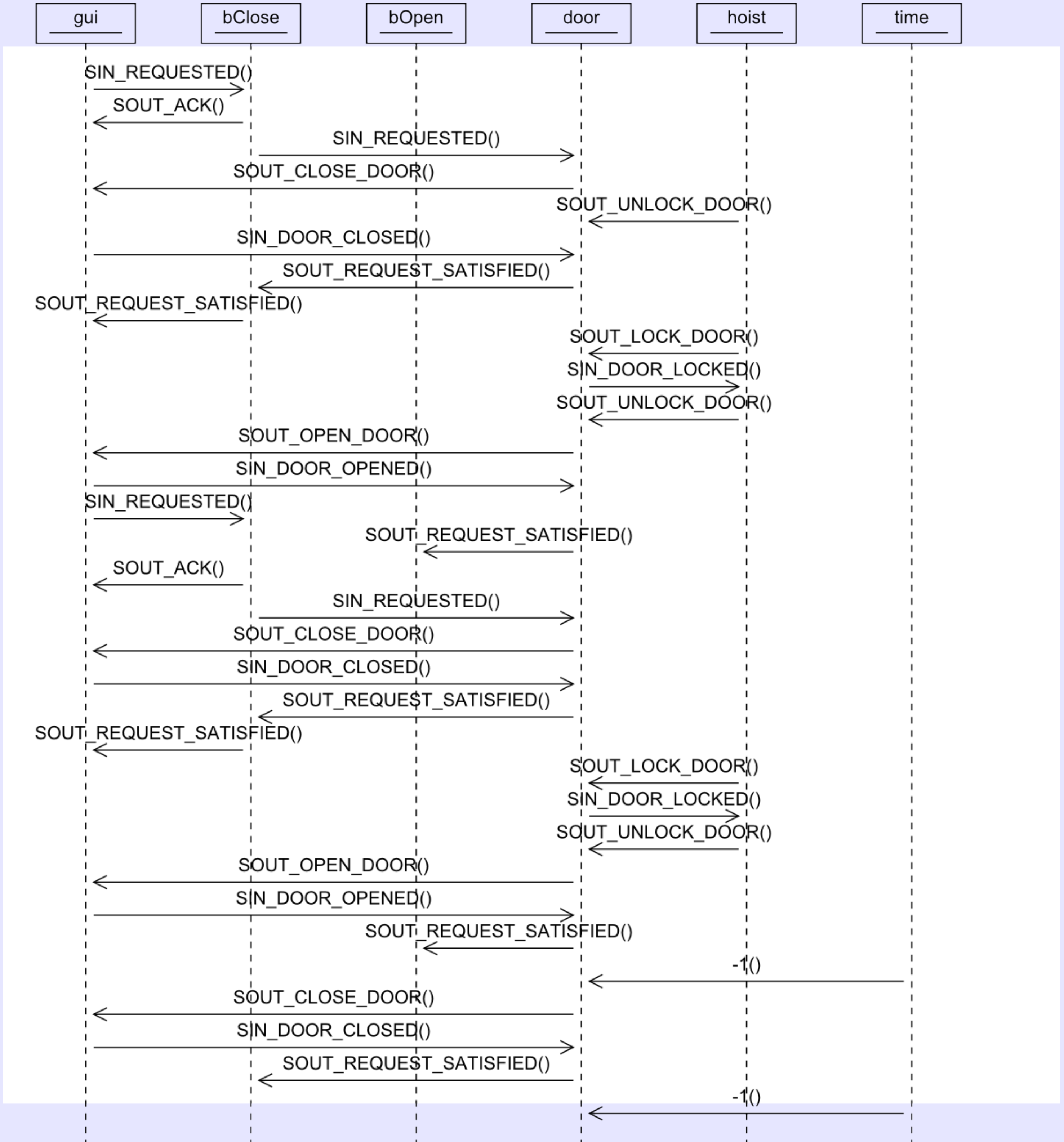

StateMachine_UserJTree
/
whichTestScenario
[getTestScenario() == 101]
scenario101 : Scenario101
[getTestScenario() == 102]
scenario102 : Scenario102
[getTestScenario() == 103]
scenario103 : Scenario103
[else]
otherScenario : OtherScenario

Test Scenario 1

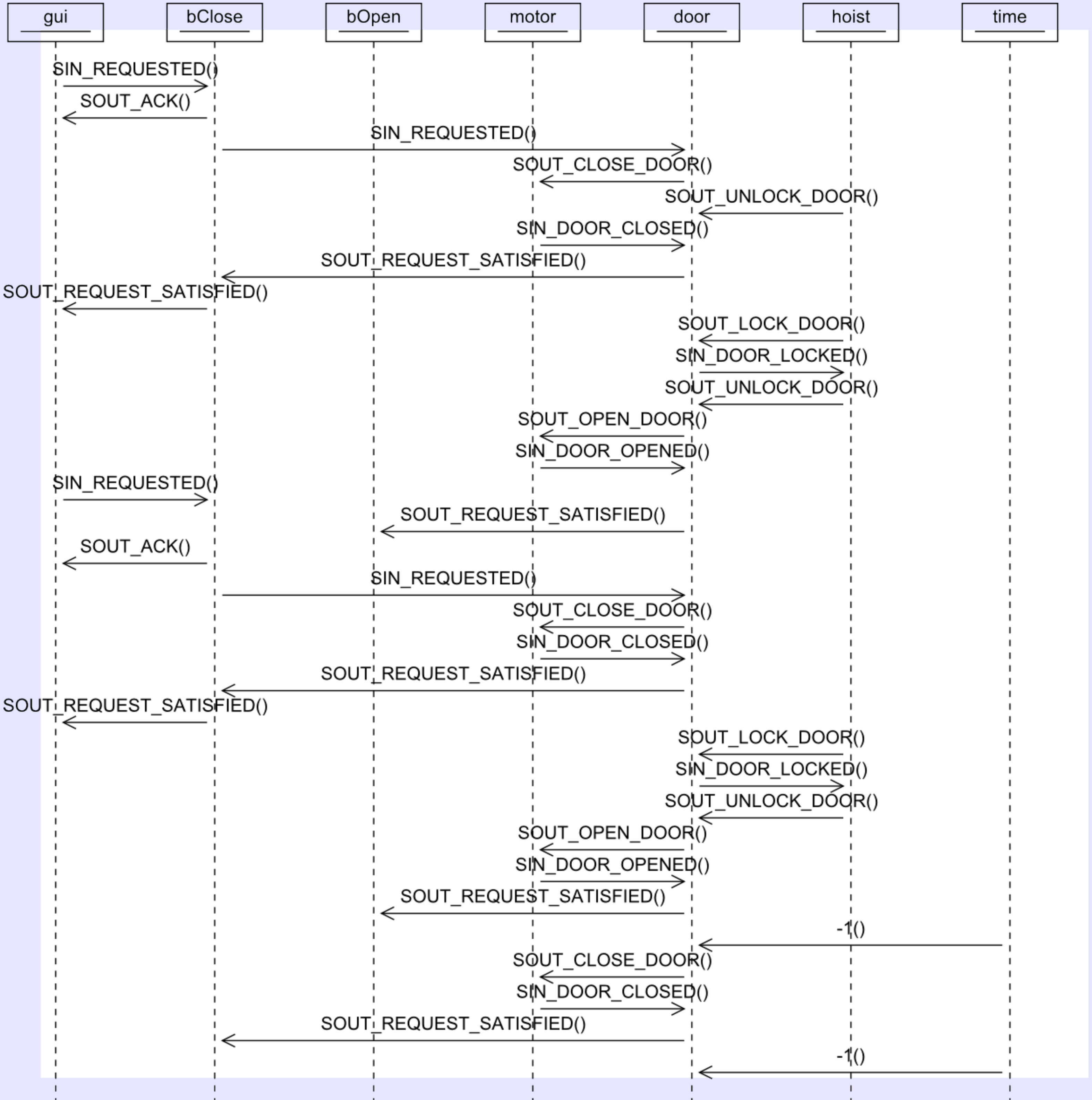


Test Scenario 2

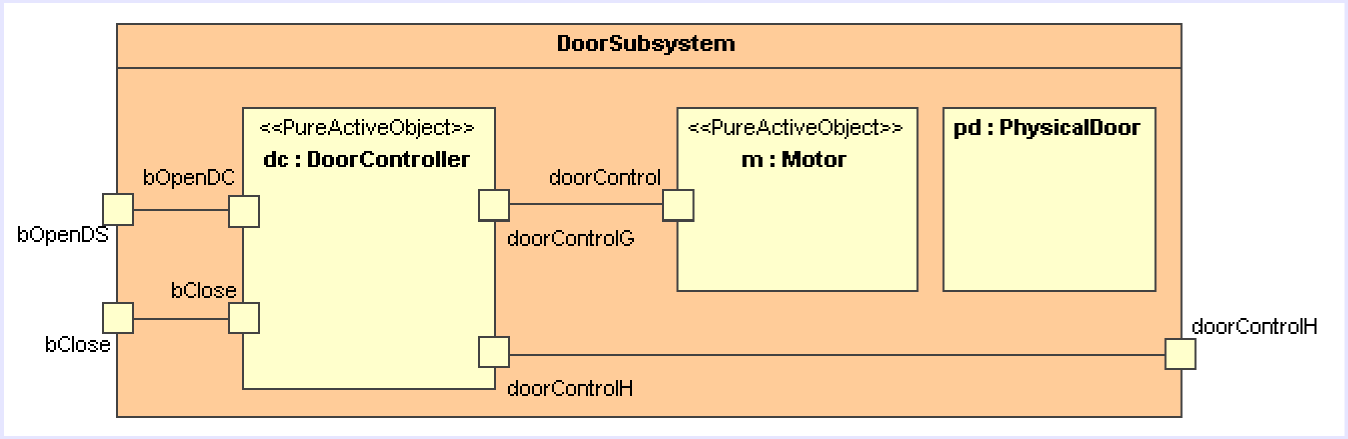



gui
bClose
bOpen
door
hoist
time
SIN_REQUESTED()
SOUT_ACK()
SIN_REQUESTED()
SOUT_CLOSE_DOOR()
SOUT_UNLOCK_DOOR()
SIN_DOOR_CLOSED()
SOUT_REQUEST_SATISFIED()
SOUT_REQUEST_SATISFIED()
SOUT_LOCK_DOOR()
SIN_DOOR_LOCKED()
SOUT_UNLOCK_DOOR()
SOUT_OPEN_DOOR()
SIN_DOOR_OPENED()
SIN_REQUESTED()
SOUT_REQUEST_SATISFIED()
SOUT_ACK()
SIN_REQUESTED()
SOUT_CLOSE_DOOR()
SIN_DOOR_CLOSED()
SOUT_REQUEST_SATISFIED()
SOUT_REQUEST_SATISFIED()
SOUT_LOCK_DOOR()
SIN_DOOR_LOCKED()
SOUT_UNLOCK_DOOR()
SOUT_OPEN_DOOR()
SIN_DOOR_OPENED()
SOUT_REQUEST_SATISFIED()
-1()
SOUT_CLOSE_DOOR()
SIN_DOOR_CLOSED()
SOUT_REQUEST_SATISFIED()
-1()

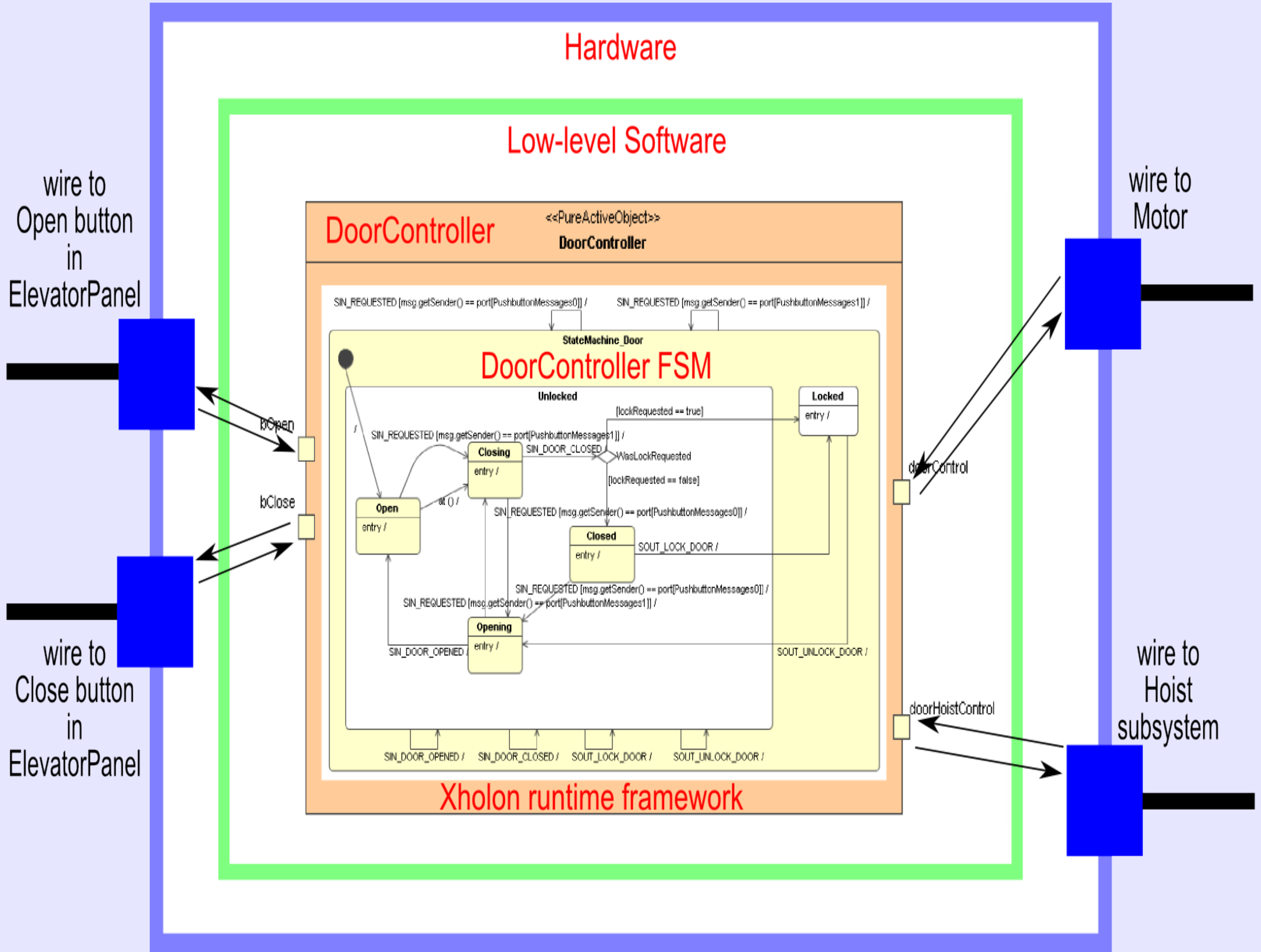
Modified Door


gui
bClose
bOpen
motor
door
hoist
time
SIN_REQUESTED()
SOUT_ACK()
SIN_REQUESTED()
SOUT_CLOSE_DOOR()
SOUT_UNLOCK_DOOR()
SIN_DOOR_CLOSED()
SOUT_REQUEST_SATISFIED()
SOUT_REQUEST_SATISFIED()
SOUT_LOCK_DOOR()
SIN_DOOR_LOCKED()
SOUT_UNLOCK_DOOR()
SOUT_OPEN_DOOR()
SIN_DOOR_OPENED()
SIN_REQUESTED()
SOUT_REQUEST_SATISFIED()
SOUT_ACK()
SIN_REQUESTED()
SOUT_CLOSE_DOOR()
SIN_DOOR_CLOSED()
SOUT_REQUEST_SATISFIED()
SOUT_REQUEST_SATISFIED()
SOUT_LOCK_DOOR()
SIN_DOOR_LOCKED()
SOUT_UNLOCK_DOOR()
SOUT_OPEN_DOOR()
SIN_DOOR_OPENED()
SOUT_REQUEST_SATISFIED()
-1()
SOUT_CLOSE_DOOR()
SIN_DOOR_CLOSED()
SOUT_REQUEST_SATISFIED()
-1()

DoorSubsystem composite structure



DoorController embedded in Hardware



Embedded Java - possibilities

- Java Micro Edition (J2ME)
  - Connected Device Configuration (CDC)
  - Connected Limited Device Configuration (CLDC), Mobile Information Device Profile (MIDP)
- Gnu Compiler for Java (gcj)
- Robots
  - Lego Mindstorms NXT, Lejos/iCommand open-source
- More traditional embedded systems; no GUI.
- Will focus on cell phone or Palm PDA for now.

Wrapup

- Xholon project needs developers.
  - Embedded version for Java J2ME.
- Can be downloaded from: http://sourceforge.net/projects/xholon/
- Project web site: http://www.primordion.com/Xholon
- Simple tutorials:
  - http://www.primordion.com/Xholon/doc/tutorial.html
  - http://www.primordion.com/Xholon/doc/tutorial_helloworld_md.html
- Questions?

Bio

Ken Webb has worked for over 25 years in the Ottawa high tech and consulting community. He was an employee of ObjecTime for several years where he became intimately familiar with the UML 2 concepts. He is the author of a published book, Oracle Distributed Systems: A C programmer's development guide. He has a BA in Cognitive Science from Carleton University, and an MSc from the Evolutionary and Adaptive Systems program (Informatics Department) at the Univerity of Sussex in England. He is currently the architect of the Xholon open-source project through his company Primordion.

Email: ken AT primordion.com