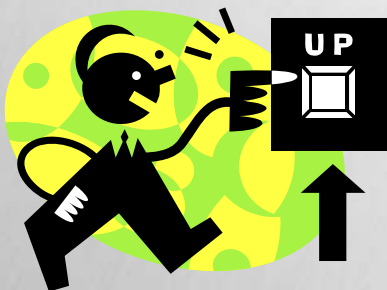




Elevator 2.0

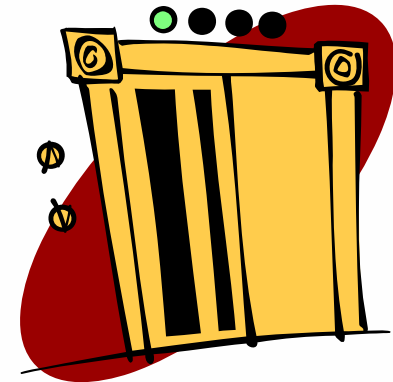
A DETAILED LOOK AT AN xtUML CASE STUDY FOR DEVELOPERS AND ARCHITECTS

MODEL INTEGRATION, LLC



The Elevator Project

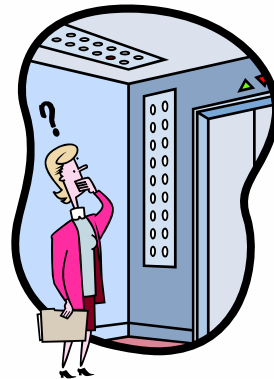
- + Build a complete working xtUML project
- + Demonstrate practical analysis and modeling techniques
- + Highlight good and bad in the development environment
- + Build GUI, equipment and platform independent models
- + Do it in our spare time



Why an Elevator?

Familiar application

Easy to reverse engineer



Multiple domains to explore

Stable requirements

Analysis and Modeling Goals

- + No manager classes
 -
- + Good domain separation
 -
- + Straightforward threads of control
 -
- + Use statecharts to model lifecycles
 -
- + Models are supported by analysis
 -
- + Data Driven
 -



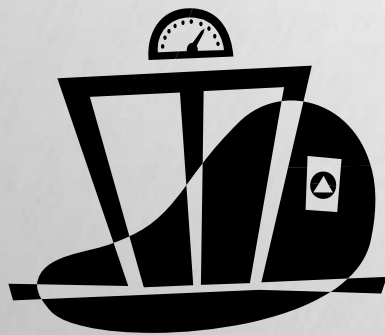
Coming up...

- + Quick Elevator 1.0 Trace
- + Tips and lessons in model improvement 1.0 -> 2.0
- + GUI Development and Demonstration
- + Initialization and Bridging
- + Testing
- + Next steps
- + Stuff you can download



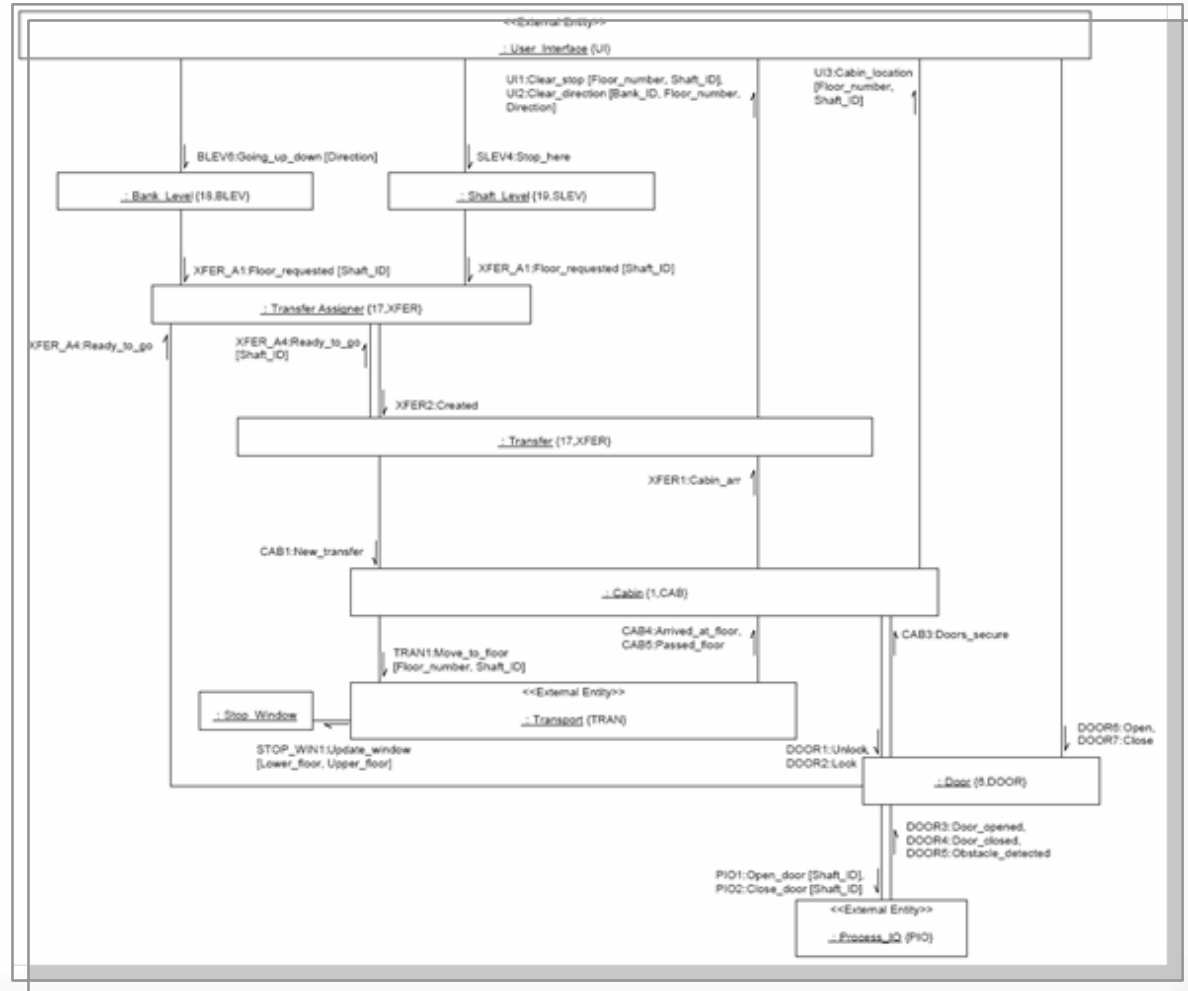


Elevator 1.0 Quick Trace

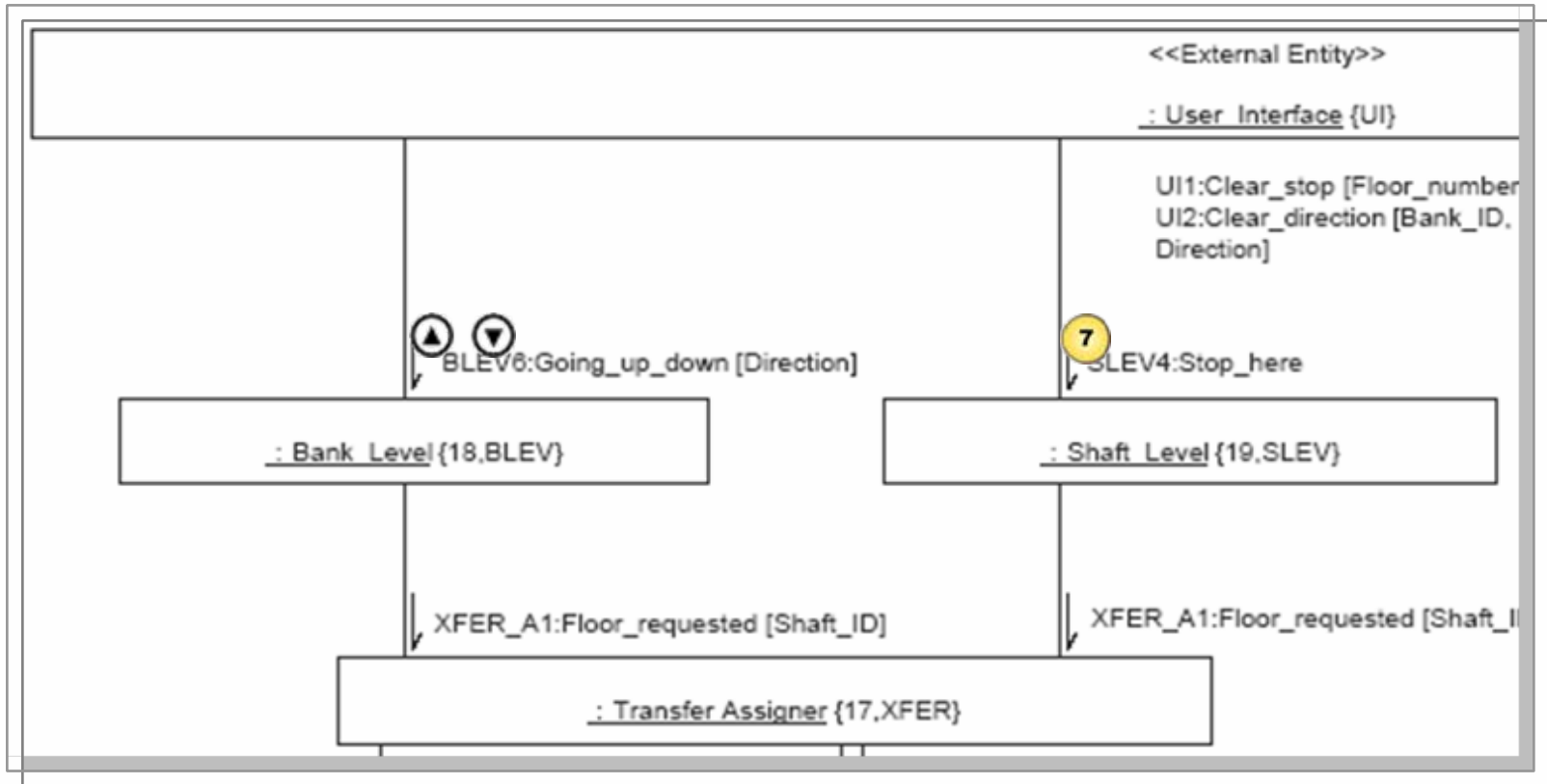


Elevator 1.0 Collaboration

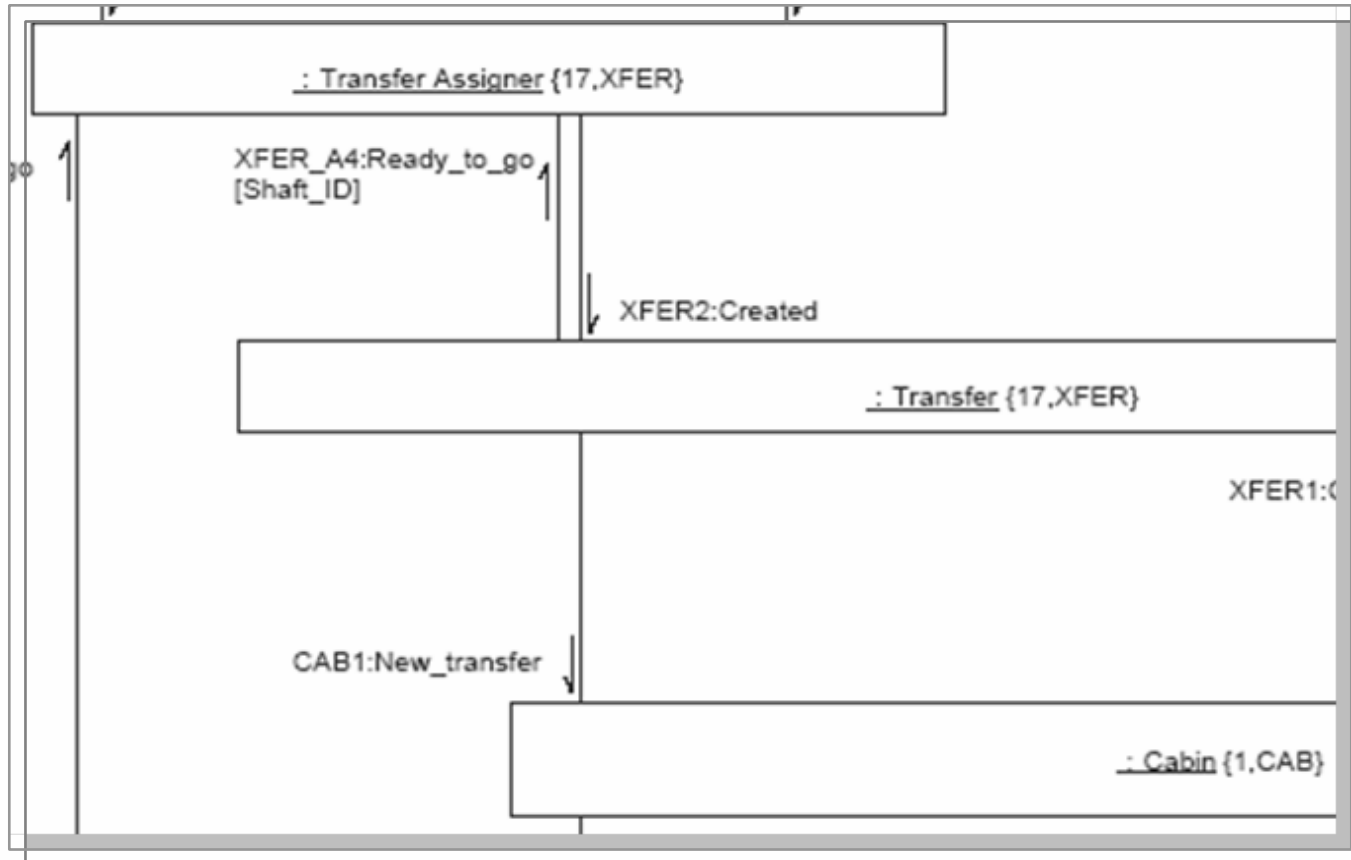
Limitations



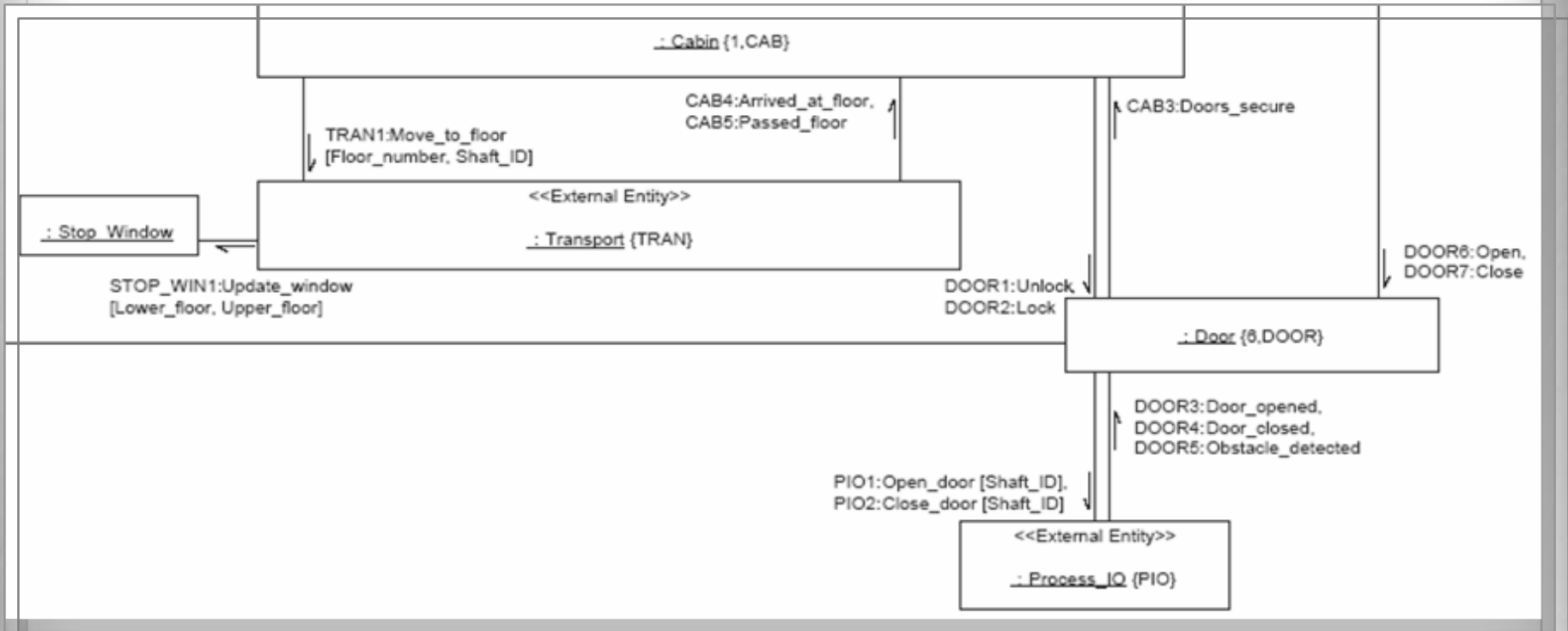
Passenger makes a request



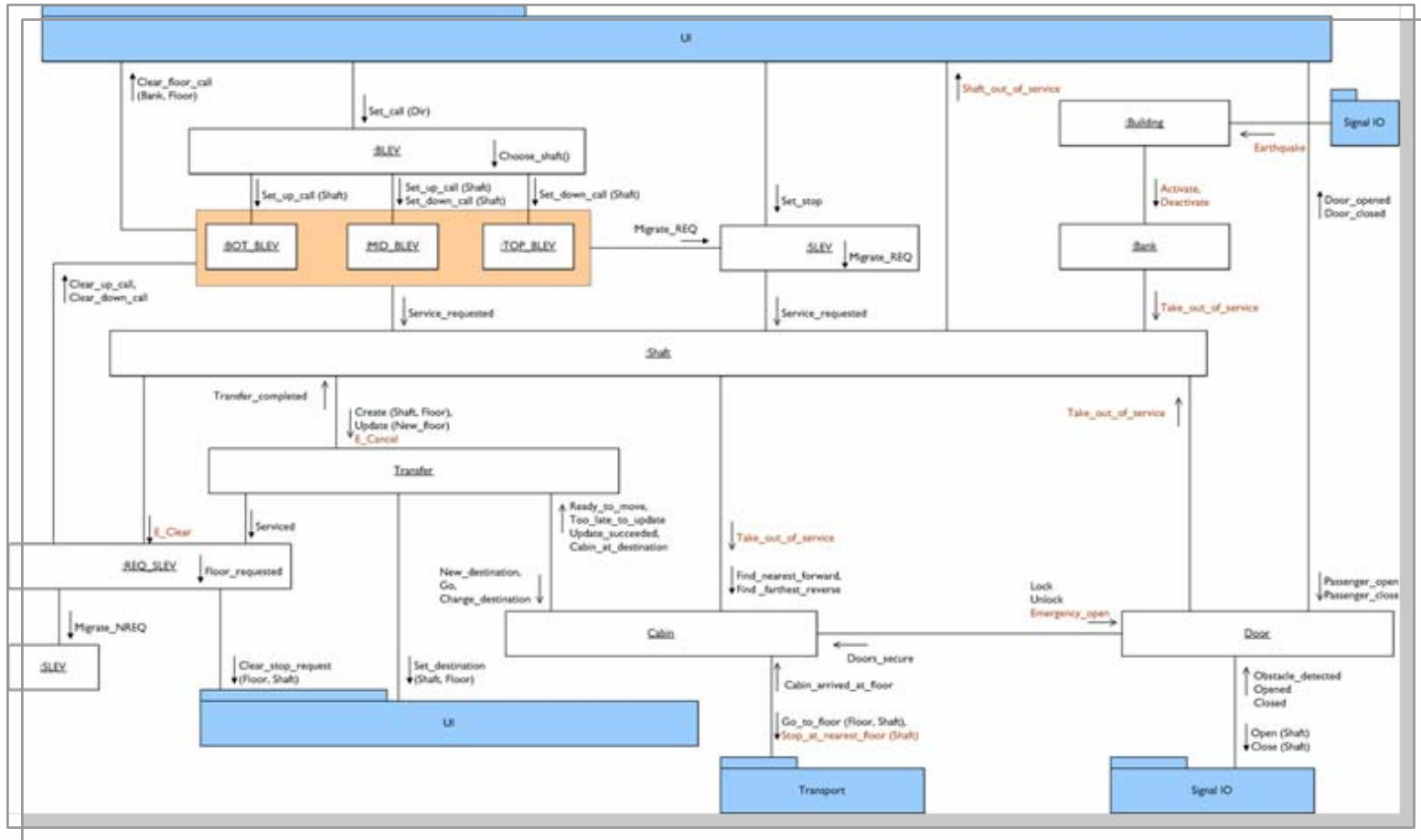
Dispatch the cabin



Move to floor and open up



Elevator 2.0 Collaboration

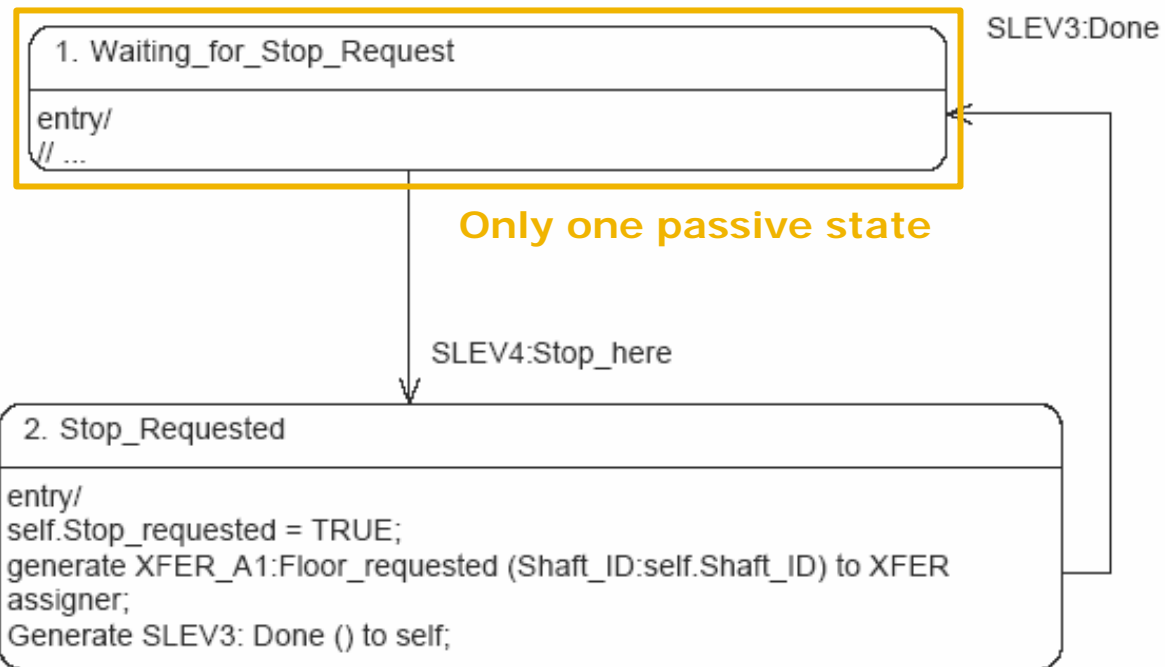




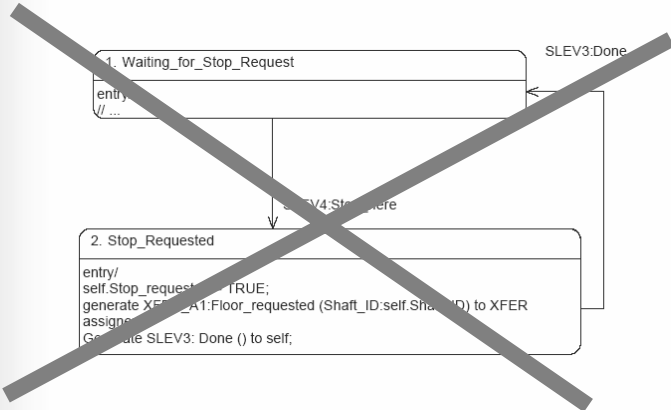
Eliminate Command Processors And Other Modeless Statecharts



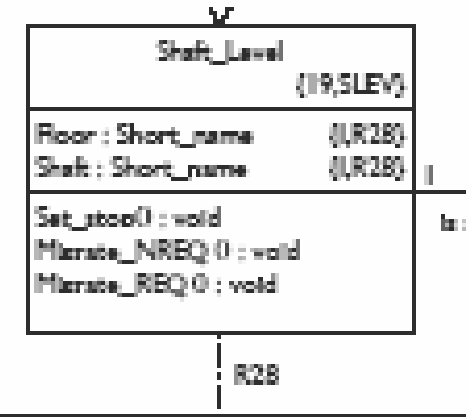
This kind of thing is unnecessary



No synchronization - no states



Even without states, it is still possible to "disable" an operation.



```

// Set stop - sets a stop request at this Shaft Level
select one my_Shaft related by self->SHAFT[R28];
if (my_Shaft.In_service)
    self.Migrate_REQ();
select one self_Req related by self->REQ_SLEV[R54];
self_Req.Stop_requested = true;
generate SHAFT1:Service_requested to my_Shaft;
end if;
  
```



Use Domain Specific Datatypes And Operations



Functions and datatypes

Why are these the only data types?

Integer (+, -, *, %)

Real (+, -, *, /)

String (+)

Datatypes should be domain specific and type safe.

Direction (Toggle)

Face (Rotate)

Angle (+, -)

Counter (incr, decr)

Vector (*)

```
::ppF (f:rect_A.Face)
```

```
// ppF - simulates unary ++ operator on a Face variable
```

```
//      rotates to the next adjacent face in the clockwise direction
```

```
if ( param.f == Face::T )
```

```
    param.f = Face::R;
```

```
elif ( param.f == Face::R )
```

```
    param.f = Face::B;
```

```
elif ( param.f == Face::B )
```

```
    param.f = Face::L;
```

```
elif ( param.f == Face::L )
```

```
    param.f = Face::T;
```

```
end if;
```

```
::decr (Value:self.Cycles)
```

```
    if (param.Value > 0)
```

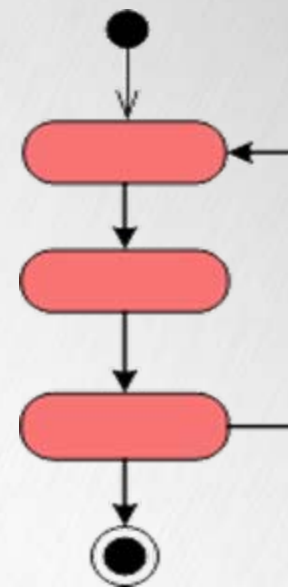
```
        param.Value = param.Value - 1;
```

```
    end if;
```

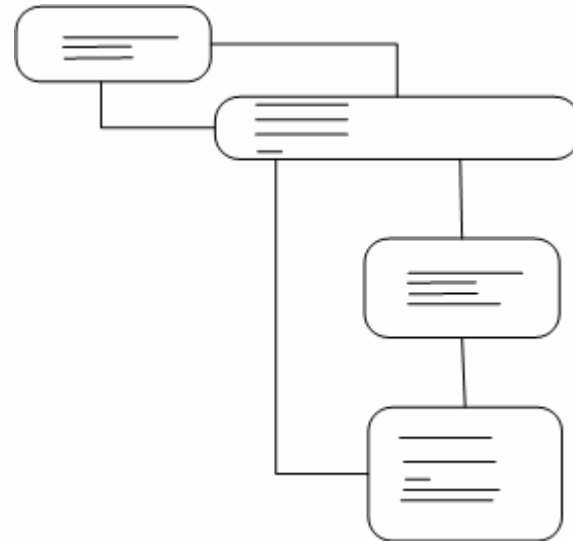
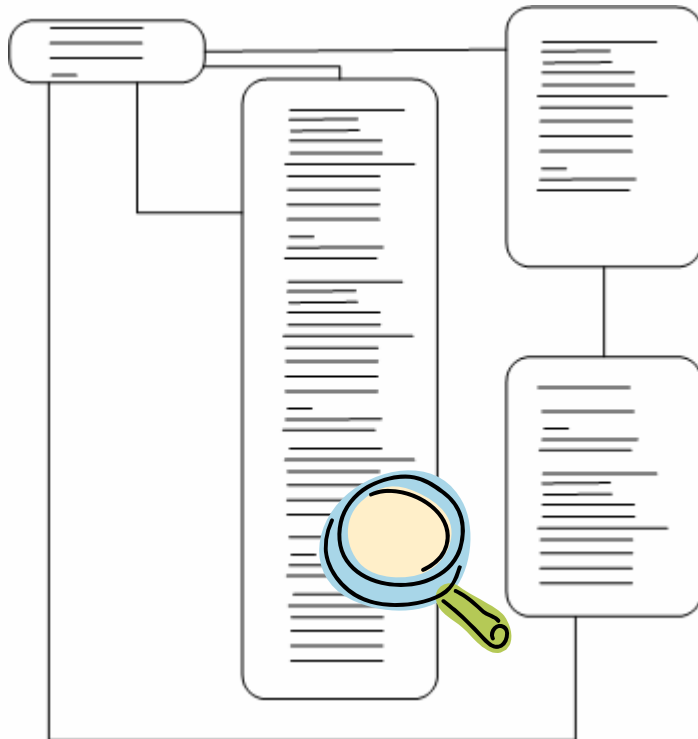
```
::Toggle(dir:my_Cabin.Travel_dir)
```

No Big States

THE STATE AND PROCESS MODELS SOLVE DIFFERENT PROBLEMS



Big states are unnecessary



Separate state and process models

Process Model – Shaft.Ping(OUT_Destination)

```

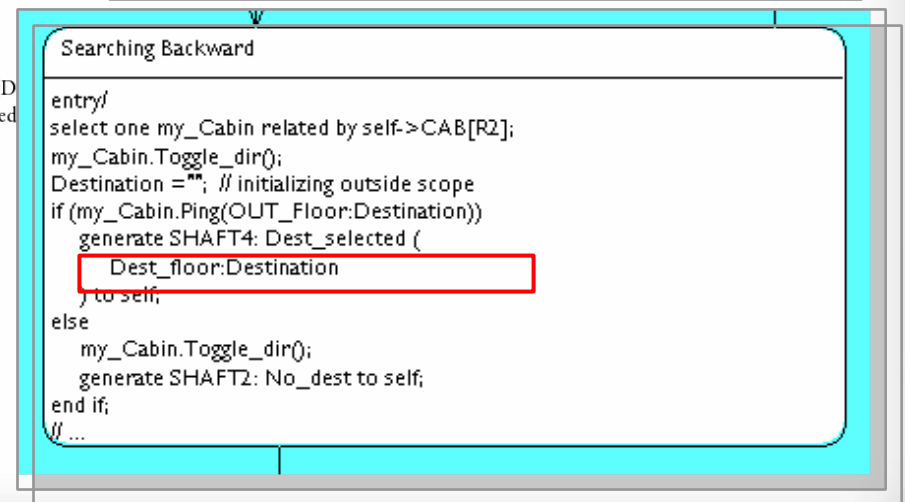
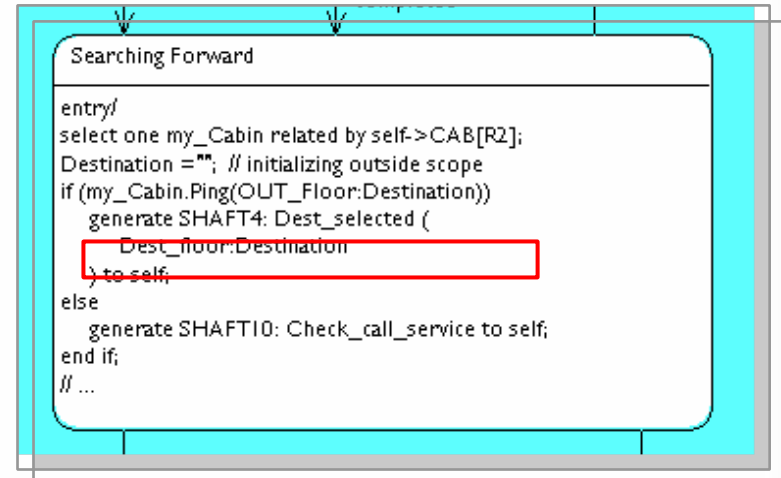
select one my_Bank related by self->SHAFT[R2]->BANK[R1];
select one my_Floor related by self->SLEV[R43]->FLOOR[R28];
select any search_BLEV related by my_Floor->BLEV[R29] where
    (selected.Bank == my_Bank.Name); // Cabin's current position

End_of_shaft = false; Request_found = false;

while (not End_of_shaft)

    if (not_empty search_BLEV) // Not the end of the shaft
        select any requested_SLEV related by // Is there a requested SLEV here?
            search_BLEV->FLOOR[R29]->
            SLEV[R28]->REQ_SLEV[R54]
            where (selected.Shaft == self.Shaft);
        if (not_empty requested_SLEV) // SLEV at this BLEV is requested
            if ( // The request is a stop or call matching my travel direction
                (requested_SLEV.Stop_requested) or
                ((self.Travel_direction == Direction::up) and (requested_SLEV.Floor_requested(D
                (self.Travel_direction == Direction::down) and (requested_SLEV.Floor_requested
            )
                Request_found = true;
                param.OUT_Floor = requested_SLEV.Floor; // Return the floor name and quit
                break;
            end if; // Request qualifies
        end if; // SLEV requested
    else // no more BLEVs
        End_of_shaft = true;
    end if; // End of shaft

    // Select the next BLEV away from me
    if (self.Travel_direction == Direction::up)
    
```



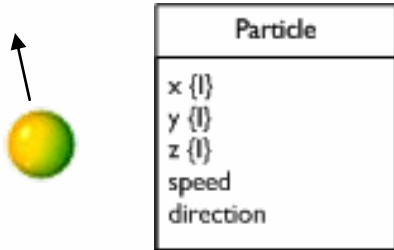


Don't Second Guess The Model Compiler

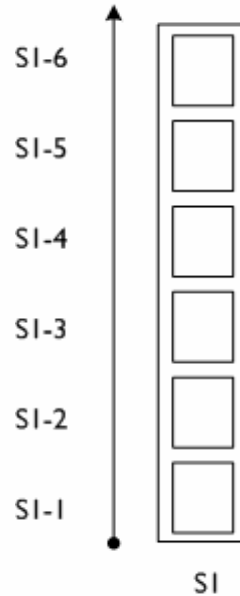


Floor Access Analysis 1.0

Coordinate System Attributes



The x, y and z axes are often not modeled as classes.



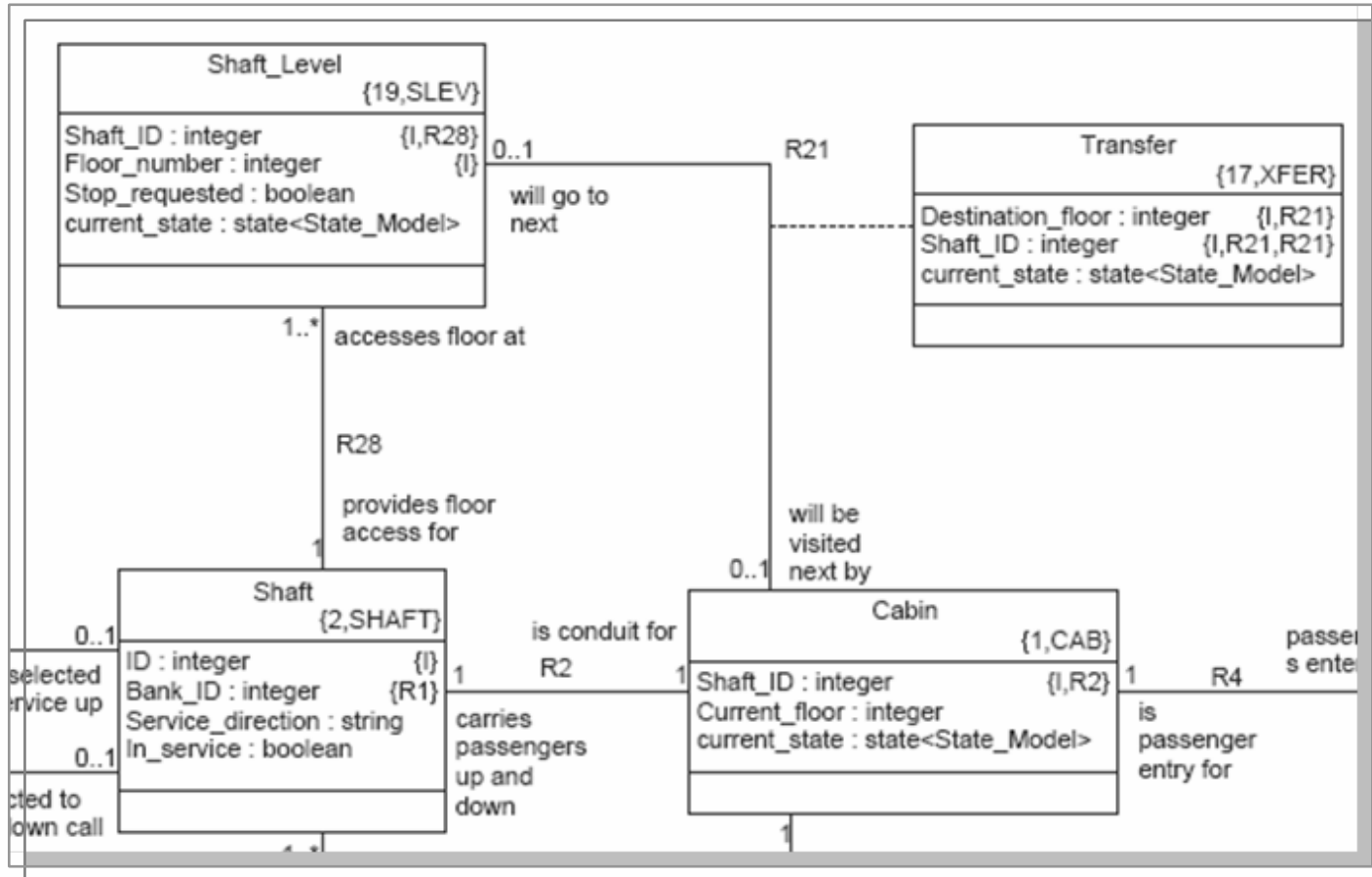
You could similarly incorporate the z-coordinate into each Shaft-Level identifier and dispense with the floor as a class.



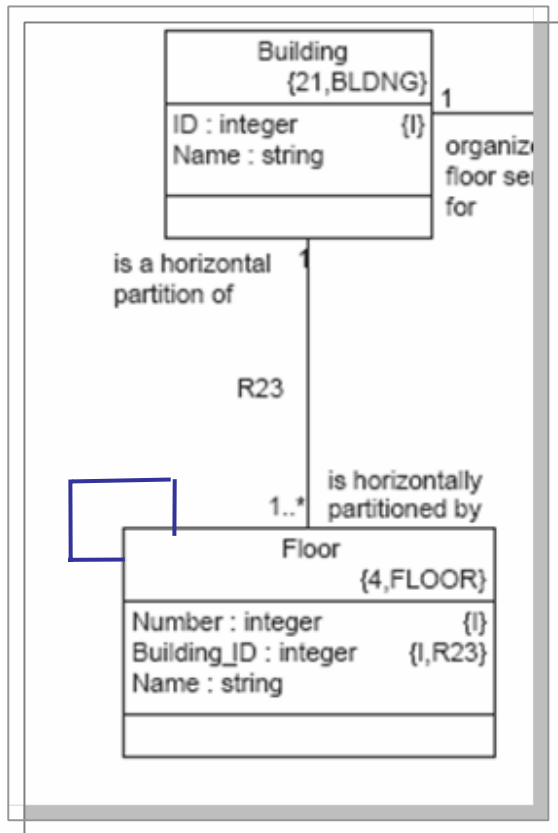
Now you can search through Shaft Level instances using a counter.

if no requests at current floor
++ curr_Floor;

Floor Access Model 1.0



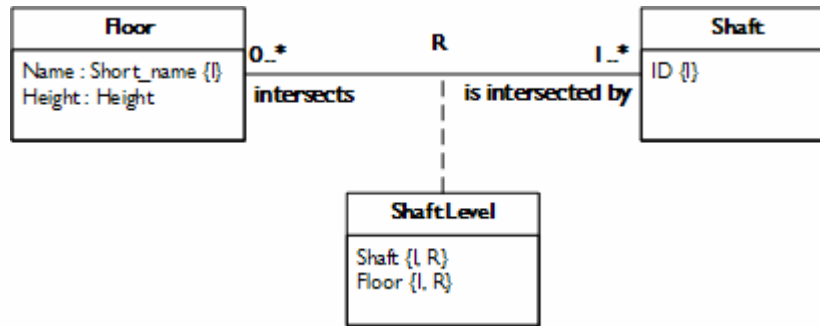
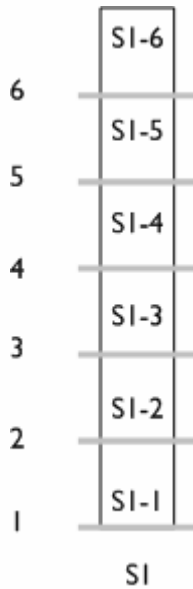
Floor Access Model 1.0



There's just no getting around the notion of Floor as a class.

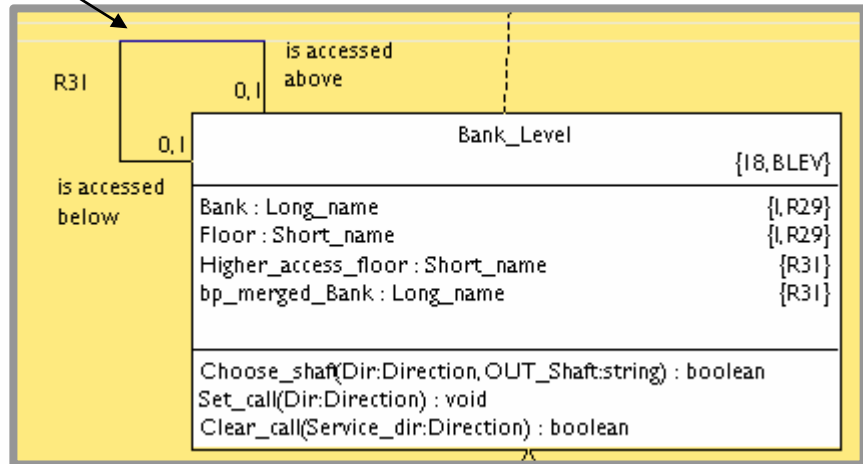
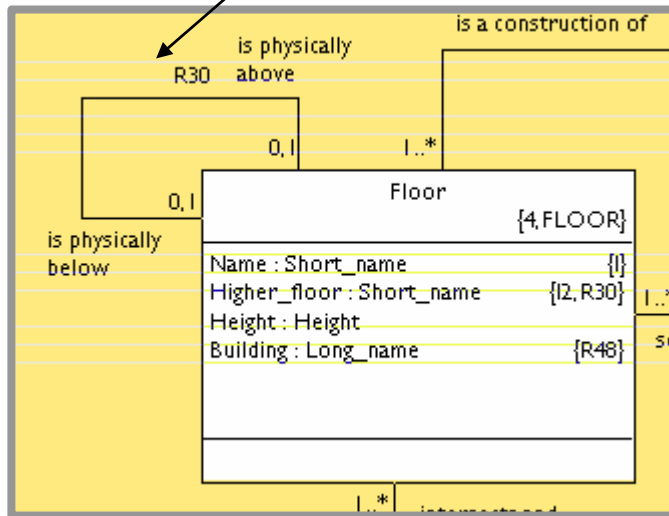
And there is really no benefit to the z-coordinate axis idea.

Floor Access Model 2.0



Floor and BLEV Sequencing

Not the same!



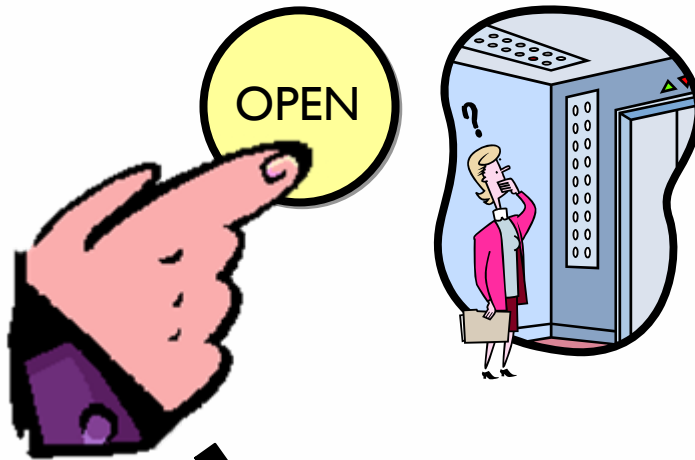


Synchronize Everything!

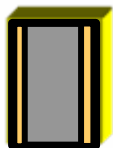


Resolving race conditions

What happens when you push the OPEN button inside the cabin?



If the cabin is waiting, open the door.

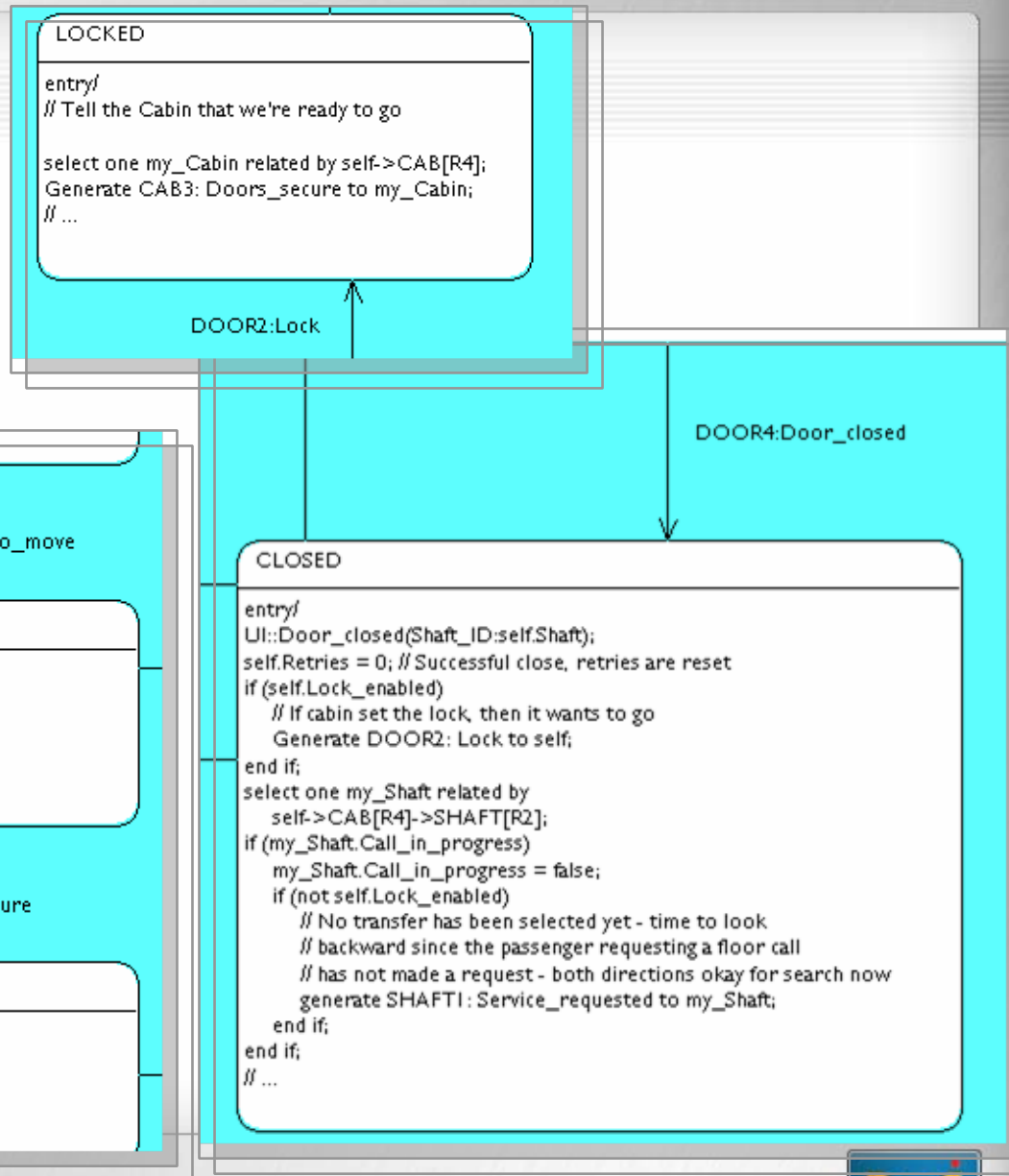


But if the cabin is moving, do NOT open the door!



So what happens when the button is pressed and the cabin starts moving at the exact same point in time?

Cabin cannot move until the door is locked. While locked, all user open events are ignored. Only the cabin can unlock the door.

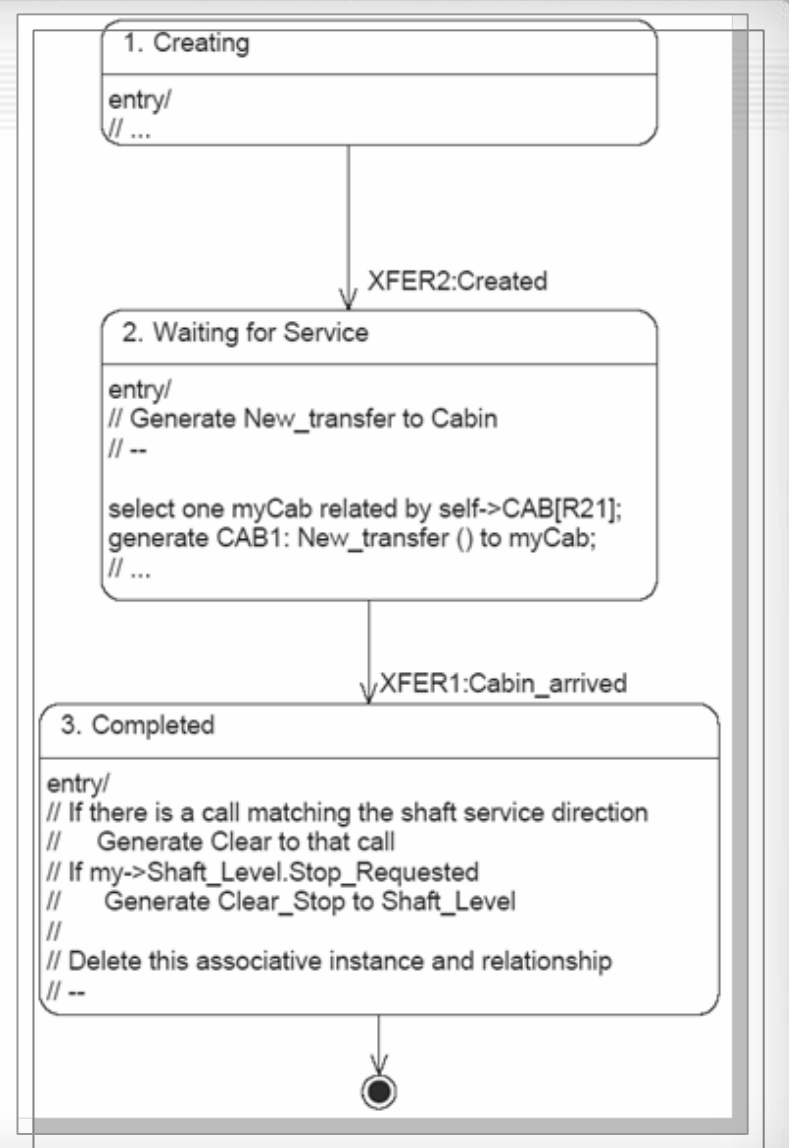


Xfer Race Condition

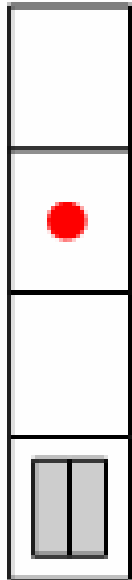
Only one instance per cabin is allowed at one time.

What happens when a better destination is discovered while this transfer is in progress?

In Elevator 1.0, the instance is remotely deleted. Not a good plan.



Xfer 2.0 Solution



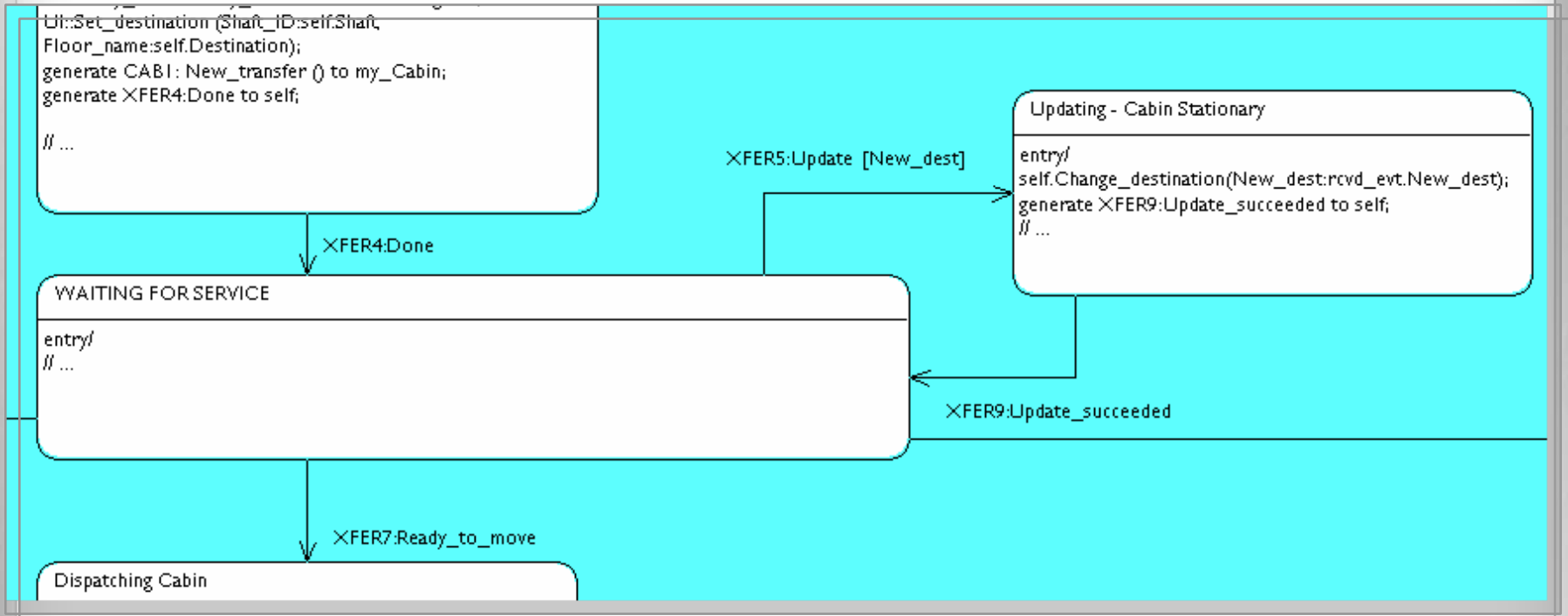
If XFER in progress and a better destination is found, Shaft generates change **request** to XFER.

XFER ignores if too late, updates if cabin not moving, otherwise **asks** cabin to redirect if possible.

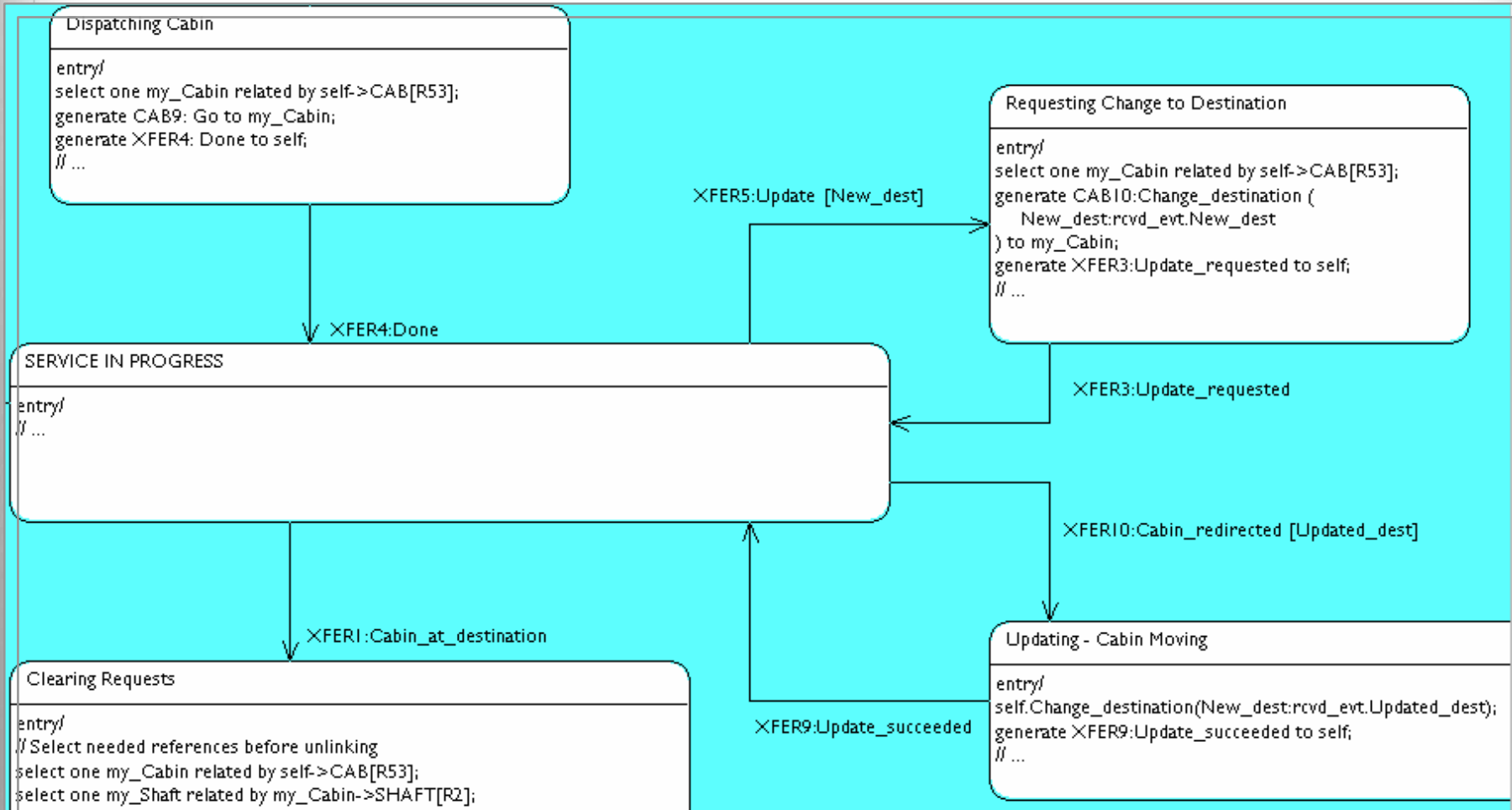


Cabin tries to accommodate, redirects Transport and, if successful, updates the XFER destination.

XFER idle - immediate change

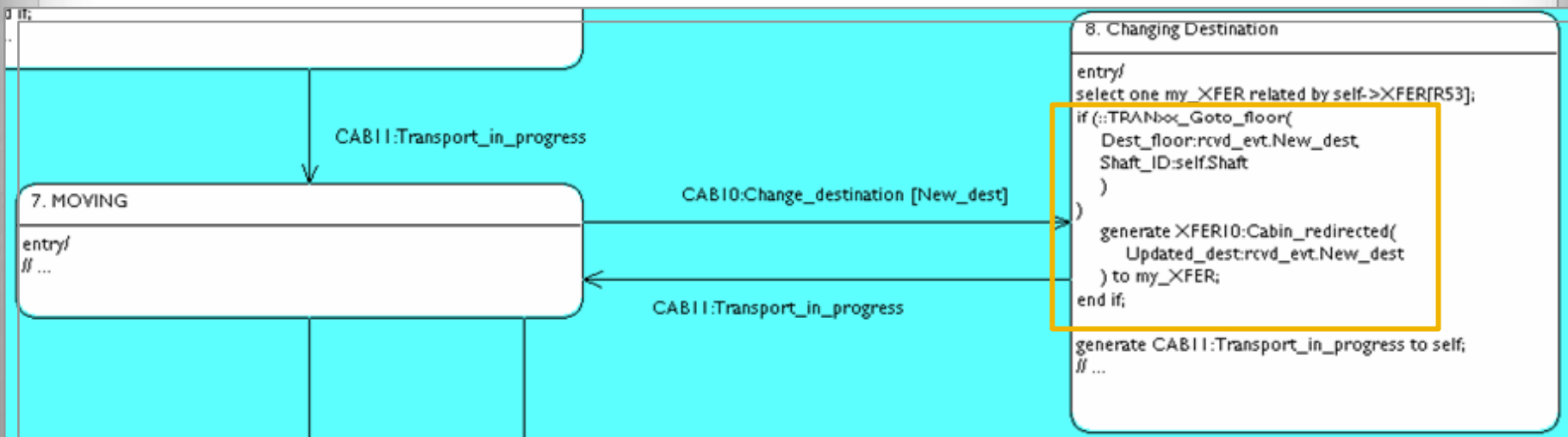


XFER active - Ask Cabin to change

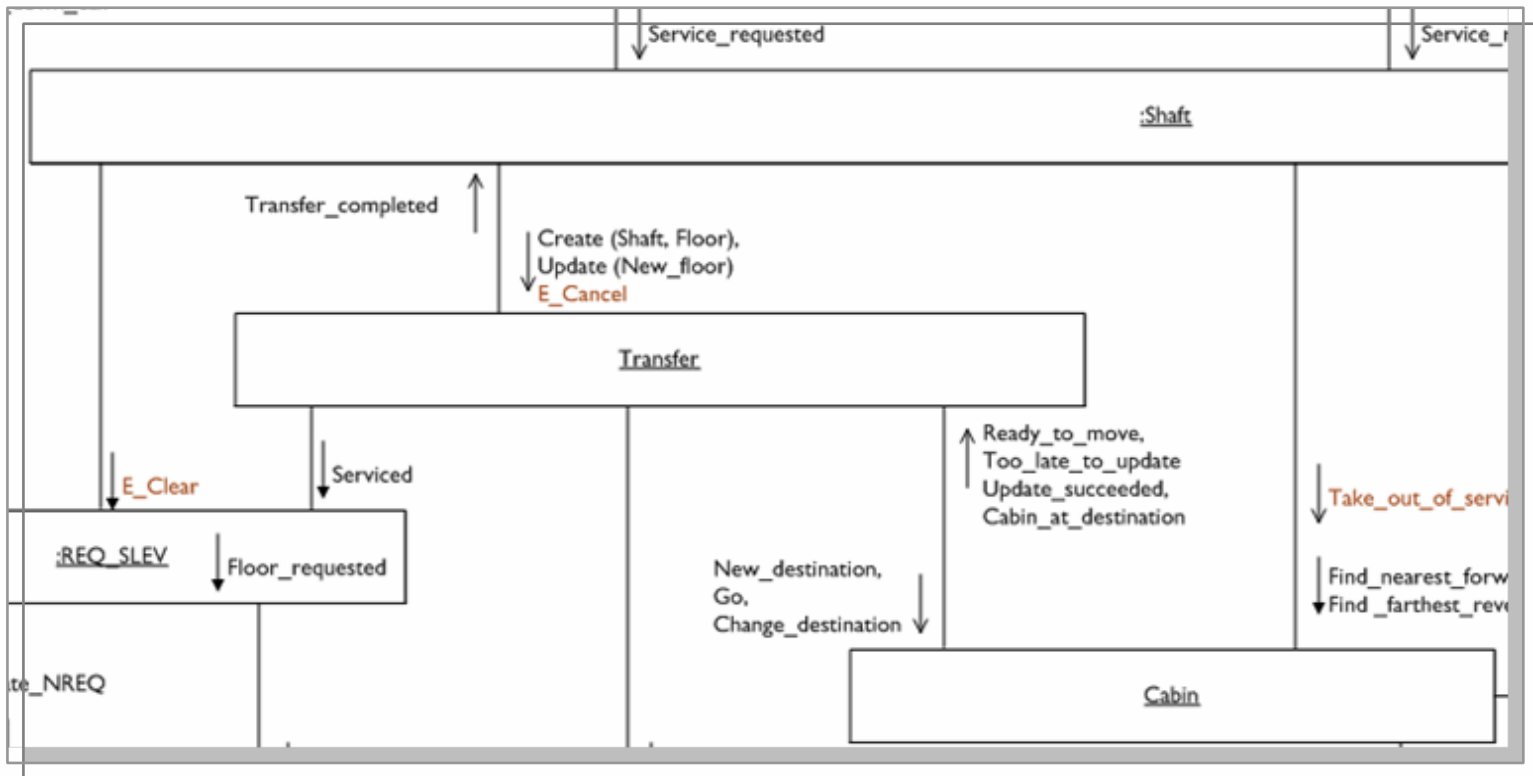


Cabin notifies XFER if successful

Transport will reject the request if the stop is too short.
If successful, the XFER will be updated.



Race condition avoidance





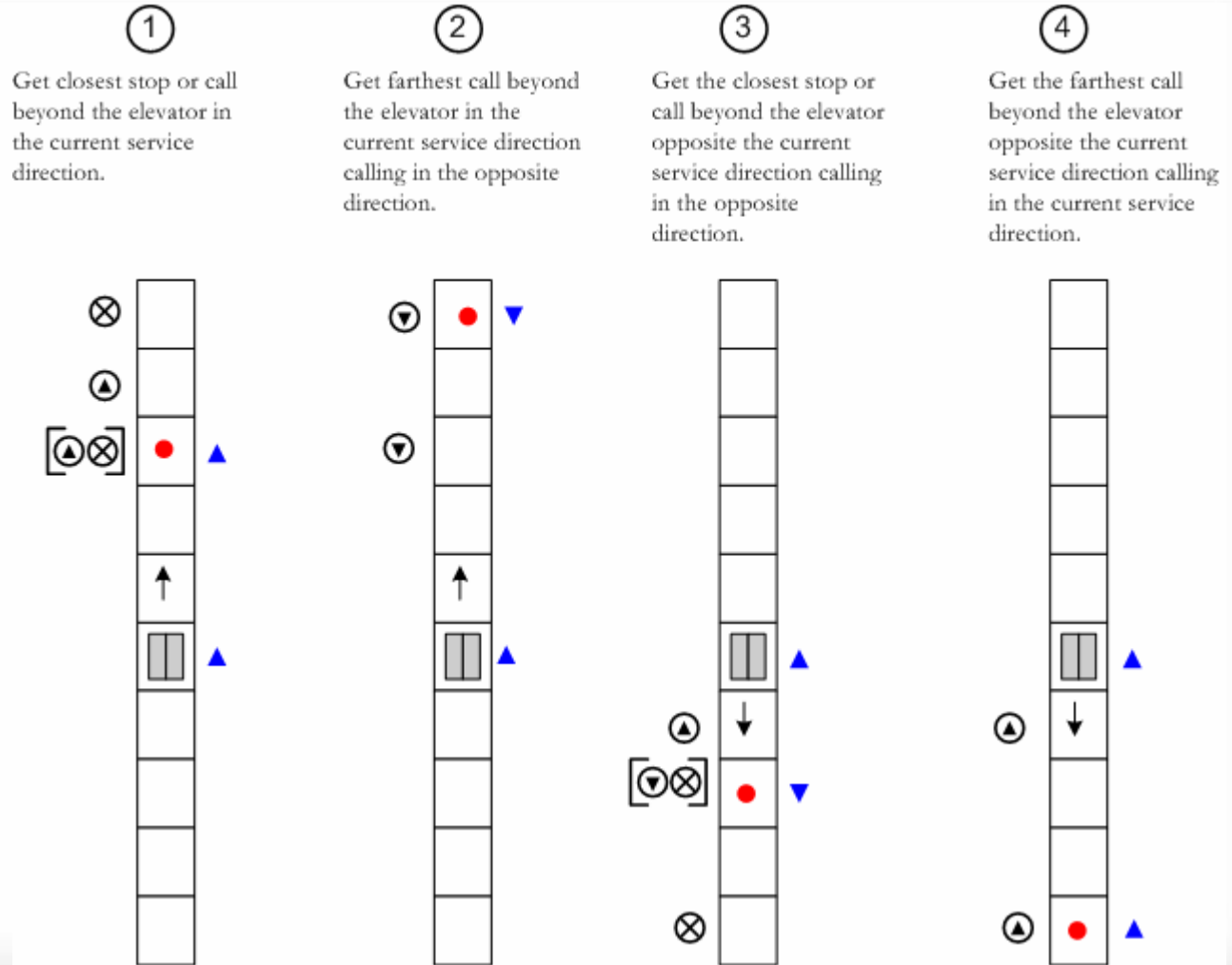
Careful Analysis Pays Off.



Same old selection algorithm.

The floor selection algorithm has not changed.

Modeling of this algorithm has evolved considerably.

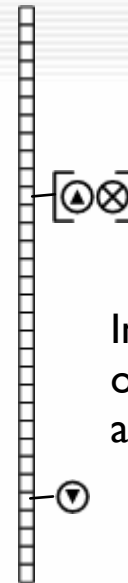
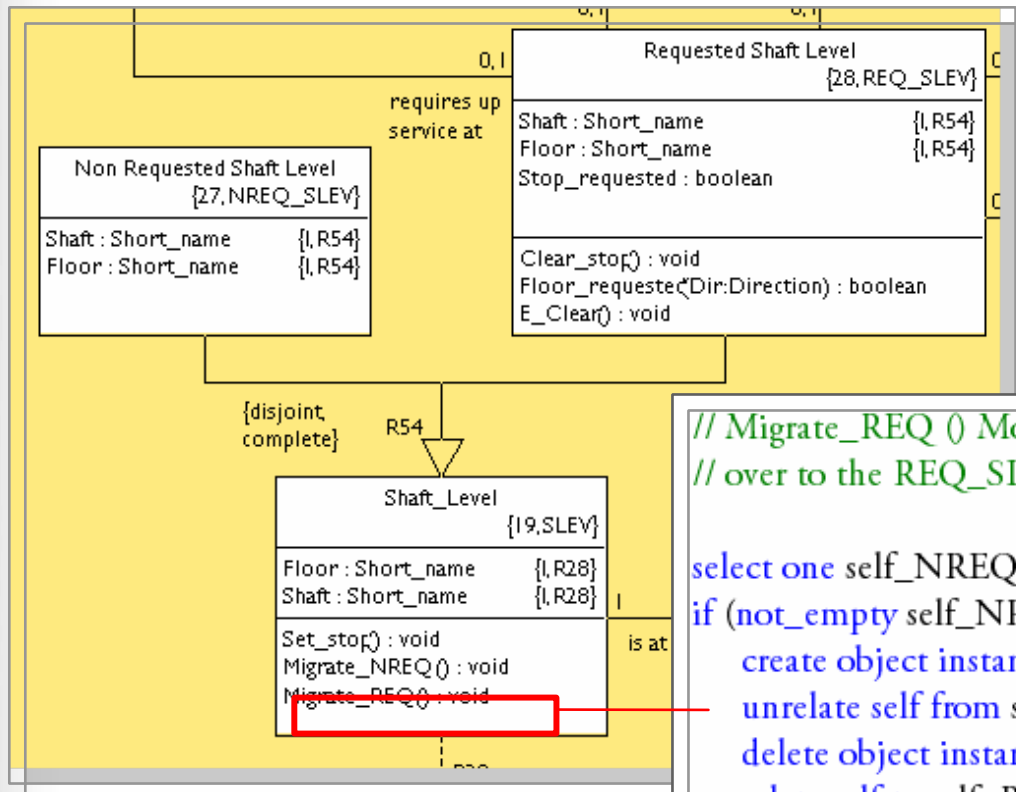




Migration And Polymorphism



Only requested levels are evaluated.

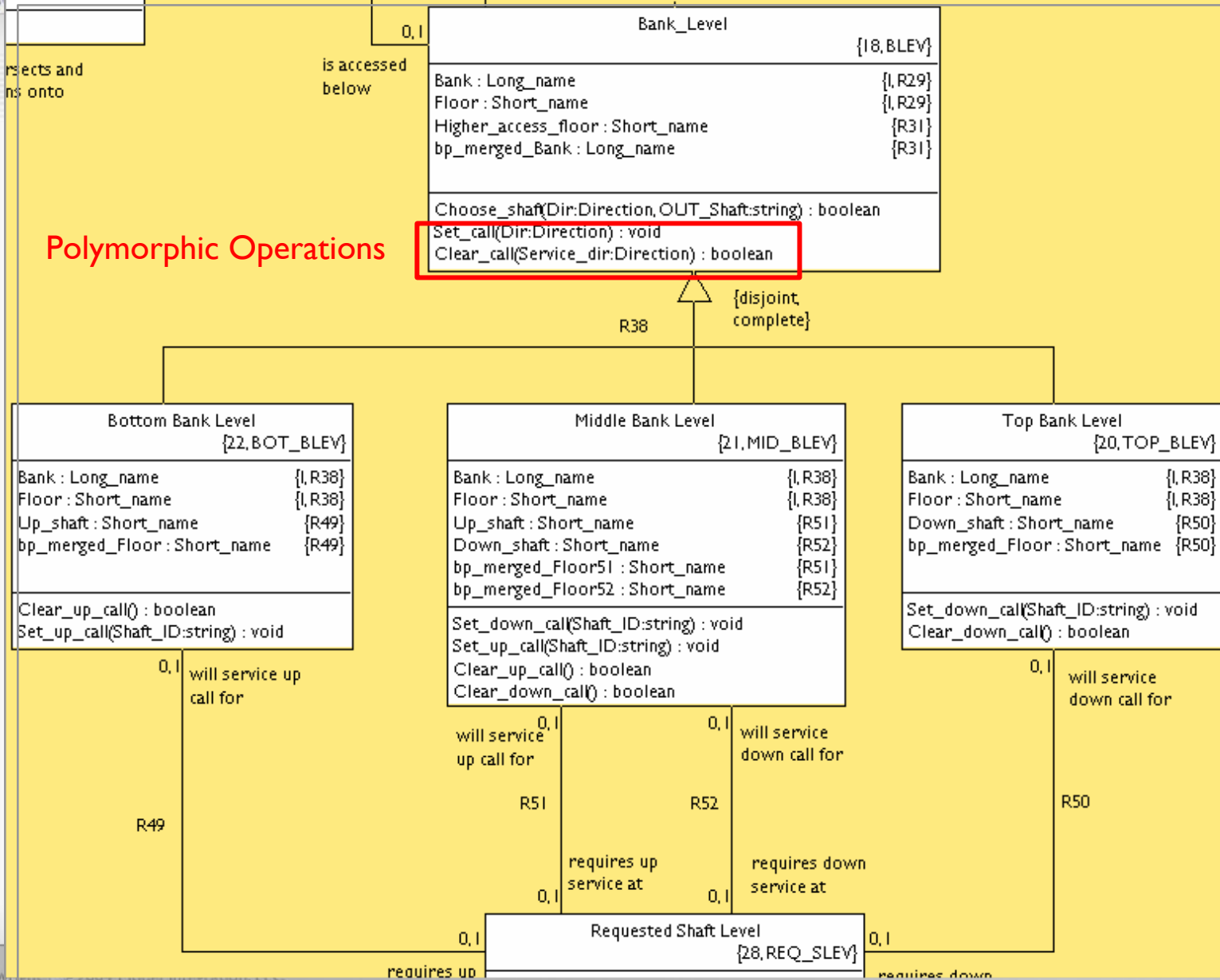


In this case, only two levels are requested.

```

// Migrate_REQ () Move an NREQ_SLEV instance
// over to the REQ_SLEV subclass

select one self_NREQ related by self->NREQ_SLEV[R54];
if (not_empty self_NREQ)
    create object instance self_REQ of REQ_SLEV;
    unrelate self from self_NREQ across R54;
    delete object instance self_NREQ;
    relate self to self_REQ across R54;
end if;
    
```





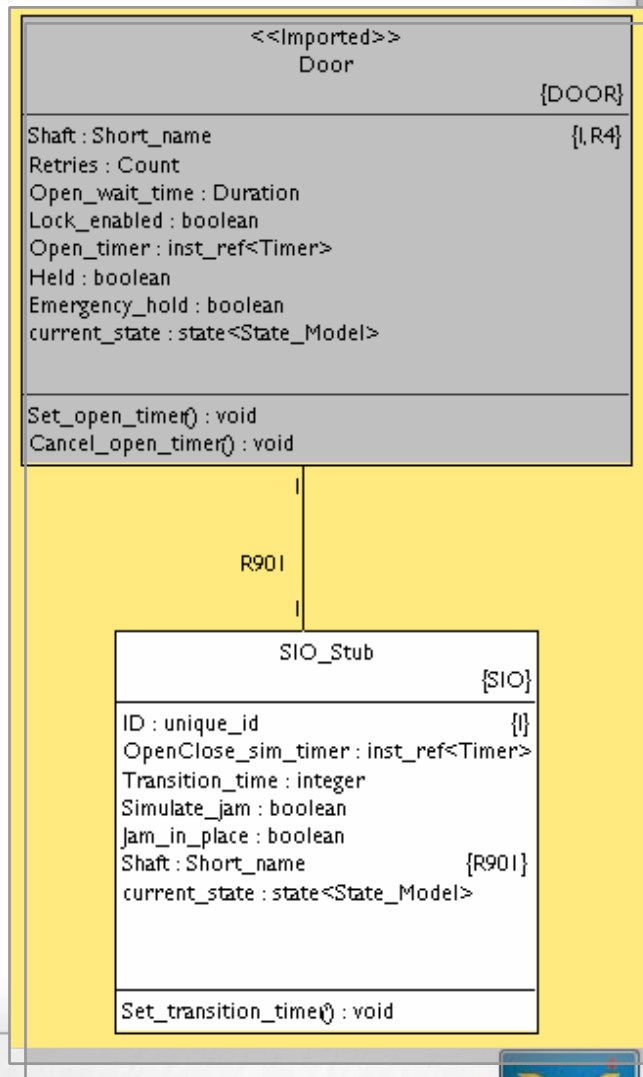
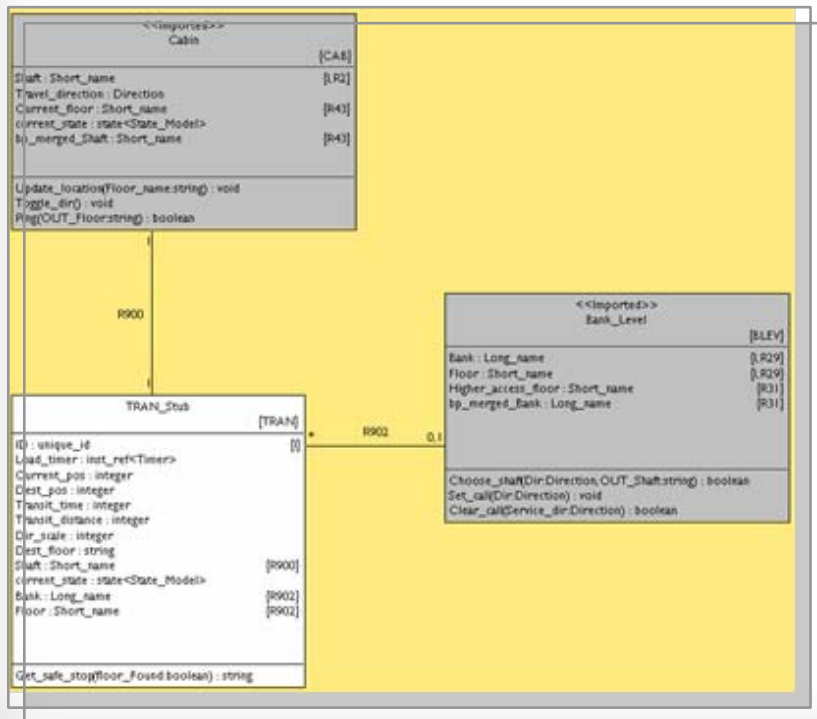
Testing



Transparent stubs

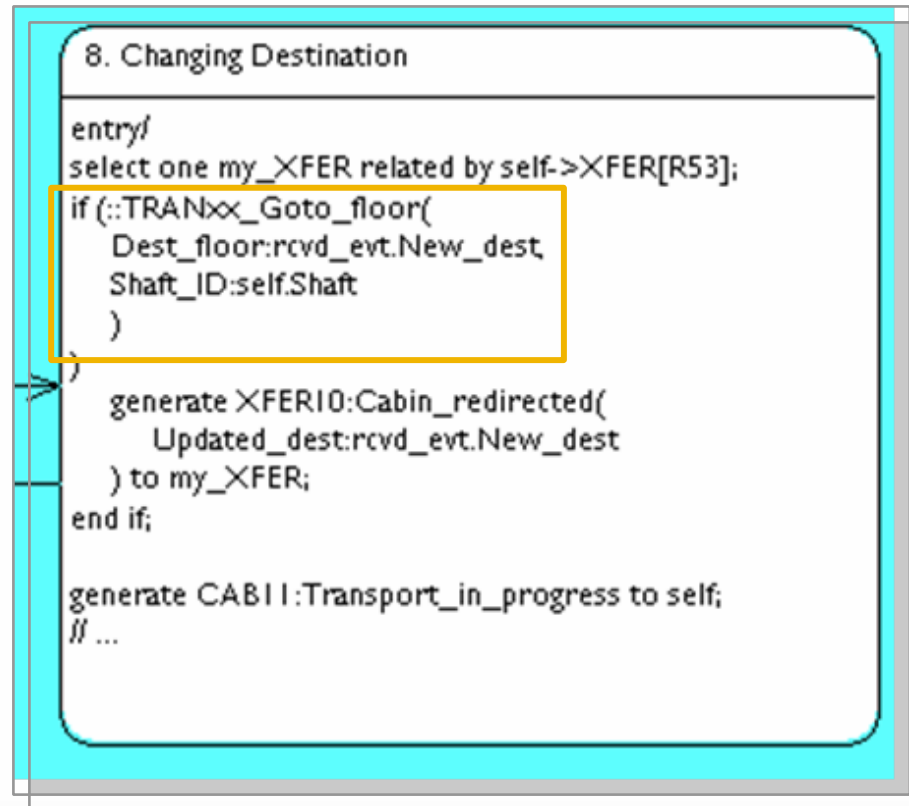
A separate subsystem is dedicated to all stubs.

The stub subsystem imports from the application so the application contains no stub references.



Functions in place of bridge calls

When the stubs are replaced by a service domain, `::Namexx` is replaced by `Name::`



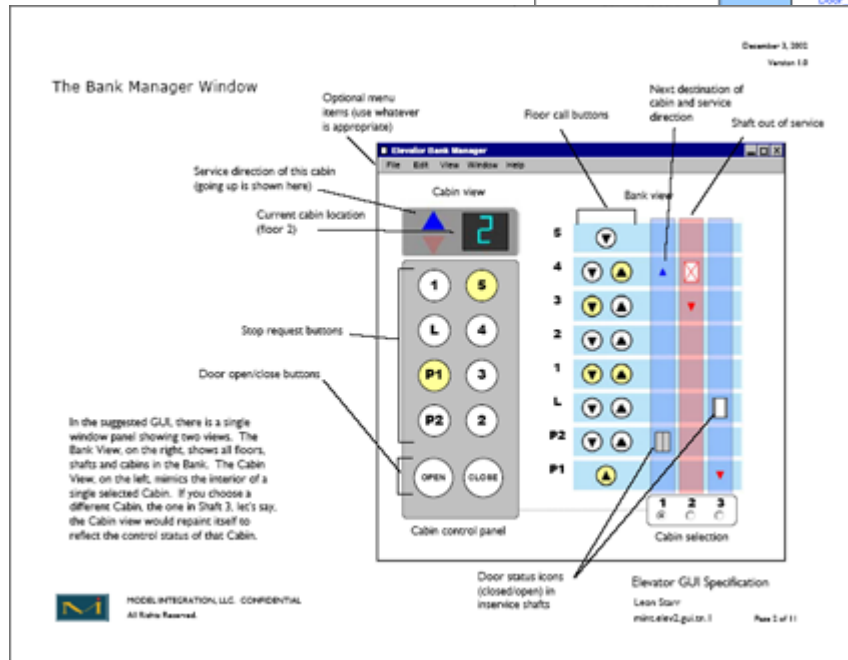
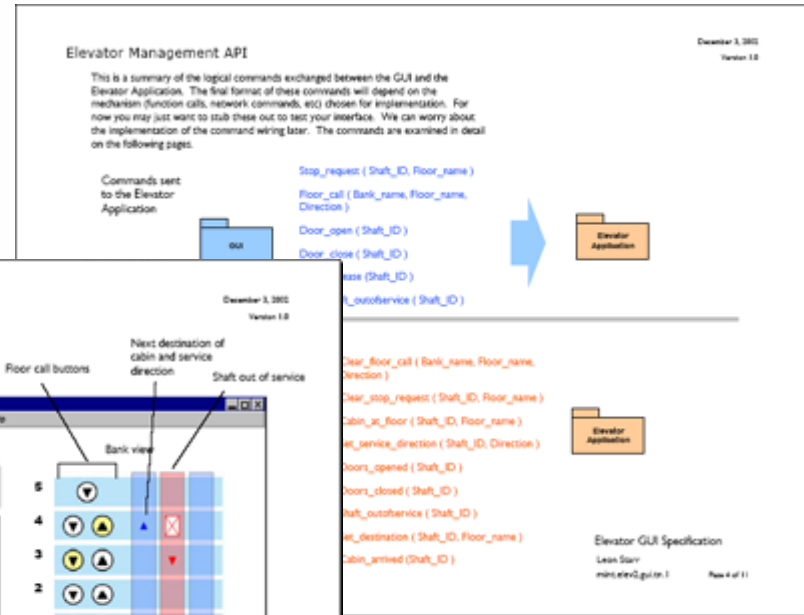


User Interface



User Interface

- + Introduction
- + Leon's phone call
- + Delivery of the GUI Spec





Challenges

- + Challenges

-
-

(Seattle, San Francisco, Denver)

- + Solution:

-

- + Proof of Concept

-



Designing the Interface

+ Designed UI using the GUI specifications

-
-

The screenshot shows the 'ELEVATOR BANK MANAGER' interface. At the top, a status bar reads 'SELECT A CABIN TO VIEW' followed by a list of cabin numbers from 1 to 13. Below this is a grid with columns for 'FLOORS', 'CALLS', and 'SHAFTS'. The 'FLOORS' column lists levels 5, 4, 3, 2, 1, L, P2, P1, P0, A2, A1, 75-55, B1, and B0. The 'CALLS' column shows green and yellow call icons. The 'SHAFTS' column shows blue and green shaft icons. A tooltip over a yellow call icon on floor 3 says 'TO TAKE ME OUT OF SERVICE CLICK ME!'. At the bottom right, a note says 'CLICK ON A CABIN TO TAKE OUT OF SERVICE' and 'EXPRESS BANK'. On the left side of the screenshot, there are floor call buttons (4, 5, 2, 3, L, 1, P1, P2) and door status buttons (OPEN, CLOSE). A digital display shows '04'.

The inset diagram, titled 'The Bank Manager Window', provides a detailed view of the GUI components. It includes labels for:

- Optional menu: menu size whatever is appropriate
- Service direction of this cabin (going up is shown here)
- Current cabin location (floor 3)
- Stop request buttons (L, 4, P1, 3, P2, 2)
- Door open/close buttons (OPEN, CLOSE)
- Cabin view
- Bank view
- Floor call buttons
- Next destination of cabin and service direction
- Shaft out of service
- Cabin selection (1, 2, 3)
- Door status icons (closed/open) in inactive shafts

 A note in the diagram states: 'In the suggested GUI, there is a single window panel showing two views. The Bank View, on the right, shows all floors, shafts and cabins in the Bank. The Cabin View, on the left, mimics the interior of a single selected Cabin. If you choose a different Cabin, the one in Shaft 3, let's say, the Cabin view would reinit itself to reflect the control status of that Cabin.'



User Interface Creation

+ Flash Development

-
-
-
-





Models

+ Connecting to the models:

-





Flash

+ Using as a GUI:

-
-





Starting the simulation

- + Startup considerations

-
-





Initialization



Application procedural init function

```
// R4
```

```
relate the_Door to the_Cabin across R4;
```

```
// R28
```

```
relate F_B to the_Shaft across R28 using Shaft_Level_B;
```

```
relate F_L to the_Shaft across R28 using Shaft_Level_L;
```

```
relate F_1 to the_Shaft across R28 using Shaft_Level_1;
```

```
relate F_2 to the_Shaft across R28 using Shaft_Level_2;
```

```
relate F_3 to the_Shaft across R28 using Shaft_Level_3;
```

```
// R29
```

```
relate F_B to the_Bank across R29 using BLEV_S1_FB;
```

```
relate F_L to the_Bank across R29 using BLEV_S1_FL;
```

```
relate F_1 to the_Bank across R29 using BLEV_S1_F1;
```

```
relate F_2 to the_Bank across R29 using BLEV_S1_F2;
```

```
relate F_3 to the_Bank across R29 using BLEV_S1_F3;
```

```
// R30
```

```
relate F_B to F_L across R30.'is physically below';
```

```
relate F_L to F_1 across R30.'is physically below';
```

```
relate F_1 to F_2 across R30.'is physically below';
```

```
relate F_2 to F_3 across R30.'is physically below';
```

This goes on for about six pages.



Tcl/tk GUI declarative init file

```
# Elevator Application Initialization
#
# bank <name>
# floor <bank_name> <floor_name> <floor_position>
# shaft <name> <id> <inservice> <floor_name> <cabin_position> <direction> <door state>

bank Express
floor Express P2 0
floor Express P1 100
floor Express L 200
floor Express 1 300
floor Express 2 400
floor Express 3 500
floor Express 4 600
floor Express 5 700
shaft Express 1 1 P2 0 up closed
shaft Express 2 1 L 200 down closed
shaft Express 3 1 L 200 up closed

bank Penthouse
floor Penthouse P2 0
floor Penthouse P1 100
floor Penthouse L 200
floor Penthouse 5 700
shaft Penthouse 4 1 L 200 up closed
shaft Penthouse 5 1 L 200 down closed

bank Exclusive
floor Exclusive P2 0
floor Exclusive 5 700
shaft Exclusive 6 1 5 700 up closed
```

Just one page.





Bridges





```
//=====
// Bridge: Set_service_direction
//=====
void
UI::Set_service_direction(
    PTC_CString_c const & ee_direction,
    PTC_CString_c const & ee_shaft_id )
{
    ostringstream myMessage;

    myMessage << "set_service_direction " << ee_shaft_id << " " << ee_direction;
    UIPortal::sendtoUI (myMessage);
}
```

Bridge calls out from the application are wired into functions that construct UI command strings.

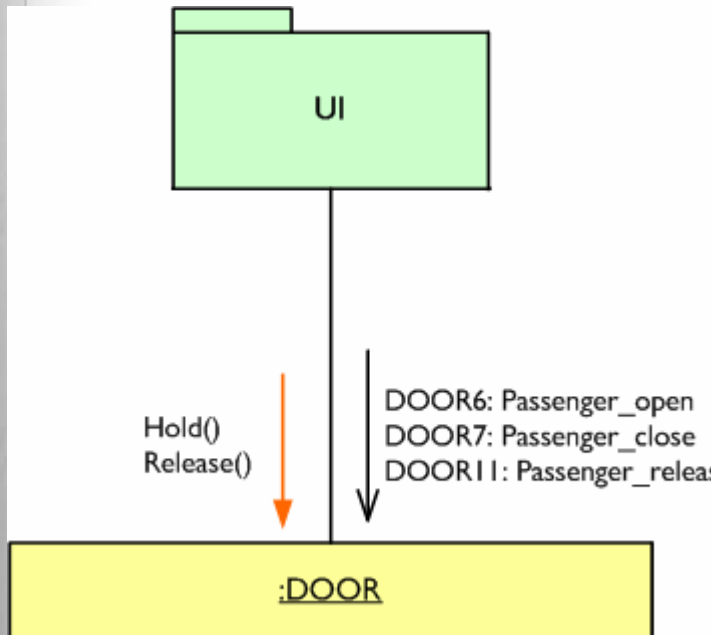
The command strings are fed into Mtalk (Model Integration's TCP/IP plugin)

```
bool UIPortal::sendtoUI(ostringstream &modelMessage)
{
    modelMessage << "\r\n";
    try
    {
        if (send(dataSocket,modelMessage.str(),modelMessage.pcount(),0) < 0)
            throw "UIPortal::sendtoUI - Can't send message to UI";
    }
    catch(const char errorMessage[])
    {
        cerr << errorMessage << endl;
        return false;
    }
    return true;
}
```



Mtalk -> Valid string -> MC func

UI is hand coded. It invokes functions provided by the model compiler.



```

}
else if (strcmp(cline->sarg[0], "floor_call") == 0)
{
    if (cline->pcount != 3)
        throw "floor_call bank_name floor_name direction";
    ::EV_UI_floor_call(cline->sarg[1], cline->sarg[3], cline->sarg[2]);
}
else if (strcmp(cline->sarg[0], "door_open") == 0)
{
    if (cline->pcount != 1)
        throw "door_open shaft_id";
    ::EV_UI_door_open(cline->sarg[1]);
}
else if (strcmp(cline->sarg[0], "door_close") == 0)
{
    if (cline->pcount != 1)
        throw "door_close shaft_id";
    ::EV_UI_door_close(cline->sarg[1]);
}
else if (strcmp(cline->sarg[0], "door_release") == 0)
{

```

Function -> Select and prod

```
::EV_UI_door_open (shaft_id);
```

```
UI_Door_open (shaft_id:string)
```

Elevator

UI

Hold()

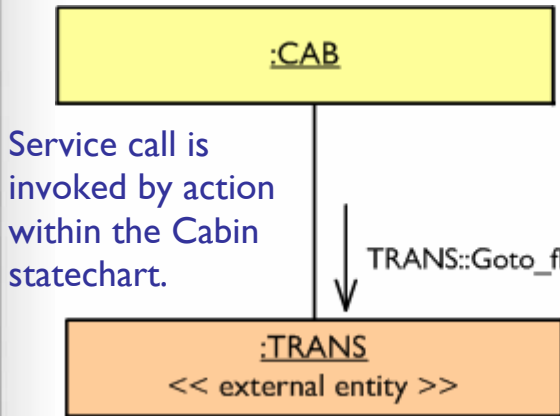
DOOR6: Passenger_open

:DOOR

```
// UI_Door_open (shaft_id:string)
//
select any theShaft from instances of SHAFT where
  ( selected.ID == param.shaft_id );
if (not_empty theShaft)
  if (theShaft.In_service)
    select any theDoor from instances of DOOR where
      (selected.Shaft_ID == param.shaft_id);
    theDoor.Hold();
    generate DOOR6: Passenger_open () to theDoor;
  else
    UI::Error(message:"Elevator App::Shaft is out of service.");
  end if;
else
  UI::Error(message:"Elevator App::No such door or shaft.");
end if;
```

Modeled to modeled domain

Elevator collaboration diagram



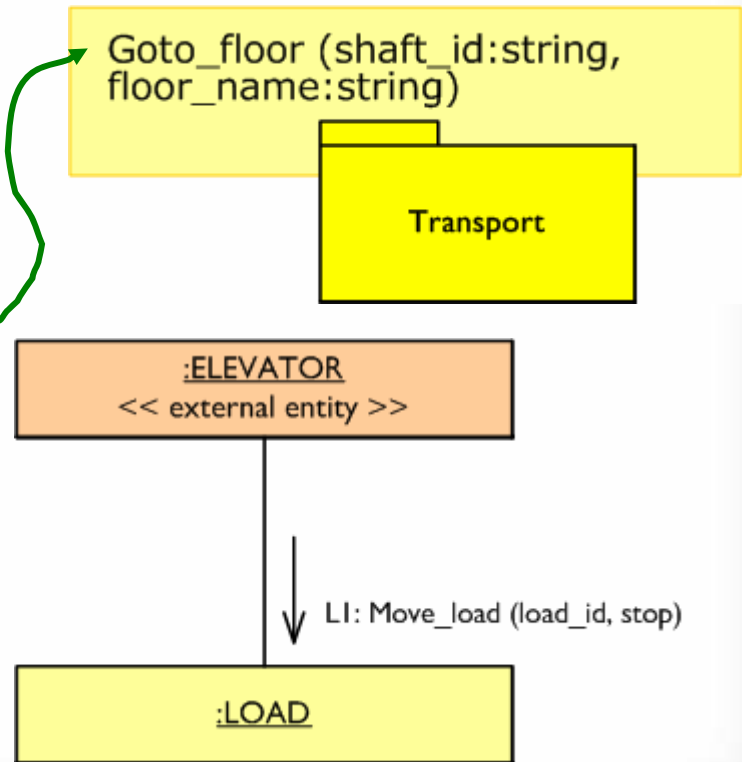
Service call is invoked by action within the Cabin statechart.

Both the Elevator and Transport domains are modeled.

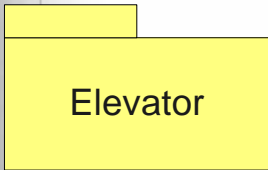
Transport collaboration diagram

Service call is wired to Transport service domain.

Action language in `Goto_floor` function maps the `shaft/floor_name` entities to `load/stop` entities and generates the LI event.



Domain separation

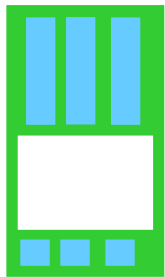


- Floor selection
- Cabin dispatching
- Door open/close timing



Elevator uses Transport Bridge between domains

- Safe acceleration
- Precise transport



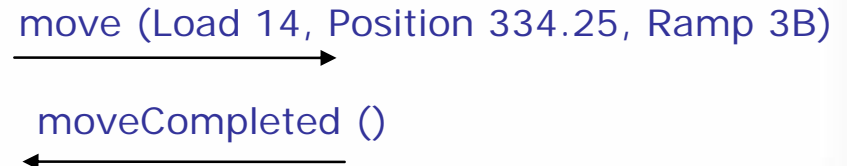
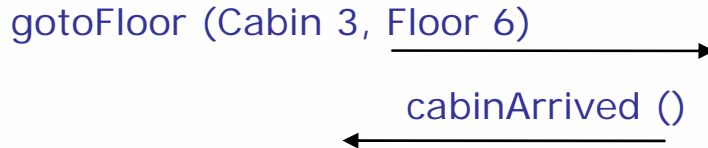
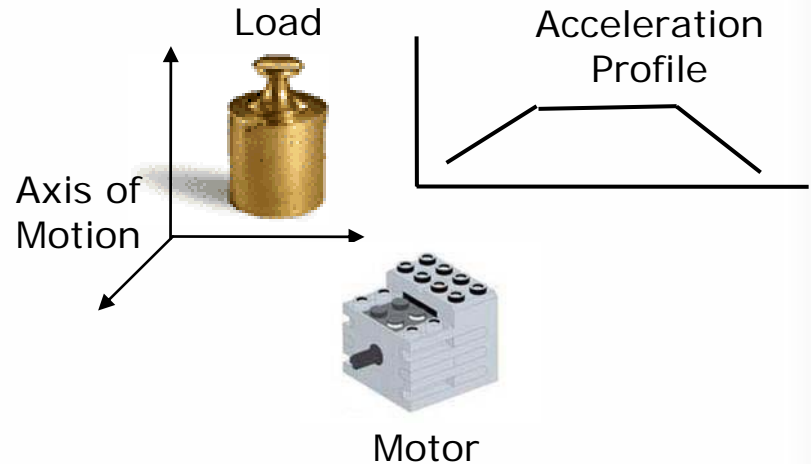
Bank



Cabin



Door



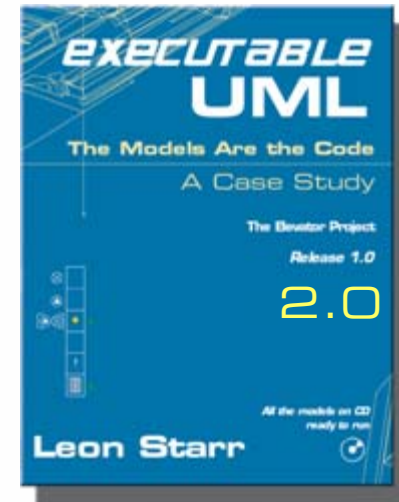


What's Next?



What's next?

- + Course Integration
- + New book(s)
- + Transport and Signal IO domains
- + Lego kit
- + Look for web goodies in 2005



MODEL INTEGRATION LLC