EXECUTABLE UML AND MBSE

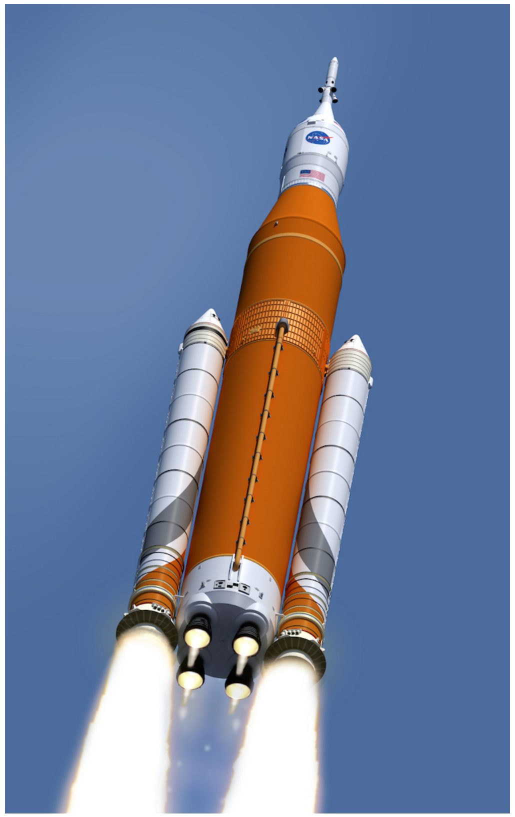
What Executable UML does,
how it is totally different from UML,
and how it fits with other Executable languages

Leon Starr
leon_starr@modelint.com

MODEL INTEGRATION LLC

www.modelint.com

FUNDAMENTAL IDEA OF MODELING



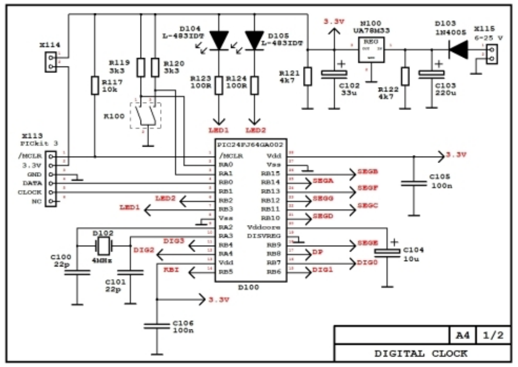
A model isn't the thing you are modeling.

To be useful, a model must omit certain aspects of the real world subject to focus on the details of interest.

EXCLUDED DETAILS ARE STILL IMPORTANT

Just because a detail is systematically ignored, doesn't mean that it is not important

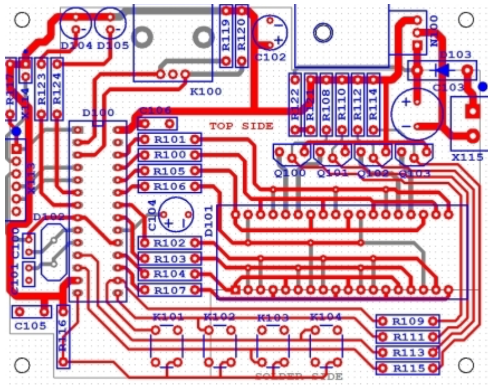
Schematic



Focuses on component properties and connectivity

Excludes layout details

Layout diagram



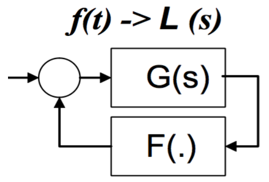
Focuses on layout geometry

Is applied to schematic

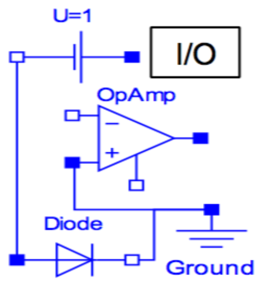
MODELING LANGUAGES, PURPOSES AND REQUIRED SKILLS

What about UML?

Simulink

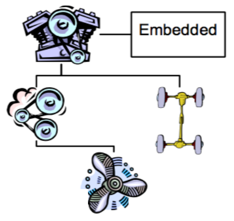


control

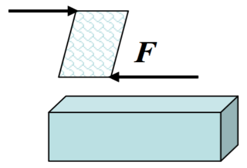


electronics/optronics

(usage at SAAB)

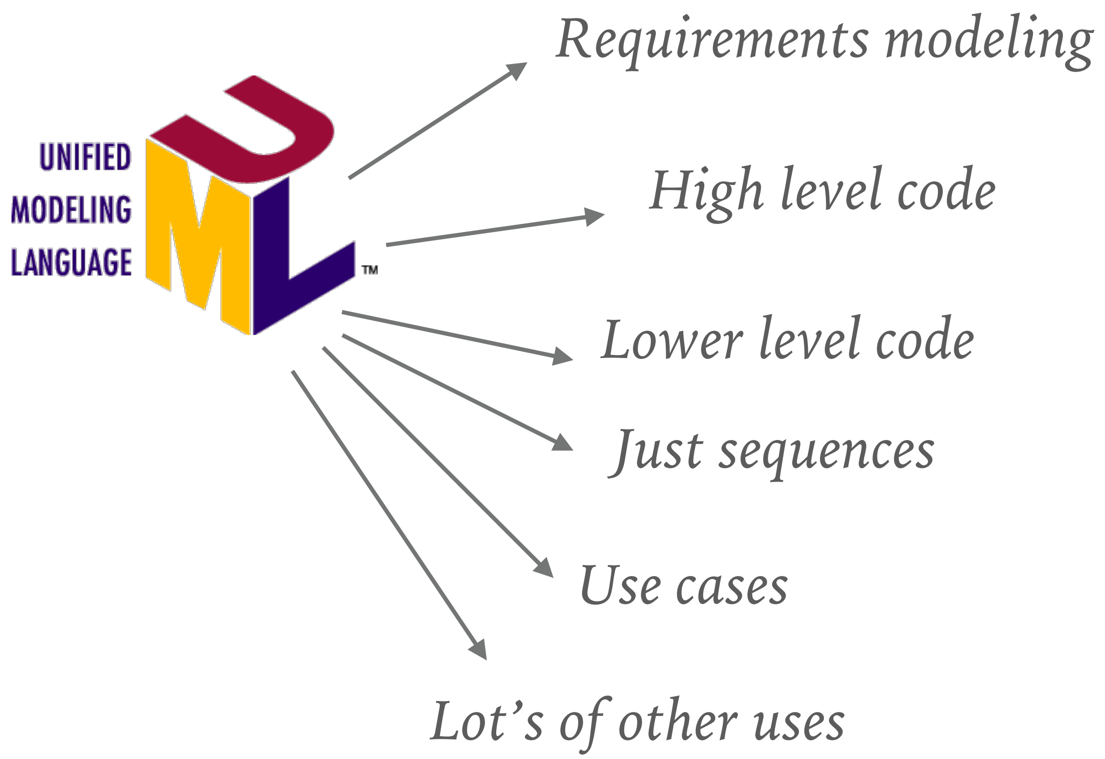


physical systems



structure

UML PURPOSE AND SKILLS



UML profiles

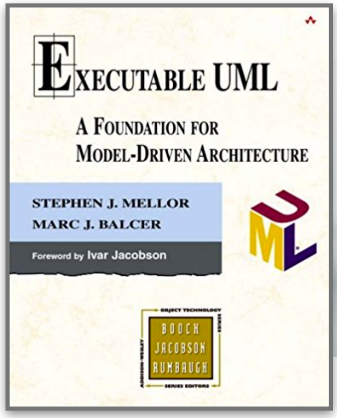
Executable UML

UML is not a modeling language.

It is a standardized set of of object-oriented notations for many purposes.

EXECUTABLE UML

Modeling language



book

white papers

(shlaer-mellor method)

Our purpose is to model requirements imposed by the real world:

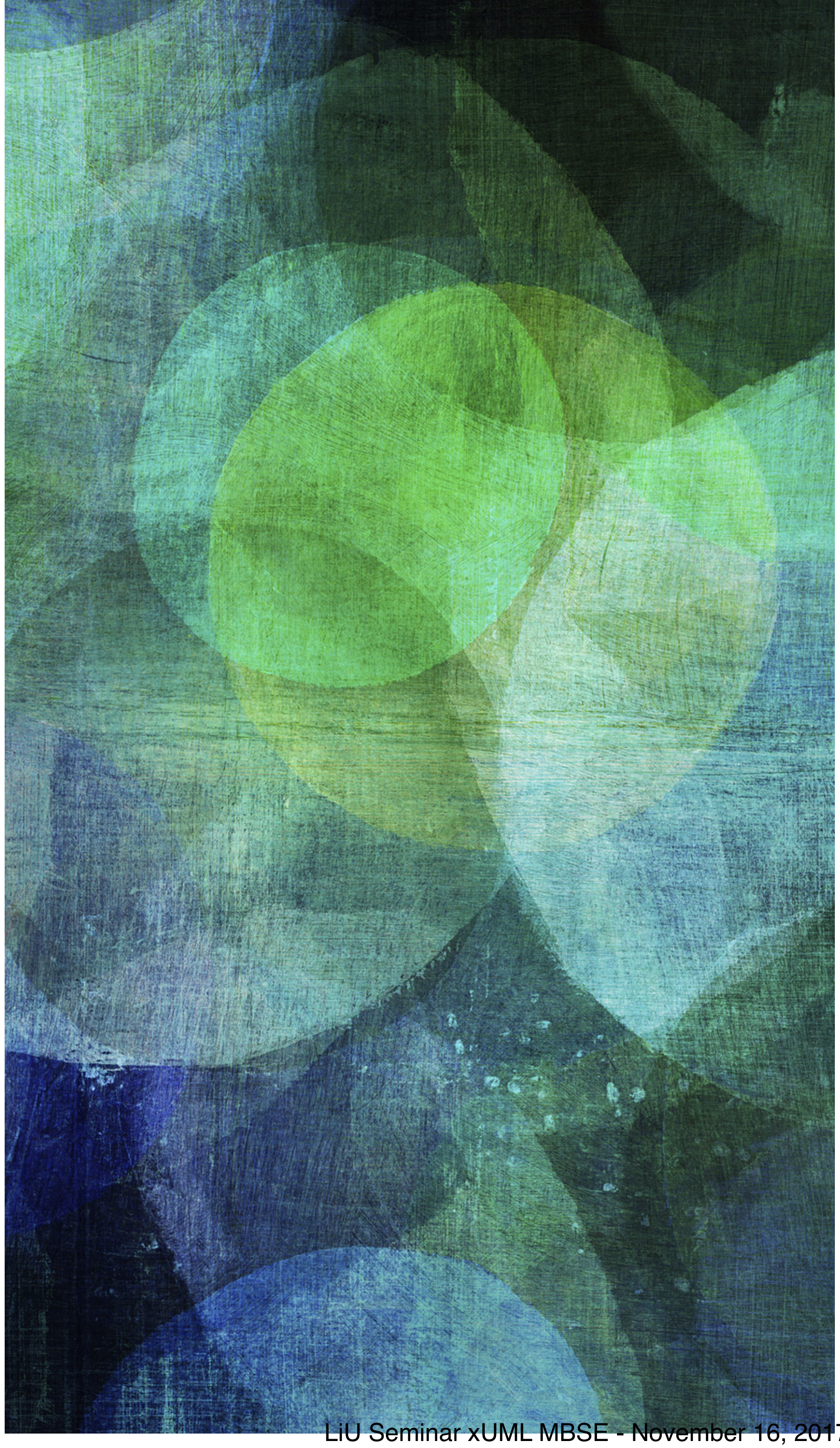
- Information
- Rules and constraints
- Real world behavior
- Essential computation

and to separate these from the platform.

Mathematical, executable semantics for each model type

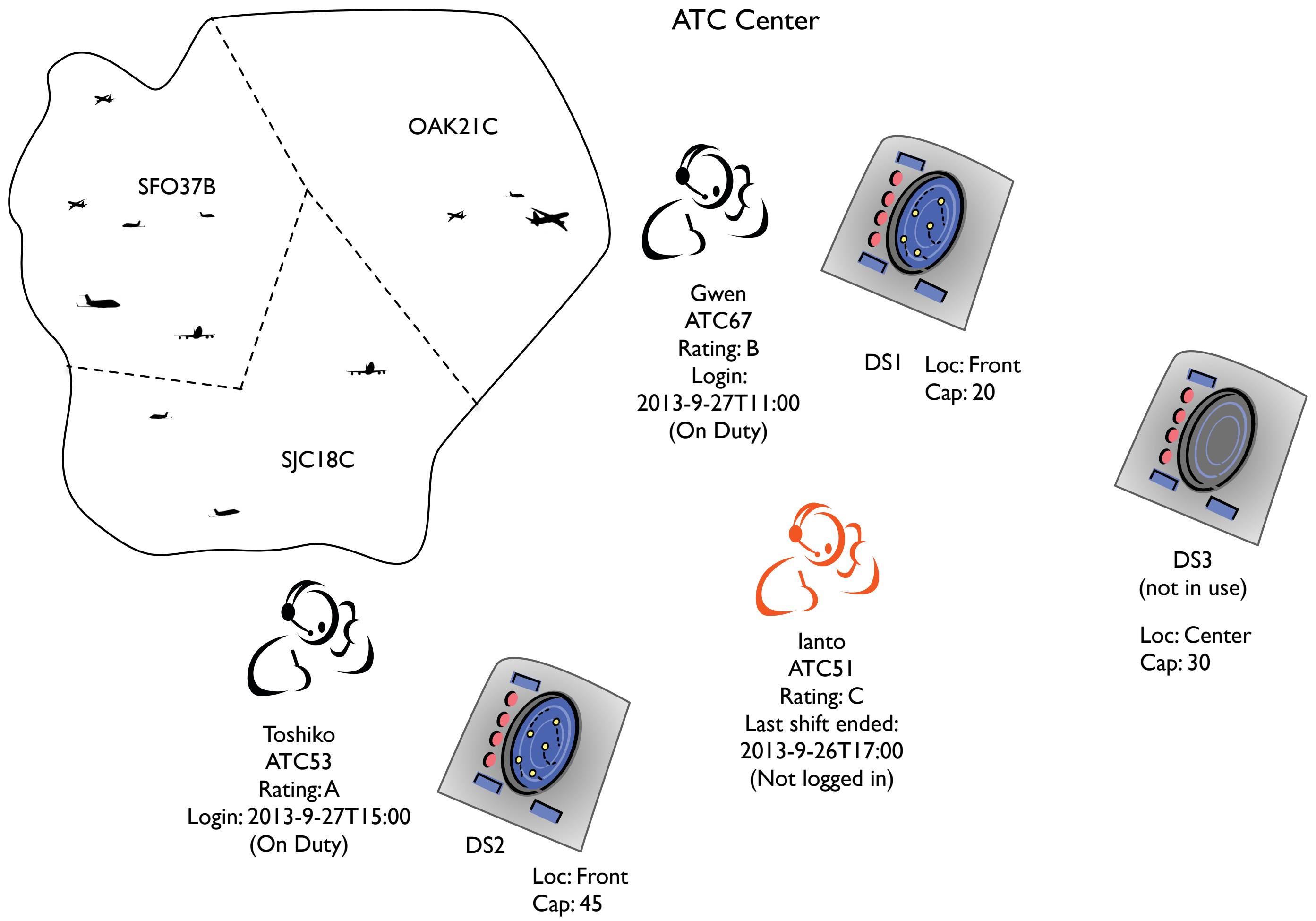
Doesn’t presume object-oriented implementation

AN EXAMPLE XUML MODEL



AIR TRAFFIC CONTROL EXAMPLE

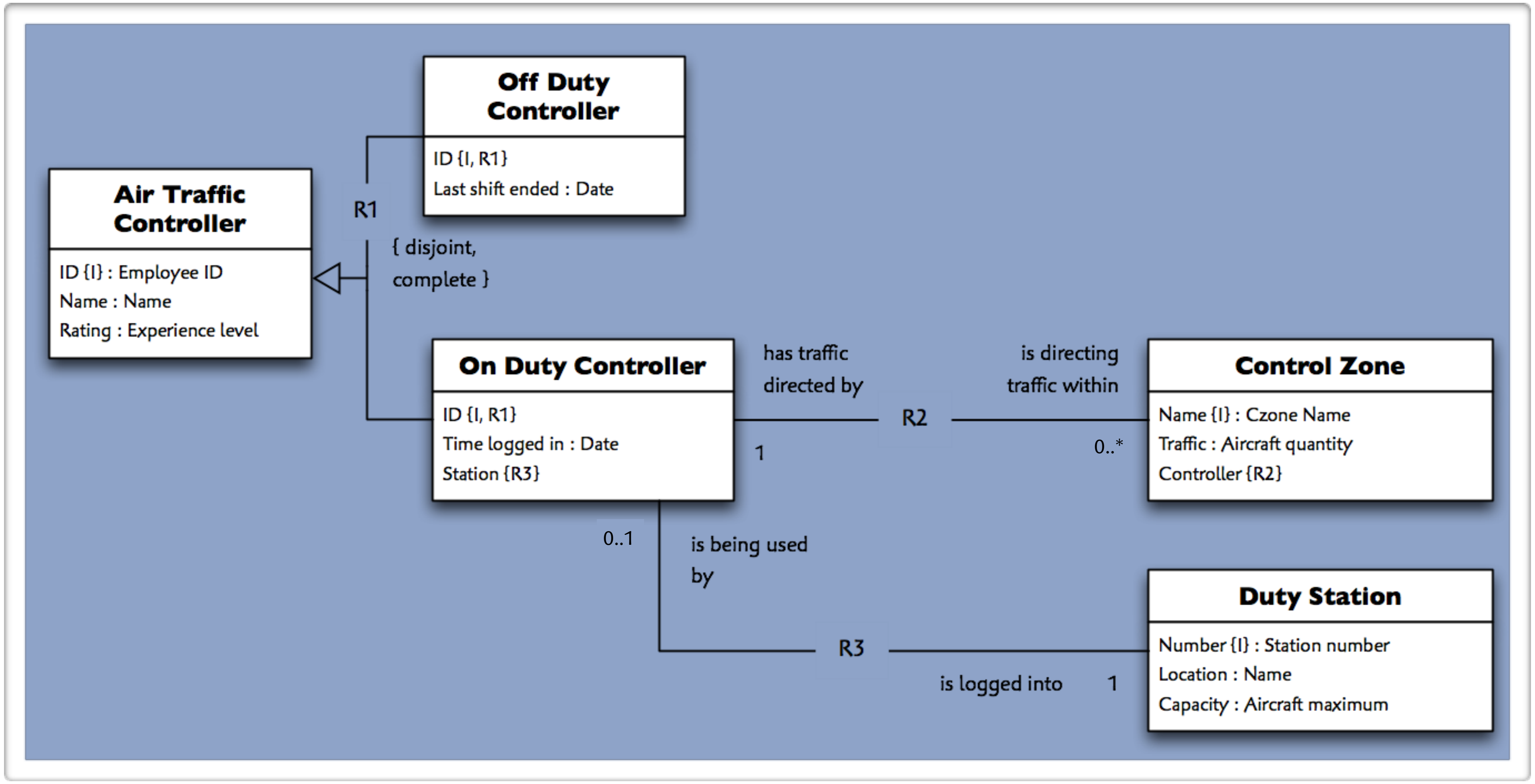
Subject matter (domain)



RULES / REQUIREMENTS

1) A controller can not direct air traffic while off duty.
2) An on duty controller must be logged into a duty station.
3) A duty station may or may not be available.
4) A control zone must have its traffic directed by one air traffic controller at all times.
5) An air traffic controller may not work a shift longer than two hours and fifteen minutes.

THE CLASS MODEL



POPULATION

Air Traffic Controllers

ID {I}	Name	Rating
ATC53	Toshiko	A
ATC67	Gwen	B
ATC51	Ianto	C

Same object

Superclass table

On Duty Controllers

ID {I, R1}	Time logged in	Duty Station {R2}
ATC53	9/27/13 15:00	DS2
ATC67	9/27/13 11:00	DS1

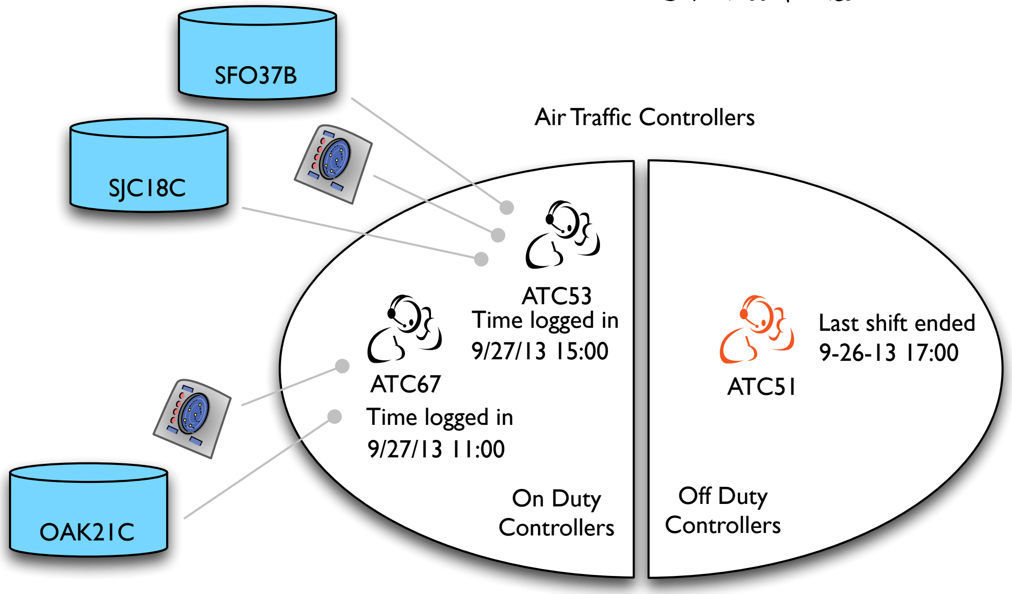
Off Duty Controllers

ID {I, R1}	Last shift ended
ATC51	9-26-13 17:00

Subclass tables

Duty Station

Number {I}	Location	Capacity
DS1	Front	20
DS3	Center	30
DS2	Front	45

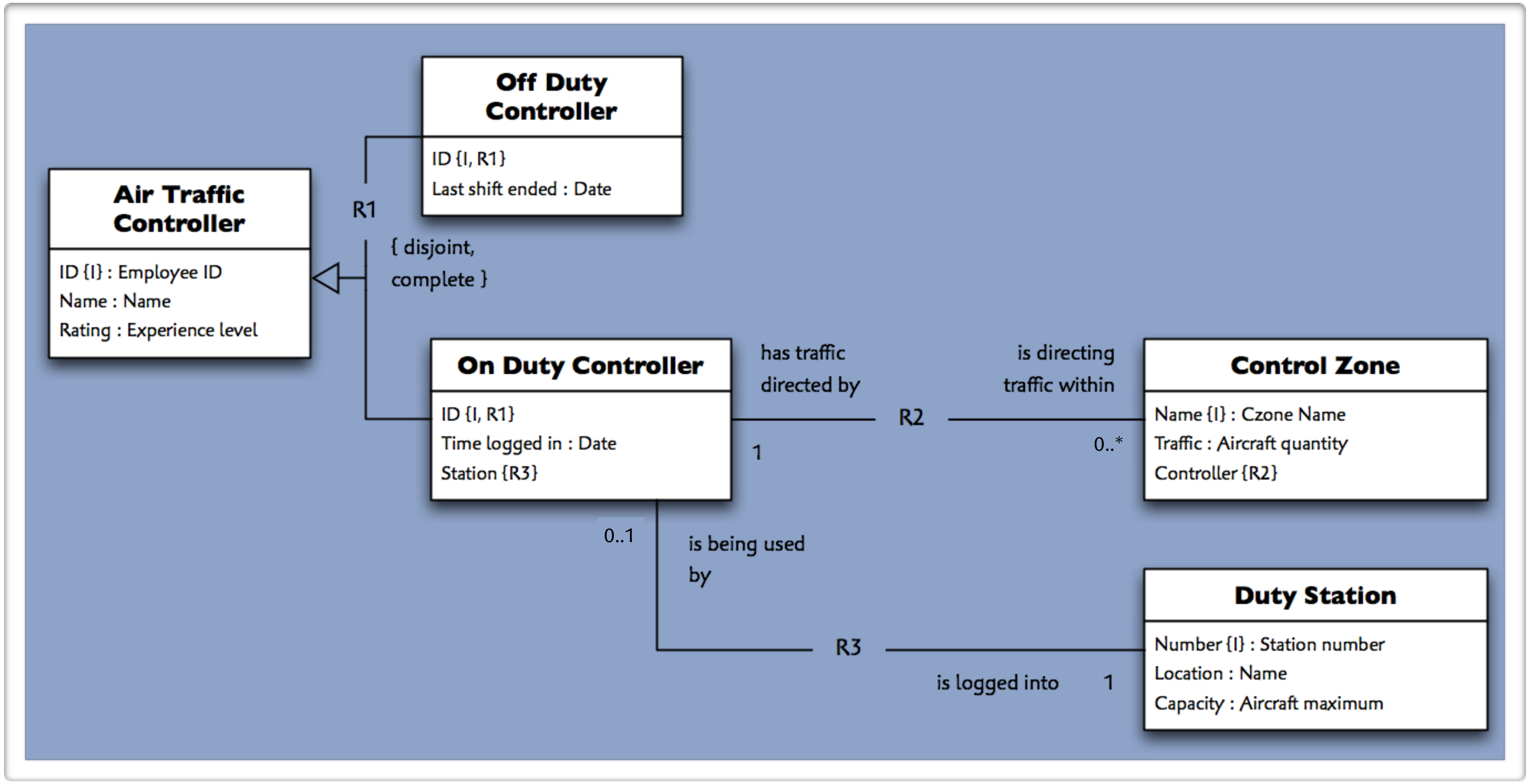


Control Zones

Name	Traffic	Controller
SJC18C	30	ATC53
SFO37B	25	ATC53
OAK21C	15	ATC67

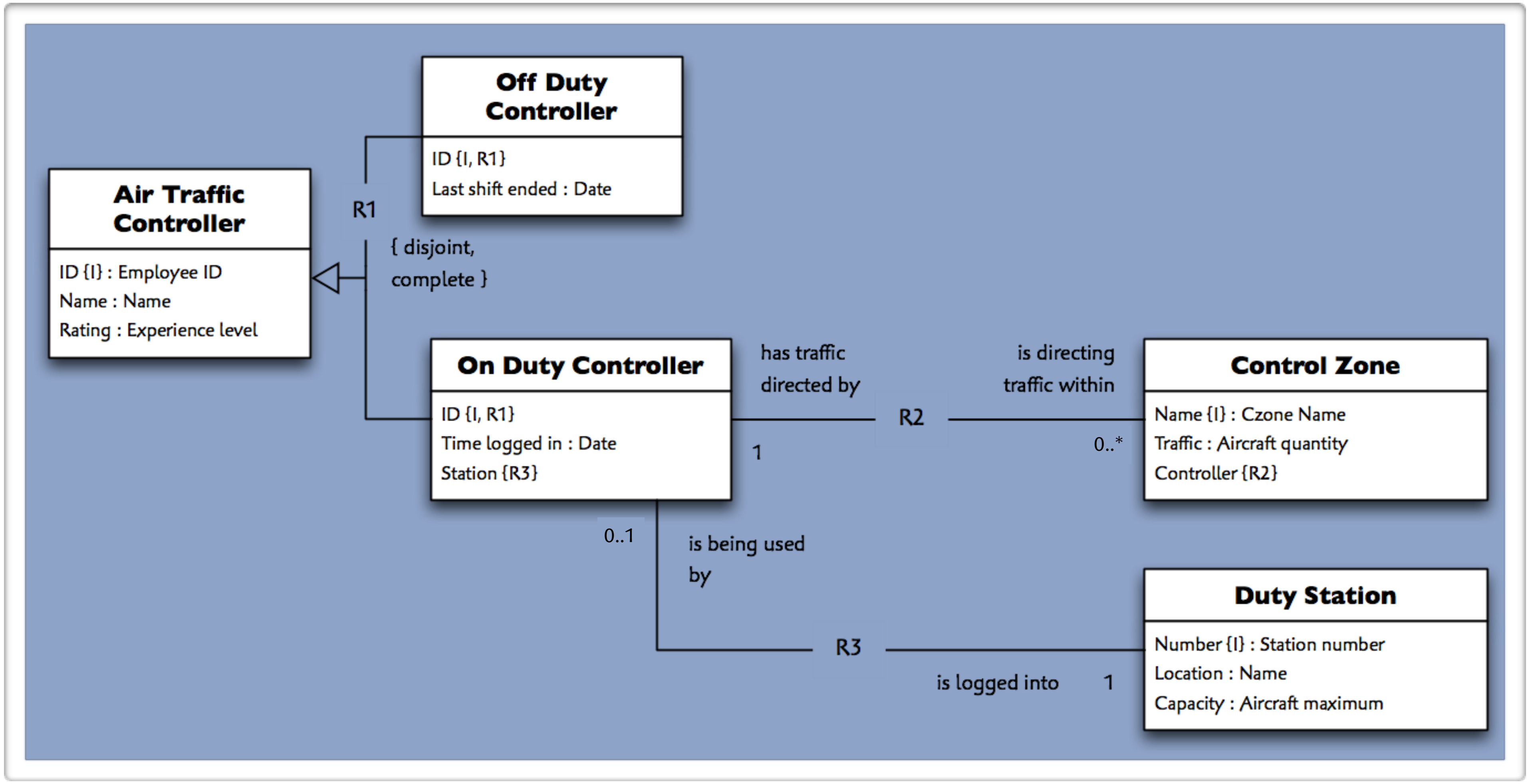
LOGIC AND CONSTRAINTS

Off Duty ATC goes On Duty?



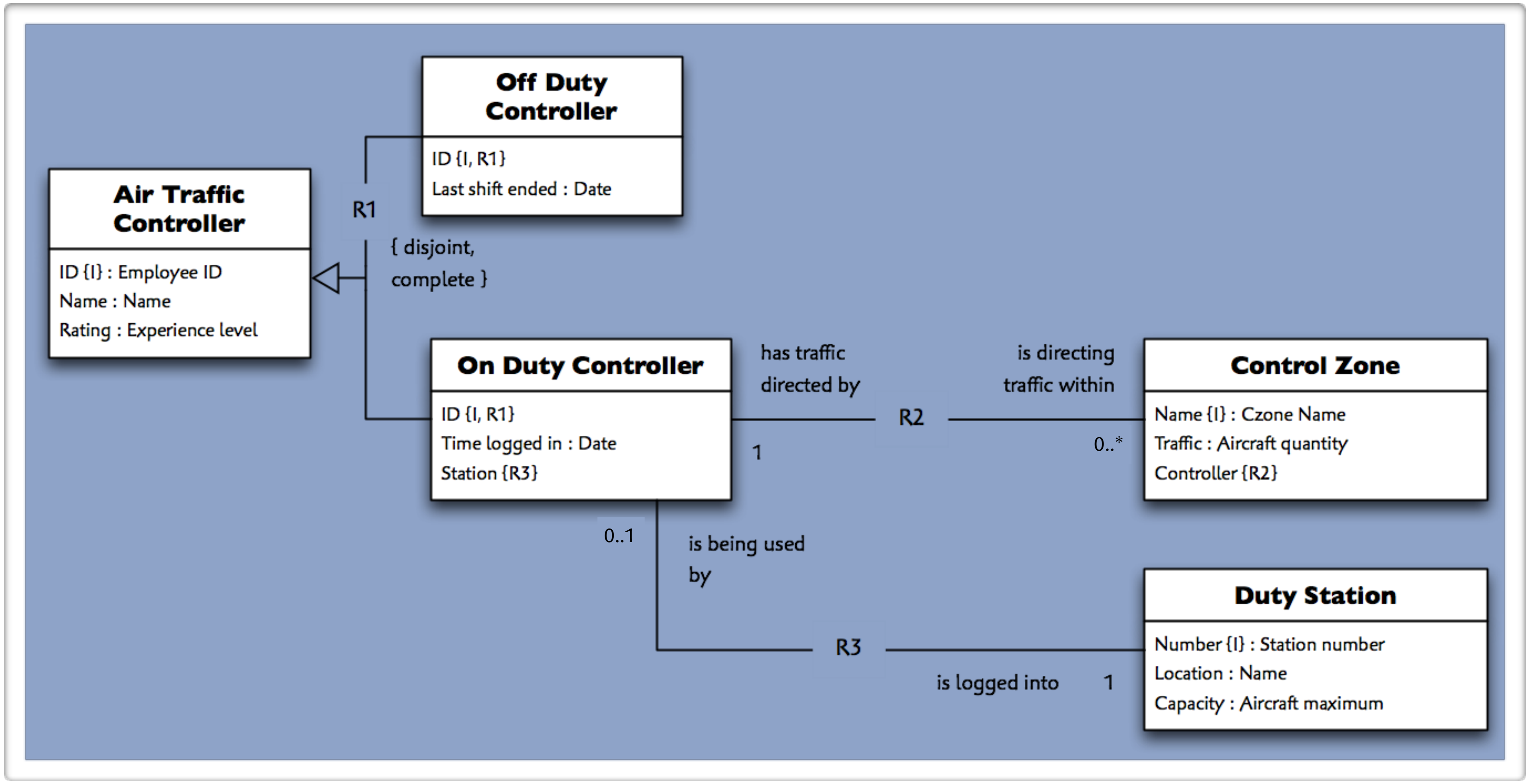
LOGIC AND CONSTRAINTS

If there is only one On Duty Controller, can he or she go Off Duty?



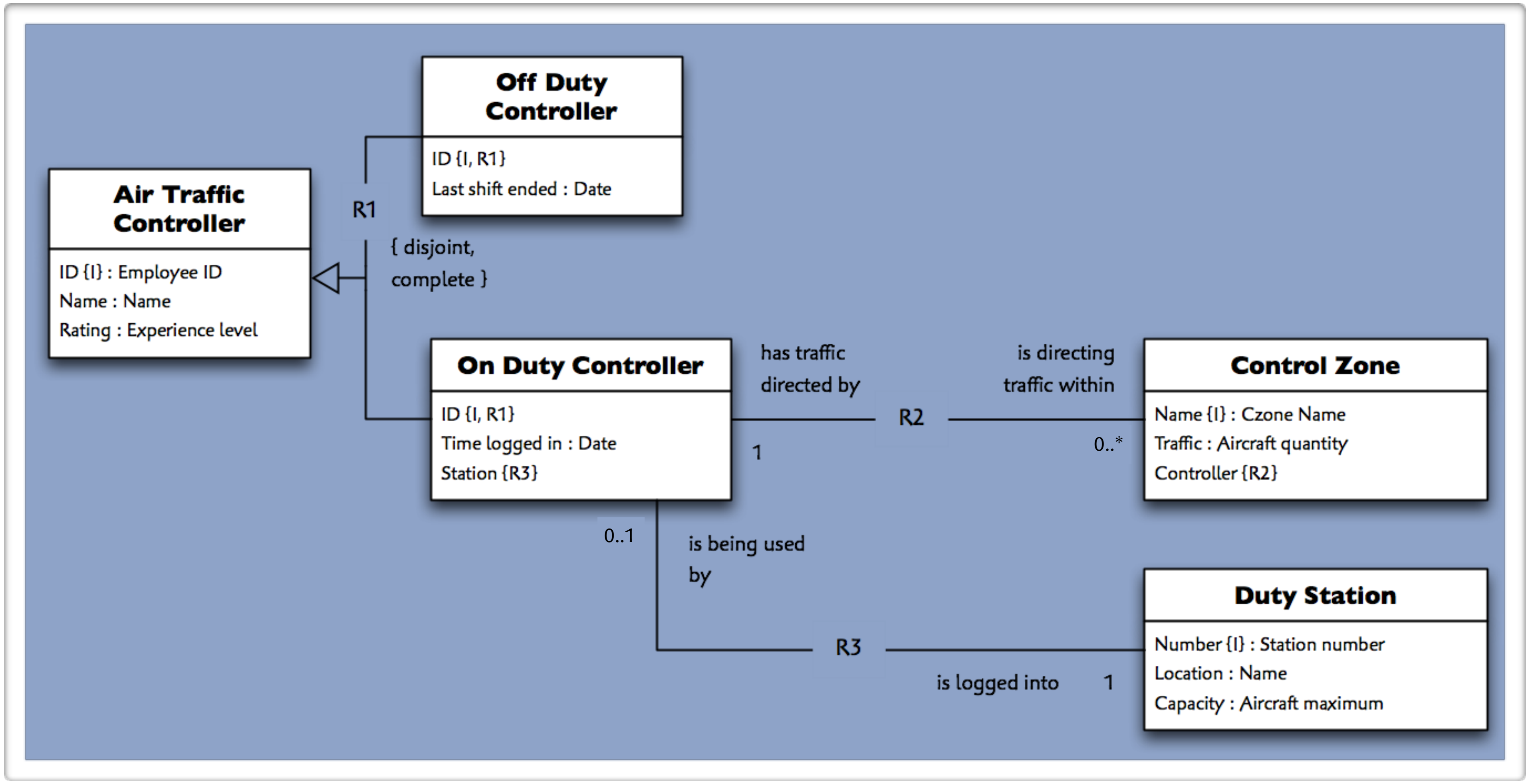
LOGIC AND CONSTRAINTS

If every Control Zone is being directed, can another Off Duty Controller log in?



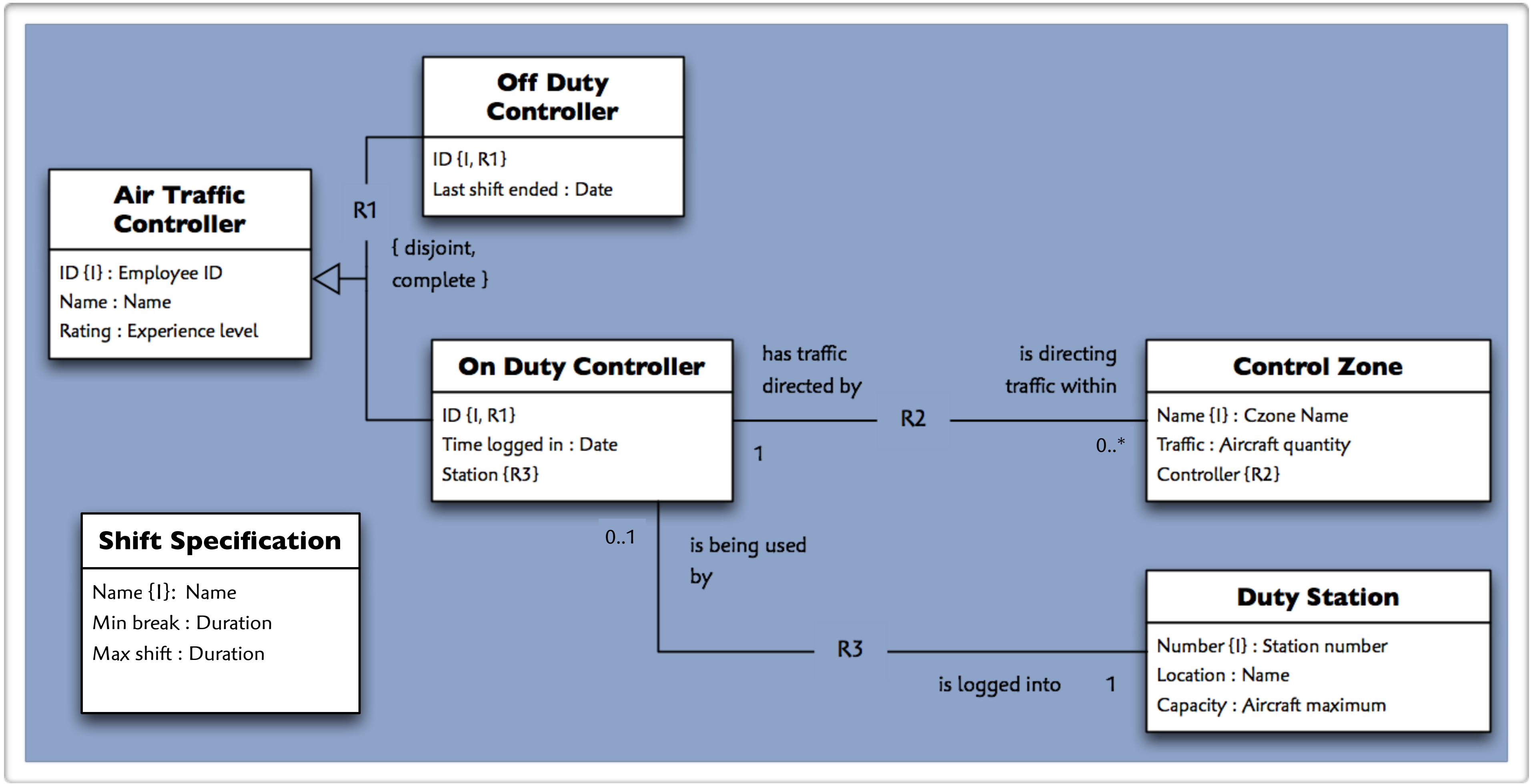
LOGIC AND CONSTRAINTS

Do we know when a shift should end?



LOGIC AND CONSTRAINTS

Do we know when a shift should end?



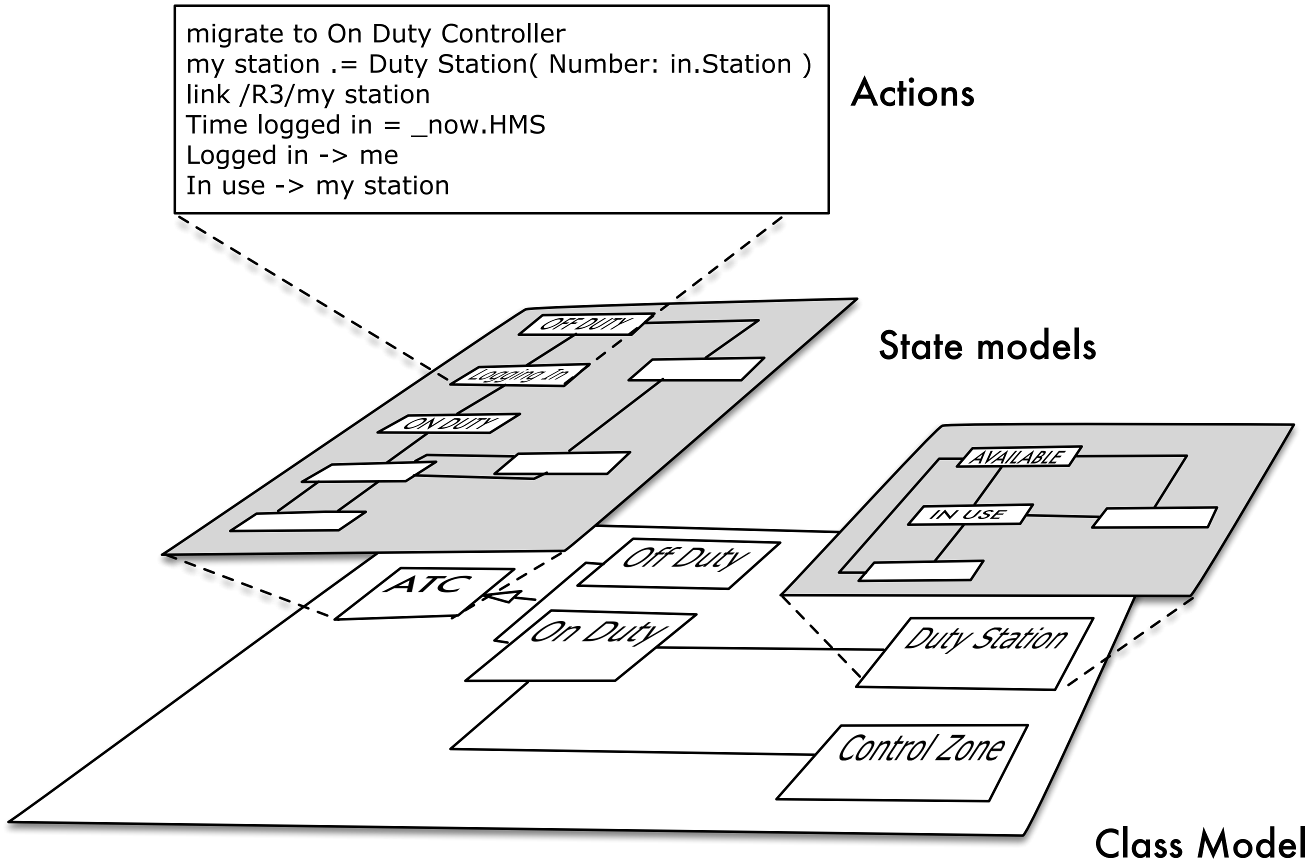
EXCLUDED DETAILS

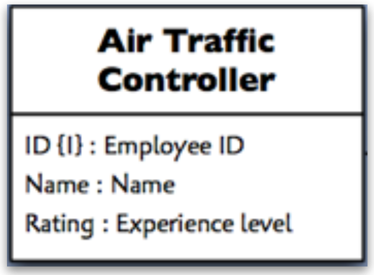
- States, algorithms, functions are filtered out
- Platform specific features are filtered out
- Implementation choices are filtered out

Lean modeling language – minimal symbols

So you can define and evaluate the application logic without distraction

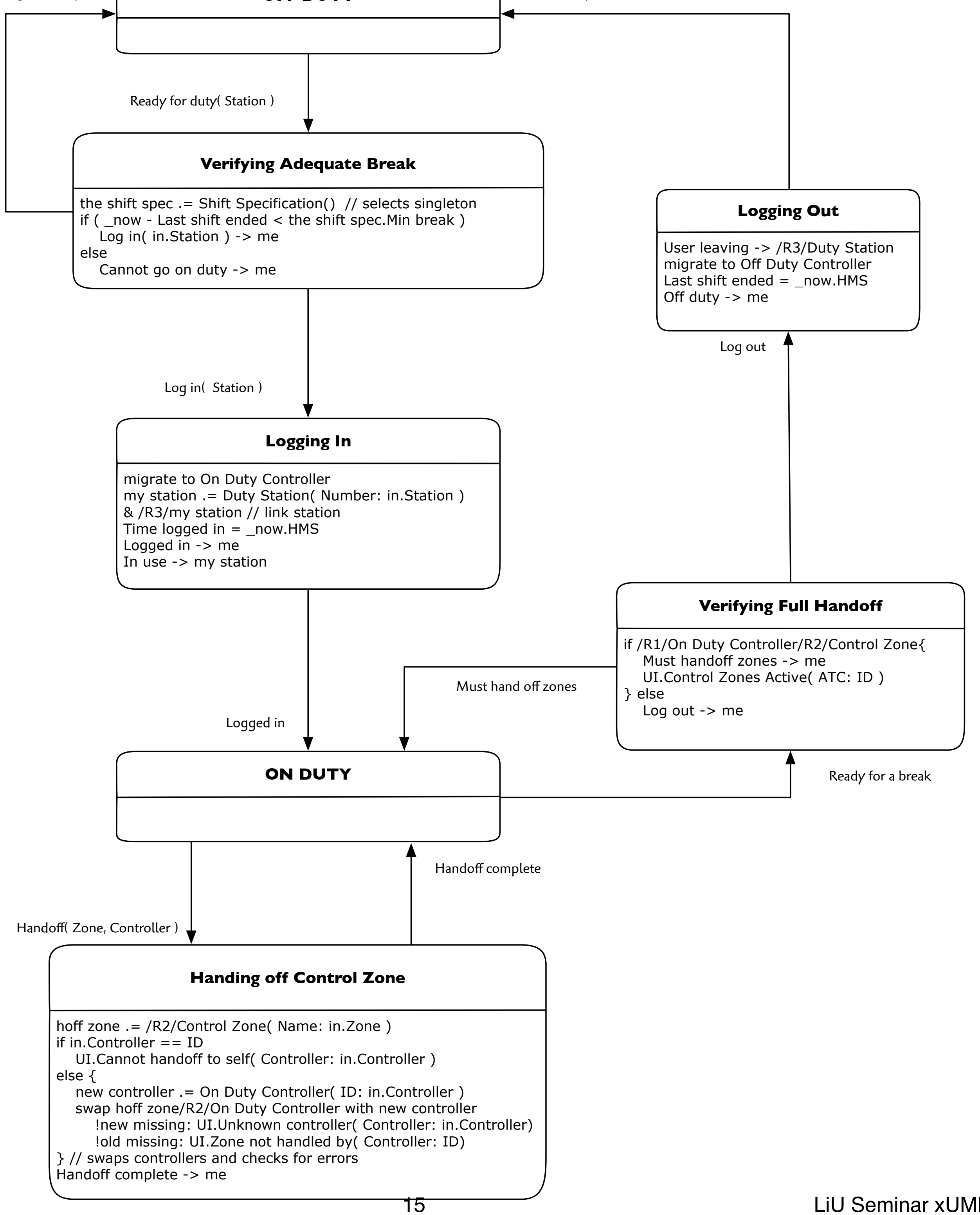
MODELING BEHAVIOR WITH STATES AND ACTIONS


```
migrate to On Duty Controller
my station .= Duty Station( Number: in.Station )
link /R3/my station
Time logged in = _now.HMS
Logged in -> me
In use -> my station
```



Air Traffic Controller

ID {I} : Employee ID
Name : Name
Rating : Experience level



Verifying Adequate Break

```
the shift spec .= Shift Specification()  // selects singleton
if ( _now - Last shift ended < the shift spec.Min break )
    Log in( in.Station ) -> me
else
    Cannot go on duty -> me
```

Logging In

```
migrate to On Duty Controller
my station .= Duty Station( Number: in.Station )
& /R3/my station // link station
Time logged in = _now.HMS
Logged in -> me
In use -> my station
```

ON DUTY

Logging Out

```
User leaving -> /R3/Duty Station
migrate to Off Duty Controller
Last shift ended = _now.HMS
Off duty -> me
```

Verifying Full Handoff

```
if /R1/On Duty Controller/R2/Control Zone{
    Must handoff zones -> me
    UI.Control Zones Active( ATC: ID )
} else
    Log out -> me
```

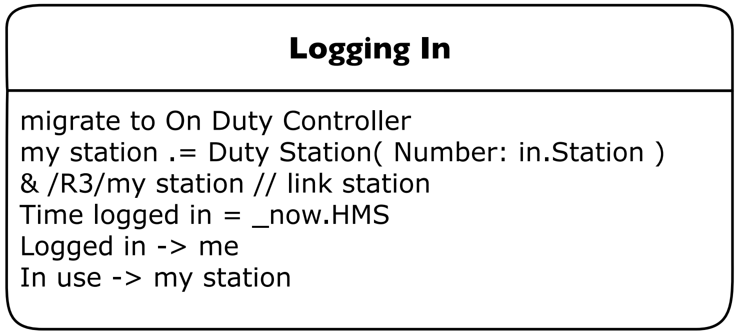
Handing off Control Zone

```
hoff zone .= /R2/Control Zone( Name: in.Zone )
if in.Controller == ID
    UI.Cannot handoff to self( Controller: in.Controller )
else {
    new controller .= On Duty Controller( ID: in.Controller )
    swap hoff zone/R2/On Duty Controller with new controller
        !new missing: UI.Unknown controller( Controller: in.Controller)
        !old missing: UI.Zone not handled by( Controller: ID)
} // swaps controllers and checks for errors
Handoff complete -> me
```

EXCLUDED DETAILS

- How execution is implemented
- How synchronization is implemented
- Distribution across processes/processors

ACTIONS – EXCLUDED DETAILS

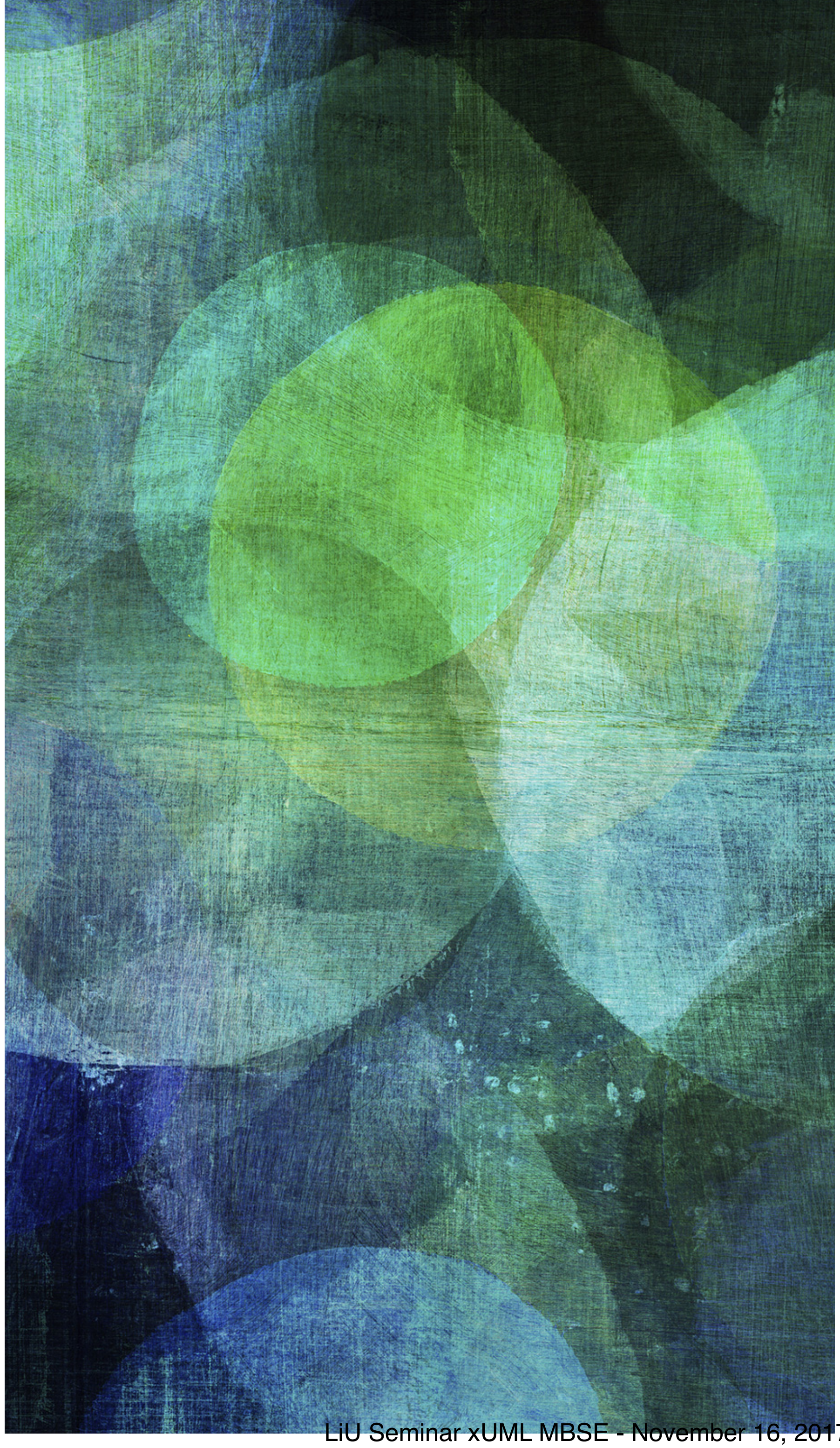


```
migrate to On Duty Controller
my station .= Duty Station( Number: in.Station )
& /R3/my station // link station
Time logged in = _now.HMS
Logged in -> me
In use -> my station
```

- Non-essential ordering of computation
- Data access implementation

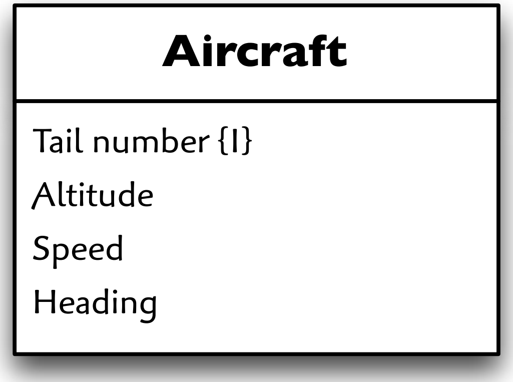
FOUNDATION SEMANTICS

of the xUML language



CLASS MODEL SEMANTICS

A class is a set of things in the real world such that all things in the set:

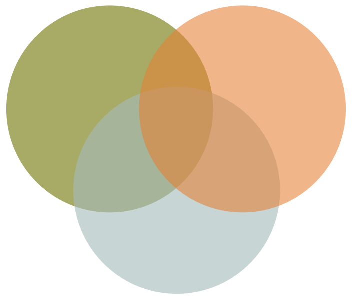


- have the same characteristics
- exhibit the same behavior
- are constrained by the same rules

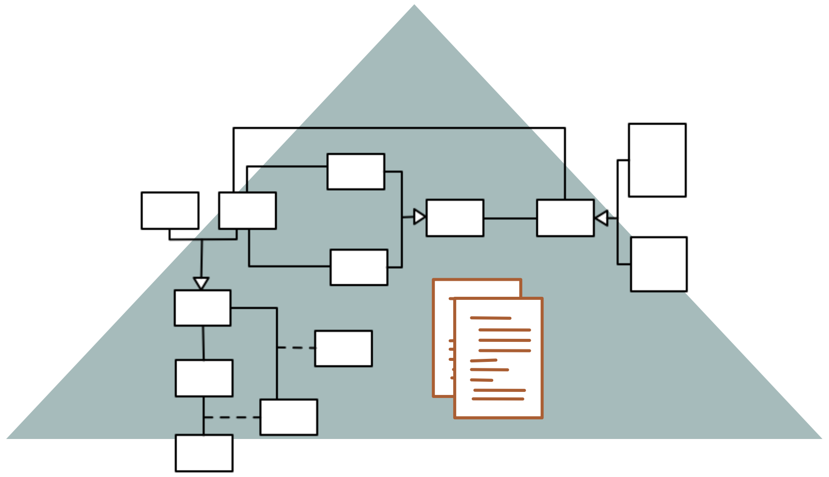
Aircraft

Tail Number {I}	Altitude	Airspeed	Heading

RELATIONAL THEORY



SET THEORY



FUNCTIONS

$y = f(x)$

PREDICATE LOGIC

forall x such that …

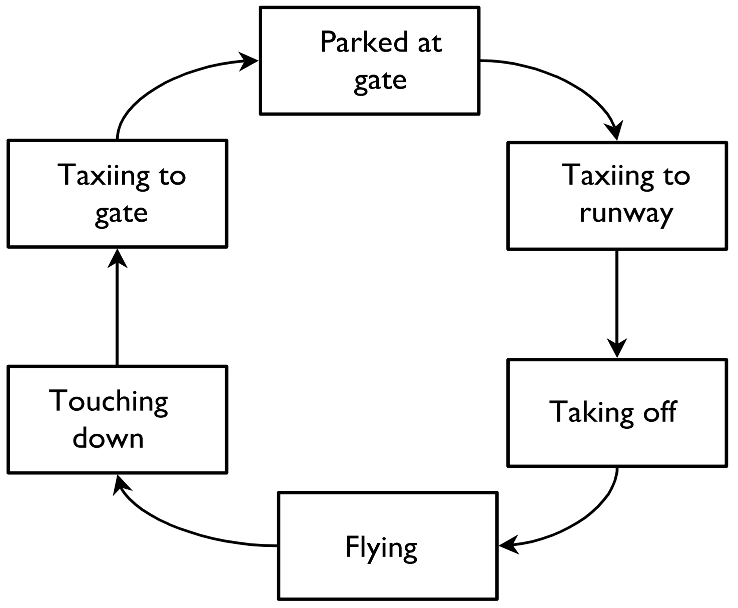
A CLASS PREDICATE

- Each class in your model is an n-ary predicate where n is the number of attributes
- An ATC has an ID *i*, a Name *n* and a Rating *r*
- We can turn this into a proposition (true/false statement) by instantiating it
- ATC53 is named Ianto and has Rating B (true/false statement)
- If the instance does not exist, it is false

STATE MODEL SEMANTICS

Aircraft
Tail number {I}
Altitude
Airspeed
Heading
Position

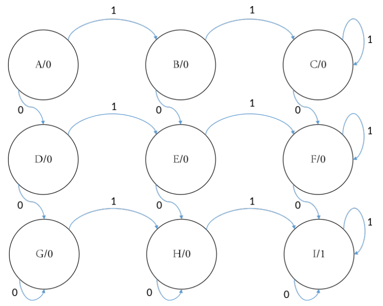
Same characteristics, rules, behavior

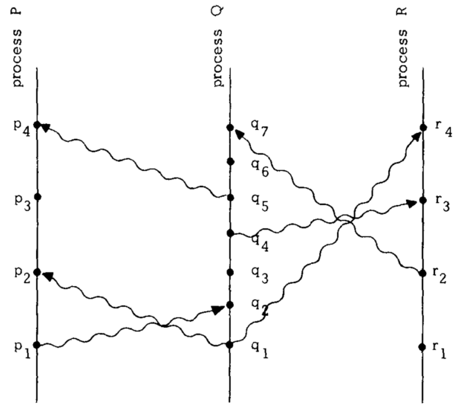


Common lifecycle

THE THEORY

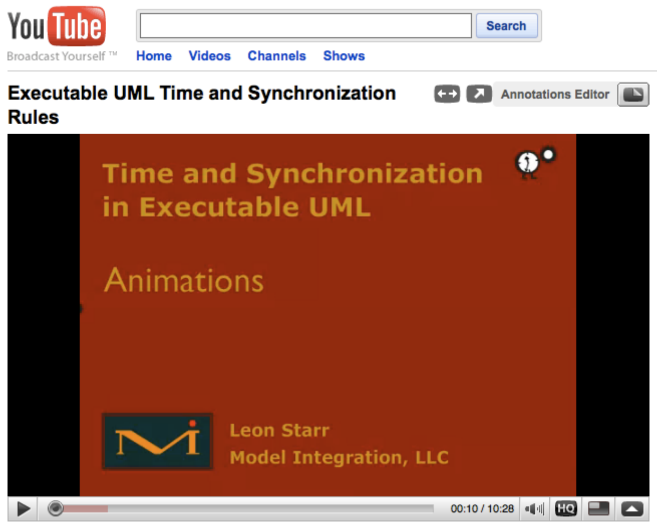
Moore state machine





Leslie Lamport

Time, Clocks, and the Ordering of Events in a Distributed System

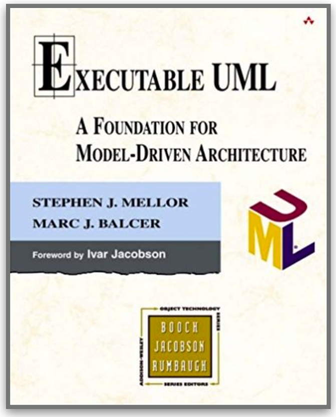

Executable UML Time and Synchronization Rules

Time and Synchronization in Executable UML

Animations

Leon Starr
Model Integration, LLC

Video animation of the platform independent synchronization rules


EXECUTABLE UML
A FOUNDATION FOR MODEL-DRIVEN ARCHITECTURE
STEPHEN J. MELLOR
MARC J. BALCER
Foreword by Ivar Jacobson

Also described here

PLATFORM INDEPENDENT EXECUTION RULES

What happens when event occurs while instance is executing a state activity?

Are events prioritized?

What is the duration of an activity?

Can event arrival order be guaranteed?

ACTION SEMANTICS

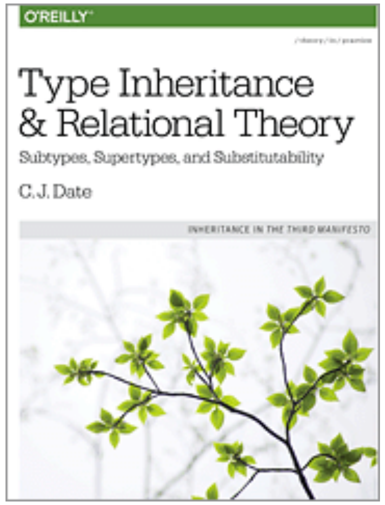
- Access data from class model
  - Based on relational semantics
- Send/receive event data on state model
  - Follows state machine execution rules
- Perform computations
  - Essential sequence only
  - No data implementation assumptions

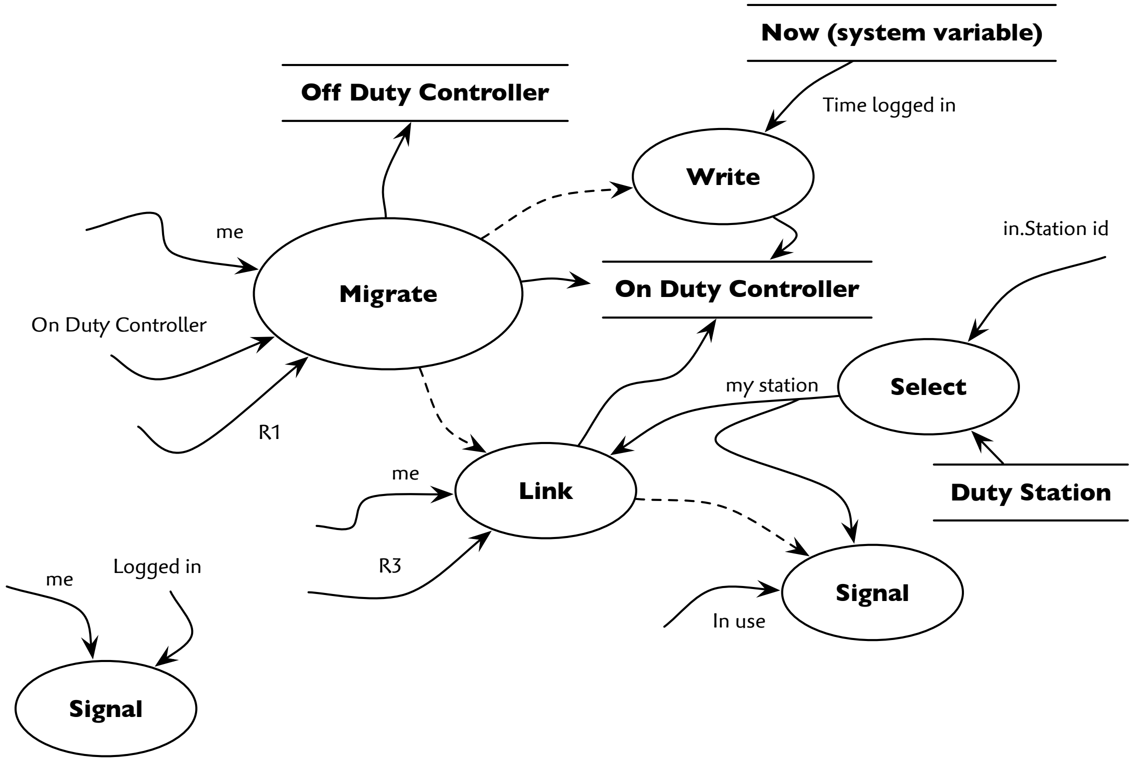
THE THEORY

Data access: Relational operations

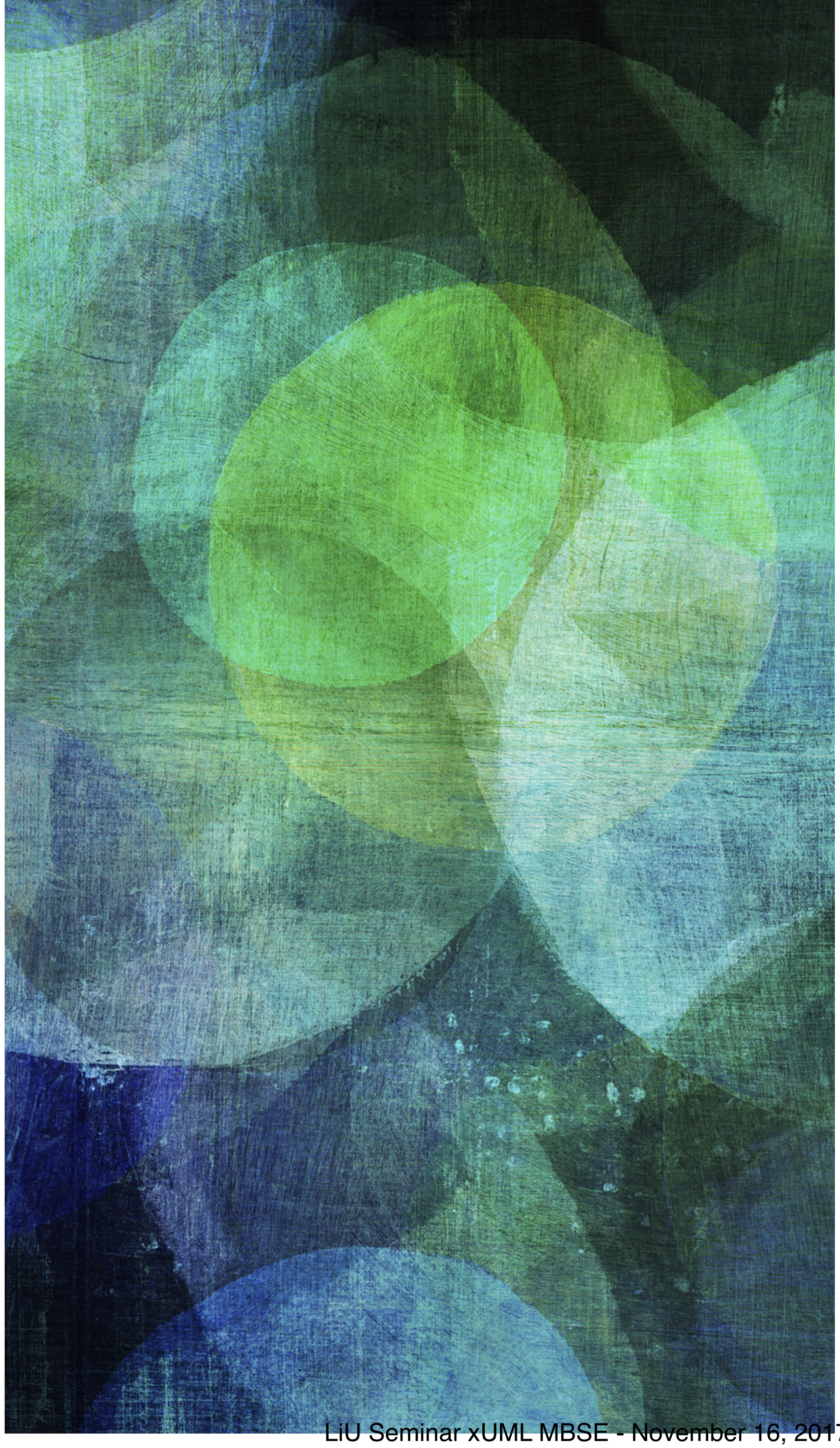
Sequencing: Data flow semantics

Data types: Type theory


O'REILLY
Type Inheritance & Relational Theory
Subtypes, Supertypes, and Substitutability
C.J. Date
INHERITANCE IN THE THIRD MANIFESTO



DOMAINS



DOMAIN DEFINITION

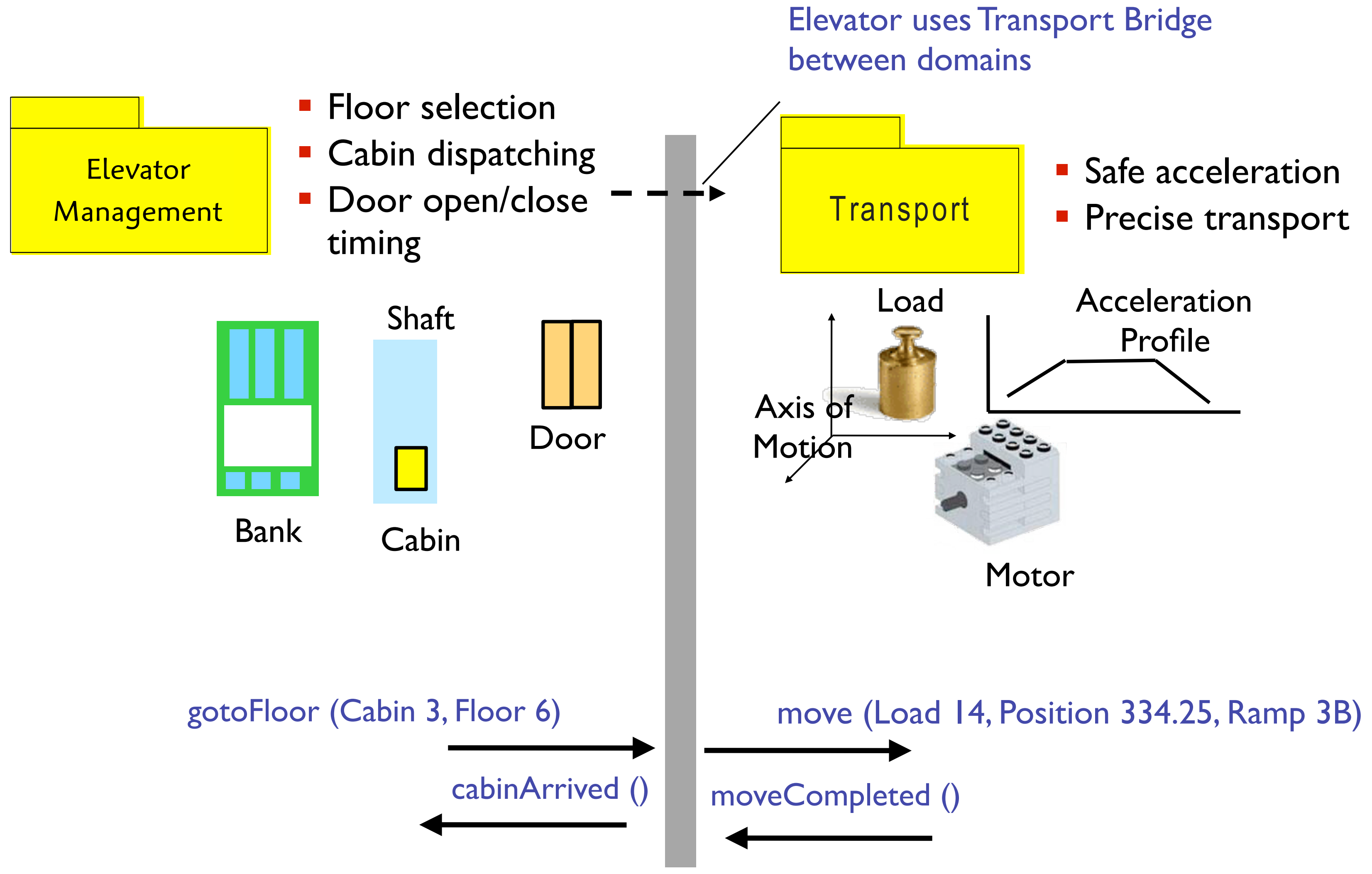
A domain is a distinct subject matter with its own vocabulary, rules, constraints and behavior.

Example: Linear Algebra

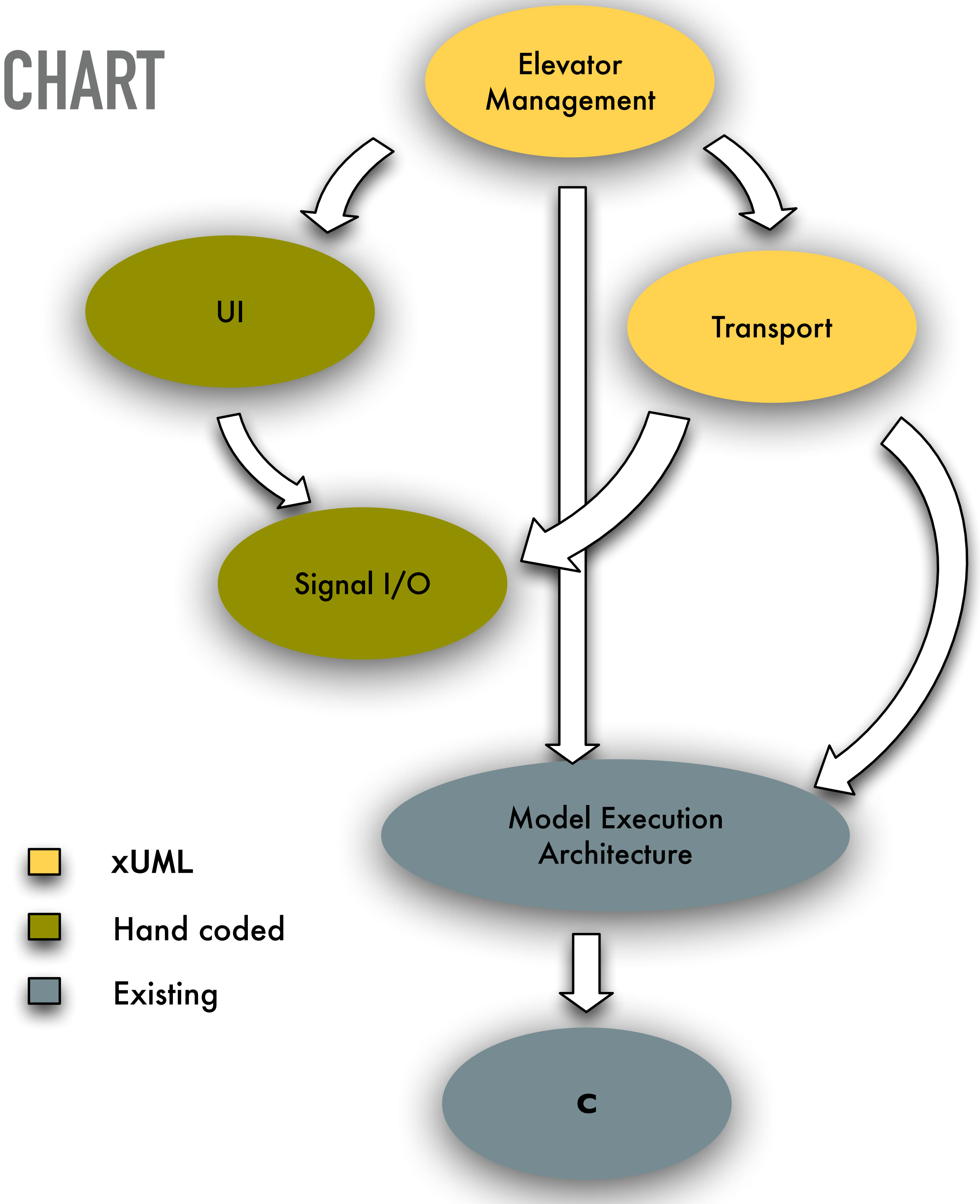
Not a domain: “Stuff that runs on processor X”, “Stuff in library Y”, “Stuff dept Z is coding up”

Domain partition of a system excludes details of deployment onto any particular platform features.

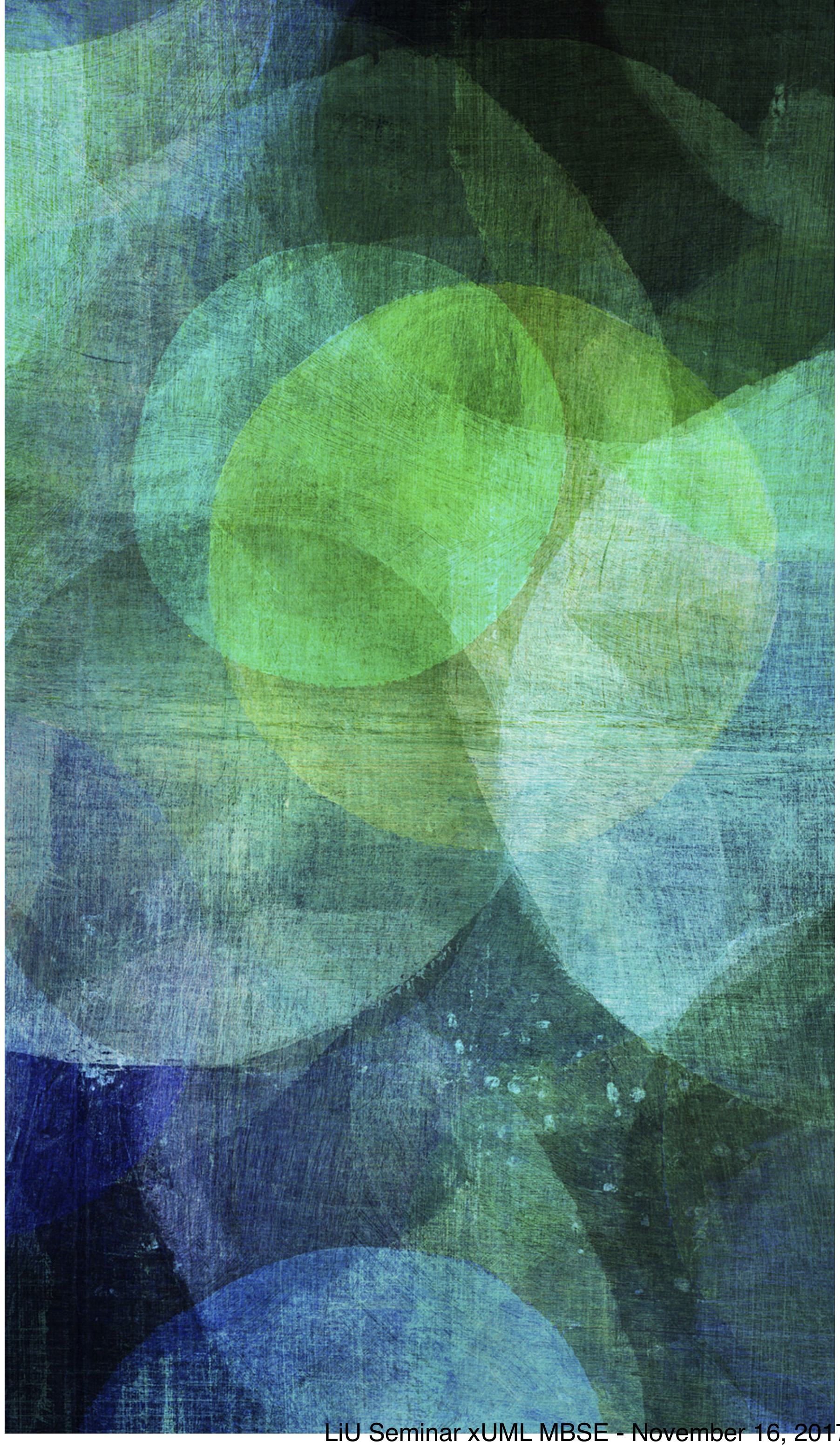
EXAMPLE DOMAIN SEPARATION



A COMPLETE DOMAIN CHART



CODE GENERATION

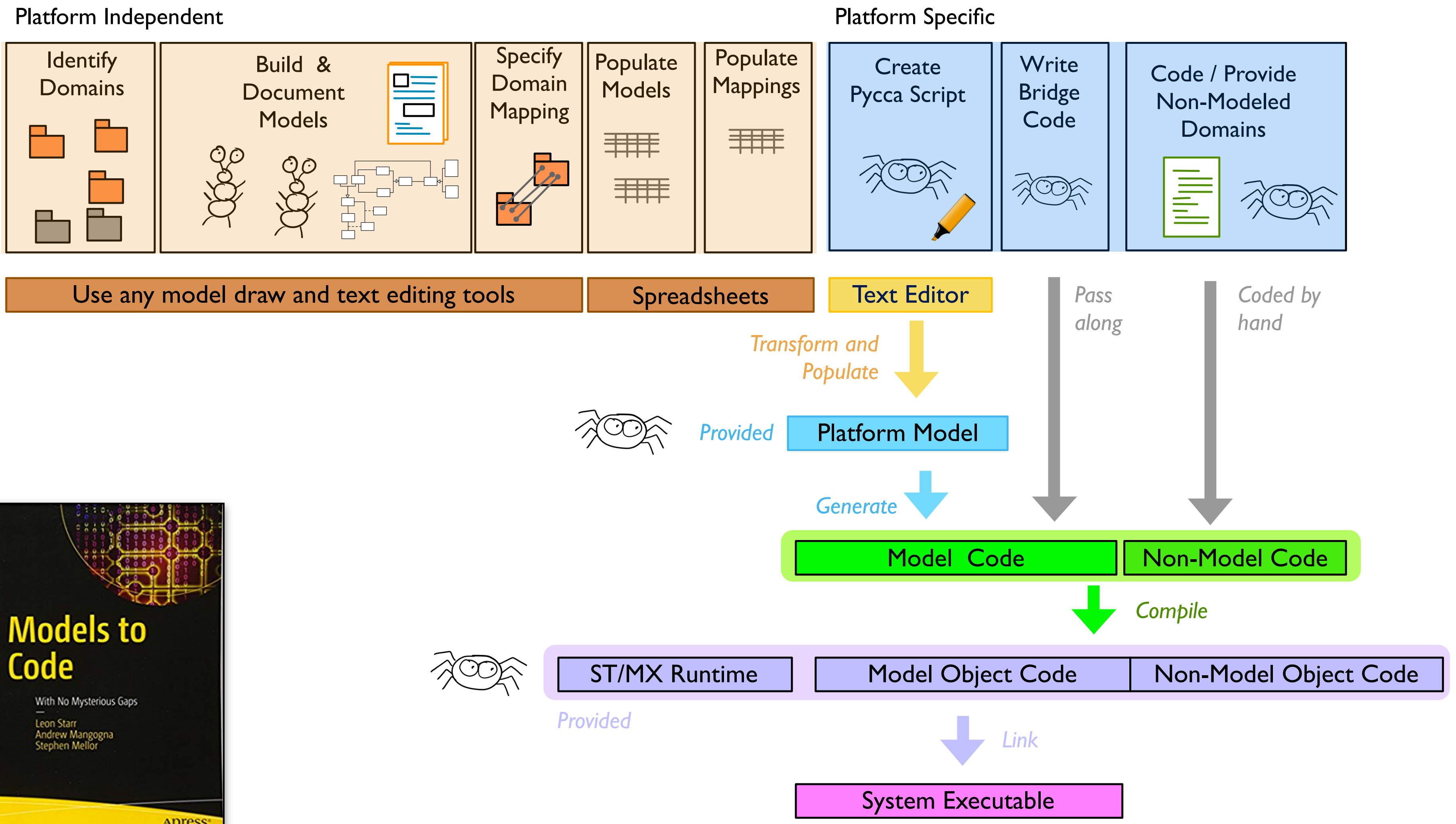


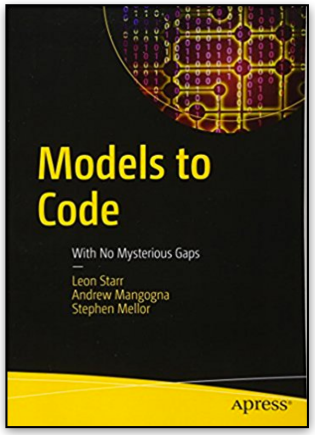
TRANSLATION PRINCIPLES

- Models are not modified by translation process
- Information is added, usually via some DSL (domain specific language / model markup)
- A runtime MX (model execution) platform is supplied
- A code generator which populates this platform (from the DSL) is provided
- The MX and code generator work for a class of platforms, e.g.
  - Embedded microcontroller
  - Cloud
  - Fault tolerant distributed

EXAMPLE APPROACH FOR A MICROCONTROLLER

Pycca Translation Workflow



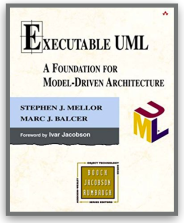
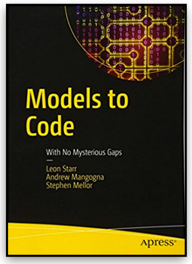


SUMMARY

- Executable UML is a complete modeling language
  - Concrete, unambiguous models of application requirements
  - Executable, platform independent modeling
  - Mathematical foundations
  - Does not presume an OO implementation
  - Domains accommodate other MBSE languages
- Model Translation
  - Models are not modified to generate code
  - So models can be reused and redeployed
  - Models are stable while platform details change

MORE RESOURCES

BOOKS

SITES

modelstocode.com

executableuml.org

www.modelint.com

MODEL INTEGRATION LLC

SOCIAL

facebook.com/modelint

twitter.com/leon_starr

EMAIL

Leon Starr

leon_starr@modelint.com