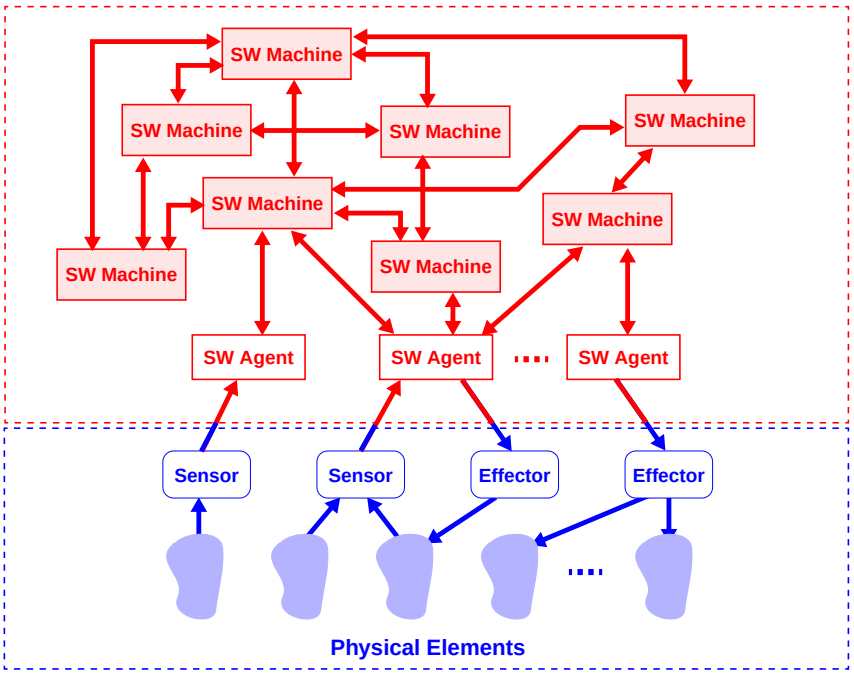
Real-Time Object-Oriented Modeling

Bran Selic
ObjecTime Limited
bran@objectime.on.ca

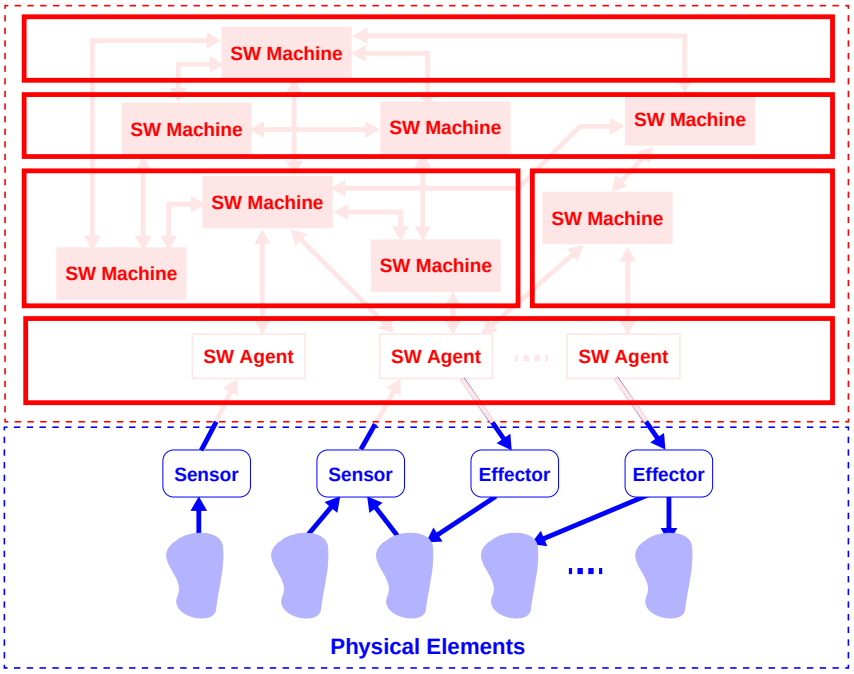
Tutorial Structure

- **Real-Time Systems and Architecture**
- **The ROOM Modeling Concepts**
- **Development Process**
- **Summary**

General Real-Time System

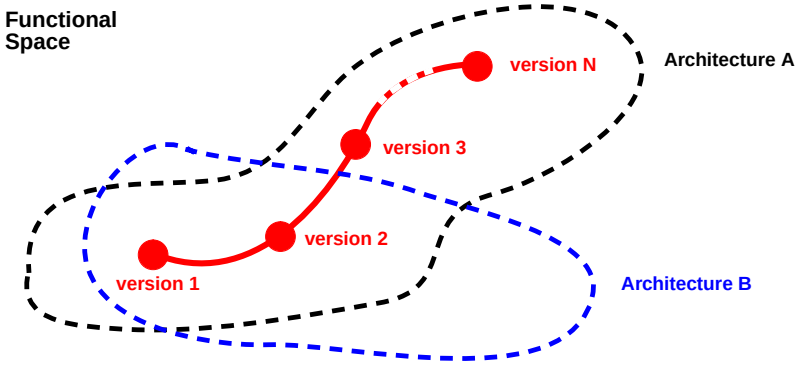


Software Architectures



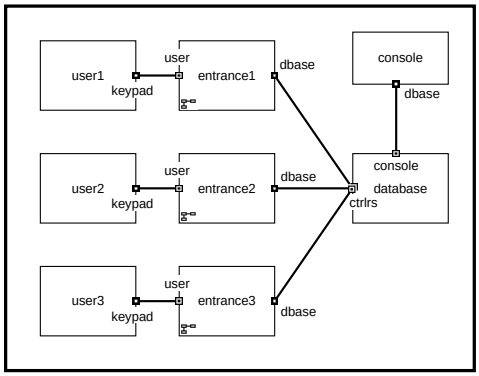
What is an Architecture?

- ***Architecture:*** **an abstract specification that constrains the *structural* and *behavioral* patterns of all system components**
- **Key to proper construction and evolution**

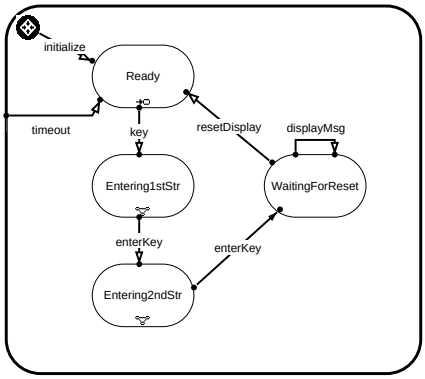


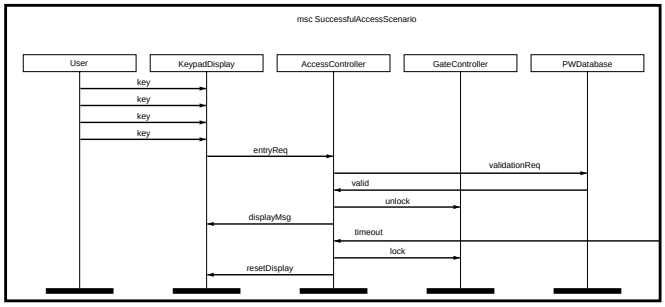
An Example Architectural Specification

High-Level System Structure



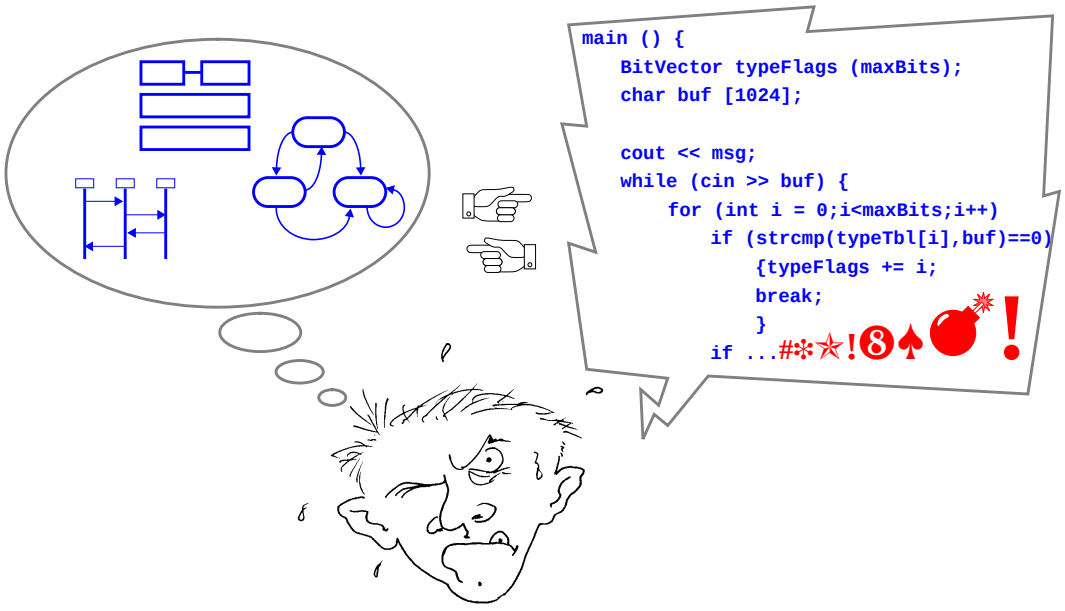
Behavior





Scenarios (Use Cases)

The Fate of Most Architectures...

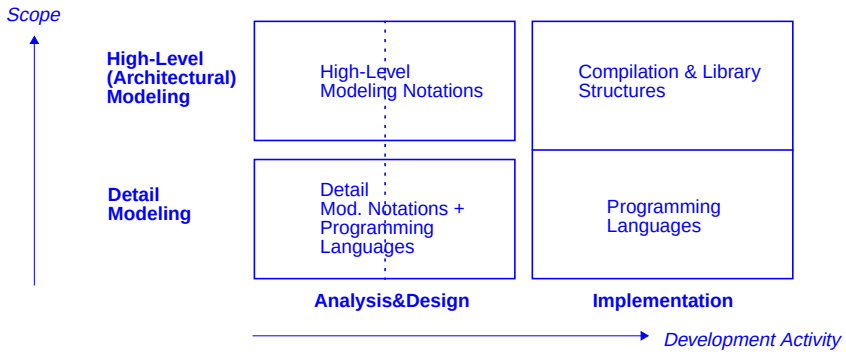


Why Architectures Fail

- **Interpretation errors**
  - ***"Hmm, I wonder what that bubble means?"***
- **Feasibility (technological) limitations**
  - ***"And, here, a miracle occurs..."***
  - ***"To simplify analysis, we will assume..."***
- **Translation errors**
  - ***"Oops!"***
- **Design changes**
  - ***"I'll show those ivory-tower types how to do it right..."***

Modeling and the Development Cycle

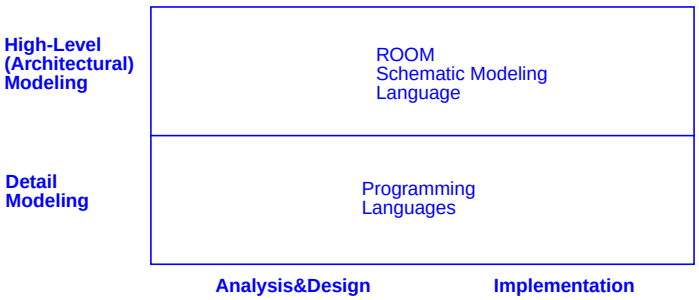
- **Traditional view**



- **Characterized by *semantic gaps* that discourage recursion and introduce errors**

The ROOM Approach

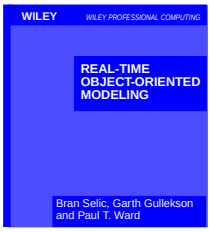
- ***R*eal-Time *O*bject-*O*riented *M*odeling**
- **Two level scheme with formally defined relationships between levels**



- **Incorporate standard programming languages (e.g., C++) for detail modeling**

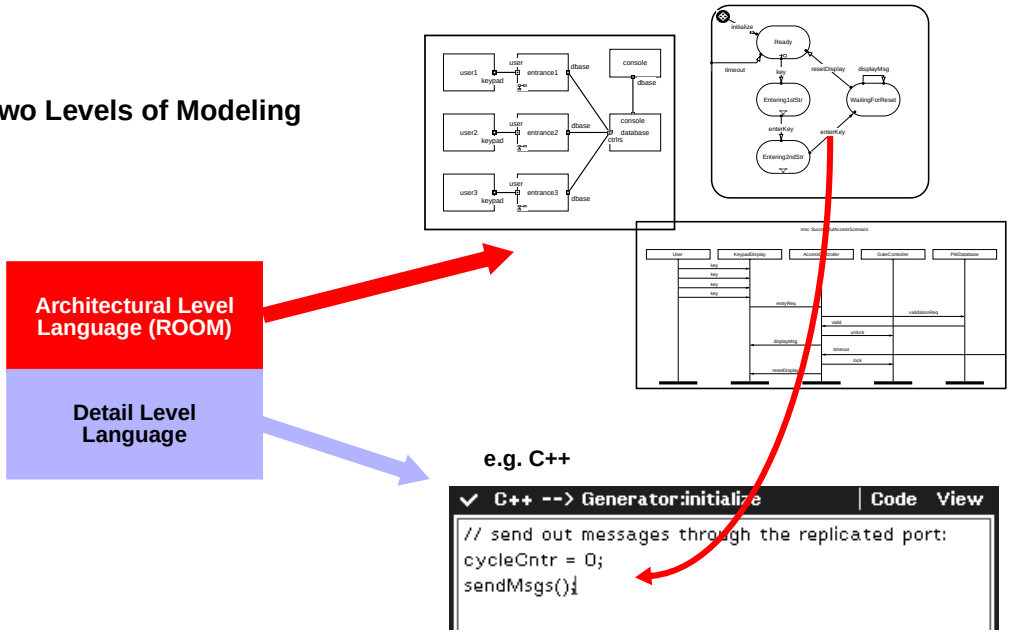
ROOM

- A method for developing software for real-time distributed systems
  - suited for event-driven systems
  - full-cycle method (A → D → I)
  - uses a formal graphical modeling language
  - suited for computer-based automation:
    - executable models
    - automatic code generation
- Originated in Bell-Northern Research



Modeling in ROOM

Two Levels of Modeling



Why Object-Oriented?

- **The structural aspect of architectures tends to be *object-based***
- **The object paradigm is a synergistic combination of proven, useful concepts**
  - **encapsulation, polymorphism, inheritance...**
- **The potential for re-use**
  - **formally supported only in implementation-level programming languages**
  - **can it be used at the architectural level?**
    - **e.g., can an architecture be inherited?**

- **Real-Time Systems and Architecture**
- **The ROOM Modeling Concepts**
- **Development Process**
- **Summary**

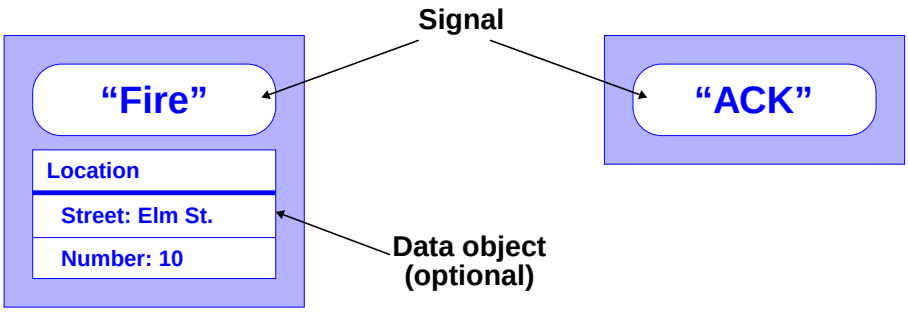
The Question of Notation...

"By relieving the mind of all unnecessary work, a good notation sets it free to concentrate on more advanced problems, and in effect increases the mental power of the race"

- Alfred N. Whitehead

Messages

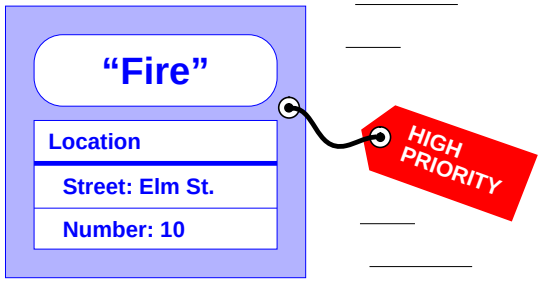
- **Objects that carry information about some event**



- **Message-based communication is well-suited to distributed systems**

Message Priorities

- A message in transit is assigned a *priority* that identifies the significance of its event



- Required to support timely responses

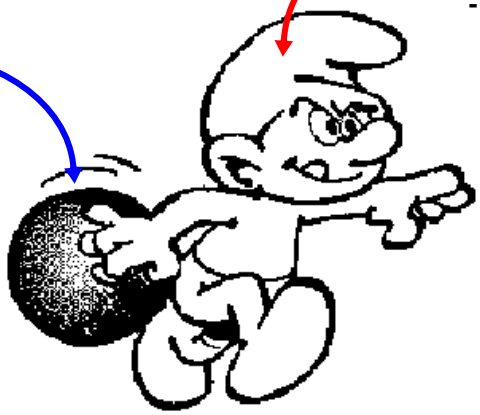
Object Types

ACTIVE

- actors
- own execution thread (concurrent)

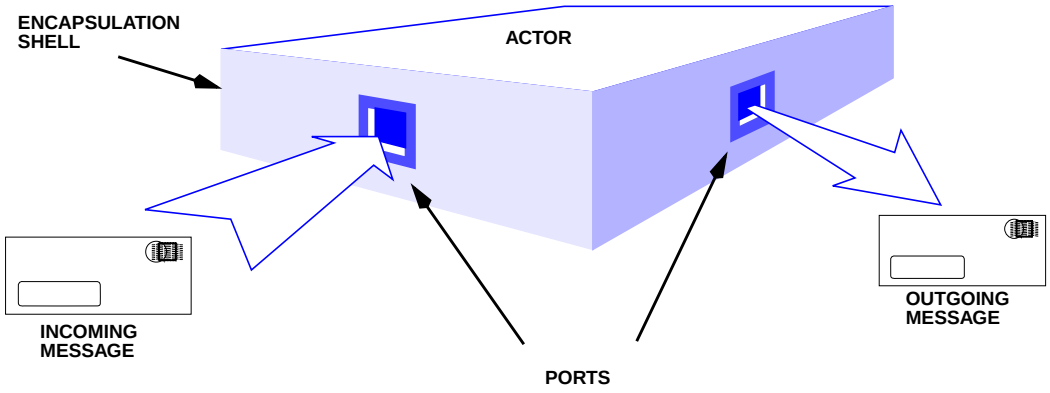
PASSIVE

- messages
- Detail Level data



Actors

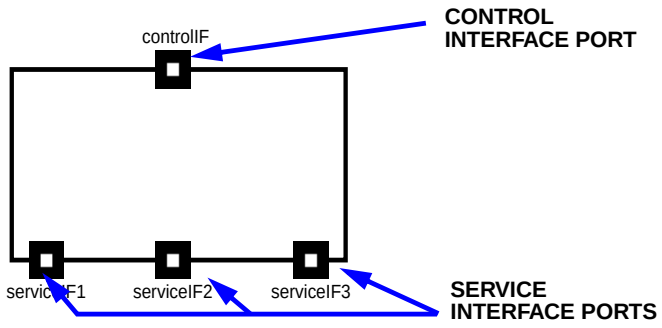
- The active objects of ROOM are called *actors*



- Actors can send and receive messages through distinct interfaces called *ports*

Actor Interfaces

- Each interface of an actor has a unique name and an associated *protocol*
- Different interfaces can use different protocols
- An actor can present different “views” of itself to different collaborators; e.g.:



Protocols

- An extension of conventional interface concept based on message exchange sequences
- A protocol is defined by:
  - a set of valid *message types* and their *directions* of flow

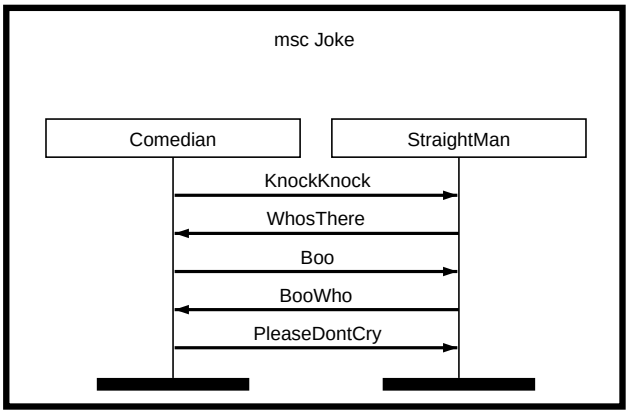
JokeProtocol | Protocol View

In Signals	Data Class	Out Signals	Data Class
KnockKnock	Null	WhosThere	Null
Boo	Null	BooWho	Null
PleaseDontCry	Null		

  - a set of valid message sequences (future extension)

Message Sequence Charts

- Message sequences are expressed by *Message Sequence Charts* (MSCs)



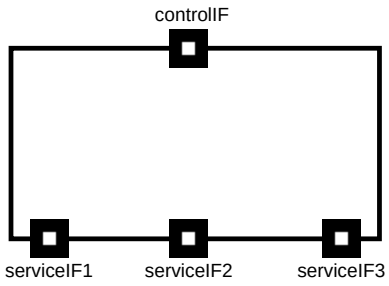
- Defined by ITU standard Z.120

Protocol Classes

- **Sometimes different actors use the same protocol definitions**
- **It is convenient to define protocols independently of the actors that use them through *protocol class* specifications**
- **Each port (actor interface) has a *type* that is defined by its protocol class**
- **Protocol classes represent reusable interface specifications; this is the basis for a type of polymorphism available in ROOM**

Actor Types

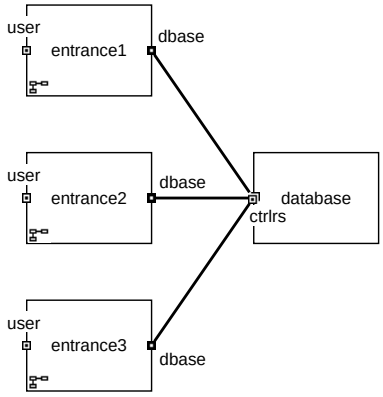
- **The collection of all interface types of an actor defines its type**



- **This is a basis for a form of *genericity***

Combining Actors — Actor Structures

- The object paradigm allows us to combine objects to create more complex systems



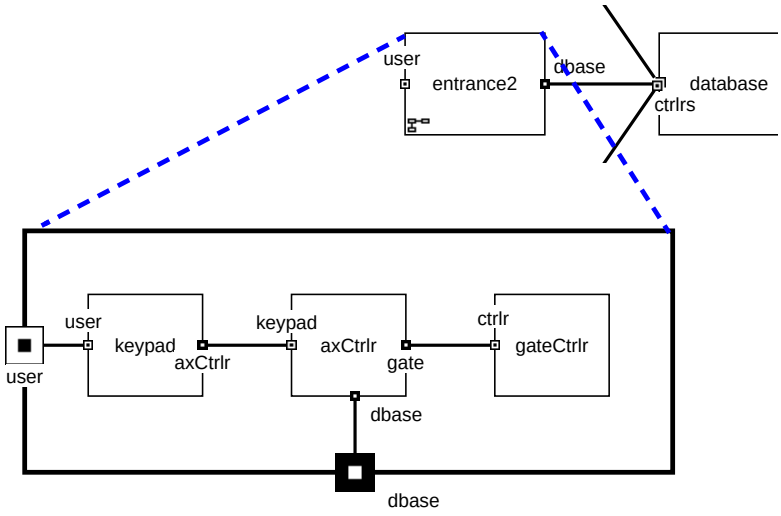
- The connections between ports are called *bindings*

Bindings

- Abstract representations of established communication channels for exchanging messages between actors
- Only ports with compatible protocols can be bound
  - Usually, a protocol class and its conjugate
- Bindings explicitly identify all valid causal linkages between actors
- Semantics of bindings are defined by the protocol classes of the ports

Looking Inside Actors

- If the functionality of an actor is complex, it may be useful to break it down into simpler components



- Actors can be defined ***recursively!***

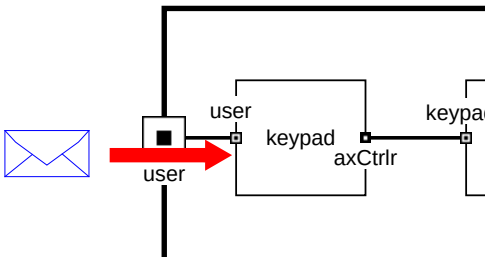
Reusing Specifications — Actor Classes

- As with protocols, it is often useful for multiple actors to share a common specification
- This is achieved through actor classes
- ROOM distinguishes between the class and the type of an actor

  actor class = actor type + actor implementation

Handling Messages and Port Types

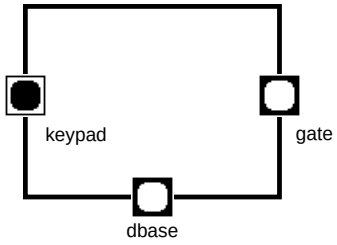
- Actors can either relay messages to internal components or handle them directly
- All messages arriving on *relay ports* are forwarded automatically



- Alternatively, messages arriving on an *end port* are processed by the actor

End Ports and Behavior

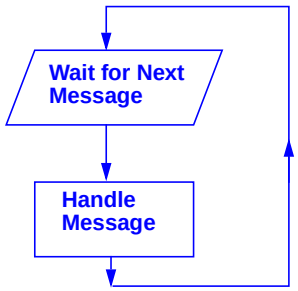
- End ports are the access points to the *behavior* (component) of an actor
- Identified by a distinct graphical notation



- The behavior component is implicit; i.e., it is not explicitly rendered in a diagram

The Behavior of Reactive Systems

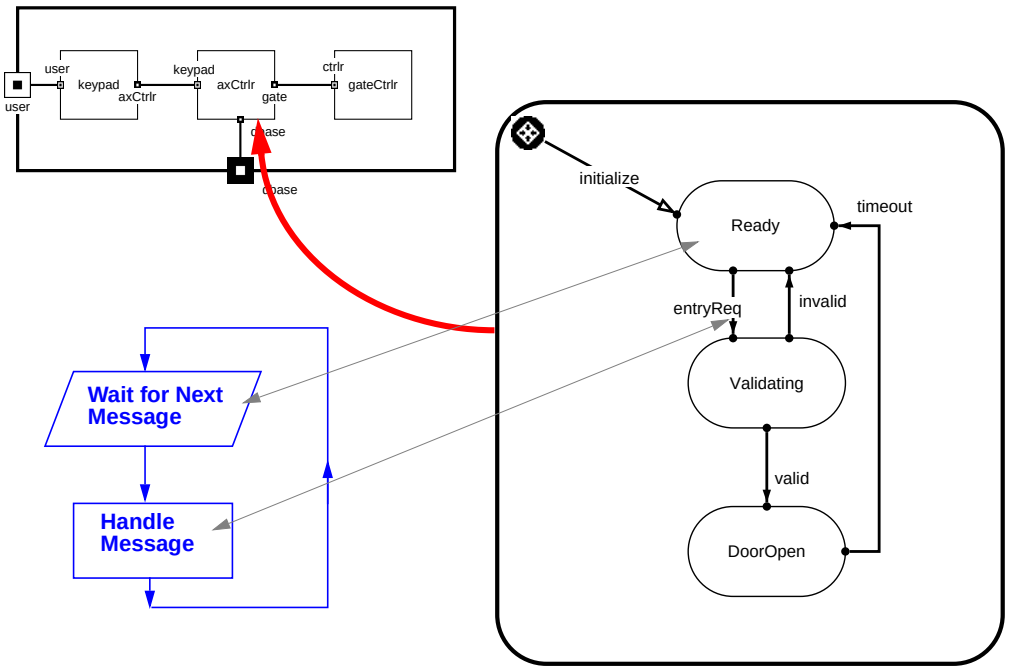
- Between successive events, the behavior is quiescent, “listening” for incoming messages
- When a message arrives, the behavior is activated and may respond by sending one or more messages of its own through its ports



Execution Paradigms

- *What happens if a new higher priority event arrives while the previous event is being processed?*
- Two options:
  - *preemptive* paradigm interrupts current event handling
  - *run-to-completion* paradigm finishes handling the current event handling before moving on to the new event
- ROOM uses the run-to-completion paradigm

ROOMcharts



Events, Triggers, and Actions

- **Each transition has**
  - **a *trigger* which specifies the message arrival event(s) that causes the transition**

entryReq Events

Ports/SAPs	Signals	Guard
keypad	entryReq	true

  - **optionally, an *action* that specifies how the event is handled**

C++ --> AccessController:entryReq | Code View

```
dbase.send (validationReq, *(msg->data));
```

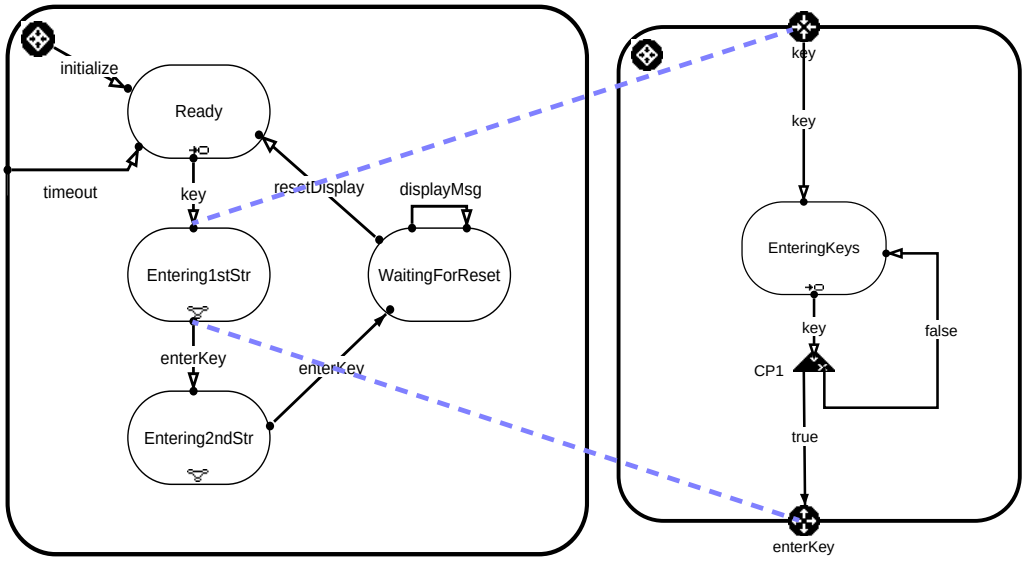
- **The action specification belongs to the Detail Level and uses the programming level of choice (e.g., C++)**

Extended State Variables

- In addition to temporary variables that may be associated with an action, ROOMcharts also allow the definition of *extended state variables*
- Variables are passive objects used to store Detail Level information that needs to be retained between transitions
  - for example, instances of C++ object classes

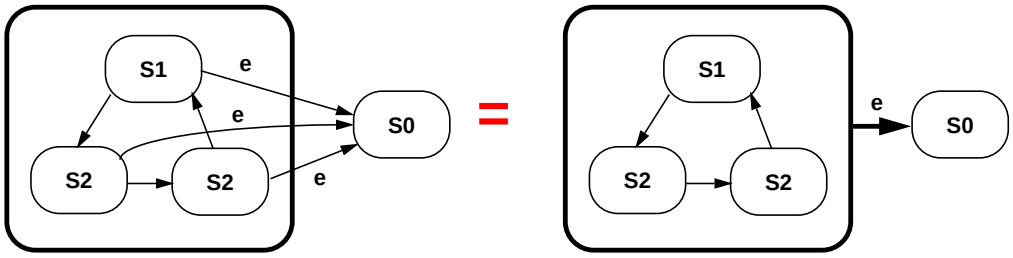
Hierarchical States

- Hierarchy is a means of dealing with complexity by focussing on only a manageable chunk at a time



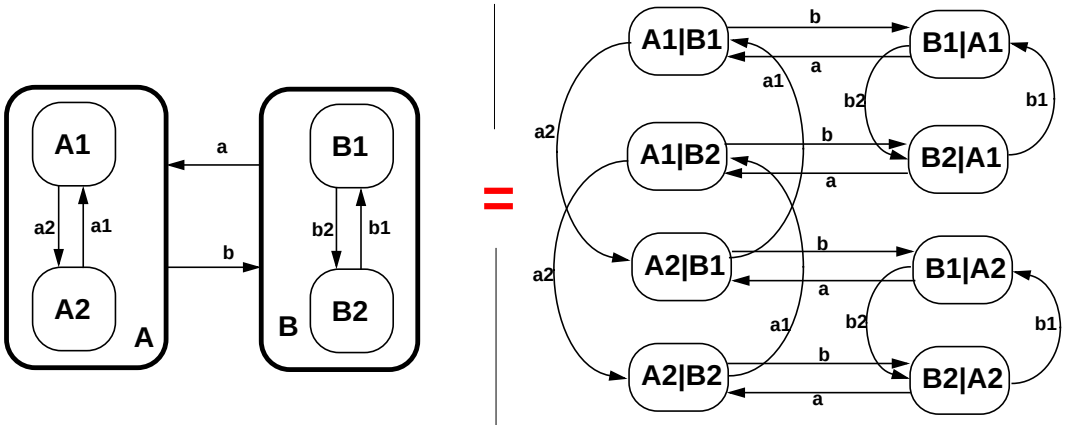
Behavior — Group Transitions

- A “shorthand” form for equivalent transitions that emanate from a common state



- Group transitions not only reduce visual clutter but often also identify “higher-order” events and provide hints about potential state aggregations

The Effectiveness of ROOMcharts



ROOMCHART:

(6 states, 6 transitions)

“FLAT” STATE MACHINE EQUIVALENT:

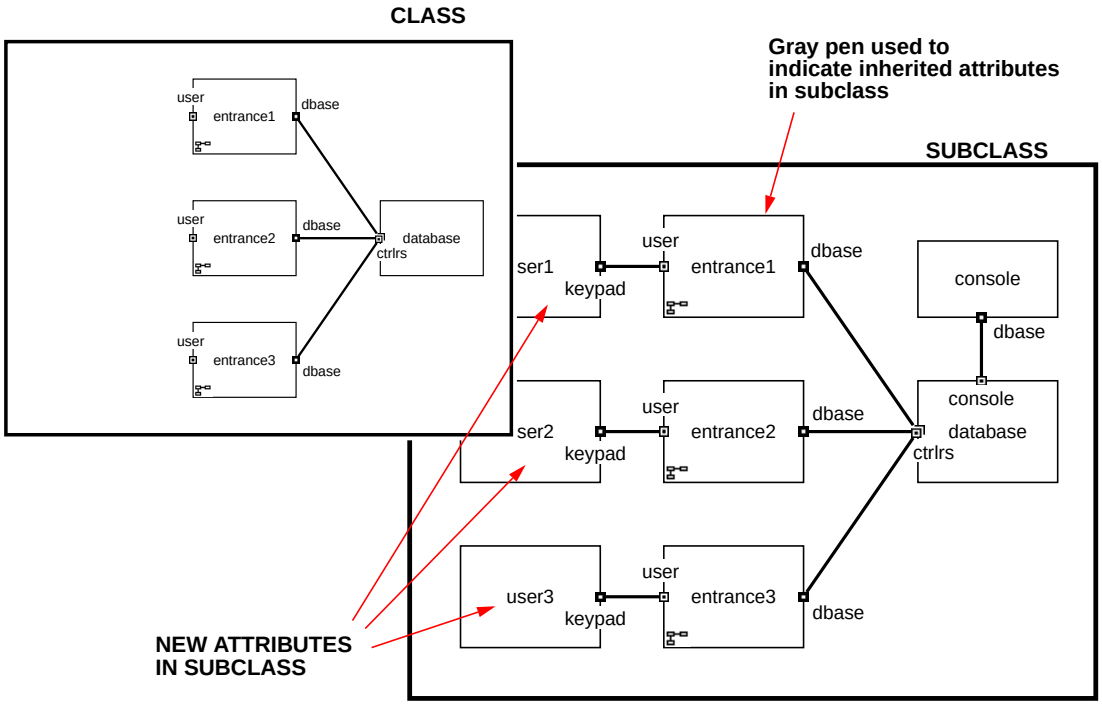
(8 states, 16 transitions)

For a 3-state equivalent: 9 states and 12 transitions (vs. 24 and 72)!

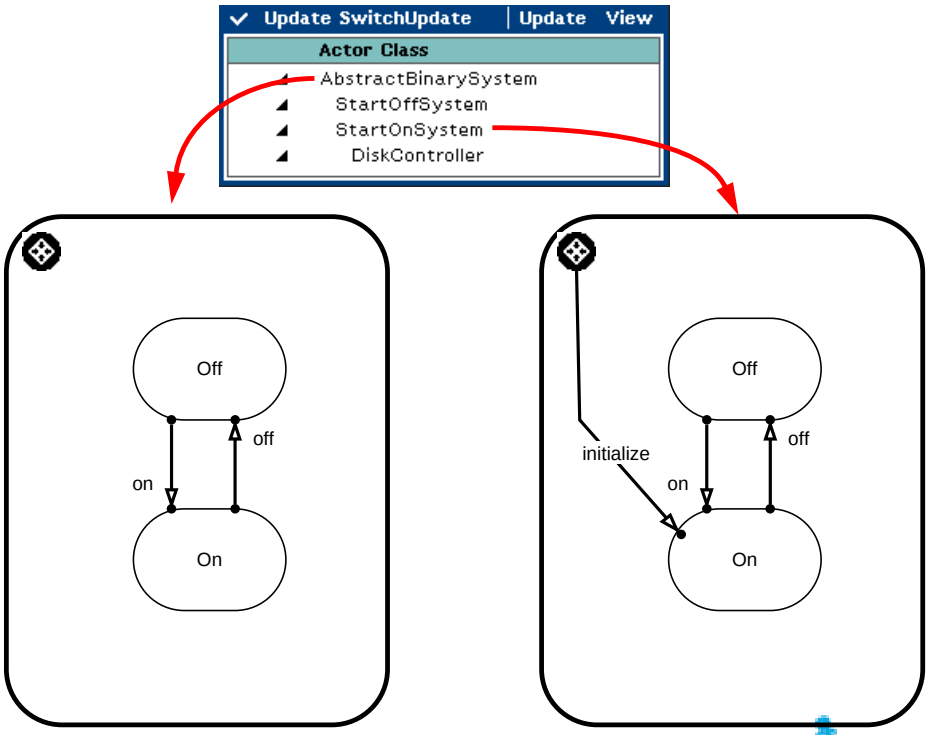
Inheritance Hierarchies in ROOM

- Each of the following class types has a distinct class hierarchy
  - Actor classes
  - Protocol classes
  - Data classes
- The rules of inheritance are not uniform in the three different hierarchies

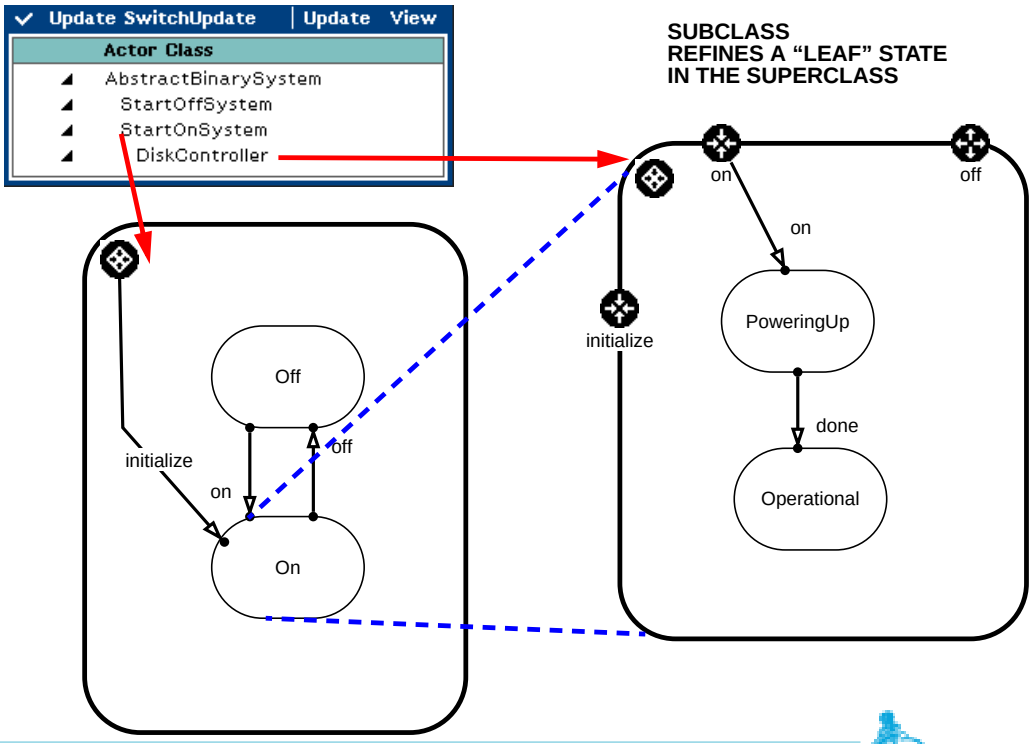
Actor Class Inheritance — Structure



Actor Class Inheritance — Behavior

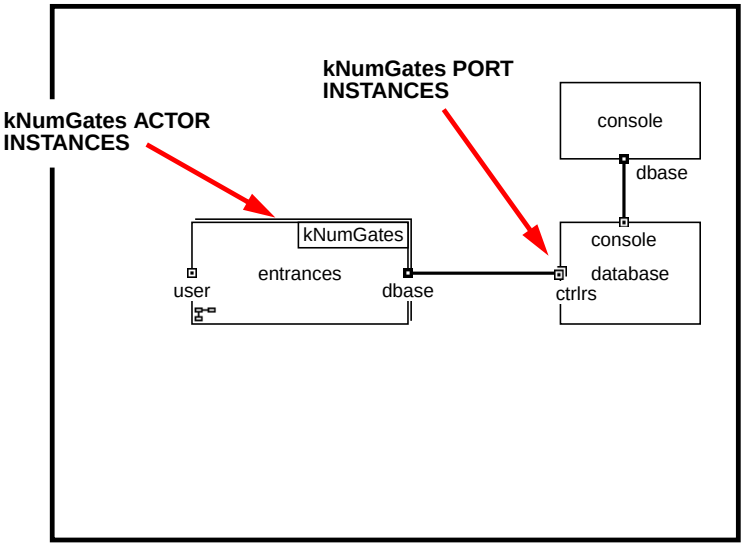


Refinement of Behavior in Subclasses



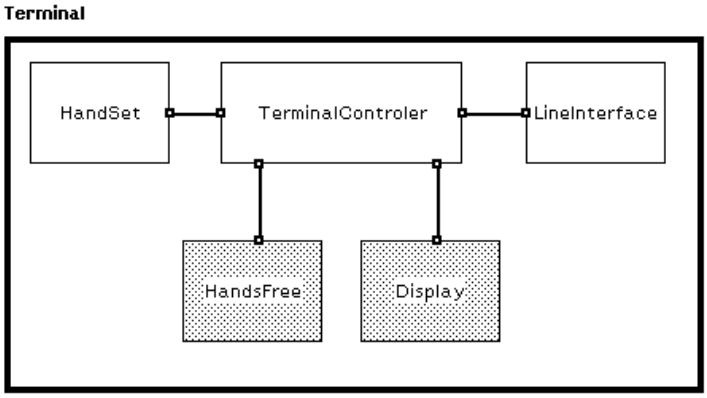
Replicated Structures

- **Useful for dealing with regular repeating structures**



Dynamic Structure — Optional Actors

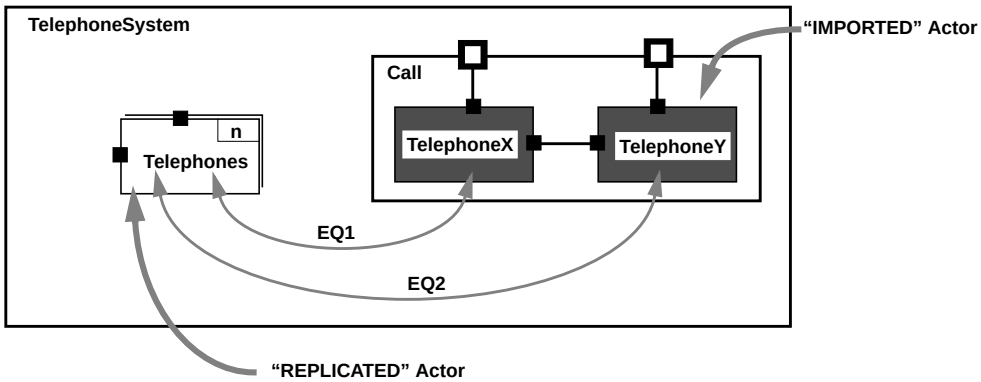
- **Components may be created *after* their container as needed**



- **Creation/destruction controlled by the container**

Multiple Containment and Importing

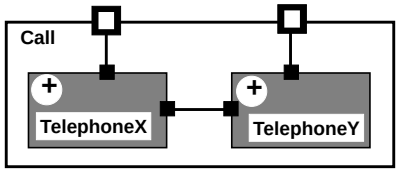
- Captures components that are simultaneously part of two or more container actors



- Component actors are "slots" to be filled as needed

Generic Structures

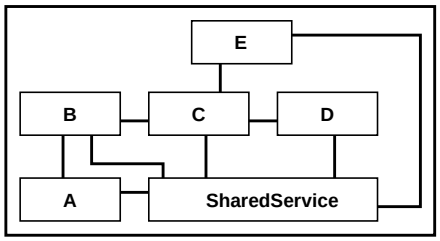
- Ability to define a component as "generic"

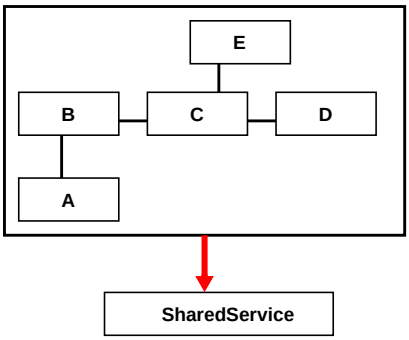


- Any actor with a compatible type can be imported into the generic "slot"
- A single model represents a collection of possible variants

Layering

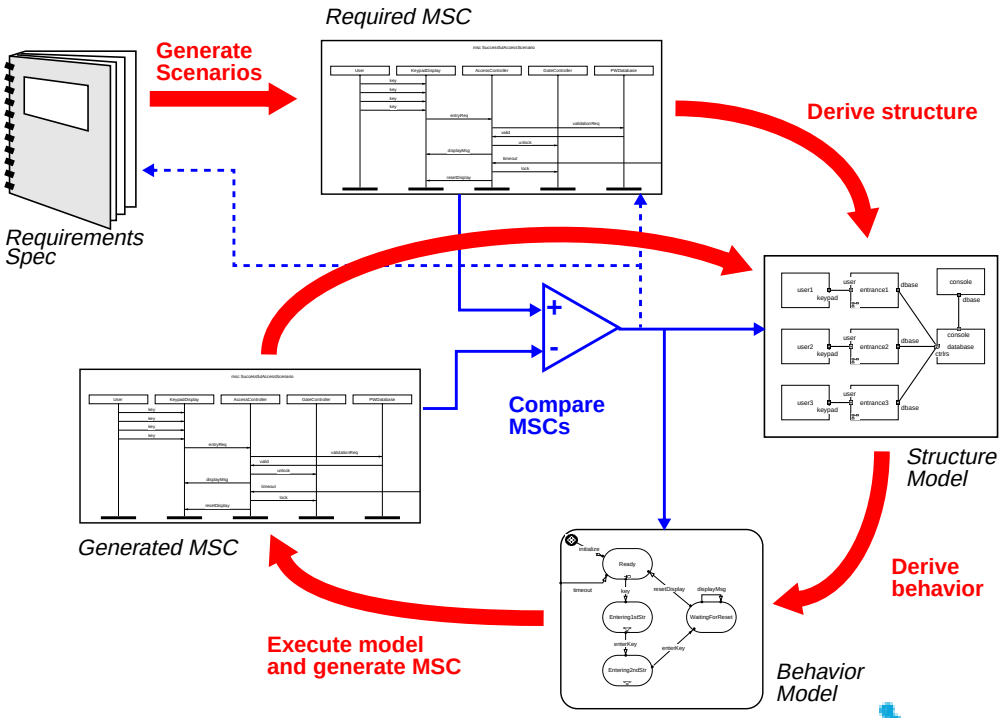
- Layer relationships:
  - Directed hierarchical relationship
  - For modeling widely-shared implementation services





- Reduces apparent complexity
- Communication model also based on protocols and port-like interfaces (Service Access Points (SAPs))

Typical Current Development Micro-Cycle



- Real-Time Systems and Architecture
- The ROOM Modeling Concepts
- Development Process
- **Summary**

Why Use ROOM?

- The essential complexity of reactive systems is sufficiently high to require specialized modeling concepts
- ROOM takes advantage of many recent developments in computer technology (formal methods, the object paradigm, graphical user interfaces)
- Also, ROOM was explicitly designed to take advantage of computer-based automation of many of the more mechanical activities of development
- This gives it a unique potential for significant gains in productivity and quality

Modeling Structure

- Use of graphical syntax for easier comprehension
- Abstractions for dealing with high-level (architectural) phenomena of large real-time systems
  - Protocol-based multiple interfaces
  - Concurrent objects (actors)
  - Dynamic structures
  - Replicated structures

Modeling Behavior

- High-level based on graphical syntax
- Uses hierarchical state machines
  - Chosen to allow powerful modeling capabilities while allowing for straightforward and efficient (automated) implementations
- Priority-based event handling for dealing with real-time requirements
- Incorporates industry standard programming languages (e.g., C++) for detailed specs

Experience

- Over a hundred different industrial projects have used ROOM
  - including complete automatic code generation
- Various application domains:
  - telecom
  - defense/aerospace
  - manufacturing
- ROOM and ObjecTime are being used as a basis for teaching real-time systems in a number of undergraduate and graduate courses