Architectural Standards

Infosys, Mysore
December 16

Objectives

- Concepts
  - What are standards?
  - Why use standards?
    - And why not? (drawbacks)
  - Deciding when to adopt a standard
- Prevalent Architectural Standards
  - Conceptual standards
  - Notational standards
  - Standard tools
  - Process standards

Objectives

- Concepts
  - What are standards?
  - Why use standards?
    - And why not? (drawbacks)
  - Deciding when to adopt a standard
- Prevalent Architectural Standards
  - Conceptual standards
  - Notational standards
  - Standard tools
  - Process standards

What are standards?

- **Definition:** a *standard* is a form of agreement between parties
- Many kinds of standards
  - For notations, tools, processes, organizations, domains
- There is a prevalent view that complying to standard 'X' ensures that a constructed system has high quality
  - This is almost never strictly true
  - But that doesn't mean standards are worthless!
  - Here, we will attempt to put standards in perspective

De jure and *de facto* standards

- Some standards are controlled by a body considered authoritative
  - ANSI, ISO, ECMA, W3C, IETF
- These standards are called *de jure* ("from law")
- *De jure* standards usually
  - are formally defined and documented
  - are evolved through a rigorous, well-known process
  - are managed by an independent body, governmental agency, or multi-organizational coalition rather than a single individual or company

De jure and *de facto* standards (cont'd)

- Some standards emerge through widespread awareness and use
- These standards are called *de facto* ("in practice")
- *De facto* standards usually
  - are created by a single individual organization to address a particular need
  - are adopted due to technical superiority or market dominance of the creating organization
  - evolve through an emergent, market-driven process
  - are managed by the creating organization or the users themselves, rather than through a formal custodial body

Examples of *de jure* and *de facto*

- *De jure* standards
  - UML (managed by OMG)
  - CORBA (also managed by OMG)
  - HTTP protocol (managed by IETF)
- *De facto* standards
  - PDF format (managed by Adobe)
    - May become *de jure* through ISO
  - Windows (managed by Microsoft)
- There is a substantial gray area between these two

Gray-area Standards

- HTML
  - Officially standardized by W3C, indicating *de jure*
  - Flavors and browser-specific extensions developed by Microsoft, Netscape, and others, creating *de facto* variants
  - None of these has power to force users to use standard
- JavaScript
  - Developed by Netscape; copied (as JScript) by Microsoft
  - After substantial adoption and possibly under threat of forking/splintering, Netscape submits it to ECMA
  - Now standardized as ECMAScript (*de jure*)
  - JavaScript and variants continue to be developed as compatible extensions of ECMAScript

Another spectrum

- Standards (whether *de jure* or *de facto*) can be:
  - Open
    - Allow public participation in the standardization process
    - Anyone can submit ideas or changes for review
  - Closed (a.k.a. proprietary)
    - Only the custodians of a standard can participate in its evolution

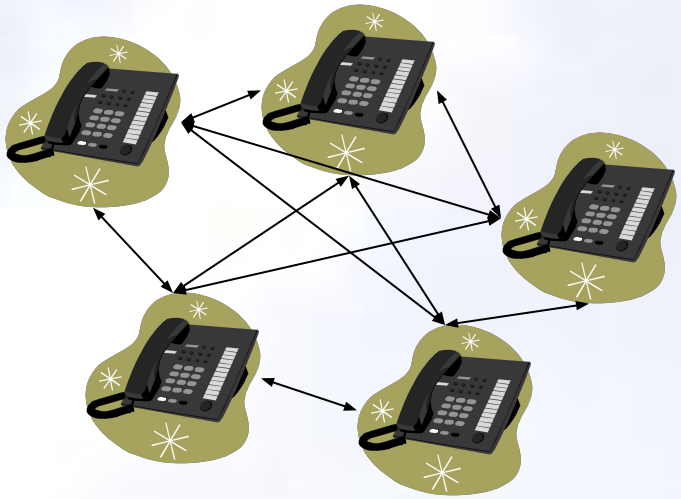
Open vs. closed standards

- Another spectrum with a gray area
  - Some standards bodies have high barriers to entry (e.g., steep membership fees, vote of existing membership)
  - Some standards (e.g., Java) have aspects of both
    - Sun Microsystems is effectively in control of Java as a *de facto* standard
    - There is an open "community process" by which external parties can participate in a limited way

Why use standards?

- Standards are an excellent way to create and exploit *network effects*
- A network effect exists if the value of participation increases as the number of users of the standard increases

versus



- Other network effects:
  - TCP/IP, HTTP & HTML, UML…

Why use standards? (cont'd)

- To ensure interoperability between products developed by different organizations
  - Usually in the interest of fostering a network effect
- To carry hard-won engineering knowledge from one project to another
  - To take advantage of hard-won engineering knowledge created by others
- As an effort to attract tool vendors
  - To create economies of scale in tools
- To attempt to control the standard's evolution in your favor

Drawbacks of standards

- Limits your agility
  - Remember that doing 'good' architecture-based development means identifying what is important in *your* project
- Standards often attempt to apply the same techniques to a too-broad variety of situations
- The most widely adopted standards are often the most general

Overspecification vs. underspecification

- A perennial tension in standards use and development
- Overspecification
  - A standard prescribes too much and therefore limits its applicability too much
- Underspecification
  - A standard prescribes too little and therefore doesn't provide enough guidance
    - Possibly in an effort to broaden adoption

Two different kinds of underspecification

- Two compromises often made in negotiation when disagreements occur
  - Leave the disagreeable part of the standard unspecified or purposefully ambiguous
  - Include both opinions in the standard but make them both optional
- Both of these weaken the standard's value
  - Consider the different kinds of reduction in interoperability imposed by these strategies
- Although they may improve adoption!

When to adopt a standard?

- Early adoption
  - Benefits
    - Improved ability to influence the standard
      - Get your own goals incorporated; exclude competitors
    - Early to market
      - If standard becomes successful, early marketers will profit
    - Early experience
      - Leverage enhanced experience to your benefit

When to adopt a standard? (cont'd)

- Early adoption
  - Drawbacks
    - Risk of failure
      - Standard may not be successful for reasons out of your control
    - Moving target
      - Early standards tend to evolve and 'churn' more than mature ones, and may be 'buggy'
    - Lack of support
      - Early standards tend to have immature (or no) support from tool and solution vendors

When to adopt a standard? (cont'd)

- Late adoption
  - Benefits
    - Maturity of standard
    - Better support
  - Drawbacks
    - Inability to influence the standard
    - Restriction of innovation

Objectives

- Concepts
  - What are standards?
  - Why use standards?
    - And why not? (drawbacks)
  - Deciding when to adopt a standard
- Prevalent Architectural Standards
  - Conceptual standards
  - Notational standards
  - Standard tools
  - Process standards

IEEE 1471

- Recommended practice for architecture description
  - Often mandated for use in government projects
- Scope is limited to architecture descriptions (as opposed to processes, etc.)
- Does not prescribe a particular notation for models
  - Does prescribe a minimal amount of content that should be contained in models
- Identifies the importance of stakeholders and advocates models that are tailored to stakeholder needs
- A notion of views and viewpoints similar to the ones used in this course

IEEE 1471 (cont'd)

- Very high level
  - Purposefully light on specification
  - Does not advocate any specific notation or process
- Useful as a starting point for thinking about architecture
  - Defines key terms
  - Advocates focus on stakeholders
- Being compliant does NOT ensure that you are doing good architecture-centric development

Department of Defense Architecture Framework

- DoDAF, evolved from C4ISR
  - Has some other international analogs (MoDAF)
  - 'Framework' here refers to a process or set of viewpoints that should be used in capturing an architecture
    - Not necessarily an architecture implementation framework
- Identifies specific viewpoints that should be captured
  - Includes what kinds of information should be captured
  - Does not prescribe a particular notation for doing the capture

DoDAF (cont'd)

Concept	Our Term	DoDAF Term
A set of perspectives from which descriptions are developed	Viewpoint set	View
A perspective from which descriptions are developed	Viewpoint	(Kind of) product
An artifact describing a system from a particular perspective	View	Product

- Some vocabulary inconsistency with our terms (and IEEE 1471 among others)

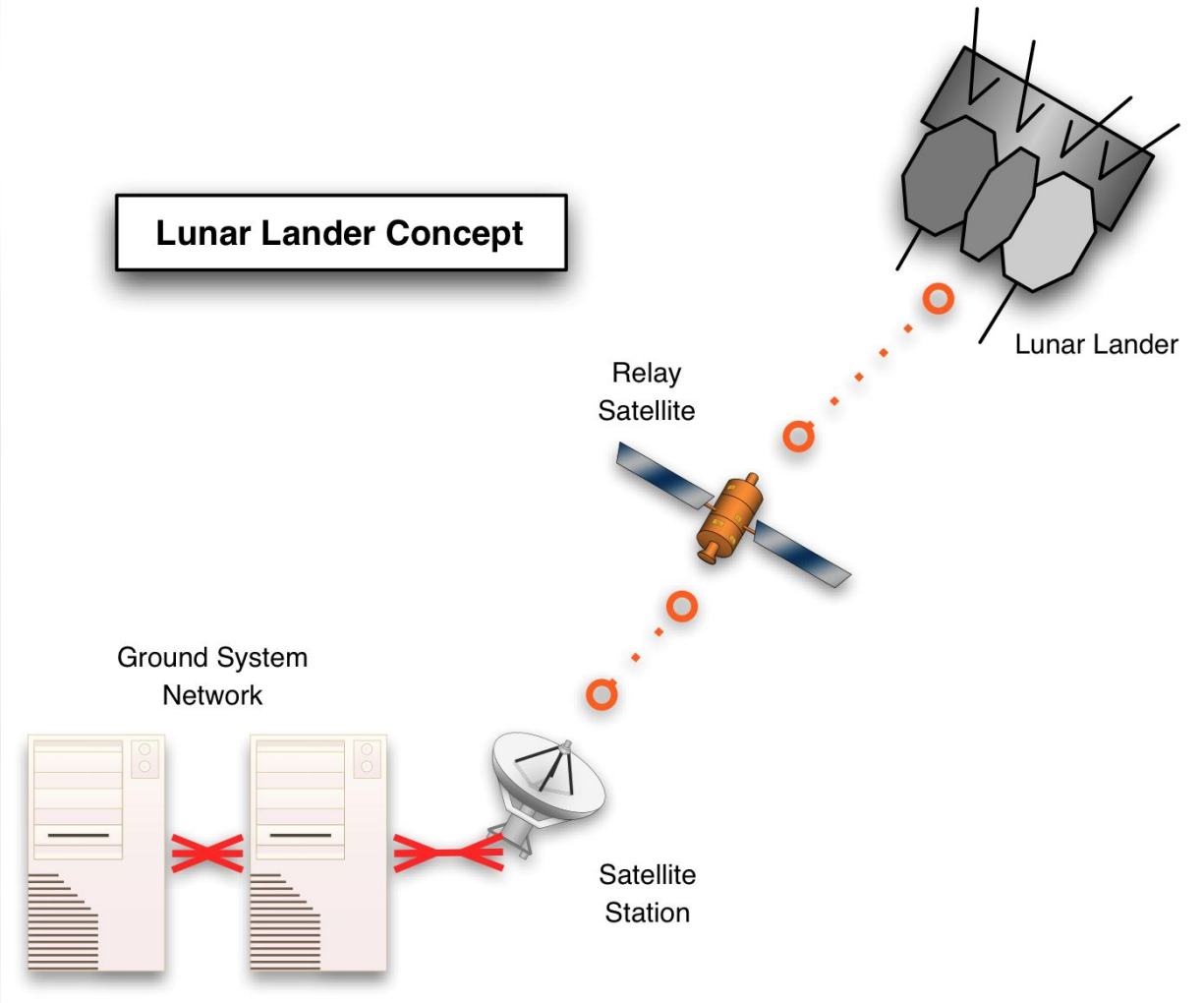
DoDAF (cont'd)

- Three views (in our terms: viewpoint sets)
  - Operational View (OV)
    - "Identifies what needs to be accomplished and who does it"
    - Defines processes and activities, the operational elements that participate in those activities, and the information exchanges between the elements
  - Systems View (SV)
    - Describe the systems that provide or support operational functions and the interconnections between them
    - Systems in SV associated with elements in OV

DoDAF (cont'd)

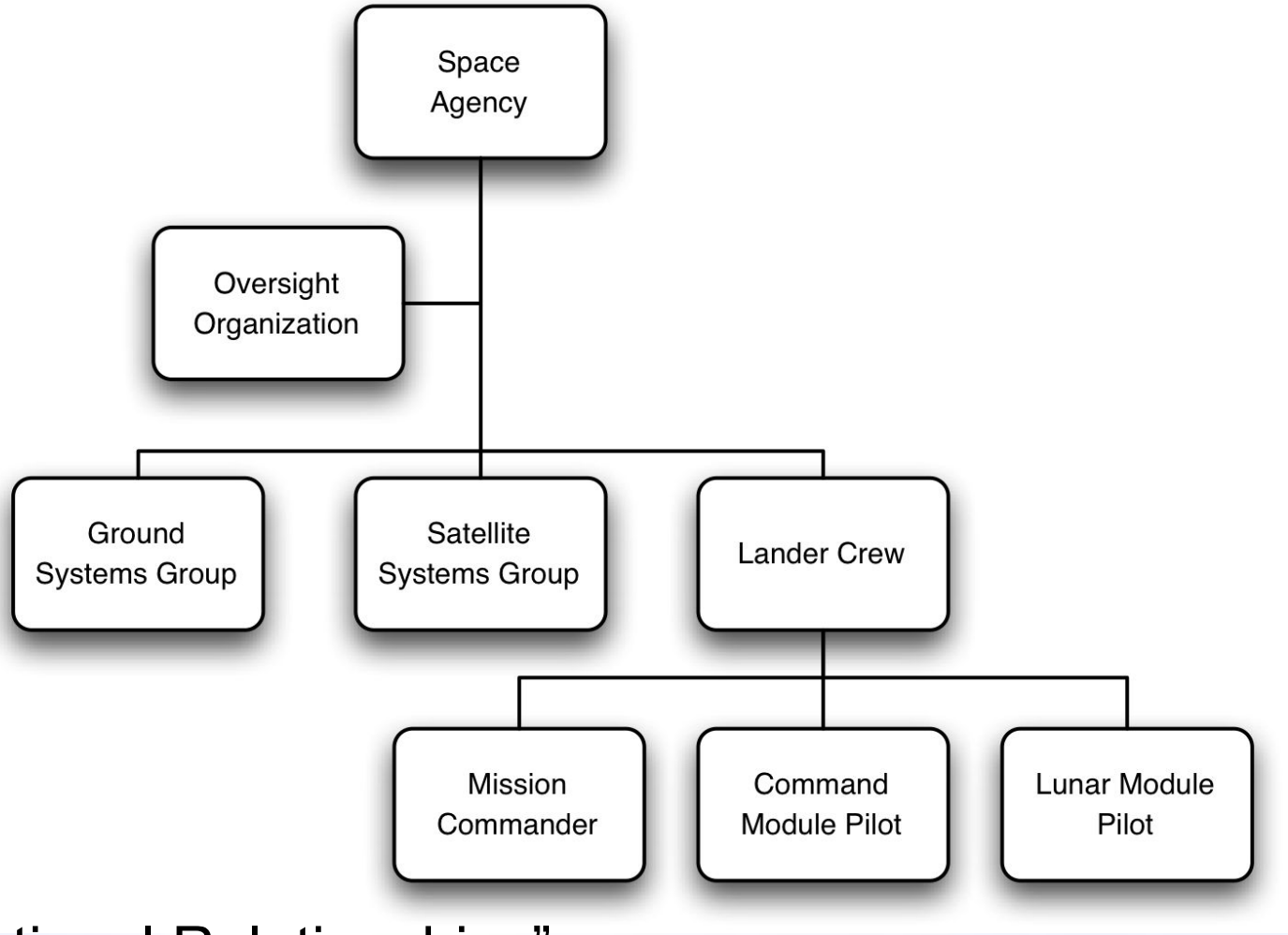
- Three views (in our terms: viewpoint sets) – continued
  - Technical Standards View (TV)
    - Identify standards, (engineering) guidelines, rules, conventions, and other documents
    - To ensure that implemented systems meet their requirements and are consistent with respect to the fact that they are implemented according to a common set of rules
- Also a few products address cross cutting concerns that affect All Views (AV)
  - E.g., dictionary of terms

DoDAF Examples



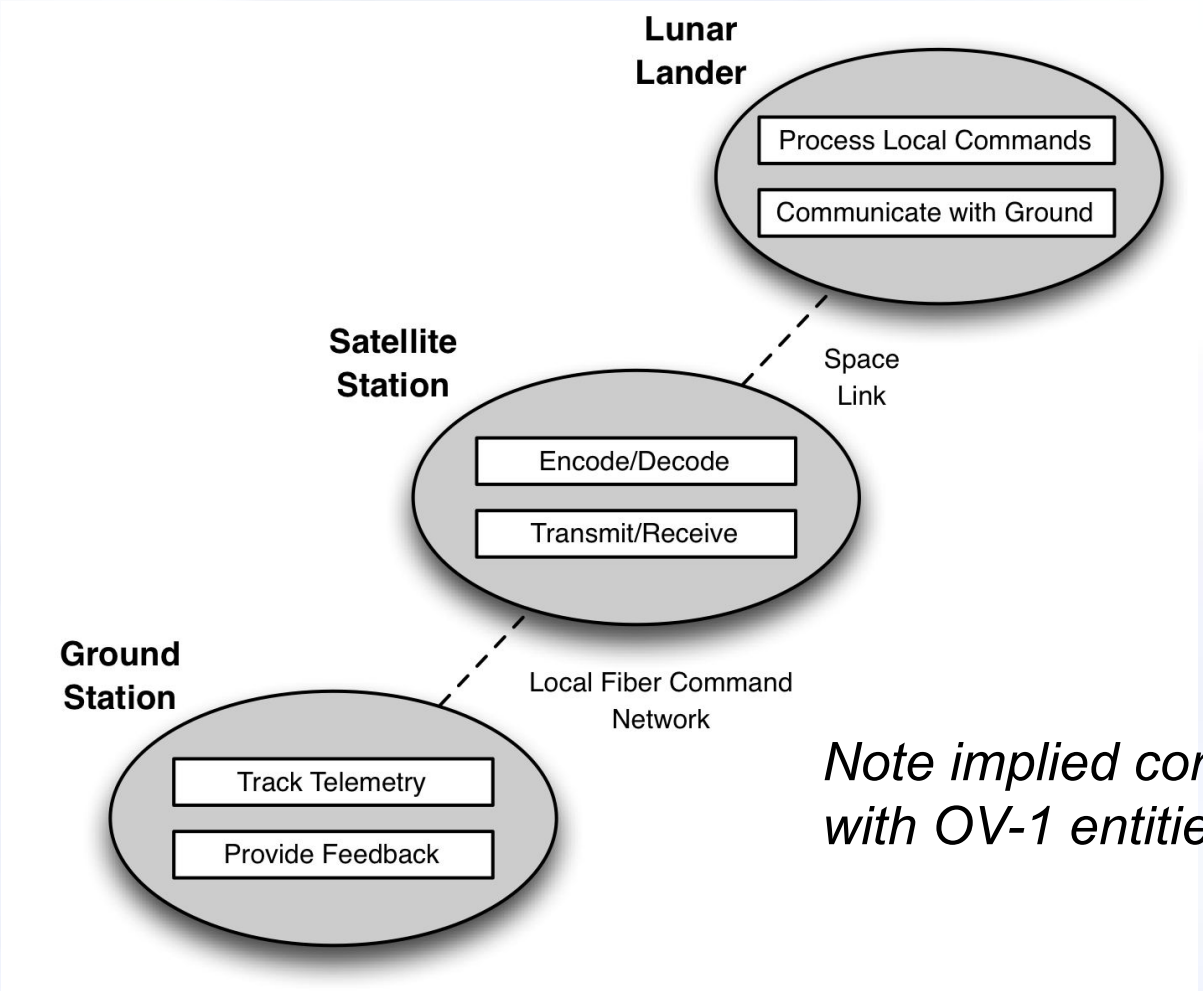
OV-1
"High-Level Operational Concept Graphic"

DoDAF Examples (cont'd)



OV-4
"Organizational Relationships"

DoDAF Examples (cont'd)



Note implied correspondence with OV-1 entities

SV-1
"Systems Interface Description"

DoDAF Examples (cont'd)

from \ to	Ground Station	Satellite Station	Lunar Lander
Ground Station		Ground Feedback (TCP/IP)	
Satellite Station	Lander Transmissions (TCP/IP)		Ground Feedback (Space Protocol)
Lunar Lander		Lander Transmissions (Space Protocol)	

SV-3
"Systems-Systems Matrix"

One of several "N^2" views in DoDAF

DoDAF Examples (cont'd)

TV-1 "Technical Standards Profile "

Standards for SV-1 Systems			Ground Station	Satellite Station	Lunar Lander
Service	*Service Area*	*Standard*			
Information Technology Standards	Operating System Standard	ISO/IEC 9945-1:1996, Information Technology - Portable Operating System Interface (POSIX) - Part 1: System Application Program Interface (API)	Baseline: 1 Jan 2020	Baseline	Baseline + 3 mos
Information Transfer Standards	Data Flow Network	Extensible Markup Language (XML) 1.0 (Fourth Edition) W3C Recommendation 16 August 2006	Baseline + 6 mos	Baseline + 6 mos	
	Physical Layer	FDDI / ANSI X3.148-1988, Physical Layer Protocol (PHY) -- also ISO 9314-1	Baseline + 3 mos	Baseline + 3 mos	

DoDAF Takeaways

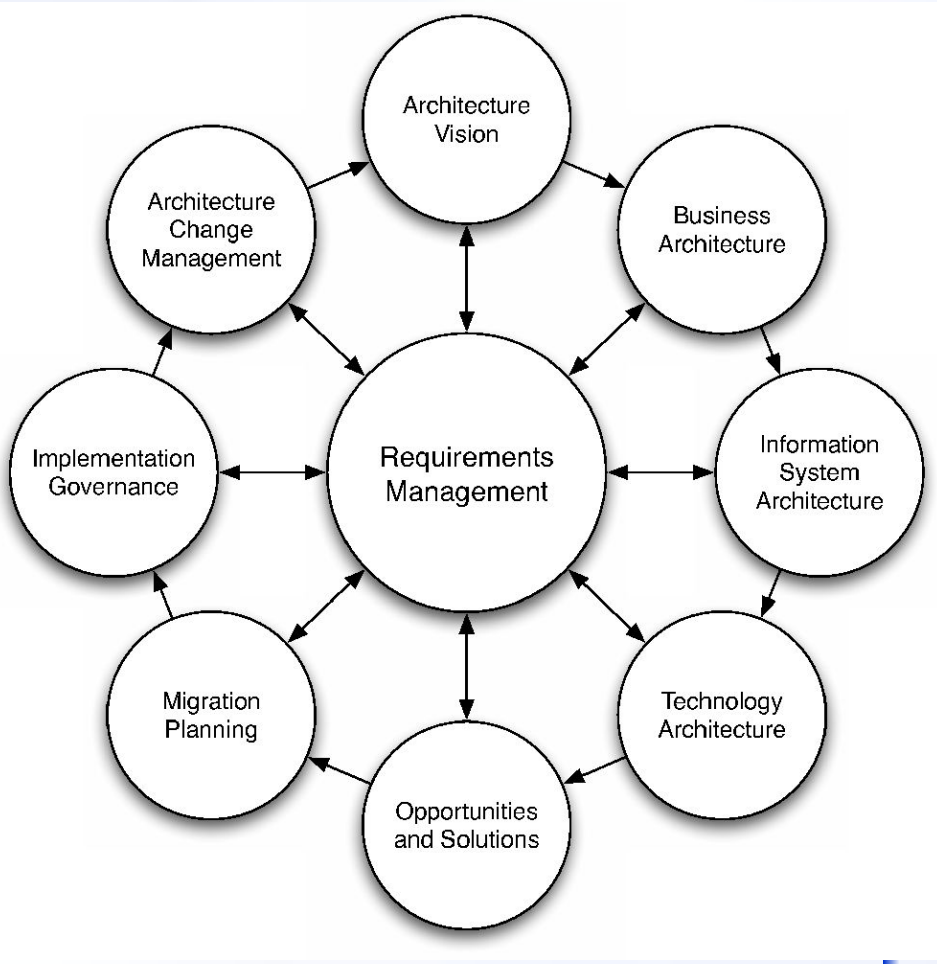
- Extremely comprehensive standard advocating capture of many views
  - Takes a high-level organizational perspective
  - OV views tend to deal with human and systems organizations
  - SV views tend to deal with technical aspects of systems (most like the architectural descriptions we have been talking about)
  - TV views tend to deal with practical issues of reuse and leveraging existing technology
- Tells us a lot about WHAT to model, but nearly nothing about HOW to model it

The Open Group Architecture Framework

- TOGAF – an "enterprise architecture" framework
  - Focuses beyond hardware/software
  - How can enterprises build systems to achieve business goals?
- Four key areas addressed
  - **Business concerns**, which address business strategies, organizations, and processes;
  - **Application concerns**, which address applications to be deployed, their interactions, and their relationships to business processes;
  - **Data concerns**, which address the structure of physical and logical data assets of an organization and the resources that manage these assets; and
  - **Technology concerns**, which address issues of infrastructure and middleware.

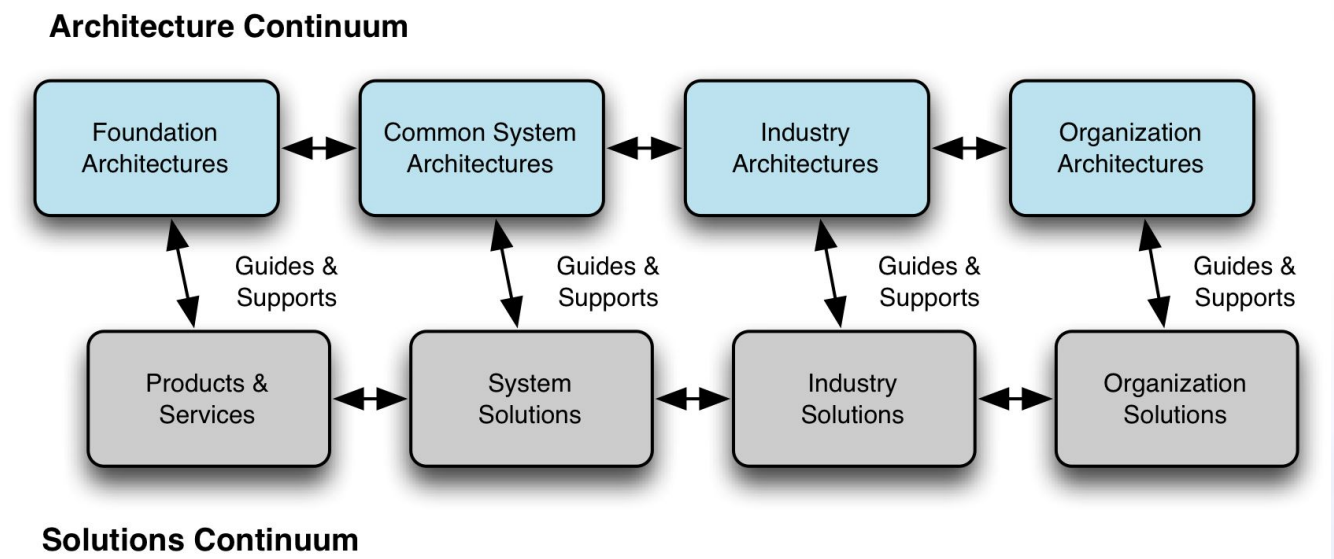
TOGAF Part 1: ADM

- An iterative process for architecture-centric development
- Each step in the processes associated with views to be captured
- Early phases focus on conceptual issues; later phases move toward reduction to practice



Redrawn from the TOGAF Specification

TOGAF Part 2: Enterprise Continuum



- Taxonomizes different kinds of architectures and the solutions that are supported by those
  - Left side is more technical and concrete
  - Right side is more organizational
- TOGAF Technical Reference Model and Standards Information Base identifies and taxonomizes many solution elements

TOGAF Part 3: TOGAF Resource Base

- A collection of useful information and resources that can be employed in following the ADM process
- Includes
  - advice on how to set up boards and contracts for managing architecture
  - checklists for various phases of the ADM process
  - a catalog of different models that exist for evaluating architectures
  - how to identify and prioritize different skills needed to develop architectures
  - ....

TOGAF Takeaways

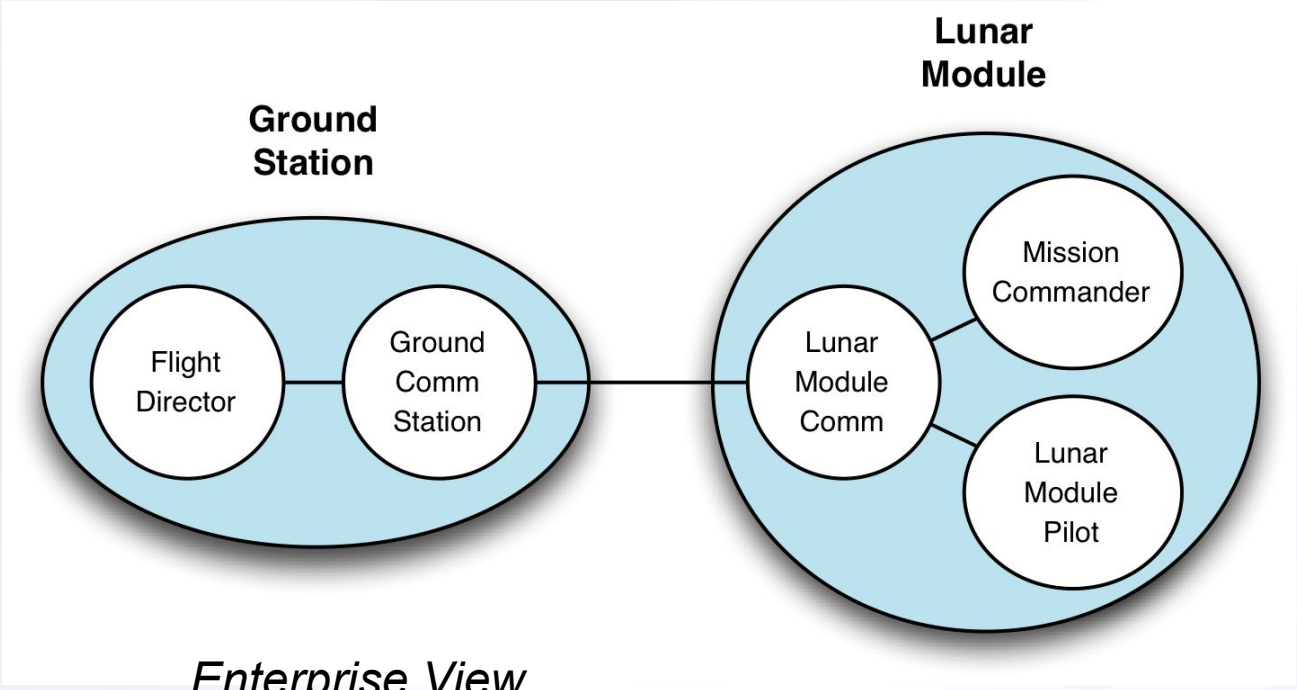
- Large size and broad scope looks at systems development from an enterprise perspective
- More suited to developing entire organizational information systems rather than indivdiual applications
- A collection and clearinghouse for IT “best practices” of all sorts

RM-ODP

- Another standard for viewpoints, similar to DoDAF but more limited in scope; resemble DoDAF SV
- Prescribes 5 viewpoints for distributed systems
  - Enterprise – system, environment, context
  - Information – information processing
  - Computational – architectural structure and component distribution
  - Engineering – distribution infrastructure
  - Technology – technology choices available

RM-ODP

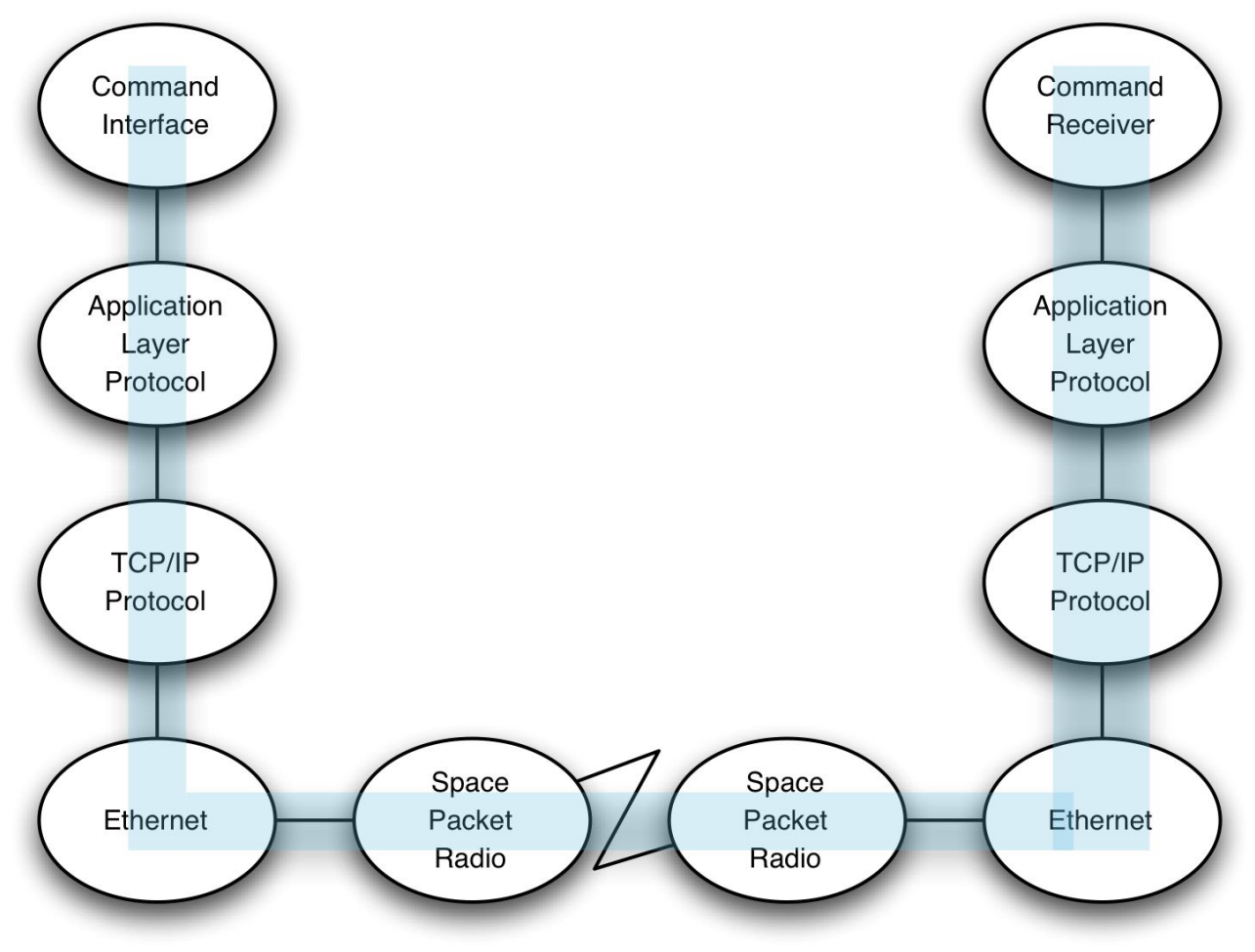
- Another standard for viewpoints, similar to DoDAF but more limited in scope; resemble DoDAF SV



Enterprise View

RM-ODP

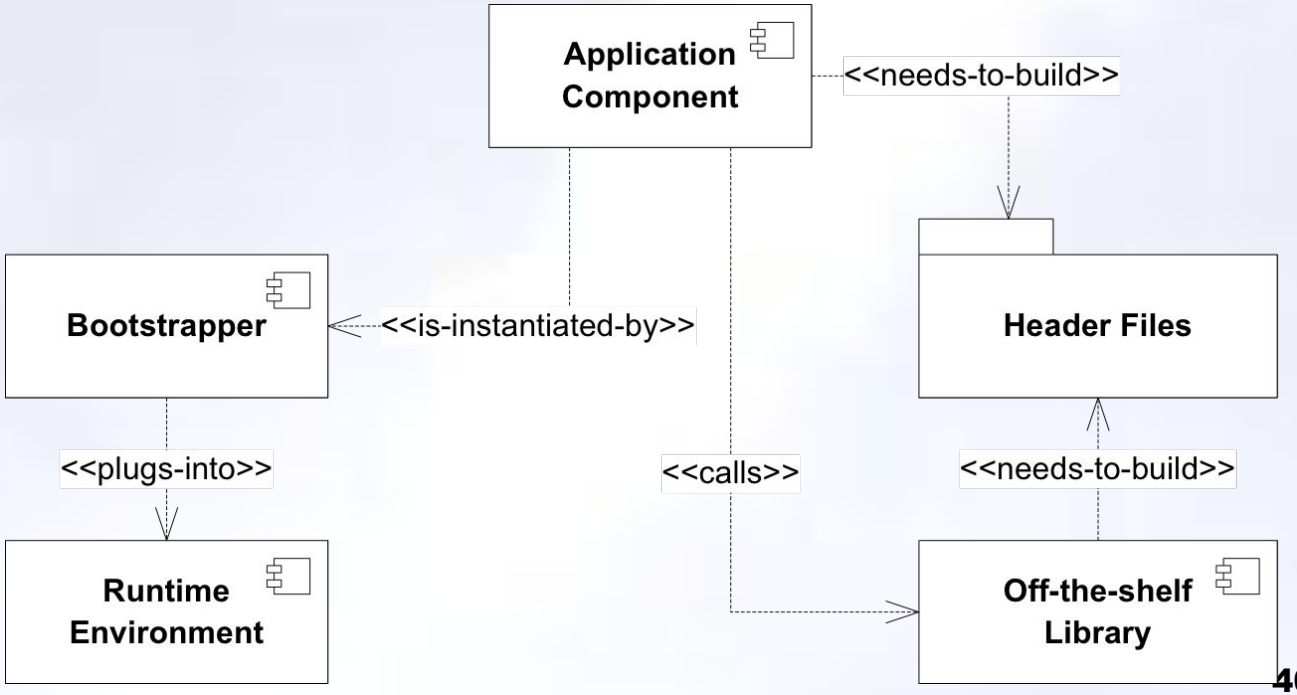
- Another stand
more limited



Engineering View

UML

- As a standard, primarily prescribes a syntax
- Some semantics with purposeful ambiguity
- Encourages specialization of the standard through the use of profiles, which are mini-standards



UML Takeaways

- Provide a common syntactic framework to express many common types of design decisions
- Profiles are needed to improve rigor
  - But profiles can only specialize existing UML diagram types, not create new ones
- Documenting a system in UML does not ensure overall system quality
  - You can document a bad architecture in UML as easily as a good one

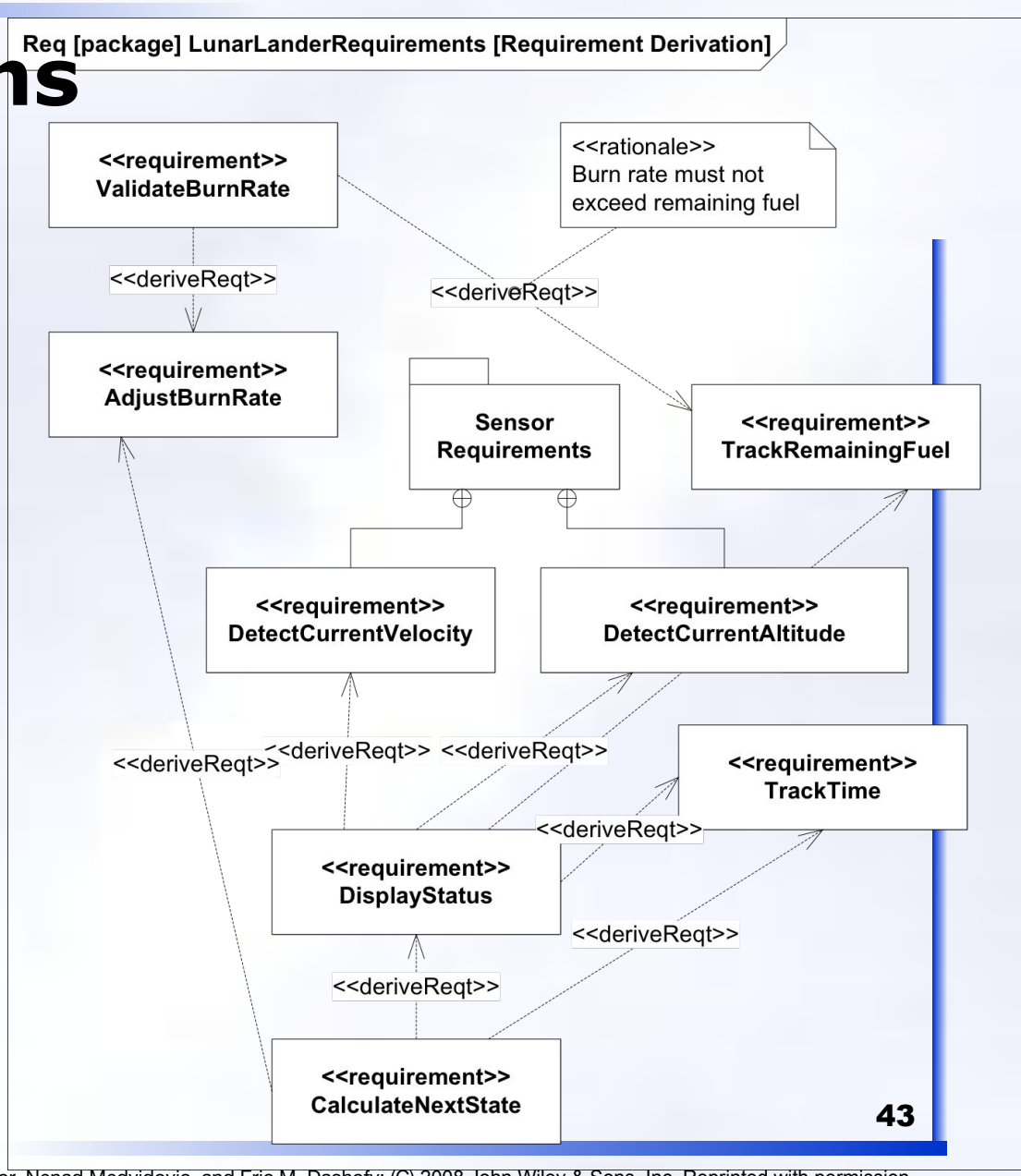
SysML

- An extended version of UML
- Developed by a large consortium of organizations (mainly large system integrators and developers)
- Intended to mitigate UML's "software bias"

- SysML group found UML standard insufficient and profiles not enough to resolve this
  - Developed new diagram types to capture system-engineering specific views
  - Limited momentum among tool vendors; focus shifting to more heavily use UML profiles

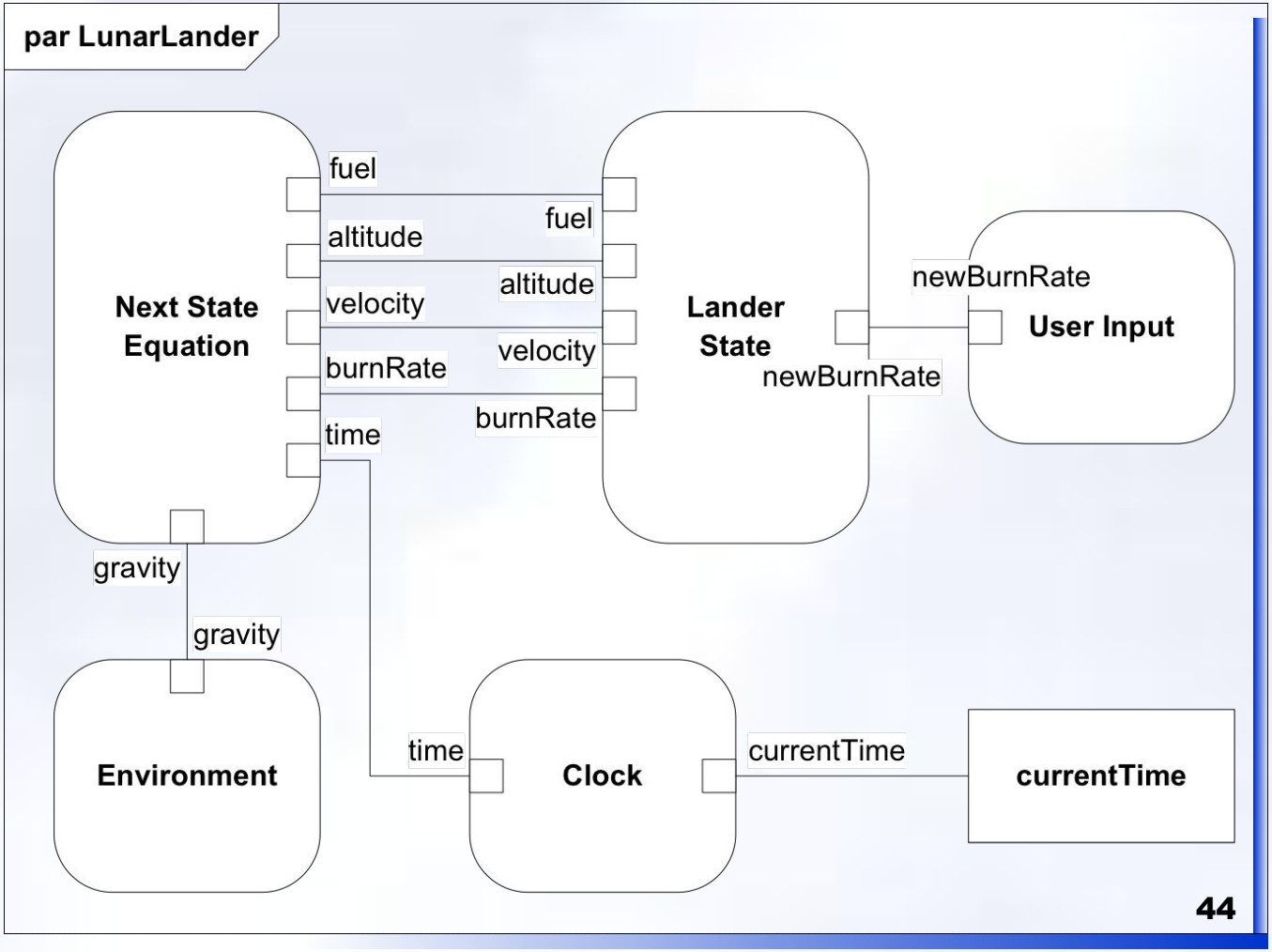
SysML Diagrams

SysML Requirement Diagram



SysML Diagrams

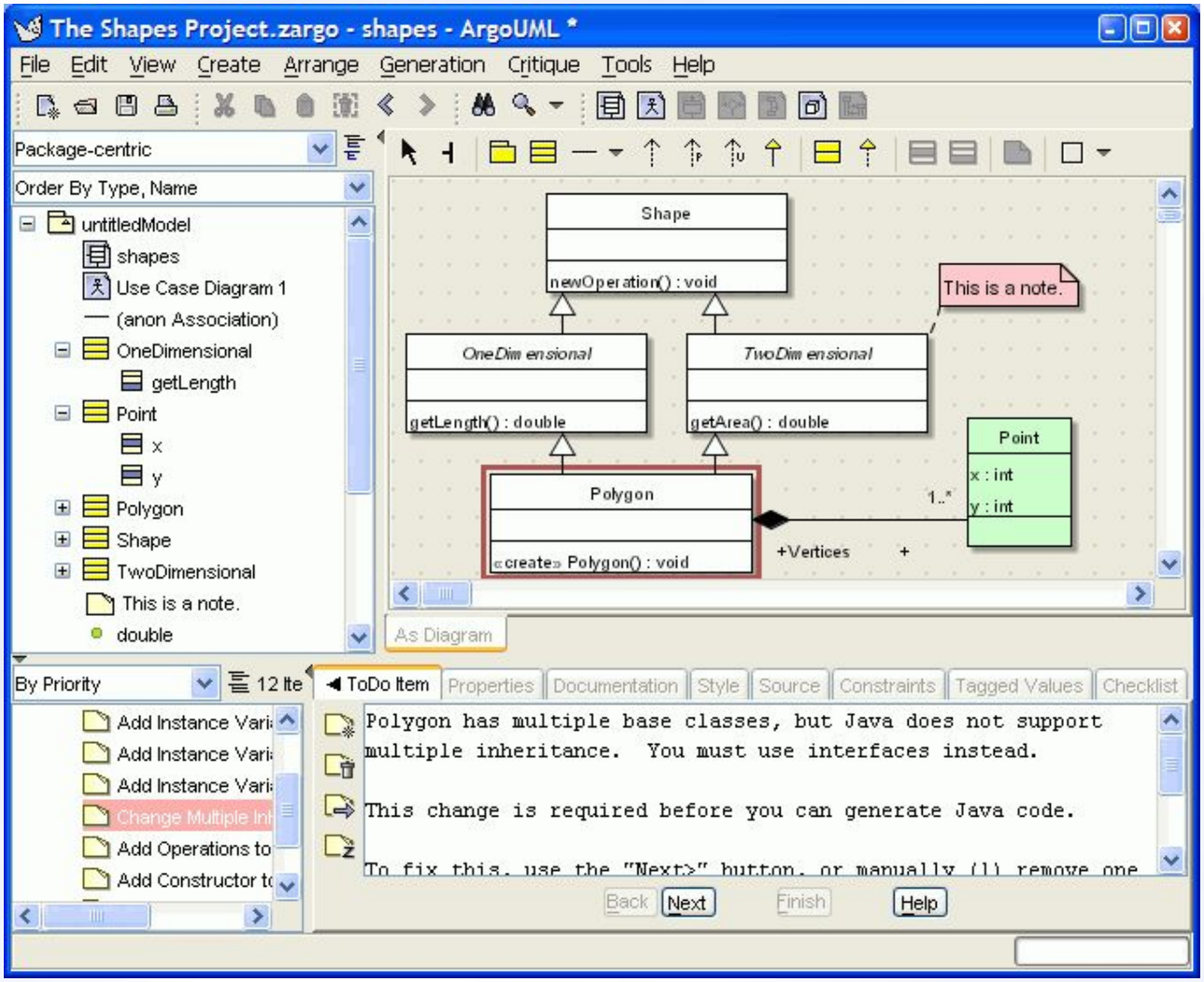
SysML Parametric Diagram



Standard UML Tools

- E.g., Rational Rose, ArgoUML, Microsoft Visio
- These are *de facto* standards
- All support drawing UML diagrams
- Vary along several dimensions
  - Support for built-in UML extension mechanisms
    - Profiles, stereotypes, tagged values, constraints
  - Support for UML consistency checking
  - Ability to generate other artifacts
  - Generation of UML from other artifacts
  - Traceability to other systems
  - Support for capturing non-UML information

ArgoUML – a UML tool



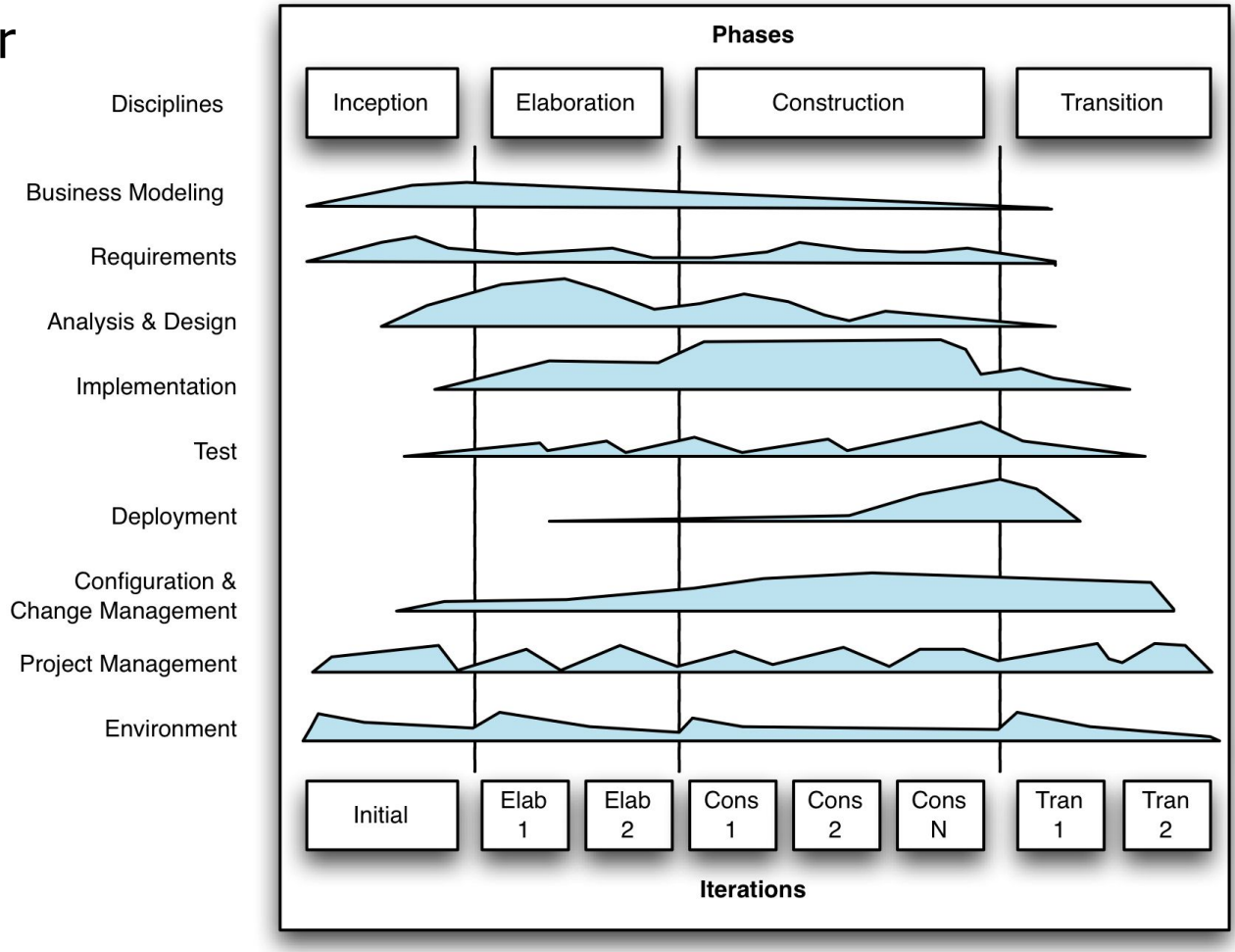
Screenshot from the Argo/UML Website and Documentation

Telelogic System Architect

- Formerly Popkin System Architect; popular among architects
  - Supports 50+ different diagram types
    - UML, IDEF, OMT, generic flowcharting, even GUI design
    - Variants for DoDAF, service-oriented architectures, enterprise resource planning
- Effectively generic diagram editor specialized for many different diagram types with different symbols, connections
  - Very little understanding of diagram semantics
  - Specialized variants have some understanding of semantics but generally less than notation-specific editors

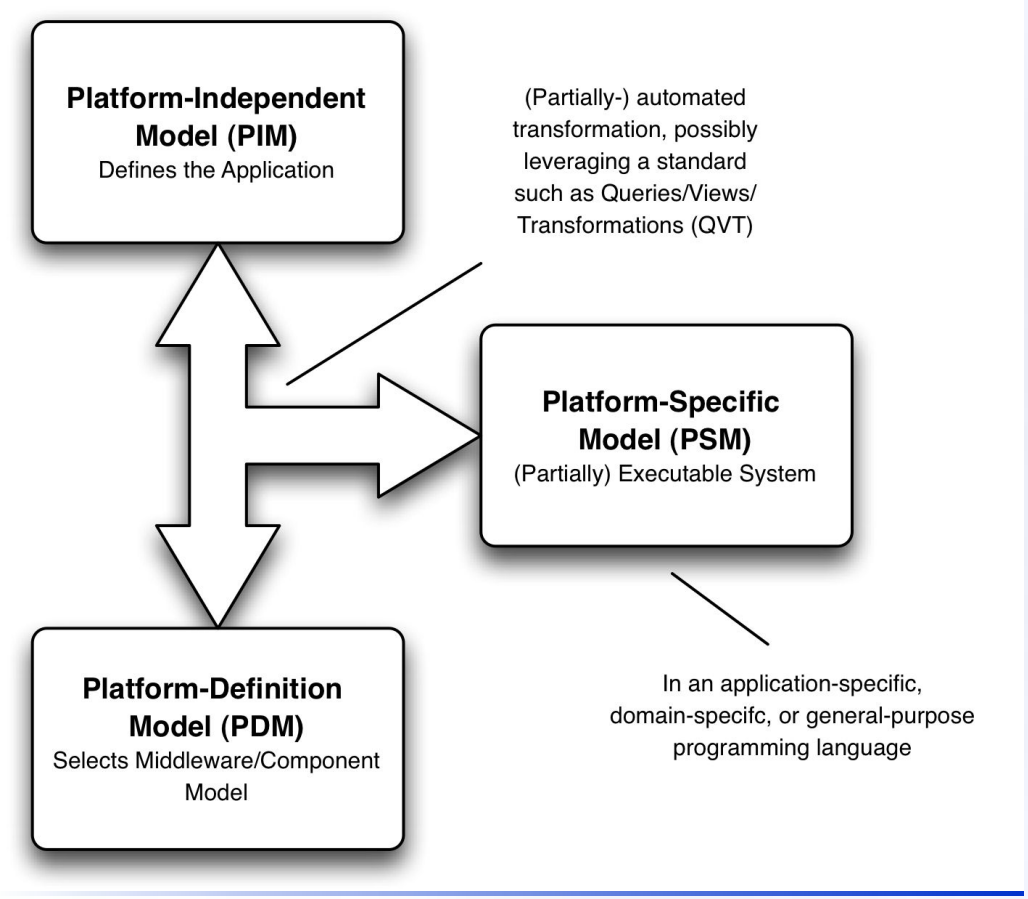
Rational Unified Process

- Phased iterative process framework/meta-process
- Like spiral model, focus on iteration and risk management
- Tends to view architecture as an artifact rather than a pervasive discipline



Model-Driven Architecture

- Also known as MDA
- Core idea: specify your architecture in detailed enough terms that implementations can be auto-generated entirely from models
- This vision is hard to achieve in general
  - May be more successful in a strong DSSE context



Overall Takeaways

- Standards confer many benefits
  - Network effects, reusable engineering knowledge, interoperability, common vocabulary and understanding
- But are not a panacea!
  - Knowledge of a breadth of standards is needed to be a good architect, but it is critical to maintain perspective
- Caveats
  - This has been a very quick tour through complex standards; many standards are hundreds of pages and can't adequately be explained in five minutes
  - Most available online – investigate yourself!

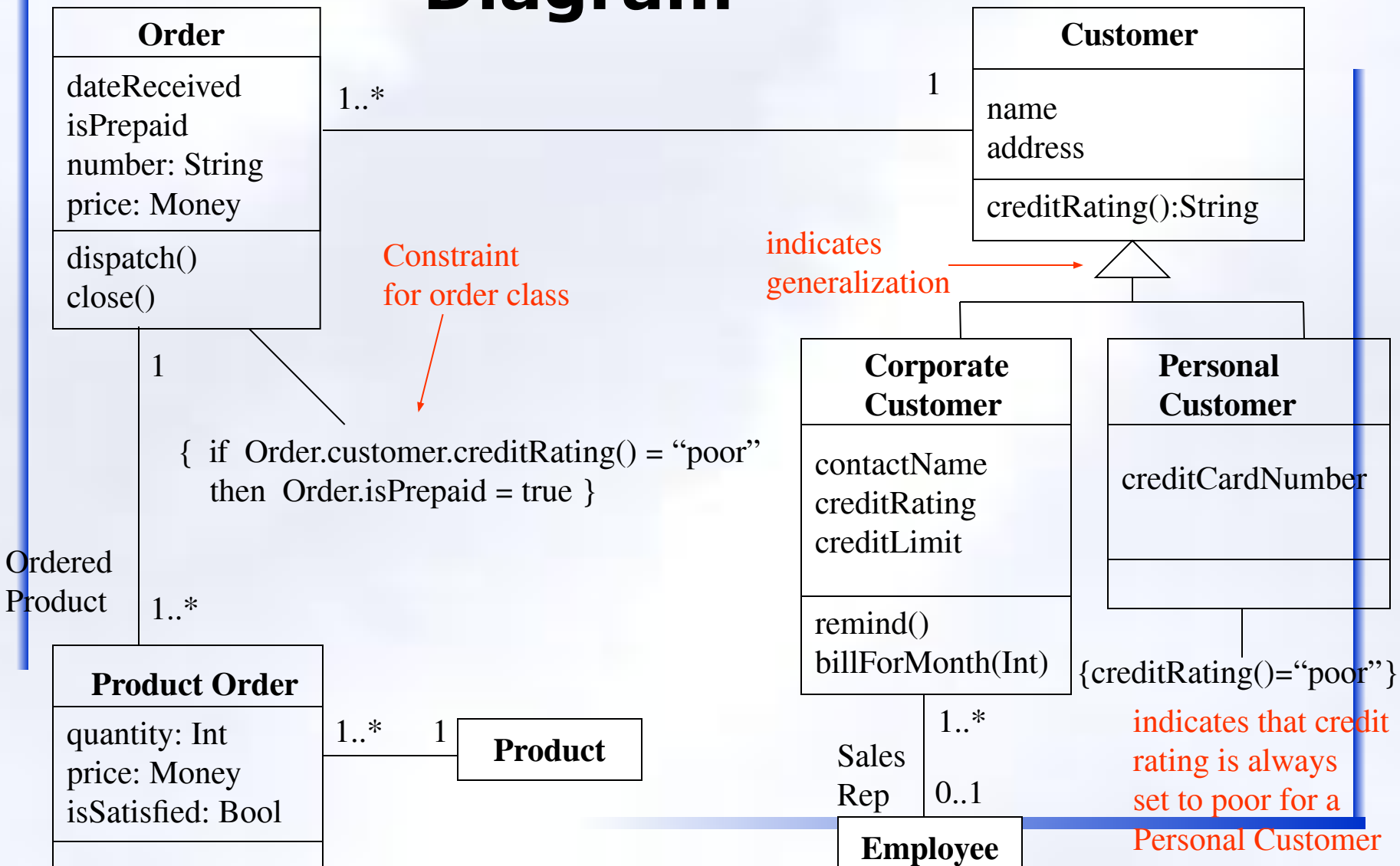
Some More on UML

- UML combines several visual specification techniques
  - use case diagrams, component diagrams, package diagrams, deployment diagrams, class diagrams, sequence diagrams, collaboration diagrams, state diagrams, activity diagrams **+ OCL**
- Semi-formal
  - Precise syntax but no formal semantics
  - There are efforts in formalizing UML semantics
- OMG defines the UML standard
  - The current UML language specification is available at: http://www.uml.org/

UML Class Diagrams

- Class diagram describes
  - Types of objects in the system
  - Static relationships among them
- Two principal kinds of static relationships
  - Associations between classes
  - Subtype relationships between classes
- Class descriptions show
  - Attributes
  - Operations

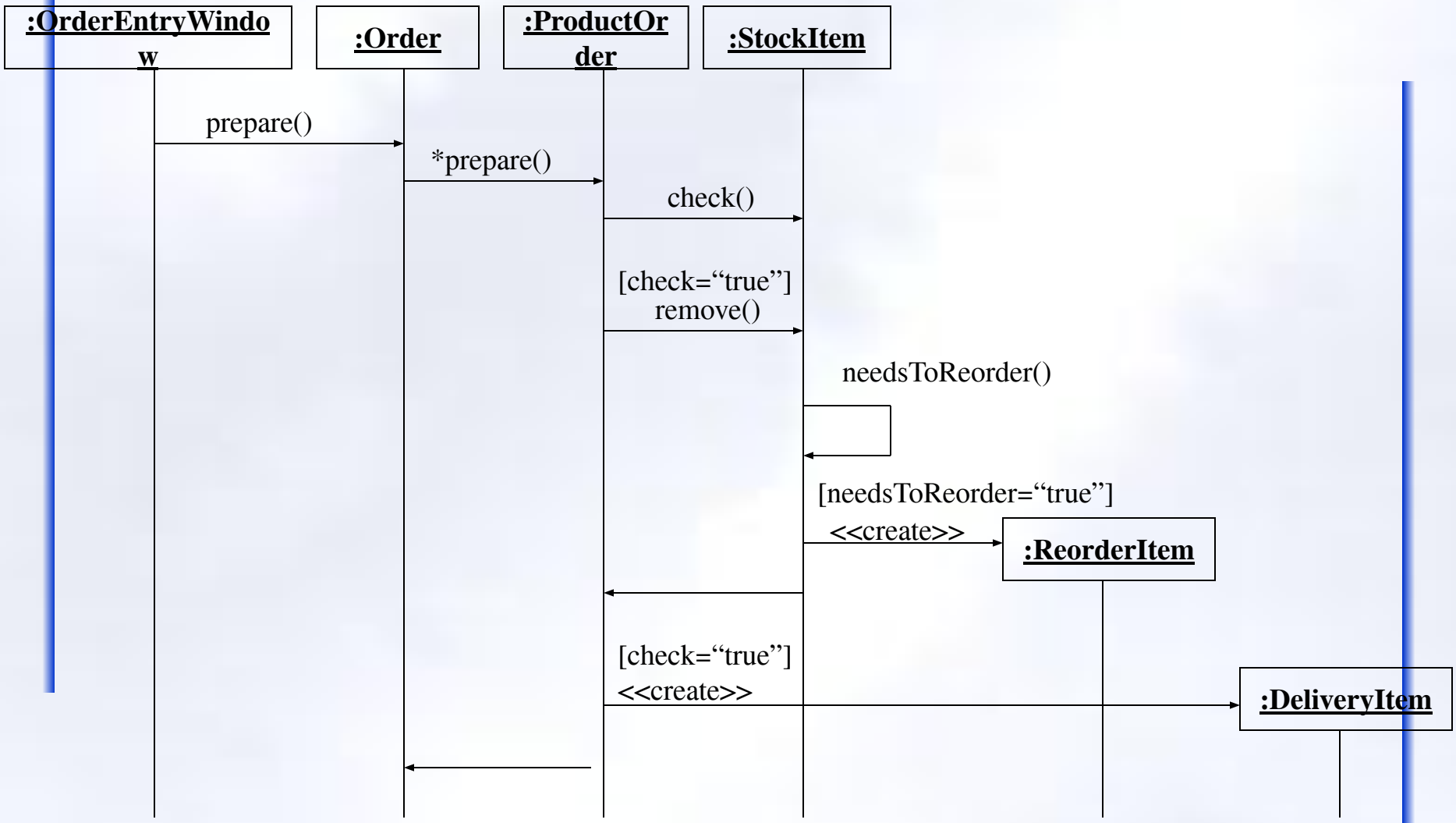
Example Class Diagram



Sequence Diagrams

- A sequence diagram shows a particular sequence of messages exchanged between a number of objects
- Sequence diagrams also show behavior by showing the ordering of message exchange
- A sequence diagram shows some particular communication sequences in some run of the system
  - it is not characterizing all possible runs

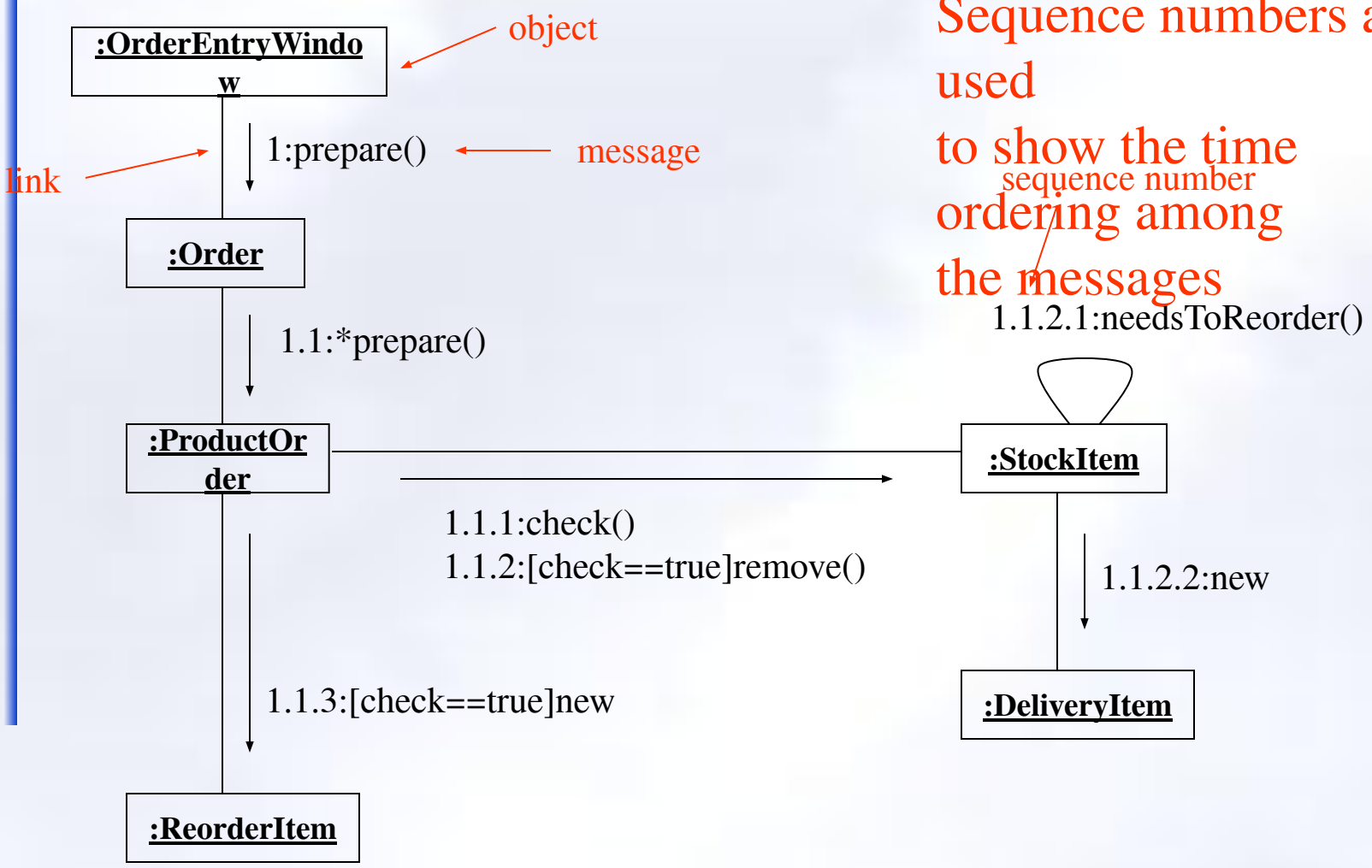
Example Sequence Diagram



Collaboration Diagrams

- Collaboration diagrams show a particular sequence of messages exchanged between a number of objects
  - this is what sequence diagrams do too!
- Use sequence diagrams to model flows of control by time ordering
  - sequence diagrams can be better for demonstrating the ordering of the messages
  - not suitable for complex iteration and branching
- Use collaboration diagrams to model flows of control by organization
  - collaboration diagrams are good at showing the static connections among the objects while also demonstrating a particular sequence of messages

Example Collaboration Diagram



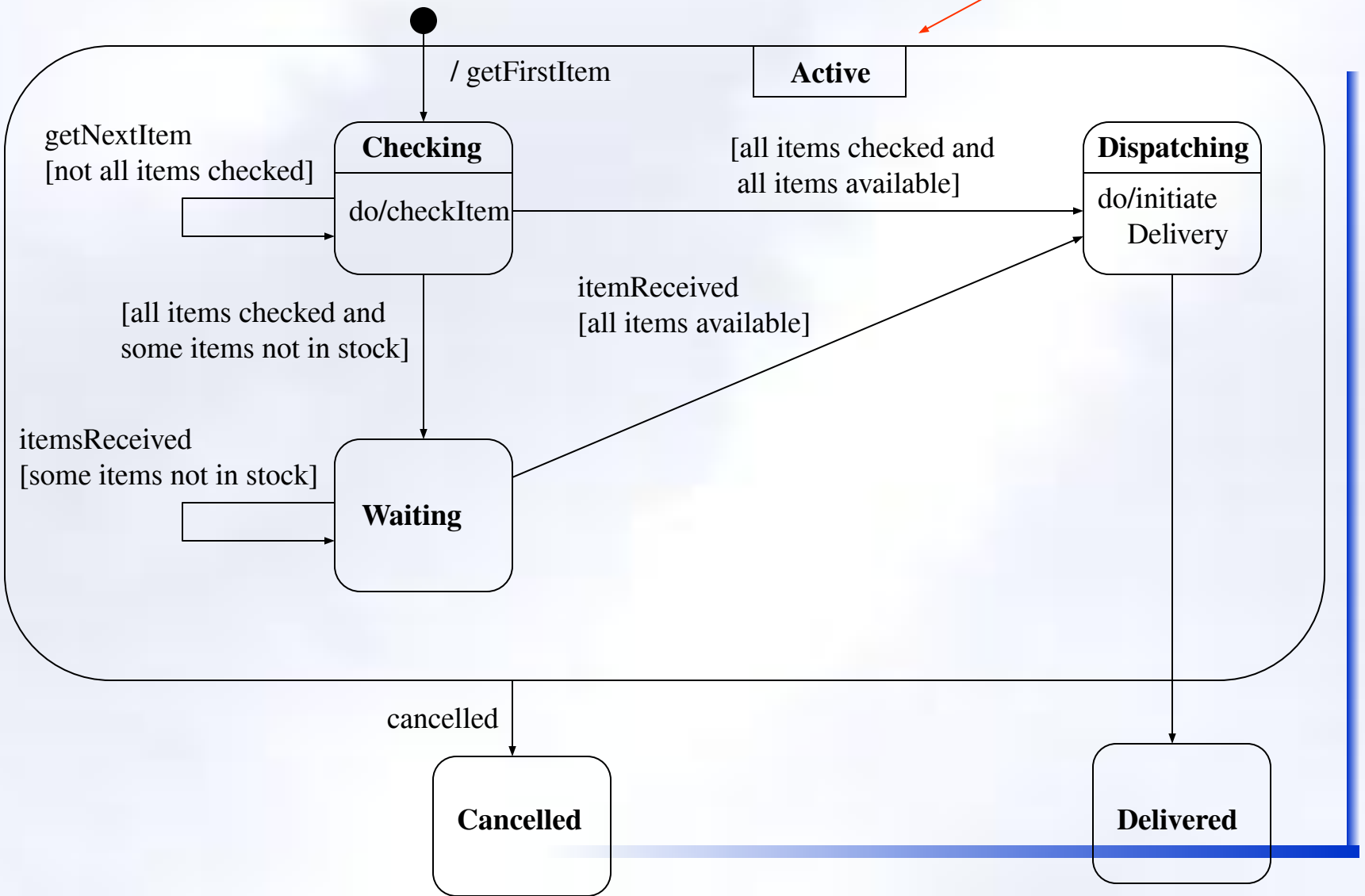
State Diagrams

- State diagrams are used to show possible states a single object can get into
- How object changes state in response to events
  - shows transitions between states
- Uses ideas from statecharts and adds some concepts such as internal transitions, deferred events etc.
  - "A Visual Formalism for Complex Systems," David Harel, Science of Computer Programming, 1987
  - hierarchical state machines with formal semantics

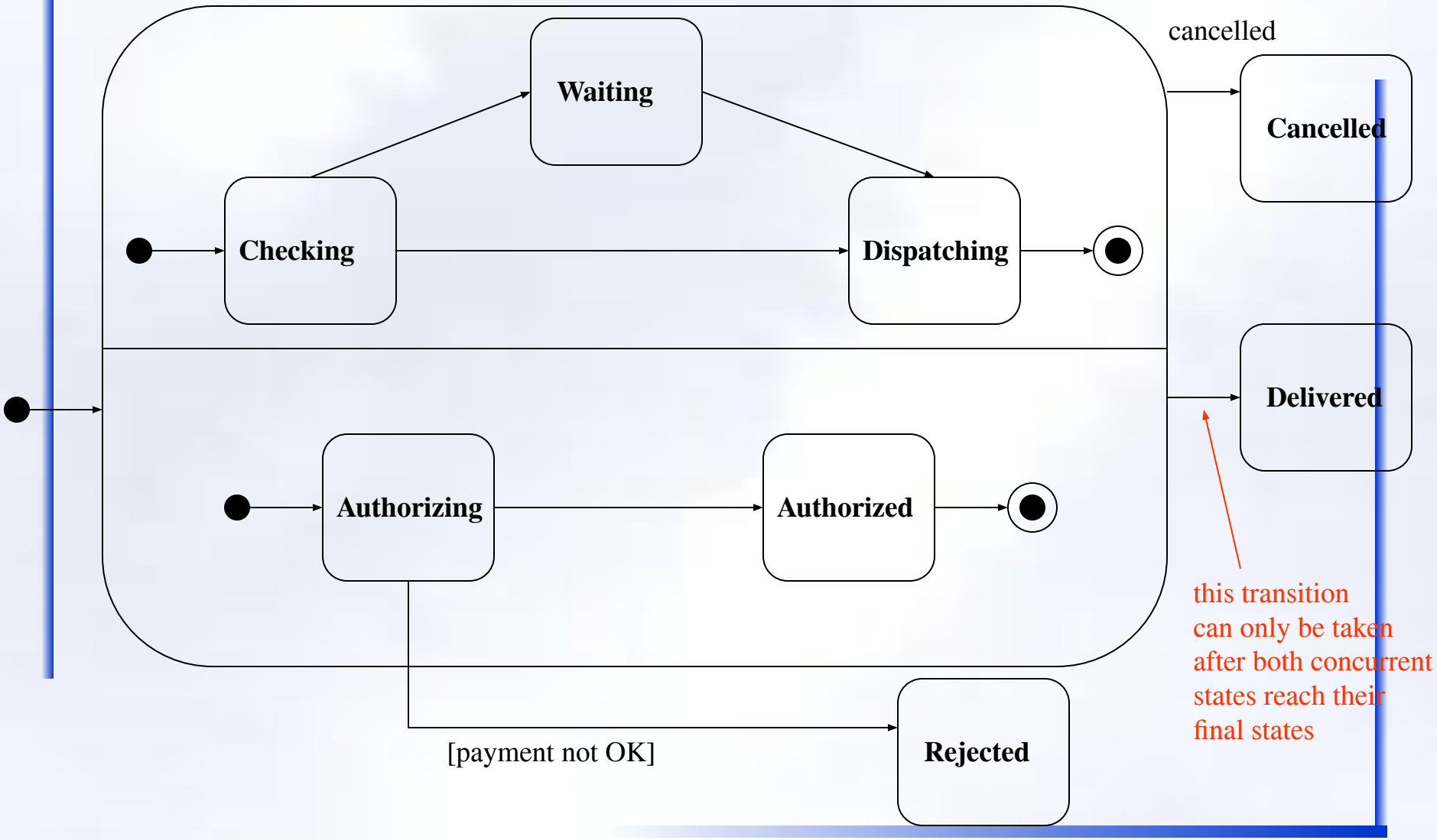
State Diagrams

- Hierarchical grouping of states
  - composite states are formed by grouping other states
  - A composite state has a set of sub-states
- Concurrent composite states can be used to express concurrency
  - When the system is in a concurrent composite state, it is in all of its substates at the same time
  - When the system is in a normal (non-concurrrent) composite state, it is in only one of its substates
  - If a state has no substates it is an atomic state

Superstates

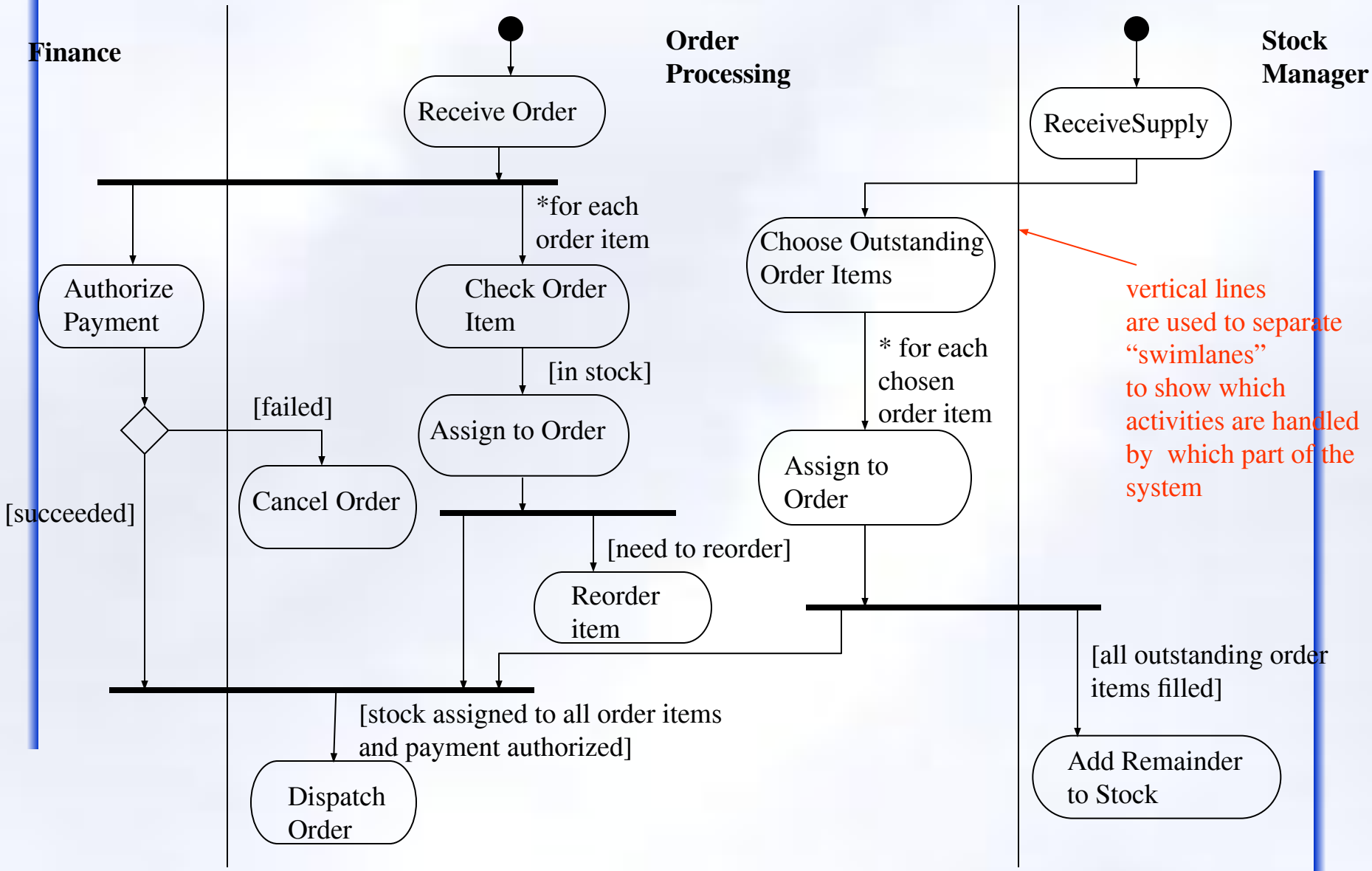


Concurrent States



Activity Diagrams

- Activity diagrams show the flow among activities and actions associated with a given object using:
  - activity and actions
  - transitions
  - branches
  - merges
  - forks
  - Joins
- Activity diagrams are basically an advanced version of flowcharts


Finance
Order Processing
Stock Manager
Receive Order
ReceiveSupply
*for each order item
Choose Outstanding Order Items
Authorize Payment
Check Order Item
[in stock]
* for each chosen order item
[failed]
Assign to Order
Assign to Order
Cancel Order
[succeeded]
[need to reorder]
Reorder item
[all outstanding order items filled]
[stock assigned to all order items and payment authorized]
Dispatch Order
Add Remainder to Stock
vertical lines are used to separate "swimlanes" to show which activities are handled by which part of the system

UML Diagrams

- Functionality, requirements
  - use case diagrams
- Architecture, modularization, decomposition
  - class diagrams (class structure)
  - component diagrams, package diagrams, deployment diagrams (architecture)
- Behavior
  - state diagrams, activity diagrams
- Communication, interaction
  - sequence diagrams, collaboration diagrams

How do they all fit together?

- Requirements analysis and specification
  - use-cases, use-case diagrams, sequence diagrams
- Design and Implementation
  - Class diagrams show decomposition of the design
  - Activity diagrams specify behaviors described in use cases
  - State diagrams specify behavior of individual objects
  - Sequence and collaboration diagrams show interaction among different objects
  - Component, package, and deployment diagrams show the high level architecture
  - Use cases and sequence diagrams can help derive test cases

Object Constraint Language

- Object Constraint Language (OCL) is part of UML
- OCL was developed at IBM by Jos Warmer as a language for business modeling within IBM
- OCL specification is available here:

http://www.omg.org/technology/documents/formal/ocl.htm

- More information:
  - "The Object Constraint Language: Precise Modeling with UML", by Jos Warmer and Anneke Kleppe

Object Constraint Language (OCL)

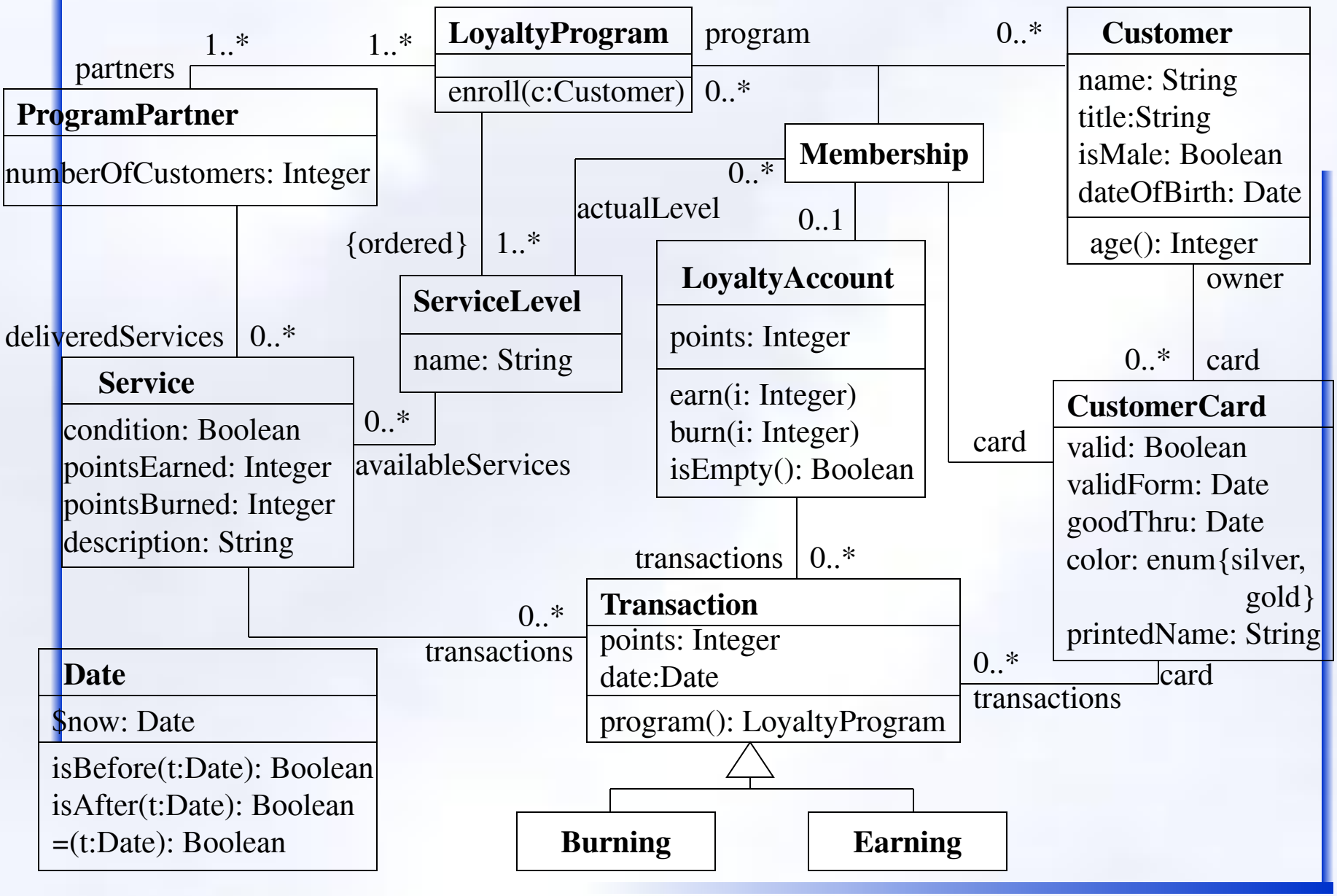
- OCL provides a way to develop more precise models using UML
- What is a constraint in Object Constraint Language?
  - A constraint is a restriction on one or more values of (part of) an object-oriented model or system

OCL Constraints

- OCL constraints are declarative
  - They specify what must be true not what must be done
- OCL constraints have no side effects
  - Evaluating an OCL expression does not change the state of the system
- OCL constraints have formal syntax and semantics
  - their interpretation is unambiguous

An Example

- Loyalty programs are used by companies to offer their customers bonuses (e.g., frequent flier miles)
- There may be more than one company participating in a loyalty program ("program partners")
- A loyalty program customer gets a membership card
- Program partners provide services to customers in their loyalty programs
- A loyalty program account can be used to save the points accumulated by a customer. Each transaction on a loyalty program account either earns or burns some points.
- Loyalty programs can have multiple service levels


LoyaltyProgram
enroll(c:Customer)
program
0..*
Customer
name: String
title:String
isMale: Boolean
dateOfBirth: Date
age(): Integer
1..*
1..*
partners
ProgramPartner
numberOfCustomers: Integer
Membership
0..*
actualLevel
0..1
{ordered}
1..*
ServiceLevel
name: String
LoyaltyAccount
points: Integer
earn(i: Integer)
burn(i: Integer)
isEmpty(): Boolean
owner
card
0..*
deliveredServices
0..*
Service
condition: Boolean
pointsEarned: Integer
pointsBurned: Integer
description: String
0..*
availableServices
CustomerCard
valid: Boolean
validForm: Date
goodThru: Date
color: enum{silver, gold}
printedName: String
card
transactions
0..*
Transaction
points: Integer
date:Date
program(): LoyaltyProgram
0..*
transactions
0..*
transactions
card
Date
$now: Date
isBefore(t:Date): Boolean
isAfter(t:Date): Boolean
=(t:Date): Boolean
Burning
Earning

Types and Instances

- OCL types are divided into following groups
  - Predefined types
    - Basic types: String, Integer, Real, Boolean
    - Collection types: Collection, Set, Bag, Sequence
  - User-defined model types
    - User defined classes such as Customer, Date, LoyaltyProgram

Operations on Boolean Type

- Boolean operators that result in boolean values

  `a or b`, `a and b`, `a xor b`, `not a`, `a = b`, `a <> b` (not equal), `a implies b`

- Another operator that takes a boolean argument is

  `if b then e1 else e2 endif`

```
Customer
title = (if isMale = true
            then 'Mr.'
            else 'Ms.'
         endif)
```

Operations on Integer and Real Types

- Operation on Real and Integer with Boolean result type

```
a = b, a <> b , a < b, a > b, a <= b, a
  >= b
```

- Operations on Real and Integer types with result type Real or Integer

```
a + b, a - b, a * b, a / b, a.abs,
  a.max(b), a.min(b)
```

- Operations on Real and Integer types with result type Integer

```
a.mod(b), a.div(b), a.round, a.floor
```

Operations on String Type

- Operations on String type with result type Boolean

  `s1 = s2, s1 <> s2`

- Operations on String type with result type String

  `s1.concat(s2), s1.toLower, s1.toUpper,`
  `s1.substring(int1, int2)`

- Operations on String type with result type Integer

  `s1.size`

Model Types

- Model types are classes, subclasses, association classes, interfaces, etc. defined in the model
- Properties of a model type are
  - attributes
  - operations and methods
  - navigations that are derived from the associations
  - enumerations defined as attribute types
- Properties of a model type can be referenced in OCL expressions

OCL expressions and constraints

- Each OCL expression has a result
  - the value that results by evaluating the expression
- The type of an OCL expression is the type of the result value
  - either a predefined type or a model type
- An OCL constraint is an OCL expression of type Boolean

Invariants

- Using OCL we can specify class invariants such as

```
Customer
age >= 18
```

- As a convention we will write the OCL expressions in the following form:

```
OCLcontext
OCLexpression
```

- For the above example, the expression `age >= 18` is an invariant of the `Customer` class, i.e. it holds for every instance of that class

Invariants

- We can also write invariants on attributes of associated classes
- Examples:

```
Membership
card.owner = customer

CustomerCard
printedName = owner.title.concat( owner.name )
```

Writing Pre and Postconditions

- One can specify the pre and postcondition of an operation of a class using OCL expressions

```
Type1::operation(arg: Type2) : ReturnType

pre: arg.attr = true

post: result = arg.attr xor self.attribute2
```

Constructs for Postconditions

- One can refer to the value of an attribute at the beginning operation in the postcondition using the `@pre` syntax

```
LoyaltyProgram::enroll(c: Customer)
pre: not customer->includes(c)
post: customer = customer@pre->including(c)
```

- You can refer to the return value of the method using the `result` keyword

```
LoyaltyAccount::isEmpty()
pre: -- none
post: result = (points = 0)
```