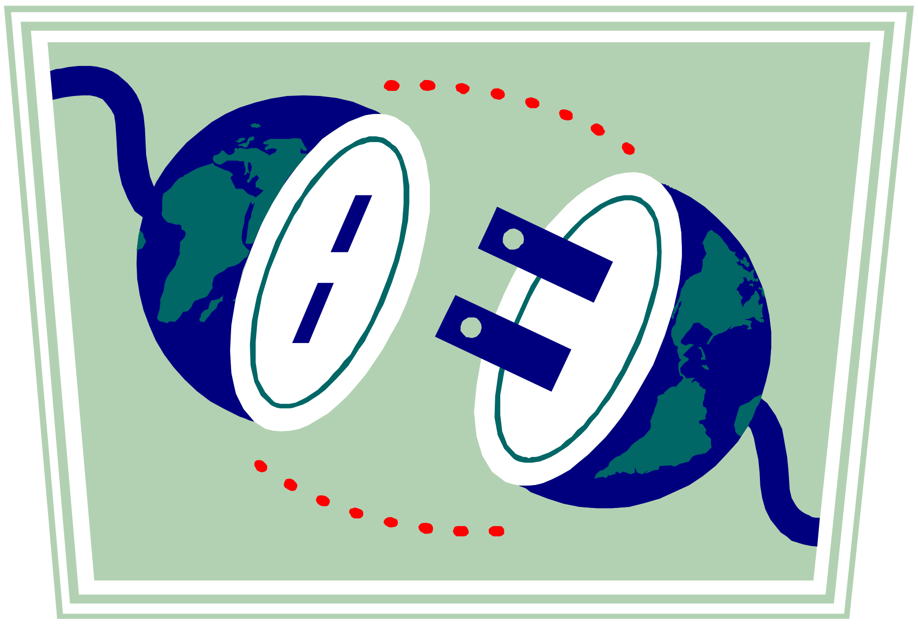
MAP

Training Course

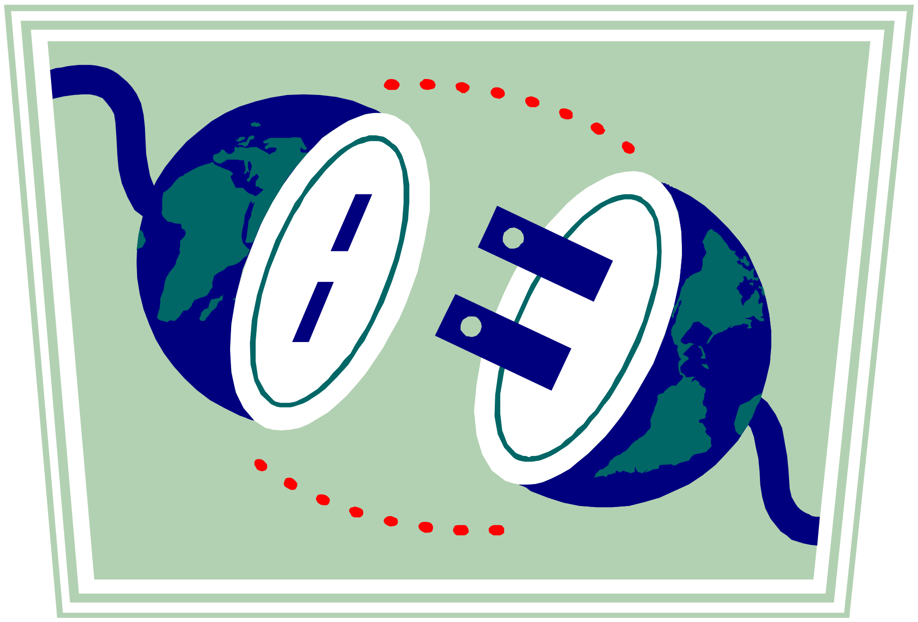


Kinetium Inc. Version 3.2

(Back of cover sheet)

(1st page after cover sheet)

MAP Training Course



Kinetium Inc.

Course Mechanics

- ♦ Let’s start with introductions, background, expectations
- ♦ Please fill out the course sign-up sheet, put your name on name cards and on your course notes
- ♦ Let’s agree planned classroom timings, breaks, etc.
- ♦ What are the facilities around the classroom?
- ♦ Some slides instructor will fill in the blanks
  - – By editing into the slide itself, on a flip chart, etc.
  - – Please fill important information into your course notes slide
- ♦ The style for class labs is
  - – Work in a team for every lab
  - – Mix business and applications people in each team
- ♦ Please fill the course evaluations before leaving

Course Objectives and Approach

- **Objectives**
  - Obtain a good end-to-end view of MAP
  - Learn to model objects, components, services in MAP
  - Understand architecture viewpoints and relationships
  - Apply modeling techniques to develop architectures
- **Non-Objectives**
  - Make you an expert in architectural design
  - Teach project management and lifecycle aspects
  - Cover broad range of architectural styles and patterns
  - Cover platform-specific technical architecture patterns
  - Cover deployment-oriented architecture patterns
- **Approach**
  - Start with basics of modeling with objects
  - Get high-level overview of the method and modeling
  - Build our toolkit to model different architectural views
  - Learn and apply toolkit incrementally on case study
  - Apply concrete examples before abstract models

Where does this course fit in?

- ♦ Recommended path to applying MAP includes
  - – Project planning and definition including MAP tailoring
  - – MAP overviews (2 hour – 1 day)
  - – MAP in-depth training course (4-5 days)
  - – Project focused workshop (1-2 days)
  - – MAP mentor working with team on their first project

- ♦ Other resources on (or on their way to) the Kinetium site include
  - – Method guide and templates
  - – Summary reference sheets
  - – Detailed documentation
  - – Tool customizations

  - – **Note**: Some materials on the web site require licenses for project use

Course Outline

		Normal Time Line
1.	Introduction ♦ Objects, Components, and Applications	[Mon 9:00 …]
2.	Overview of MAP ♦ Architecture Viewpoints, Concepts, Project Lifecycle	[Mon 10:00 …]
3.	Basic Modeling Techniques ♦ Objects, Snapshots, Types, Actions	[Mon 1:00 pm …]
4.	Integrated Information and Behavior Modeling ♦ Integrated Modeling Skills	[Tue 11 am …]
5.	Business View of <System X> ♦ Business Goals, Information Model and Behavior Model	[Wed 11 am …]
6.	Black Box View of <System X> ♦ Use Cases, Information Model, Black box Assembly	[Wed 4 pm …]
7.	White Box View of <System X> ♦ Partitioning, Collaborations, White box Assembly	[Thu 12 pm …]
8.	Technical View of <System X> ♦ Component Realization, Connector Realization	[Thu 4 pm …]
9.	Deployment View of <System X> ♦ Nodes, networks, devices, and software deployment	[Fri 1 pm …]
10.	Optional: Models and code ♦ Implementation classes and interfaces from models	

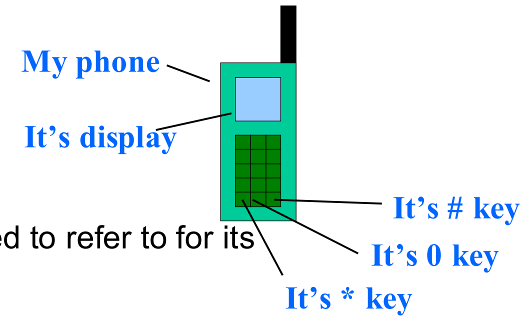
- ♦ Appendix: Key Points
- ♦ Appendix: References
- ♦ Appendix: Exercises and Solutions

Chapter 1
Introduction – Objects & Components

This chapter introduces modeling with objects and components.

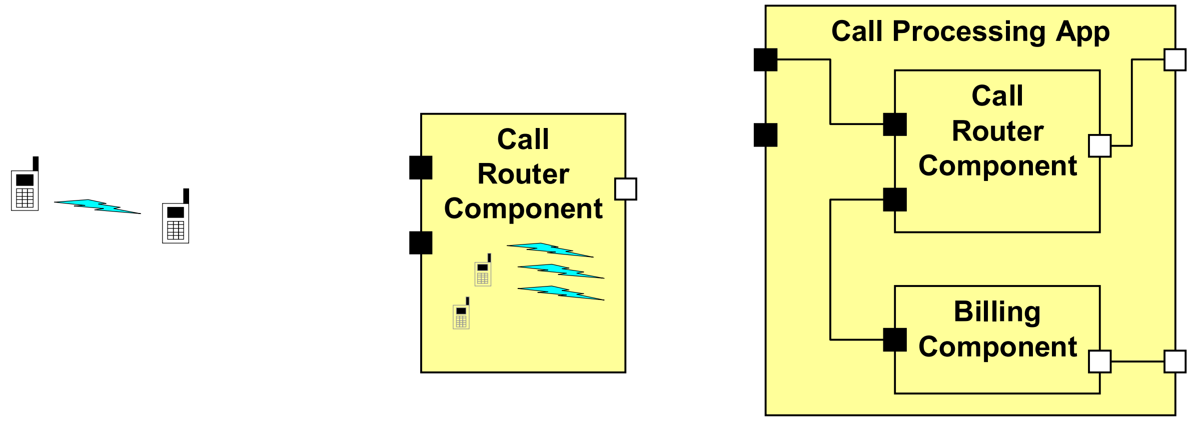
It covers:
- ♦ Objects and Actions
- ♦ Objects, Components, and Applications
- ♦ Type vs. Class
- ♦ Components
- ♦ Component specification vs. component implementation

Objects and Actions



- ♦ **Object**
  - – An individual, identifiable thing that you need to refer to for its
    - • **State**
    - • **Behavior**
  - – **Individual** – a single, well-demarcated thing distinct from every other object
    - • A phone, an application, a road
  - – **State** = **attributes** / links to other objects
    - • Phone: on/off, phone number, calling plan
    - • Road: name, intersections, traffic pattern
  - – **Behavior** = **actions** (operations, interactions, things it does or done to it)
    - • Phone: call, use feature, turn on/off
    - • Road: take out of service
- ♦ **Object Behavior and State are intertwined**
  - – Action outcome is affected by, and affects, the attributes of objects
  - – Actions are performed by objects that participate in that action

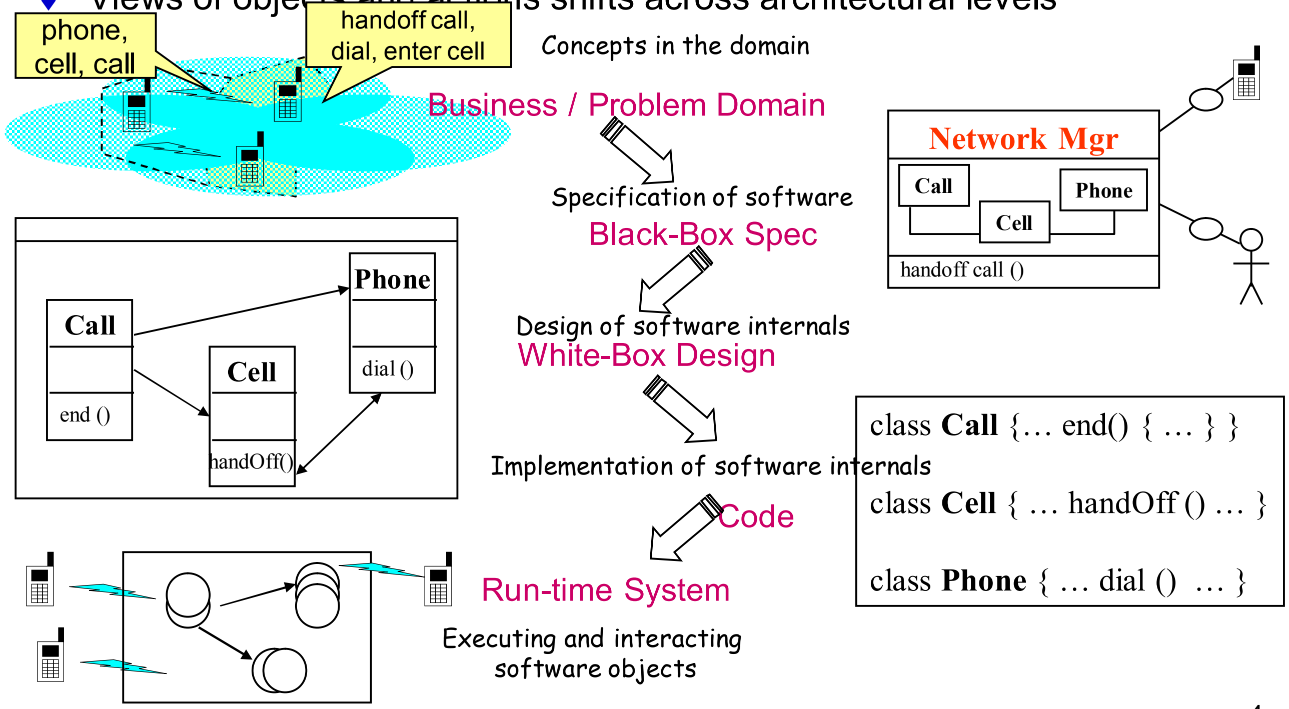
Objects: From Small Things to Components & Applications



- Objects range in “size” from cell phones and calls to entire call-processing apps
- Objects of interest have some interesting state or behavior
  - Objects in the domain – cell phone, call
  - Larger-grained software components – call router component for many calls, phones
  - Assemblies of components into applications – call processing application

Objects and Actions – Continuity across Levels

- Objects, attributes, and actions offer continuity of concept and notations
- Views of objects and actions shifts across architectural levels

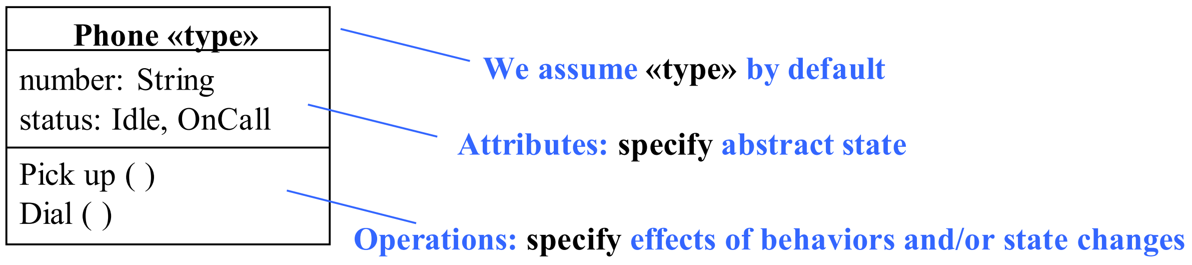


phone, cell, call
handoff call, dial, enter cell
Concepts in the domain
Business / Problem Domain
Network Mgr
Call
Phone
Cell
handoff call ()
Specification of software
Black-Box Spec
Call
end ()
Cell
handOff()
Phone
dial ()
Design of software internals
White-Box Design
Implementation of software internals
Code
Run-time System
Executing and interacting software objects

```
class Call {… end() { … } }
class Cell { … handOff () … }
class Phone { … dial ()  … }
```

What is a Type

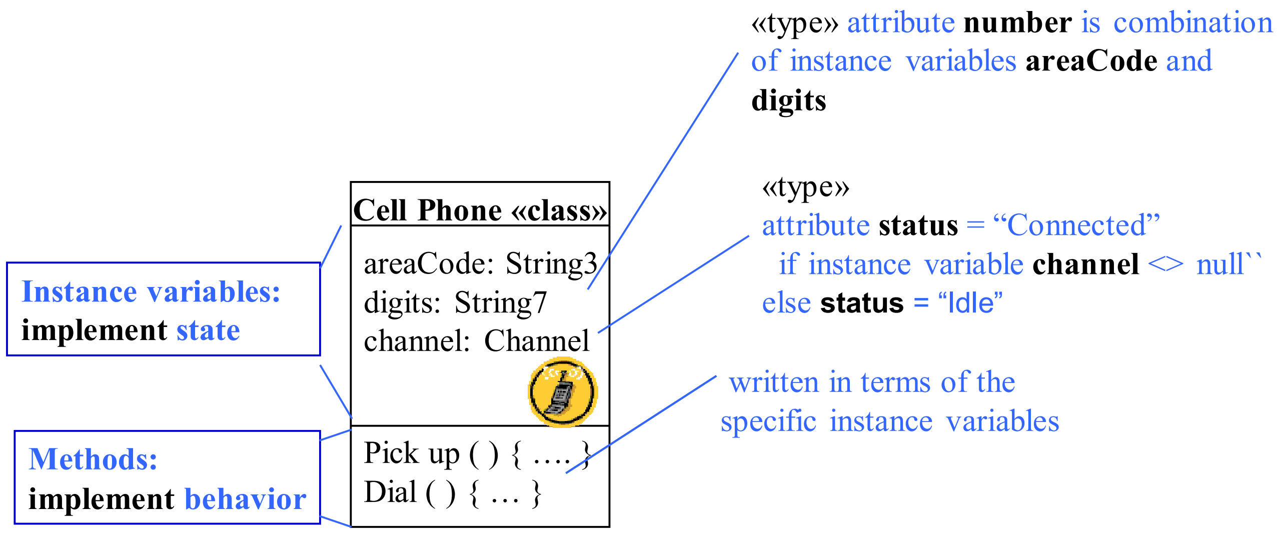
- A **type** is a **specification** of selected externally known properties of some object
  - required object “attributes” for its state and for any information exchanged
  - required object behavior as “operations”
  - all descriptions are independent of specific implementations
- An object conforms to a type if it somehow implements the external properties
  - That object is said to be a member of every type it conforms to


Phone «type»
number: String
status: Idle, OnCall
Pick up ( )
Dial ( )
We assume «type» by default
Attributes: specify abstract state
Operations: specify effects of behaviors and/or state changes

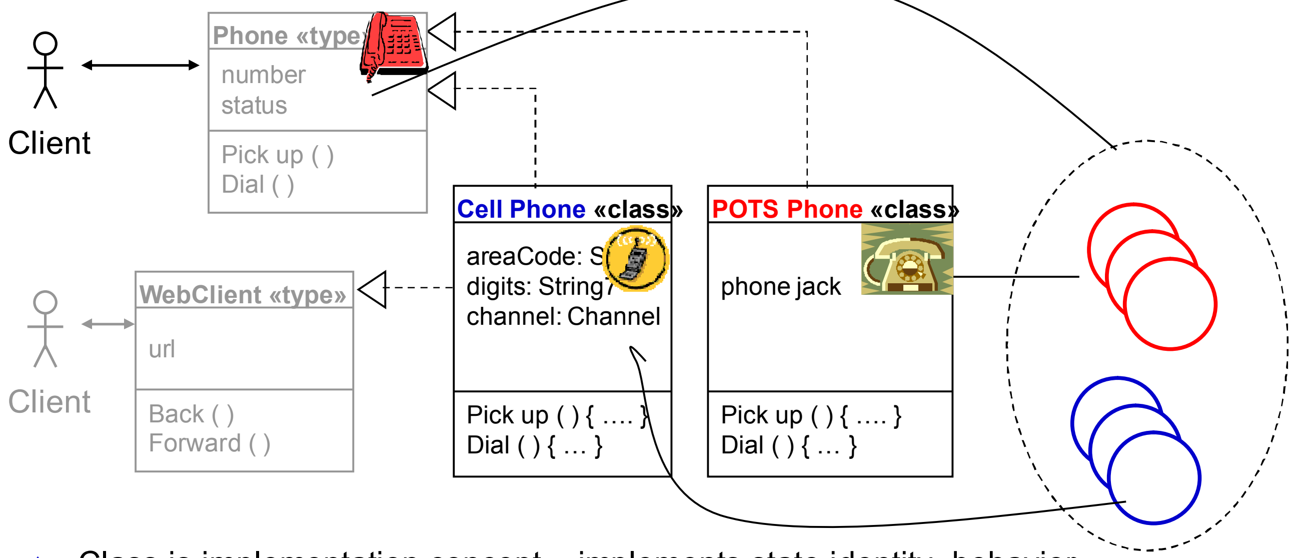
- Type can be used roughly interchangeably with
  - Interface
  - Pure abstract class
  - Role

What is a Class

- A **class** is an **implementation** template for all instances of that class
  - specific instance variables that every instance will store for state / attributes
  - specific method code that every instance will execute for behavior / operations
- An **object** is an instance of one class


«type» attribute number is combination of instance variables areaCode and digits
Cell Phone «class»
areaCode: String3
digits: String7
channel: Channel
Pick up ( ) { …. }
Dial ( ) { … }
«type» attribute status = “Connected” if instance variable channel <> null`` else status = “Idle”
Instance variables: implement state
written in terms of the specific instance variables
Methods: implement behavior

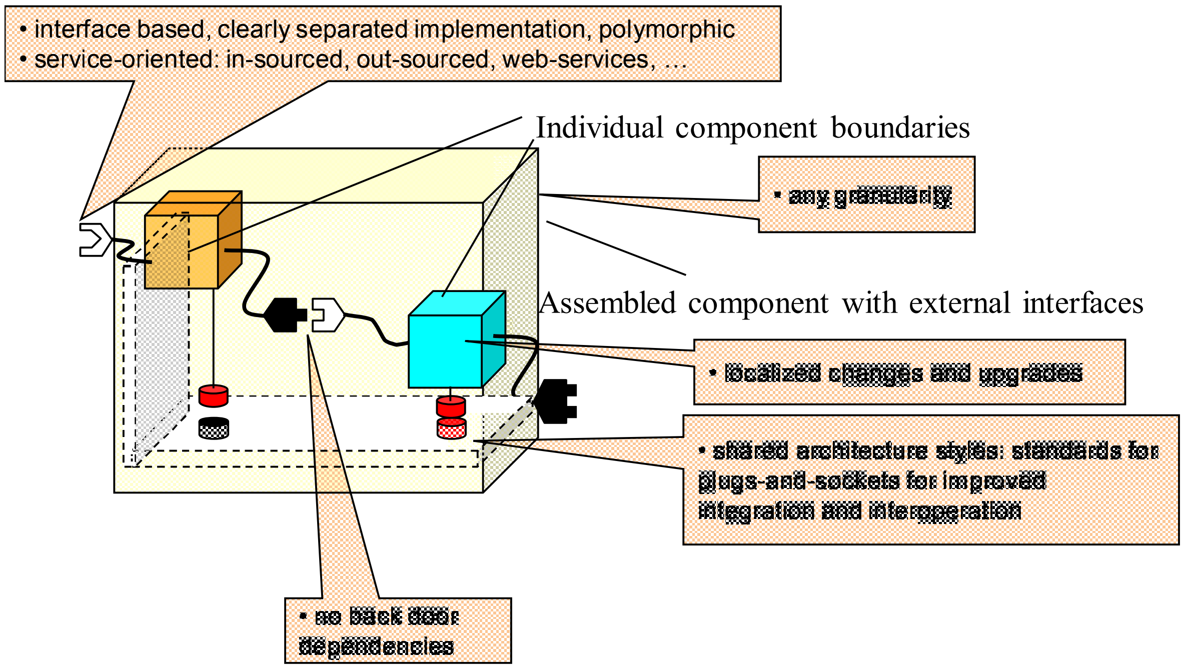
Object Vs. Type Vs. Class



- Class is implementation concept – implements state,identity, behavior
  - An object is an instance of one class
- Type is specification concept – externally visible object specification for client
  - An object conforms to one (or more) types
- A class implements one (or more) types; many classes can implement one type
- **Polymorphism:** multiple implementations of interface type, transparent to client

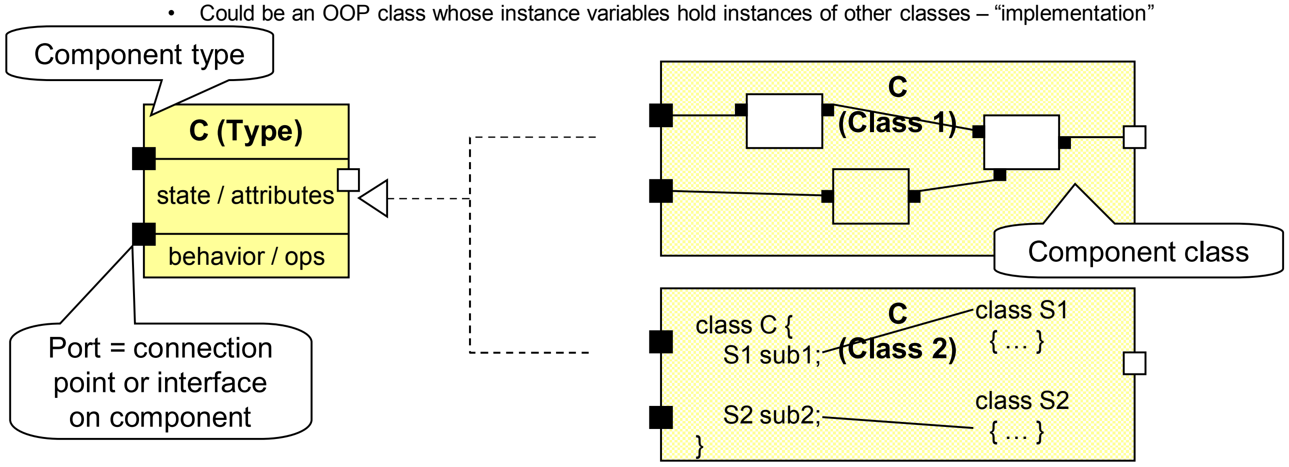
What is a Component?

A package of software that can be independently replaced. It both provides and requires services based on specified interfaces. It conforms to architectural standards to interoperate with other components. It runs like a large grained object.


• interface based, clearly separated implementation, polymorphic
• service-oriented: in-sourced, out-sourced, web-services, …
Individual component boundaries
• any granularity
Assembled component with external interfaces
• localized changes and upgrades
• shared architecture styles: standards for plugs-and-sockets for improved integration and interoperation
• no back door dependencies

Component "Type" vs. Component "Class"

- A component has an external specification and internal design / implementation
  - External specification is the component "type" – we will call this the **"black box"** view
    - It specifies component behavior visible through all its ports, each port optionally as its own type
  - Internal design/implementation is the component "class" – we will call this the **"white box"** view
    - Could be an assembly of some number of smaller components and their interconnections – "design"
    - Could be an OOP class whose instance variables hold instances of other classes – "implementation"

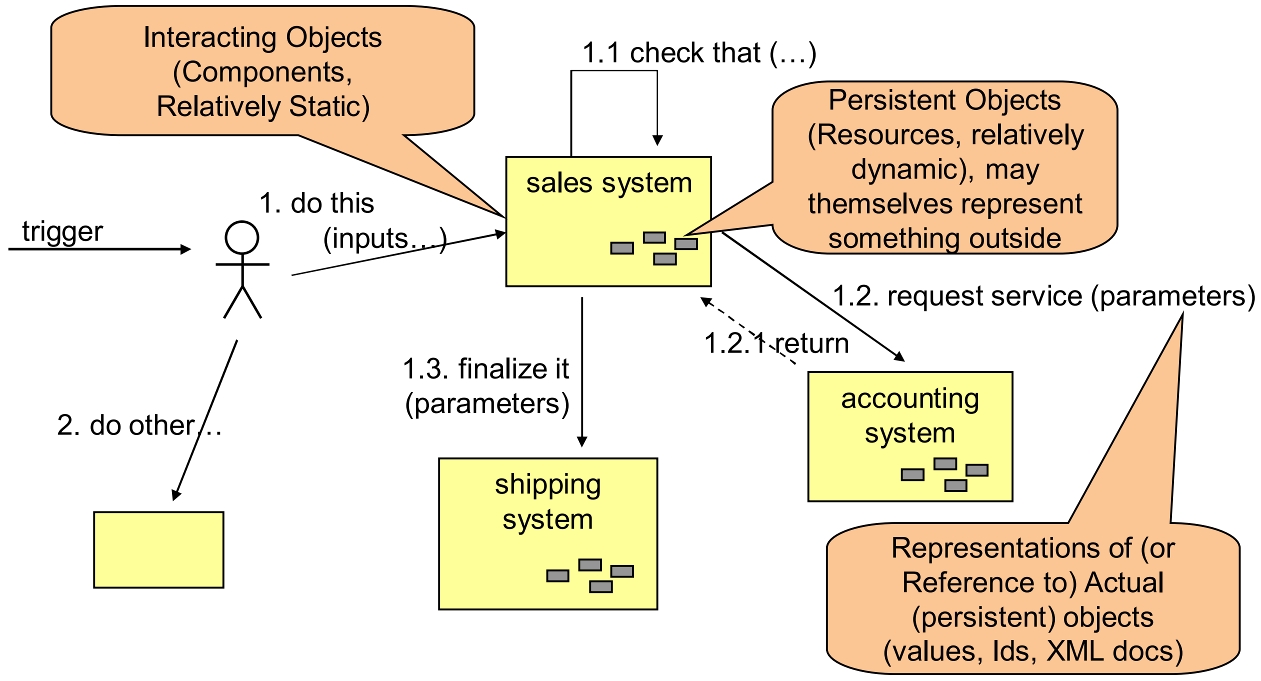


- So type and class are analogous to "component spec" and "component design"
  - Can be realized with many implementation technologies
  - Not specific to an object-oriented programming language

Active Objects vs. Information Objects

- We often need to distinguish between these two
- Active Objects: participate in interactions with other active objects
  - People
  - Applications
  - Components
  - Hardware devices
- Information Objects: information objects persisted or passed around
  - Trade Record
  - Customer Identifier
  - Hotel Reservation Record
- Information objects are often representations of some "real" objects

Interacting Objects / Components


Interacting Objects (Components, Relatively Static)
1.1 check that (…)
Persistent Objects (Resources, relatively dynamic), may themselves represent something outside
sales system
trigger
1. do this (inputs…)
1.2. request service (parameters)
1.2.1 return
1.3. finalize it (parameters)
2. do other…
accounting system
shipping system
Representations of (or Reference to) Actual (persistent) objects (values, Ids, XML docs)

- Interactions between objects shown with labeled, numbered arrows
- Nested numbering shows nested response to triggering request
- Parameters and returns (are objects that) represent other persistent objects

Exercise 1.1

Summary

This chapter has introduced:

- Objects and Actions
- Type as specification vs. class as implementation
- Modeling components as objects with all interfaces provided or required
- Component specification vs. component implementations
- Interacting components (objects) exchange other objects, or representations of other objects

Chapter 2
Overview of MAP

This chapter provides an overview of MAP, and how it describes systems in terms of interacting objects at multiple levels of abstraction.

It covers:

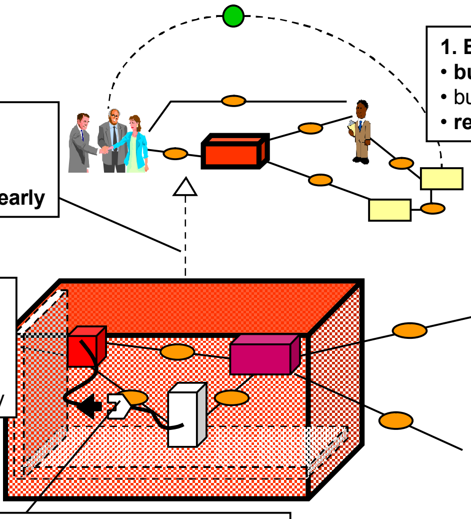
- ♦ What is MAP?
- ♦ Formal Architectural Framework
- ♦ Multiple Viewpoints and Concerns
- ♦ Process Pragmatics for Project Planning and Lifecycle
- ♦ Key Concepts of MAP

Outline

- ♦ **What is MAp?**
- ♦ MAp – Formal Architecture Framework
  - – Goal Modeling
  - – Viewpoints and Concerns – Black Box, White Box, Technical, …
  - – Architecture Style
- ♦ MAp – Process Pragmatics
  - – Iterative and Incremental
  - – Roadmap Route: Problem Definition, Current State, Target State, Migration Plan
  - – Flexible Models and Domains
- ♦ Summary

What is MAP?

- MAP – **M**odel-Driven **Ap**proach for clear, business-aligned, architectural solutions
- MAP is a systematic approach to plan, architect, develop, and evolve software systems
  - Clearly separated viewpoints and models to design or analyze an architecture



1. Business Driven
- **business goals, terminology** first
- business process to meet goals
- **relate multiple systems, programs, views**

2. Traceability
- bridge **Business** and IT
- **precise** shared vocabulary
- critical business questions **early**

4. Consistent Architecture
- blueprint for consistency
- principles, patterns, **styles**
- both business and technical
- **standards** for interoperability

5. Reuse Process
- reuse **business models**
  … and **architectures**
  … and **implementations**
- develop **for** reuse
- develop **with** reuse

3. Design and Assemble Pluggable Components
- federated components, services; not monolithic systems
- clear **interfaces** and **responsibilities**
- reusable and maintainable building blocks

Builds upon best practice and industry standards for architecture: UML, EDOC, RM-ODP, Catalysis, Simple Components, RUP, Problem Frames, Goal Modeling

Two Aspects of the MAP Framework

Formal Architecture Framework

- Goals
- Viewpoints
- Concerns
- Models
- Refinement
- Architecture Styles

- Architecture always ties back to goals.
- Different presentation suitable to audience
- Clear and unambiguous architecture
- Consistency rules across descriptions

Process Pragmatics

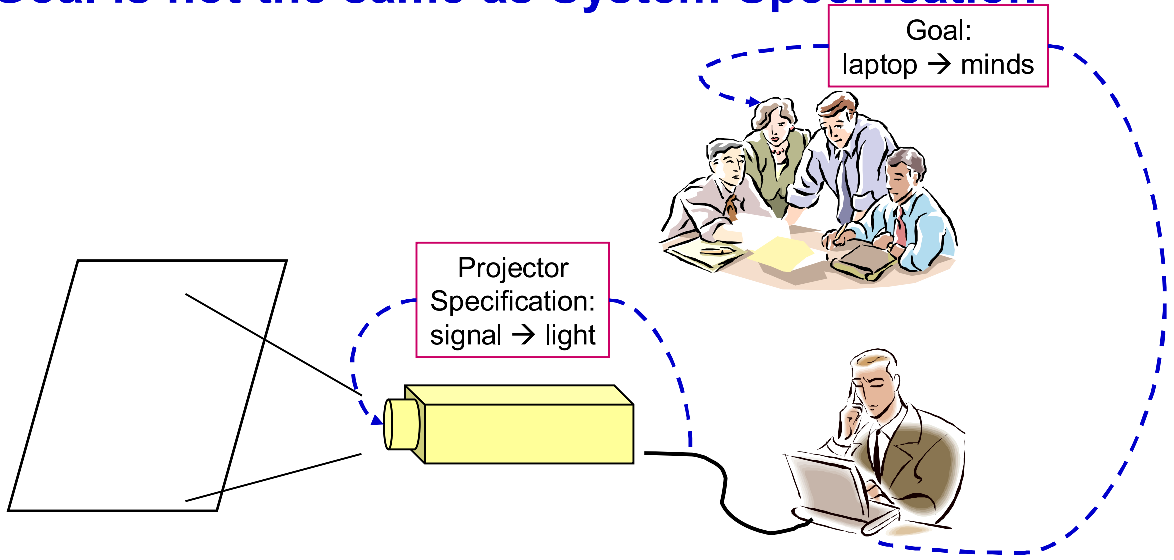
- Routes: Roadmap, Construction, …
- Different Entry Points
- Iterative and Incremental
- List, Draft, or Dressed Templates
- Guidelines
- Checklists

- How to prioritize and focus work?
- How much detail is appropriate and when?
- How to show early value and reduce risk?
- How to customize to project needs?

Outline

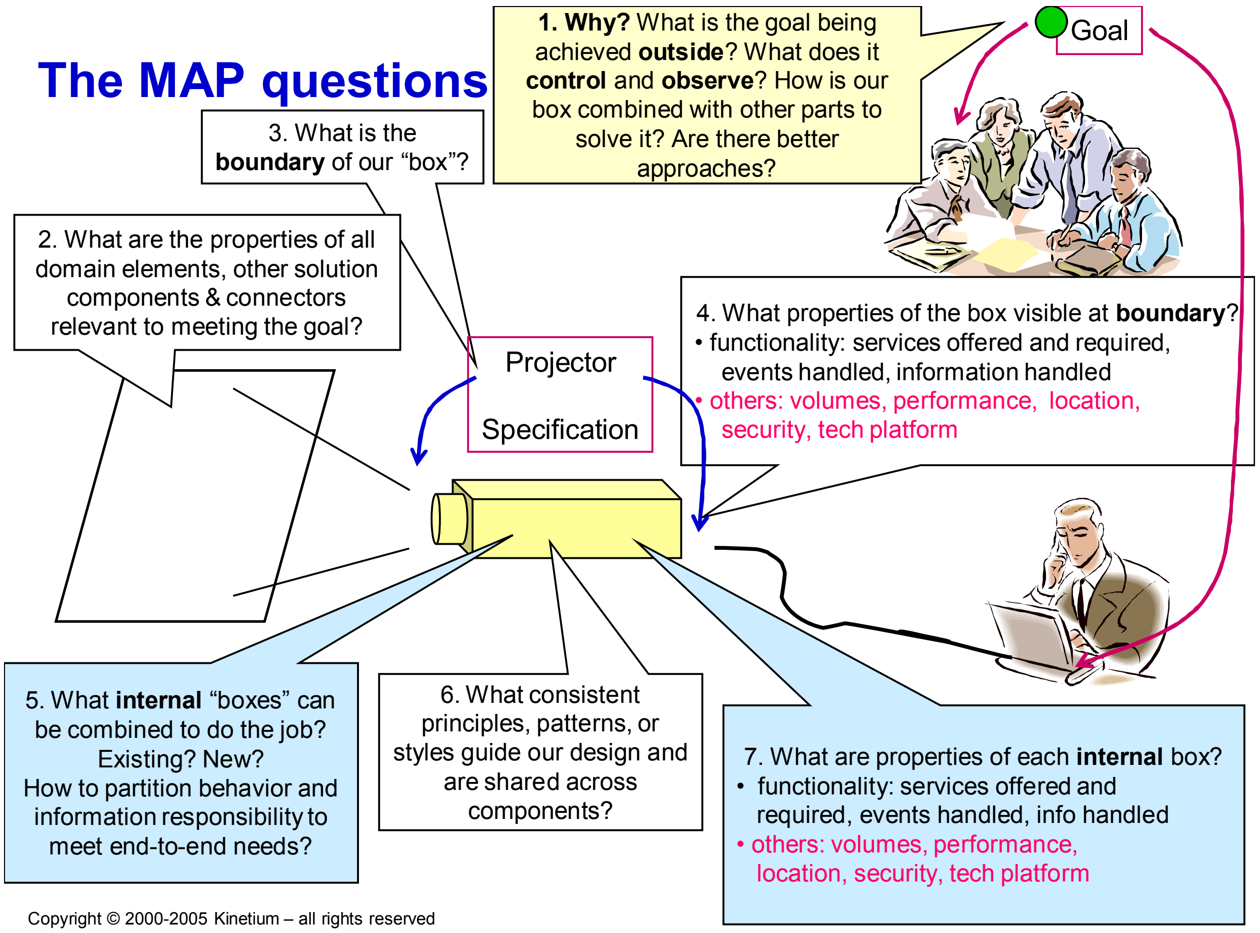
- What is MAp?
- **Formal Architecture Framework**
  - **Goal Modeling**
  - Viewpoints and Concerns – Black Box, White Box, Technical, …
  - Architecture Style
- Process Pragmatics
  - Iterative and Incremental
  - Roadmap Route: Problem Definition, Current State, Target State, Migration Plan
  - Flexible Models and Domains
- Summary

Goal is not the same as System Specification



- Goals are typically 1, 2, or more levels "removed" from the target software system
  - Goal to **observe** some things in the "problem domain" to **control** other things
- Target system + other elements + domain properties needed to meet the goal
  - Target system = projector
  - Other elements = screen, cables, laptop, speaker, audience, room
  - Domain properties = audience location, room brightness, …

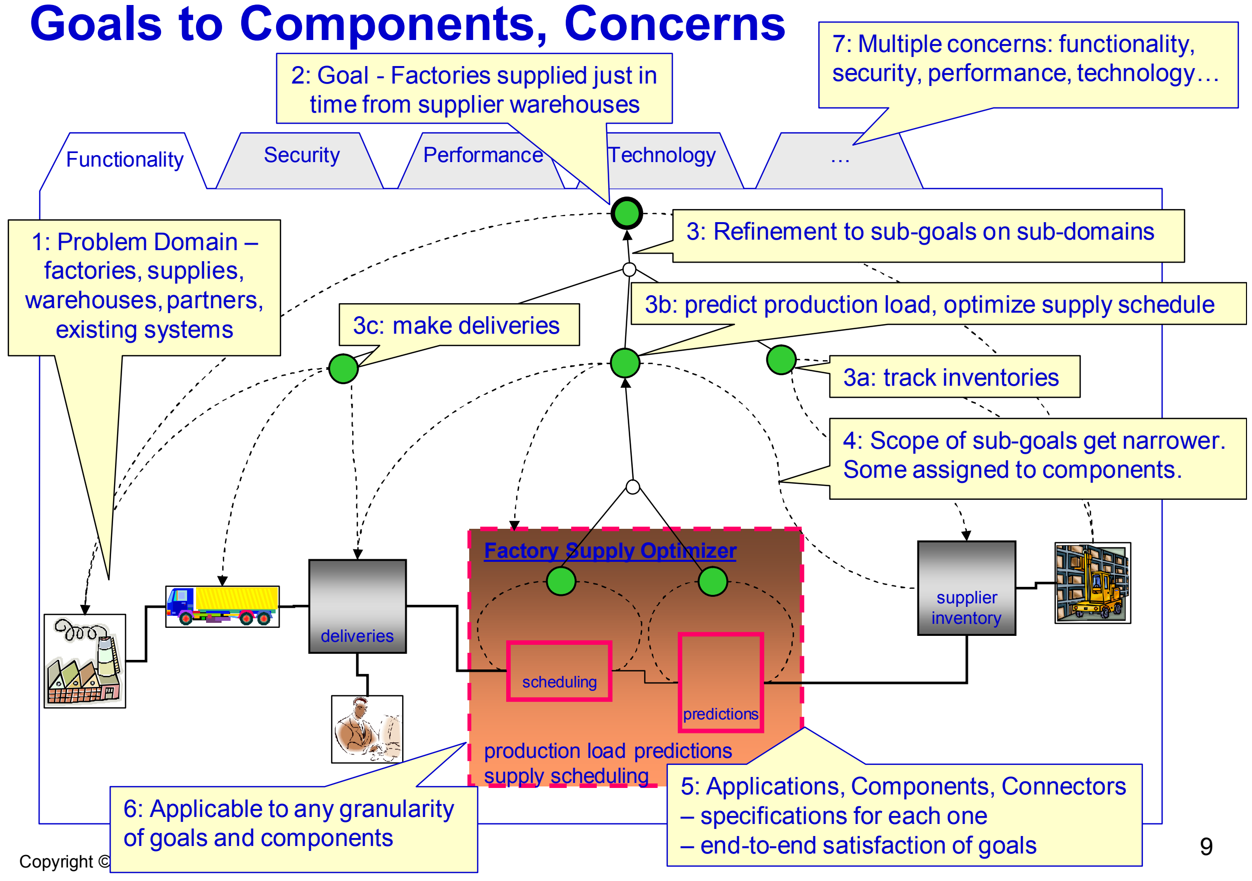
The MAP questions


1. **Why?** What is the goal being achieved **outside**? What does it **control** and **observe**? How is our box combined with other parts to solve it? Are there better approaches?
Goal
3. What is the **boundary** of our "box"?
2. What are the properties of all domain elements, other solution components & connectors relevant to meeting the goal?
Projector Specification
4. What properties of the box visible at **boundary**?
• functionality: services offered and required, events handled, information handled
• others: volumes, performance, location, security, tech platform
5. What **internal** "boxes" can be combined to do the job? Existing? New? How to partition behavior and information responsibility to meet end-to-end needs?
6. What consistent principles, patterns, or styles guide our design and are shared across components?
7. What are properties of each **internal** box?
• functionality: services offered and required, events handled, info handled
• others: volumes, performance, location, security, tech platform

Outline

- What is MAp?
- **Formal Architecture Framework**
  - Goal Modeling
  - **Viewpoints and Concerns – Black Box, White Box, Technical**
  - Architecture Style
- Process Pragmatics
  - Iterative and Incremental
  - Roadmap Route: Problem Definition, Current State, Target State, Migration Plan
  - Flexible Models and Domains
- Summary

Goals to Components, Concerns



1: Problem Domain – factories, supplies, warehouses, partners, existing systems

2: Goal - Factories supplied just in time from supplier warehouses

3: Refinement to sub-goals on sub-domains

3a: track inventories

3b: predict production load, optimize supply schedule

3c: make deliveries

4: Scope of sub-goals get narrower. Some assigned to components.

5: Applications, Components, Connectors
– specifications for each one
– end-to-end satisfaction of goals

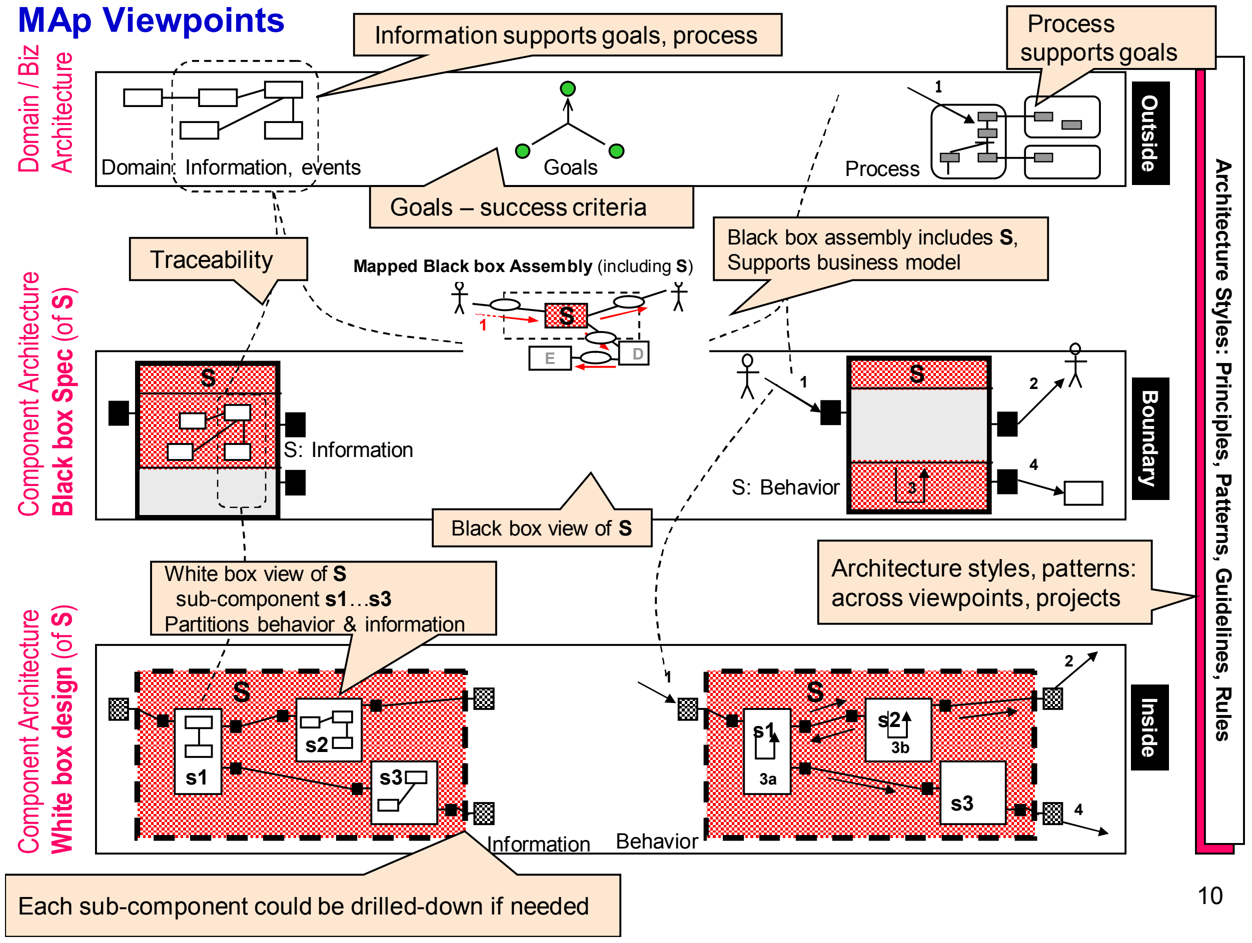
6: Applicable to any granularity of goals and components

7: Multiple concerns: functionality, security, performance, technology…

Copyright ©

9

MAp Viewpoints


Domain / Biz Architecture
Domain: Information, events
Goals
Process
1
Outside
Component Architecture Black box Spec (of S)
Mapped Black box Assembly (including S)
S
1
E
D
S
S: Information
S: Behavior
1
2
3
4
Boundary
Component Architecture White box design (of S)
S
s1
s2
s3
s1
3a
s2
3b
s3
1
2
4
Information
Behavior
Inside
Architecture Styles: Principles, Patterns, Guidelines, Rules

Information supports goals, process

Process supports goals

Goals – success criteria

Traceability

Black box assembly includes **S**, Supports business model

Black box view of **S**

White box view of **S**
sub-component **s1**…**s3**
Partitions behavior & information

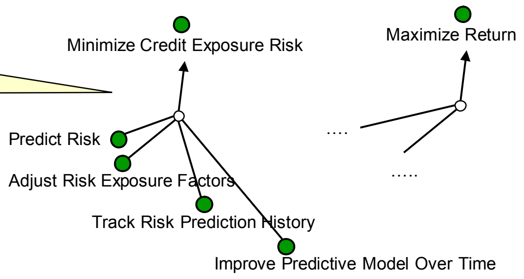
Architecture styles, patterns: across viewpoints, projects

Each sub-component could be drilled-down if needed

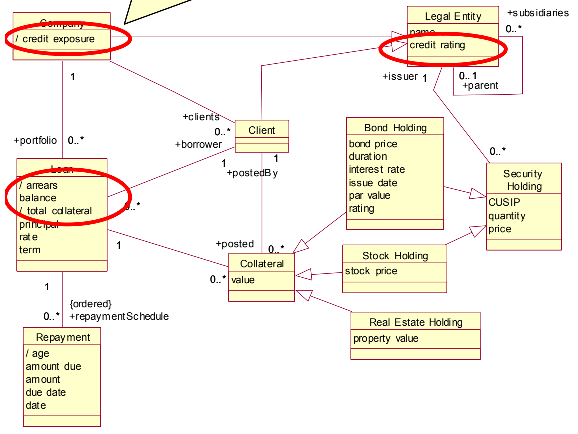
10

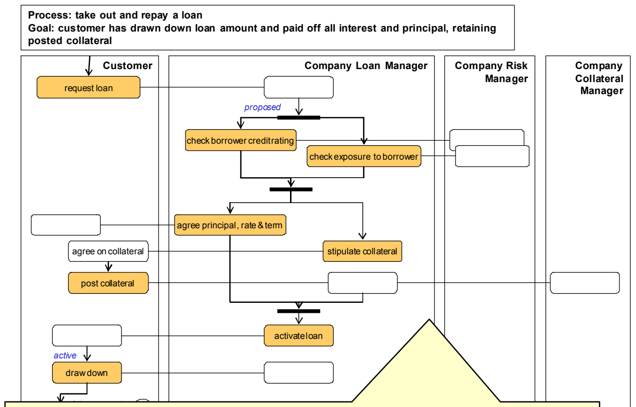
Loans Example – Goals, Domain Model, Biz Process Model

- Goals clearly defined in domain terms.
- Goals refined into adequate sub-goals
- Choice of subgoals: necessary? sufficient?


Minimize Credit Exposure Risk
Maximize Return
Predict Risk
Adjust Risk Exposure Factors
Track Risk Prediction History
Improve Predictive Model Over Time
….
…..

- Unambiguous definition of domain terms
  - Object types, attributes, events
- Terms defined in "drill-down" as needed



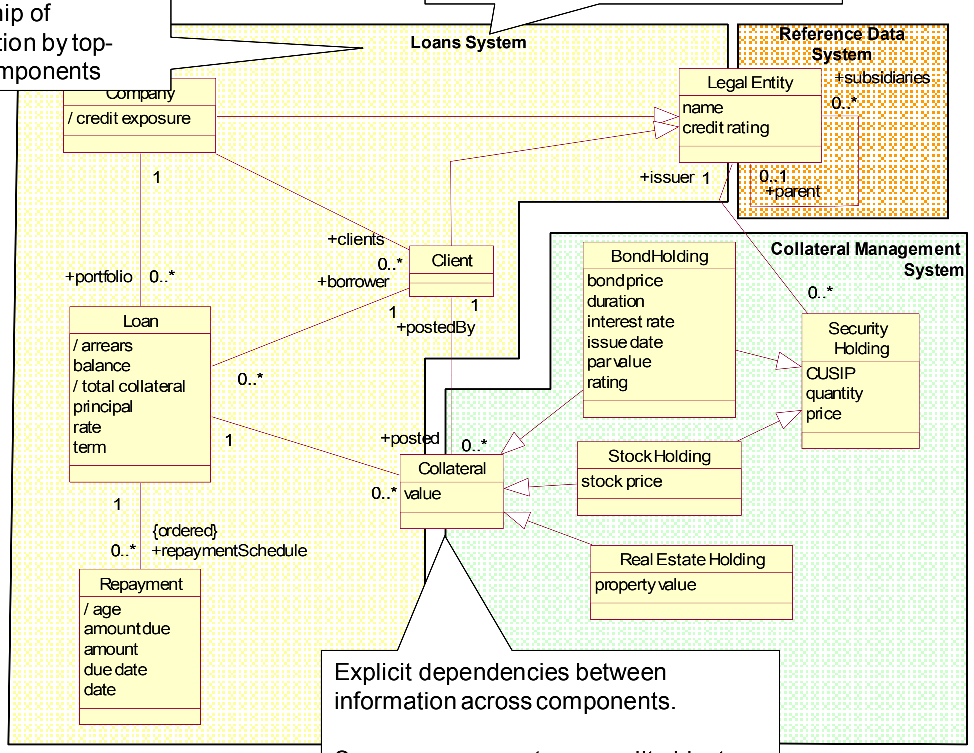

Process: take out and repay a loan
Goal: customer has drawn down loan amount and paid off all interest and principal, retaining posted collateral
Customer
Company Loan Manager
Company Risk Manager
Company Collateral Manager
request loan
proposed
check borrower creditrating
check exposure to borrower
agree principal, rate & term
agree on collateral
stipulate collateral
post collateral
activate loan
active
draw down

- Process activities clearly defined in domain terms
- Business process demonstrably supports goals

Example – Black-box partition & mapping of Domain Model

Clear authoritative ownership of information by top-level components

Top level "black-box" components

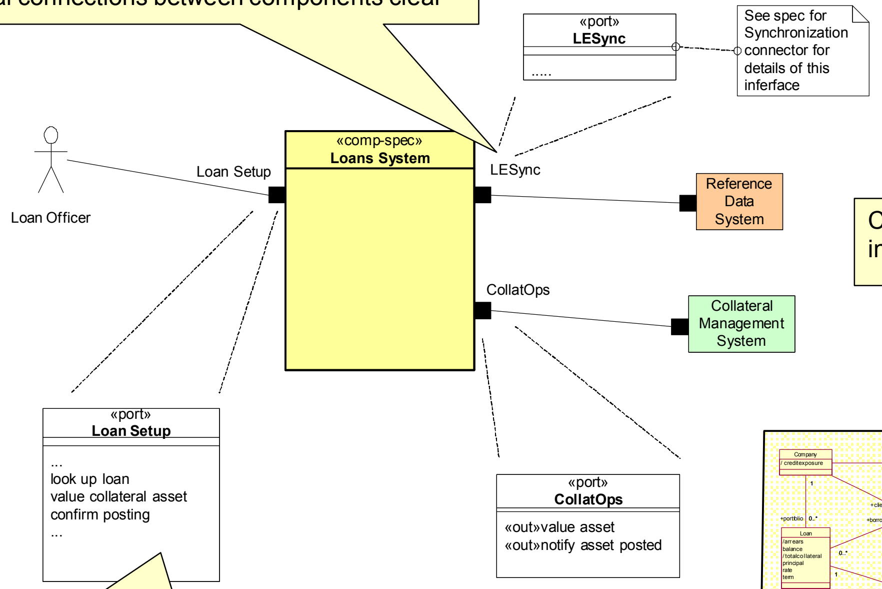


Explicit dependencies between information across components.

Some <<cross-system>> split objects

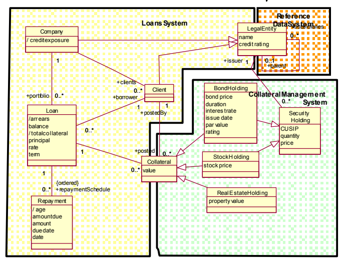
Example – Black Box Assembly

- Ports ■ define connection points on components
- Services provided and required explicit
- Logical connections between components clear



- Interfaces specified explicitly

Clear underlying information ownership



Multiple Concerns, Styles, across Viewpoints

- Functionality, security, location, technology … specified at suitable granularity and level

uild-time structure of software units including sources and tools

Location: deployment, location & connectivity: people, hardware, software

Technical: technical and platform realizations of components, ports, connectors

Performance: throughput, response time, latency, data volumes, loads

Security: who can and cannot do what and to whom

Logical Functionality: what actions with what information and what triggers

Concerns

Viewpoint

1. Business Architecture	Business perspective on the problem or concern, including goals, activities, interactions, and information.
2. component architecture – Black Box	Describe a large grained software component as a black box, with its behavior and information responsibility to others in its environment, for any aspect that is externally visible.
3. component architecture - White Box	Design the "insides" of a black box component as a collection of connected sub-components, each a black-box, and decide how they collaborate to fulfill black box responsibilities.

Outside

Boundary

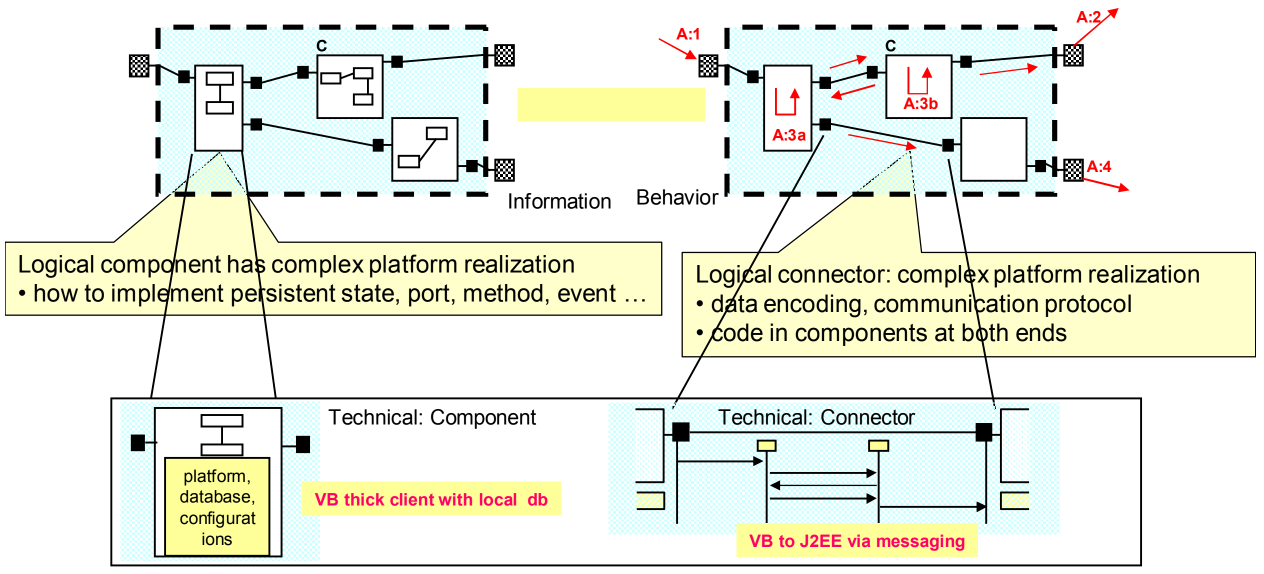
Inside

Architecture Styles: Principles, Patterns, Guidelines, Rules

Address all relevant concerns early in the process and in the appropriate viewpoint

Technical Architecture
– Complex “Overlay” on Components + Connectors at any Level
– 1st Class Connectors

White-Box: Logical Components and Connectors



Logical component has complex platform realization
• how to implement persistent state, port, method, event …

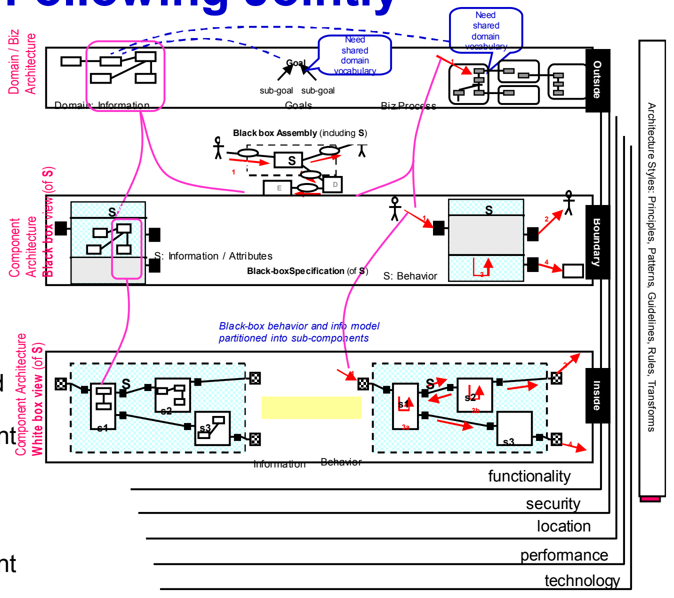
Logical connector: complex platform realization
• data encoding, communication protocol
• code in components at both ends

White-Box: Technical Realization of Components and Connectors

Exercise – Discuss The Following Jointly

Where would these fit in the formal MAp framework centered around “Entitlement Review” (“ER”)?

- ♦ A worker has a job and many roles, each requiring some system access
- ♦ Entitlement to any resource is decided by a workers job, role, the corresponding systems’ access required
- ♦ No worker should be able to access a resource unless entitled to that access
- ♦ The Entitlement Review process finds inappropriate entitlements and fixes them
- ♦ The ER component owns all current Entitlement information
- ♦ The HR component owns all job and role-related information
- ♦ The Provisioning and De-Provisioning component owns review rules
- ♦ The ER component provides services to assist entitlement managers perform their entitlement reviews
- ♦ The Provisioning and De-Provisioning component provides services to change entitlements
- ♦ Entitlement change requests from ER to Provisioning and De-Provisioning are communicated using MQ-Series
- ♦ The ER component is realized by a collaboration between a Review-Scheduler, Rules-Checker, and Violations-Handler components


Domain / Biz Architecture
Need shared domain vocabulary
Need shared domain vocabulary
Goal
sub-goal
sub-goal
Domain: Information
Goals
Biz Process
Outside
Black box Assembly (including S)
S
Component Architecture Black box view (of S)
S
S: Information / Attributes
Black-boxSpecification (of S)
S: Behavior
S
Boundary
Black-box behavior and info model partitioned into sub-components
Component Architecture White box view (of S)
S
s1
s2
s3
S
s1
s2
s3
Information
Behavior
Inside
functionality
security
location
performance
technology
Architecture Styles: Principles, Patterns, Guidelines, Rules, Transforms

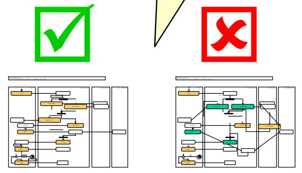
Outline

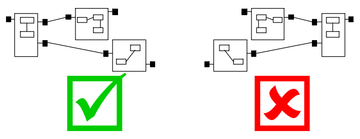
- What is MAp?
- **Formal Architecture Framework**
  - Goal Modeling
  - Viewpoints and Concerns – Black Box, White Box, Technical
  - **Architecture Style**
- Process Pragmatics
  - Iterative and Incremental
  - Roadmap Route: Problem Definition, Current State, Target State, Migration Plan
  - Flexible Models and Domains
- Summary

Architecture Principles and Styles

• Architecture styles define rules of architecture conformance

- Architecture styles define patterns of preferred or allowed architectures
- One common form: **Wherever** you have … **solve it by** …
- Very wide spectrum of styles in (and across) all viewpoints
  - **Business Process Style**: Earliest point of data capture
    - **Wherever** any process needs some information at multiple points …
    - Capture the information at the first opportunity in process. Never require it again.
  - **Technical Architecture Style**: MQ Series Configuration Style for Co-located Queues
    - **Wherever** co-located components need a messaging connector …
    - For a source S of events and a destination D that will be co-located on a single machine, use an MQ Series queue named <S><D><…>, configured as <….>, and set up the S and D ends of the queue as <….>
  - **Technical Architecture Style**: XML encoding styles
    - Use nested entities for …; use attributes for …; compress names as follows …
  - Integration styles: point-to-point, desktop, gateway, …

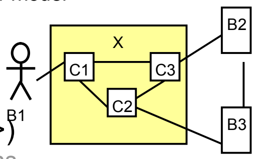




- Styles can be of more or less prescriptive and automated
  - rules-of-thumb: if high availability is needed use replication and heartbeats
  - review criteria: Component count should be constant or increase very slowly with increase of users
  - fully automated transformations: generate EJB deployment descriptors, data model from source model

Structure of "Fully Dressed" MAP Deliverables

- Styles, principles, and patterns
- Business Architecture
  - Goals model and Domain model
  - Principles and styles
  - Business process / subject area 1
    - process model + information model
  - Business / subject area 2
    - …
  - [Cross business area spec]
- Component Black Box (of <X>)
  - Styles, principles, and patterns
  - Black Box Assembly and Map
  - Black Box Spec of <X>
  - Grouped by actor, port, subject, …
    - Use case specifications
      - Ins/Outs
      - Pre and post conditions
      - Main and alternate paths
      - System operations
  - Black-box information model
    - Persistent state + input/output
  - Security, performance, location, …
  - API, UI, technology specifics



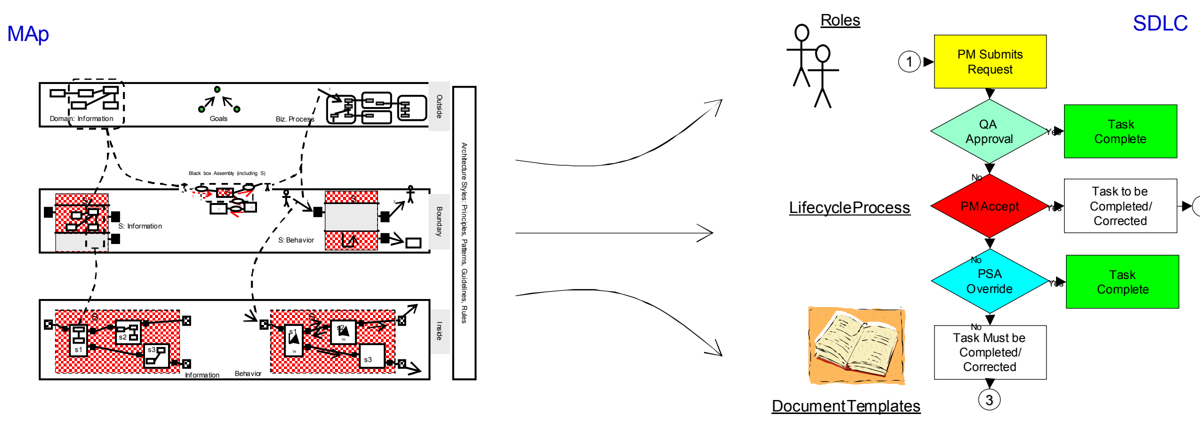
- Component White Box (of <X>)
  - Styles, principles, and patterns
  - Assembly of sub-component C1, C2, … Cn
  - For each major use case or system operation
    - Collaboration between C1 … Cn
  - Sub Component C1 – black box spec
    - Information model
    - Port 1 specification
      - Operation specifications, in, out, pre, post
    - Port 2 specification
      - Operation specifications…
  - Sub Component C2 – black box spec
- Technical Architecture (black/white box level)
  - Styles, principles, and patterns
  - Connector realizations
    - Shared connector styles – spec + realization
  - Component realizations
    - Language, platform, data base, protocols, etc.
- Deployment Architecture (black/white box level)
  - Styles, principles, and patterns
  - Location, node, network, platform configuration
  - Software deployment on to network

Documents include narrative, rationale, examples, … in addition to models

Outline

- What is MAp?
- Formal Architecture Framework
  - Goal Modeling
  - Viewpoints and Concerns – Black Box, White Box, Technical
  - Architecture Style
- **Process Pragmatics**
  - **Iterative and Incremental**
  - Roadmap Route: Problem Definition, Current State, Target State, Migration Plan
  - Flexible Models and Domains
- Summary

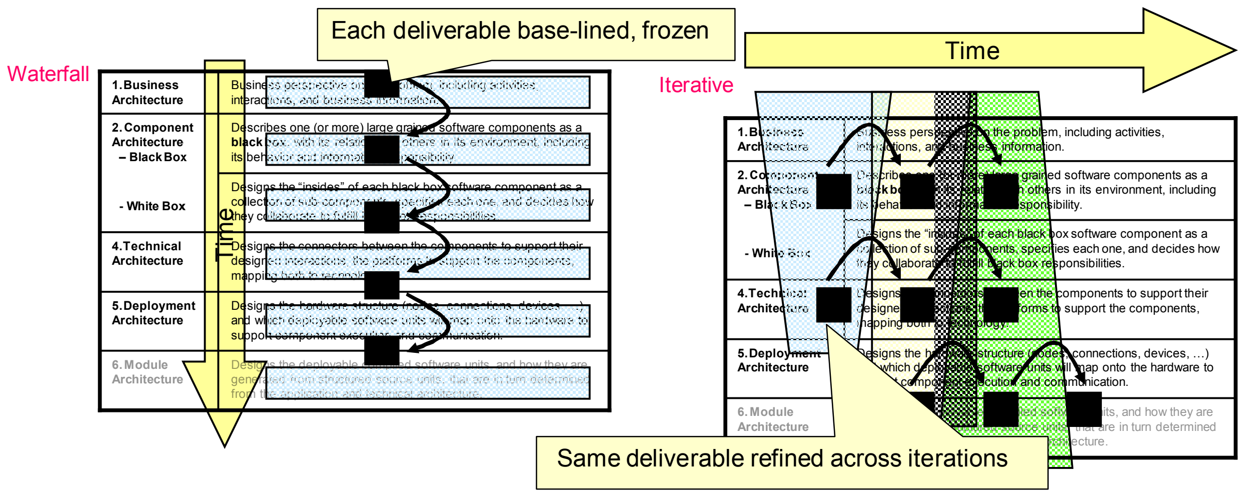
MAp can provide complementary value to an SDLC



- Adapt MAp to the SDLC structure for the organization
  - If possible follow a somewhat iterative lifecycle
- Add specific useful tools within the SDLC
  - How to frame the problem and clarify the domain terminology
  - How to derive and validate key architecture decisions
- Fill in key parts of templates e.g. Business Requirements, Functional Specs
  - Model of goals and it's refinement to architecture decisions
  - Specification of services, components, connectors
- Define relations and consistency criteria across descriptions

21

Architecture Viewpoint vs. Development Sequence



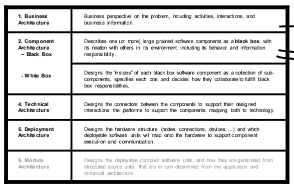
Each deliverable base-lined, frozen
Waterfall
Time
Iterative
Time
1.Business Architecture
Business perspective on the problem, including activities, interactions, and business information.
2.Component Architecture – Black Box
Describes one (or more) large grained software components as a black box, with its relationships to others in its environment, including its behavior and information responsibility.
- White Box
Designs the "insides" of each black box software component as a collection of sub-components, specifies each one, and decides how they collaborate to fulfill black box responsibilities.
4.Technical Architecture
Designs the connectors between the components to support their designed interactions, the platforms to support the components, mapping both to technology
5.Deployment Architecture
Designs the hardware structure (nodes, connections, devices, …) and which deployable software units will map onto the hardware to support component execution and communication.
6.Module Architecture
Designs the deployable packaged software units, and how they are generated from structured source units, that are in turn determined from the application and technical architecture.
Same deliverable refined across iterations

- Development need not be waterfall i.e. not 1-viewpoint at a time
- With multiple iterations, each iteration can cover multiple viewpoints
  - The same deliverables is refined across iterations
- Same models can be presented as tables, bullet lists, formal diagrams
  - Flexible form with consistent content

22

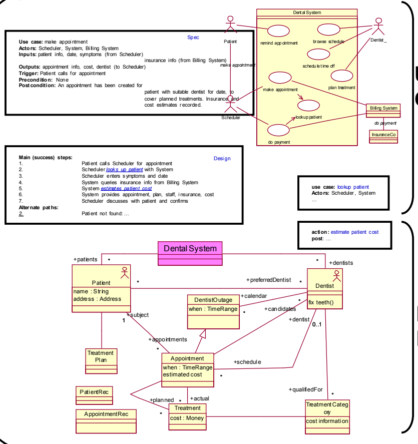
List, Draft, and Dressed Versions

Black box



1. Business Architecture	Business perspective on the problem, including activities, interactions, and business information.
2. Component Architecture – Black Box	Describes one (or more) large grained software components as a **black box**, with its relation with others in its environment, including its behavior and information responsibility.
- White Box	Designs the "insides" of each black box software component as a collection of sub-components, specifies each one, and decides how they collaborate to fulfill black box responsibilities.
4. Technical Architecture	Designs the connectors between the components to support their designed interactions, the platforms to support the components, mapping both to technology.
5. Deployment Architecture	Designs the hardware structure (nodes, connections, devices, …) and which deployable software units will map onto the hardware to support component execution and communication.
6. Module Architecture	Designs the deployable compiled software units, and how they are generated from structured source units, that are in turn determined from the application and technical architecture.

Fully Dressed *Version*



Spec

Use case: make appointment
Actors: Scheduler, System, Billing System
Inputs: patient info, date, symptoms (from Scheduler) insurance info (from Billing System)
Outputs: appointment info, cost, dentist (to Scheduler)
Trigger: Patient calls for appointment
Precondition: None
Postcondition: An appointment has been created for patient with suitable dentist for date, to cover planned treatments. Insurance and cost estimates recorded.

Design

Main (success) steps:
1. Patient calls Scheduler for appointment
2. Scheduler *looks up patient* with System
3. Scheduler enters symptoms and date
4. System queries insurance info from Billing System
5. System *estimates patient cost*
6. System provides appointment, plan, staff, insurance, cost
7. Scheduler discusses with patient and confirms

Alternate paths:
2. Patient not found: …

Use Cases

Info Model

List *Version*

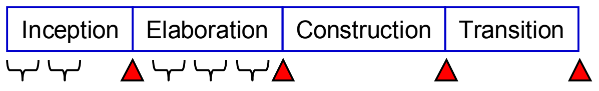
Use cases
♦ Enquire about product
♦ Place order
Information Model
♦ Customer
– need
♦ Product

Draft *Version*

Actors	Customer, Sales Agent
Use cases	***Enquire about product***: **Goal:** The system has recorded customer's need for product. Callback has been scheduled, cross-sell analysis initiated.
	Place Order: *….*
Information Model	**Customer**: a legal entity who … *need:* the set of likely future purchases **Product**: A unit of sales to customer …

- Depth, detail, and form to suit context
- Don't produce fully dressed all at once

Iterative MAP
Iterations within phases



List: Goals Domain

List: Black box White box

Initial system: List: technical

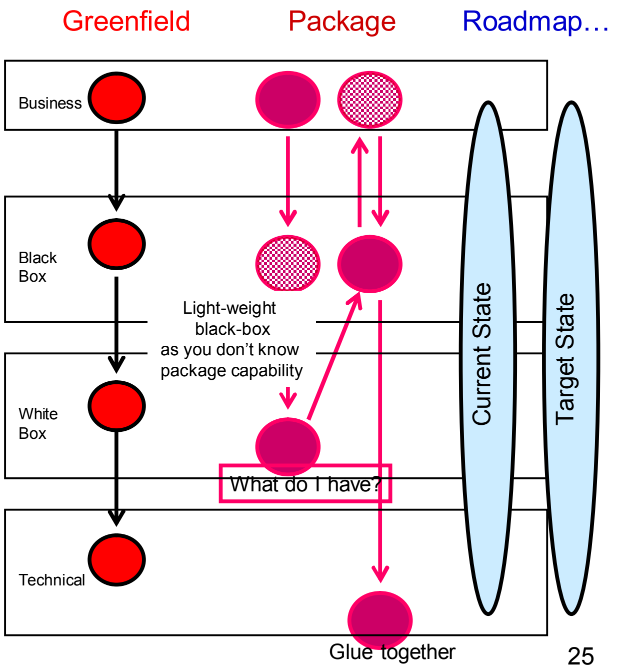
Release

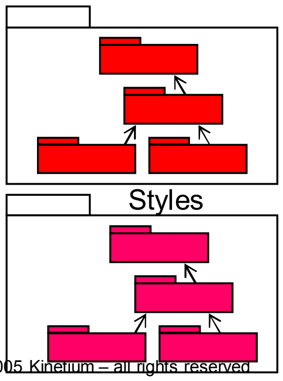
Outline

- What is MAp?
- Formal Architecture Framework
  - Goal Modeling
  - Viewpoints and Concerns – Black Box, White Box, Technical
  - Architecture Style
- **Process Pragmatics**
  - Iterative and Incremental
  - **Roadmap Route: Problem Definition, Current State, Target State, Migration Plan**
  - Flexible Models and Domains
- Summary

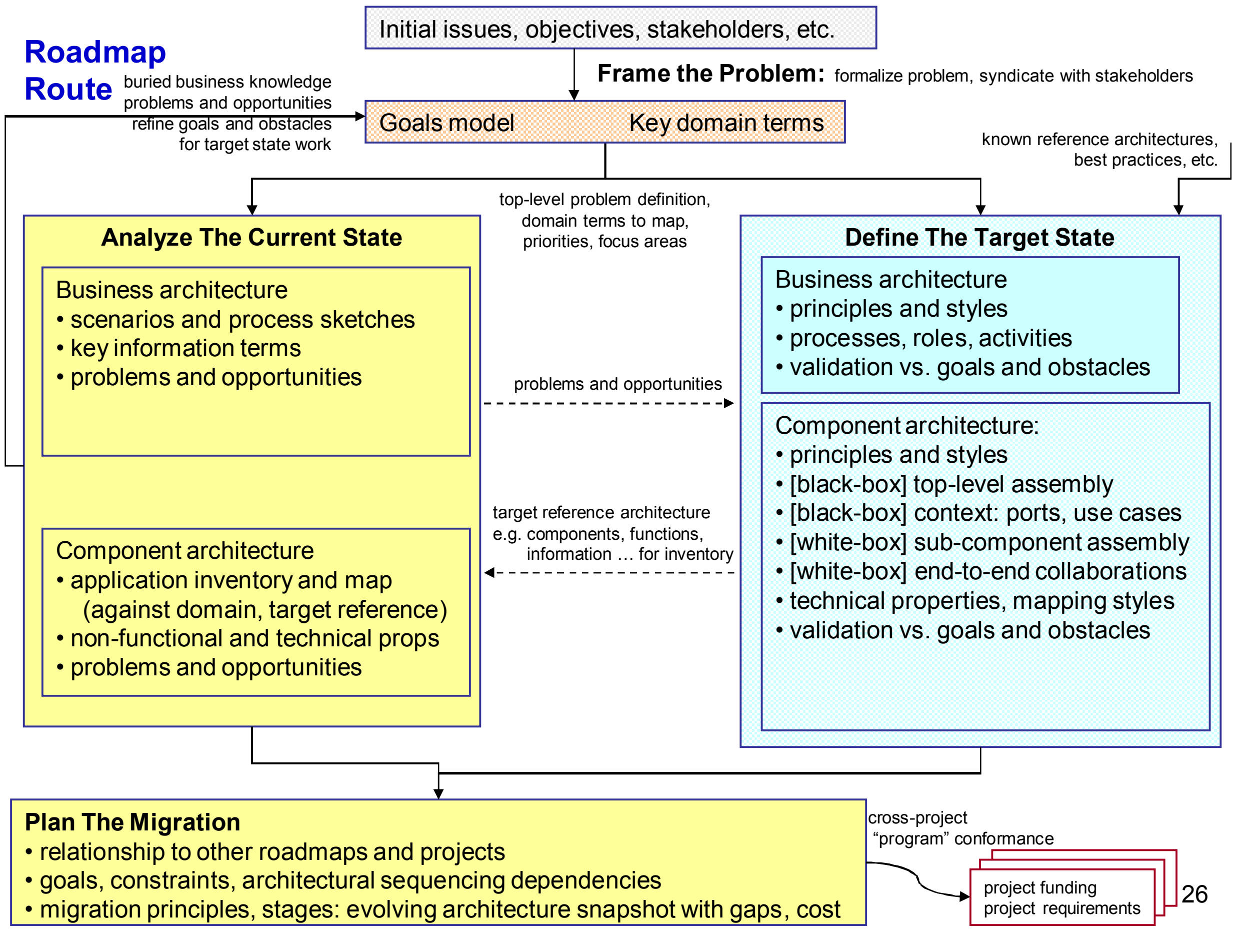
Different Project “Routes”

- Business, Black box, White box, Technical, … Architectures
  - For a project, which deliverables developed when and how far?
- Different projects, different “routes”
  - Different paths through deliverables
- Different architecture **styles**
  - … applicable for different projects





Roadmap Route



Initial issues, objectives, stakeholders, etc.

Frame the Problem: formalize problem, syndicate with stakeholders

buried business knowledge
problems and opportunities
refine goals and obstacles
for target state work

Goals model — Key domain terms

known reference architectures, best practices, etc.

top-level problem definition, domain terms to map, priorities, focus areas

Analyze The Current State

Business architecture
- scenarios and process sketches
- key information terms
- problems and opportunities

Component architecture
- application inventory and map
  (against domain, target reference)
- non-functional and technical props
- problems and opportunities

problems and opportunities

target reference architecture e.g. components, functions, information … for inventory

Define The Target State

Business architecture
- principles and styles
- processes, roles, activities
- validation vs. goals and obstacles

Component architecture:
- principles and styles
- [black-box] top-level assembly
- [black-box] context: ports, use cases
- [white-box] sub-component assembly
- [white-box] end-to-end collaborations
- technical properties, mapping styles
- validation vs. goals and obstacles

Plan The Migration
- relationship to other roadmaps and projects
- goals, constraints, architectural sequencing dependencies
- migration principles, stages: evolving architecture snapshot with gaps, cost

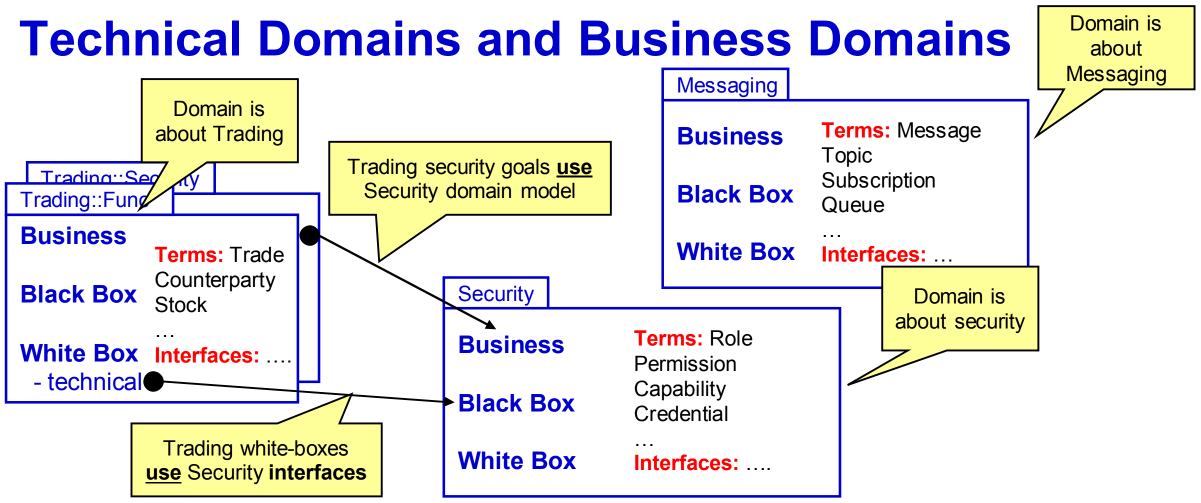
cross-project “program” conformance

project funding
project requirements

Outline

- What is MAp?
- Formal Architecture Framework
  - Goal Modeling
  - Viewpoints and Concerns – Black Box, White Box, Technical
  - Architecture Style
- **Process Pragmatics**
  - Iterative and Incremental
  - Roadmap Route: Problem Definition, Current State, Target State, Migration Plan
  - **Flexible Models and Domains**
- Summary

Technical Domains and Business Domains


Domain is about Messaging
Messaging
Business
Black Box
White Box
Terms: Message Topic Subscription Queue …
Interfaces: …
Domain is about Trading
Trading::Security
Trading::Func
Business
Black Box
White Box
- technical
Terms: Trade Counterparty Stock …
Interfaces: ….
Trading security goals use Security domain model
Security
Business
Black Box
White Box
Terms: Role Permission Capability Credential …
Interfaces: ….
Domain is about security
Trading white-boxes use Security interfaces

- Infrastructure or technical domains (e.g. Security, Messaging) have their own domain models, goals, black box, and white box models
- MAp applies to such domain in isolation, or as part of a “business” domain
- These can be developed, to some extent, in parallel with a primary business domain (e.g. Trading)
- When Trading is defined in technical or platform terms (e.g. white-box/technical), the platform is defined in part by the technical domains

MAP is …

- A consistent method to define and evolve architectures that can be used in many areas
- An approach to clarifying goals and partitioning and connecting components and applications
- Useful when used formally, on whiteboards, or just for clearer thinking: consistent content, flexible form
- A toolkit for architects, business analysts, and designers to use to clarify and improve their designs
- A way to clearly specify components for outsourced development
- A framework within which to develop and promote shared architecture principles and standards

MAP is not …

- Just about technical details
  - It addresses high-level technology-independent goals and architecture as well
- A PDLC (Project Lifecycle)
  - It prefers an iterative lifecycle, and adds architecture focus and concrete deliverables and techniques to a PDLC/SDLC
- Only applicable to business systems
  - It is also applicable to architecting technical and infrastructure domains
- An “out-of-the-box” comprehensive library of architecture styles for technical and deployment architecture

Summary and Discussion

- Model-driven approach to domain and business-aligned architecture
- Address both business and technical concerns
- Model of goals and the problem domain those goals are concerned with
- Model of components with ownership, responsibilities, and interactions
- Clear separation of viewpoints for simpler architectural descriptions
- Different entry points and routes applicable to different projects
- Can be used to differing degrees of completeness and rigor as needed

Chapter 3
Basic Modeling Techniques

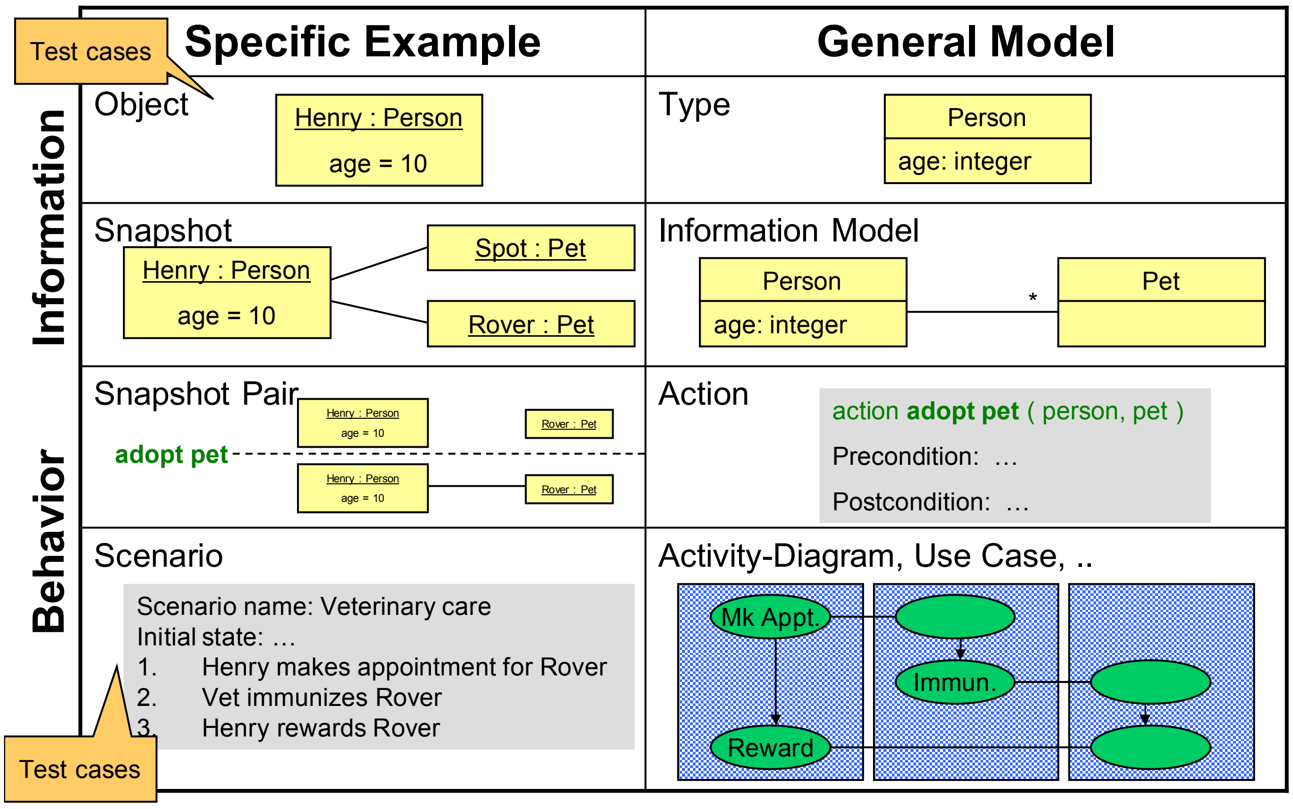
This chapter introduces basic techniques for modeling information and behavior.

Note that these are just the basic tools for our toolkit. We will apply these later in Business, Black box, and White box architecture.

It covers:

- Information Modeling
  - Objects, Types, Snapshots, Information Models
- Behavior Modeling
  - Snapshot Pairs, Actions, Scenarios, Activity Diagrams, Use Cases
- Consistency of Behavior and Information Models

Preview: Examples (Specific) and Models (General)



Copyright © 2000-2005 Kinetium

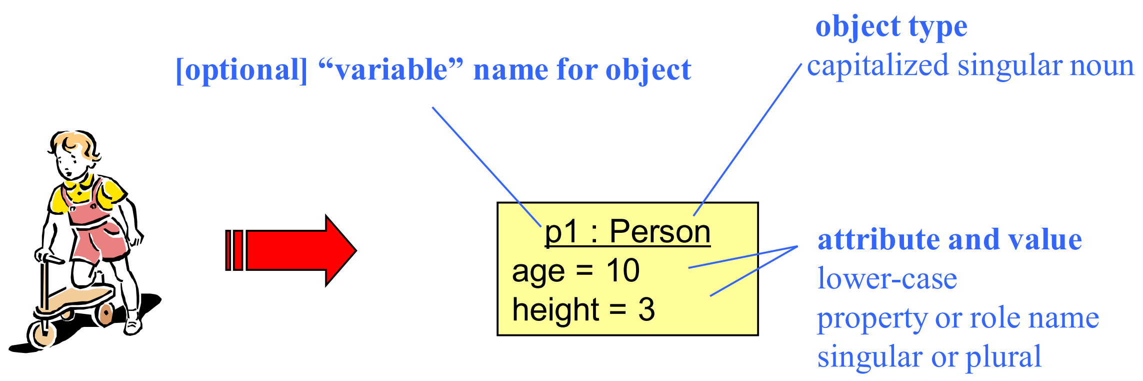
2

Objects and Types

This section covers:

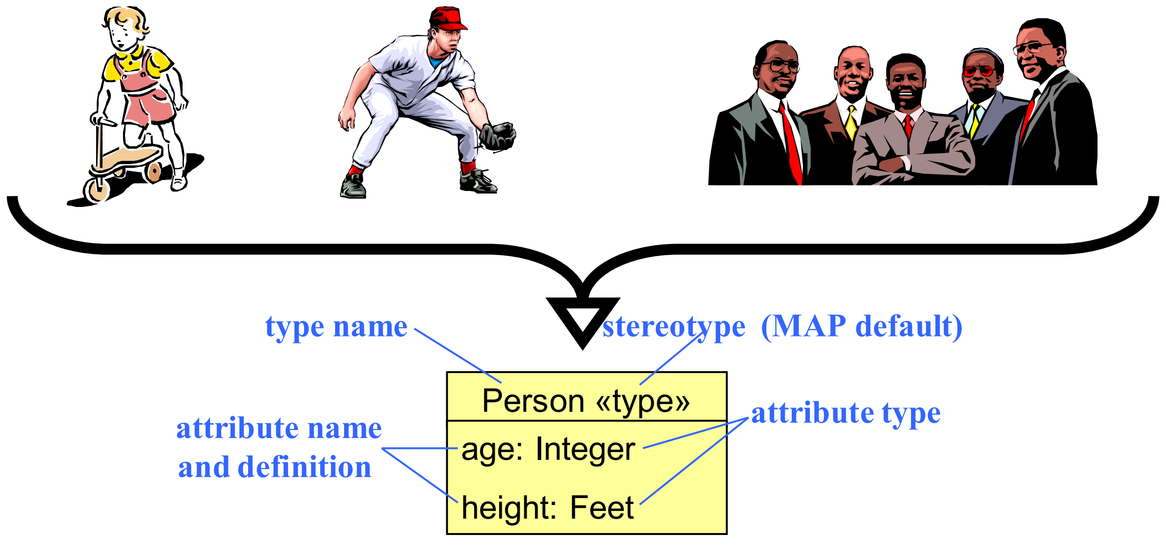
- How to model a single object
- How to model a single type

Modeling Objects



- UML notation
- Single box with underlined title
- Can refer to object by the variable name
- Can omit variable name; refer to it as **:Person** if unambiguous

Types: Generalized Objects



- Each type has a name and list of attribute definitions
  - Attributes represent property values every object of that type has
  - Attributes are used to represent structure and/or state
  - Each attribute has a type

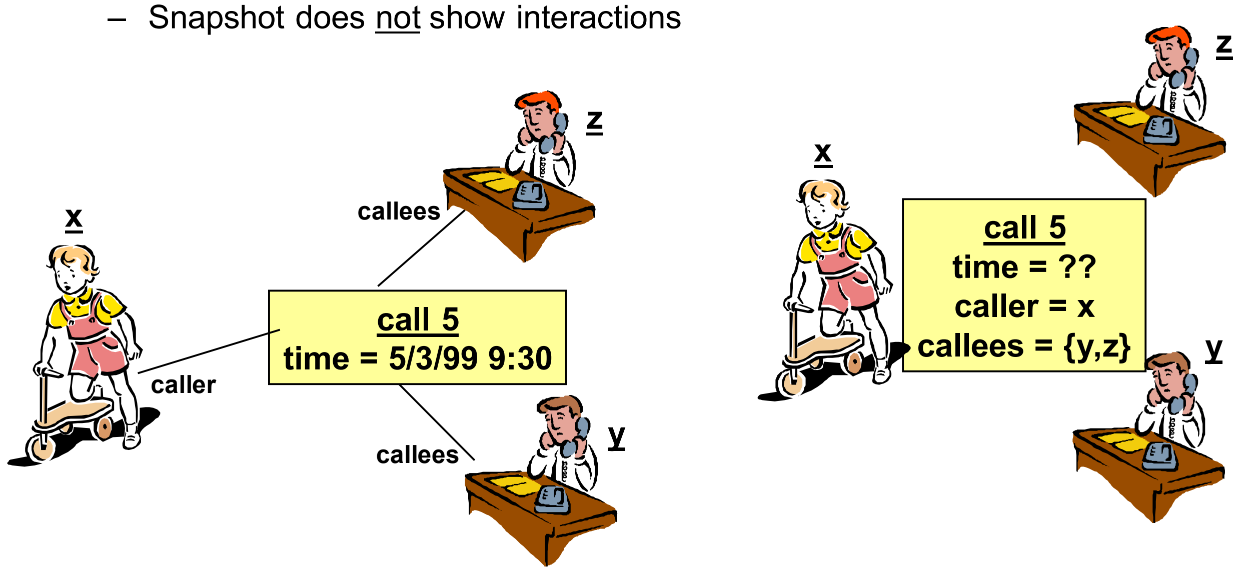
Snapshots and Information Models

This section covers:

- How to model many objects
- How to model many types
- Different ways of showing Snapshots

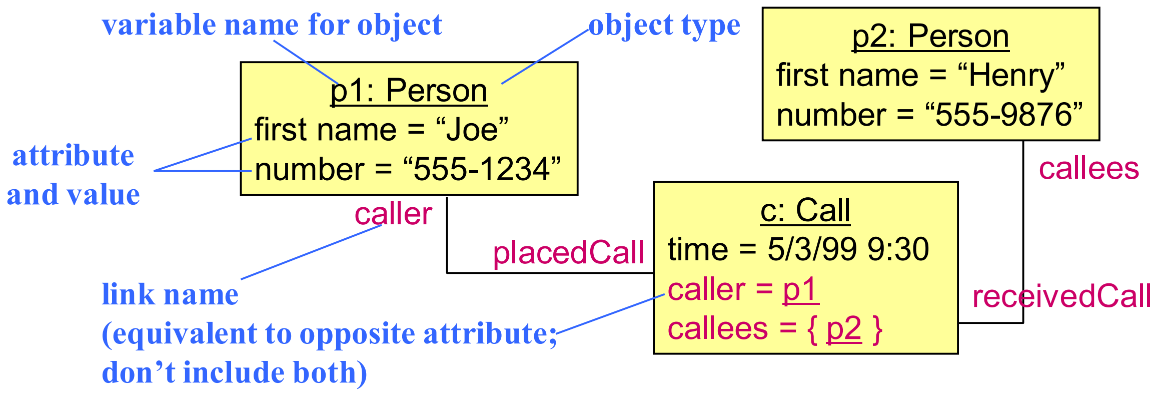
Snapshots: Linked Objects

- Snapshot: a configuration of interlinked objects at a point in time
  - A snapshot shows named links between those objects at that time
  - Attribute values can be textually written, or drawn as links
  - Snapshot shows relevant <u>state</u> of a system
  - Snapshot does <u>not</u> show interactions



How to Read a Snapshot in UML

- Uses UML "collaboration diagram" but without showing interactions



<u>p1 is a Person</u>
- its name is "Joe"
- its placedCall is <u>**c**</u>

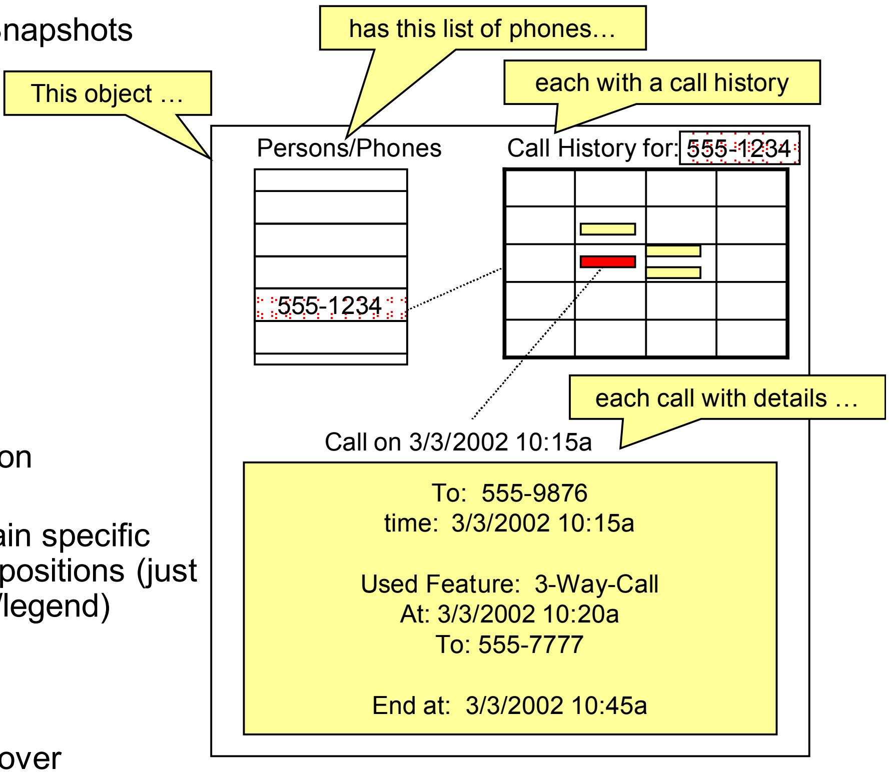
<u>**c** is a Call</u>
- its time is 5/3/99 9:30
- its caller is <u>**p1**</u>
- its callees is <u>**p2**</u>

<u>p2 is a Person</u>
- its name is "Henry"
- its receivedCall is <u>**c**</u>

- Note: Model could describe relational database or interconnected objects or a wall calendar

User-Friendly Snapshots – UIs, Graphical Notations

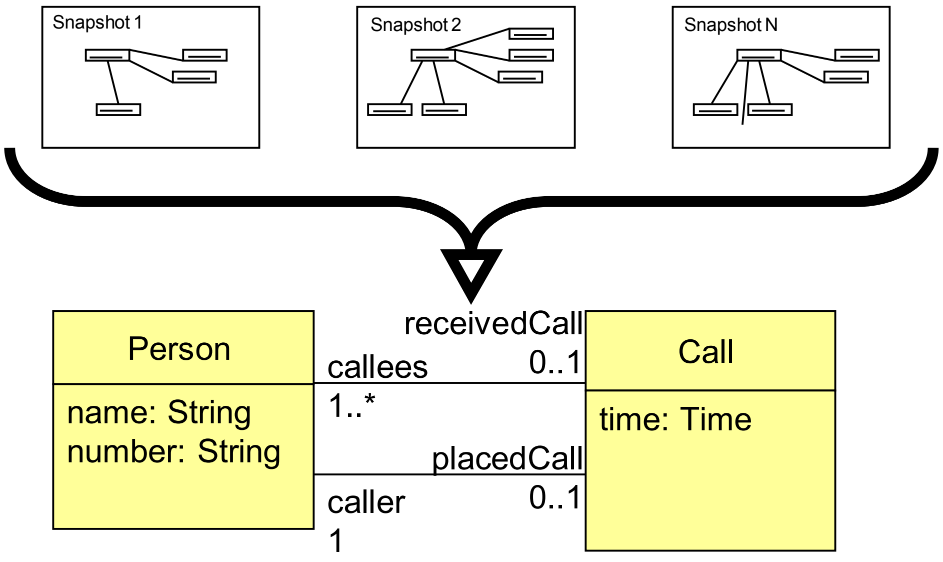
- User Interfaces show Snapshots
  - Telephone
    - Status
    - Current call
    - Available features
  - Call History
    - Duration
    - Caller / Callee
  - Call
    - Feature usage
    - Start/end times
- Useful for communication
- Alternately, use a domain specific notation, colors, icons, positions (just define the notation key/legend)
  - floor plans
  - factory flows
- Use as a means to uncover Information Model, not as an end



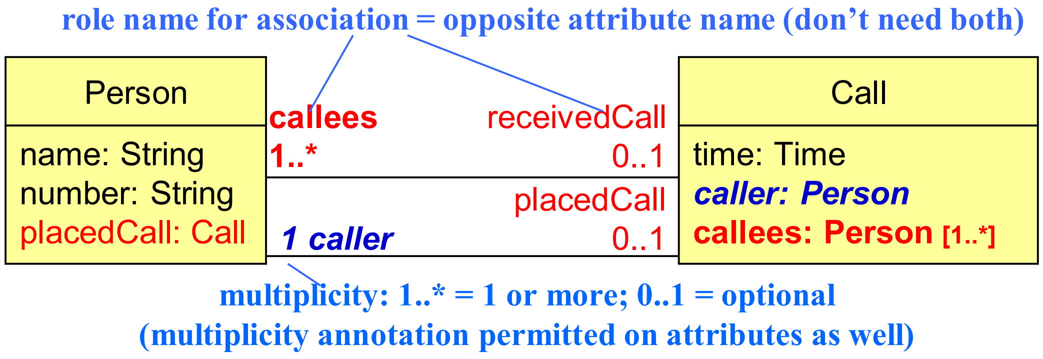
Exercise 3.1

Information Model Generalizes Snapshots

- One snapshot describes one example state / configuration
- An Information Model describes all possible snapshots
  - Also "Class Model", but emphasis on attributes and not operations



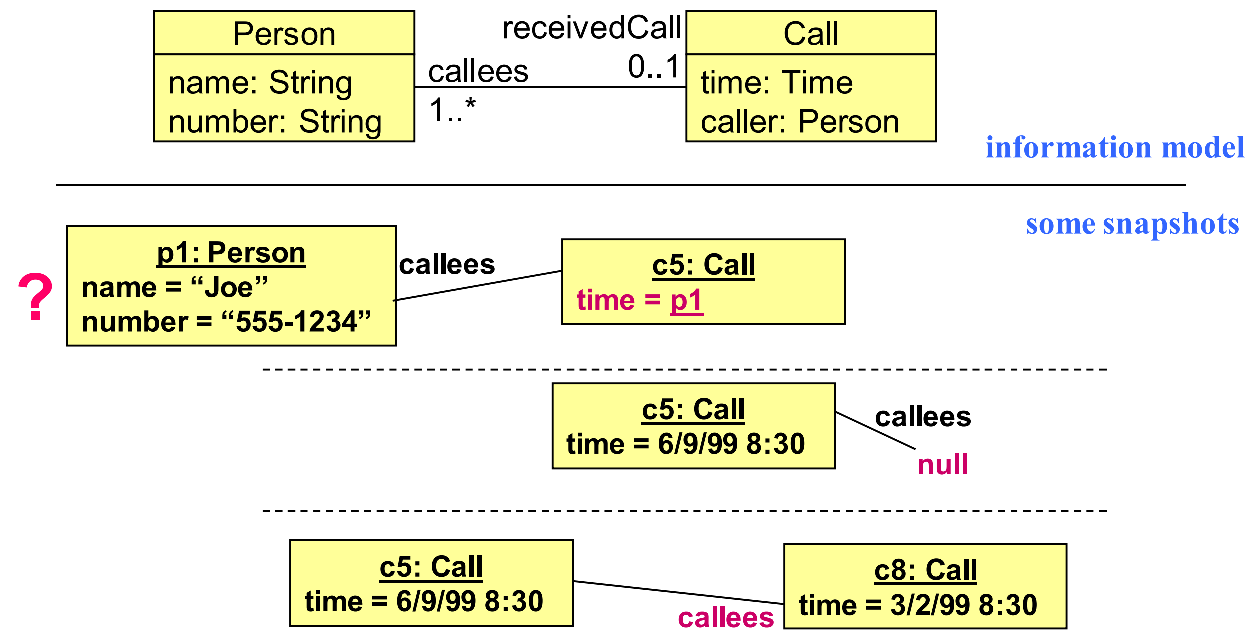
How to Read an Information Model


role name for association = opposite attribute name (don't need both)
Person
name: String
number: String
placedCall: Call
callees
1..*
receivedCall
0..1
placedCall
0..1
1 caller
Call
time: Time
caller: Person
callees: Person [1..*]
multiplicity: 1..* = 1 or more; 0..1 = optional
(multiplicity annotation permitted on attributes as well)

If association role is not named, convention is to use target type name. Almost always better to explicitly name the role similar to an attribute.

- Every person (object of type Person)
  - Has a name, which is a String
  - Has a number (phone number), which is a String
  - Optionally has a call to another person
- Every call (object of type Call)
  - Has a time: the time the call started
  - Has a caller (the person who originated the call)
  - Has one or more callees

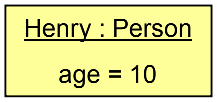
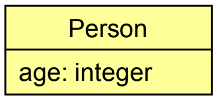
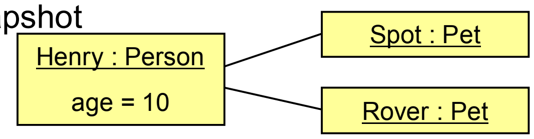
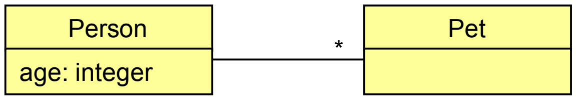
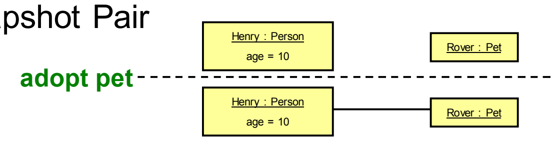
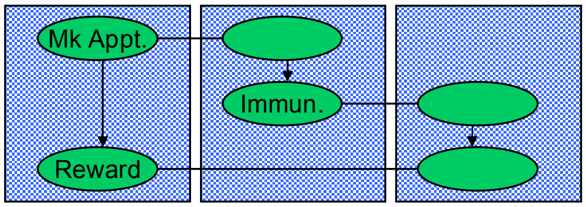
Information Model Disallows Snapshots - How?



- Snapshots can be incomplete
  - Information, attributes, links omitted are not *null* unless explicitly marked
- Which of these snapshots is invalid and why?

Exercise 3.2

Contd. Specific Examples and general Models

Specific Example	General Model
Object 	Type 
Snapshot 	Information Model 
Snapshot Pair  Henry : Person age = 10 Rover : Pet adopt pet Henry : Person age = 10 Rover : Pet	Action action **adopt pet** (person, pet) Precondition: … Postcondition: …
Scenario Scenario name: Veterinary care Initial state: … 1. Henry makes appointment for Rover 2. Vet immunizes Rover 3. Henry rewards Rover	Activity-Diagram,Use Case, .. 

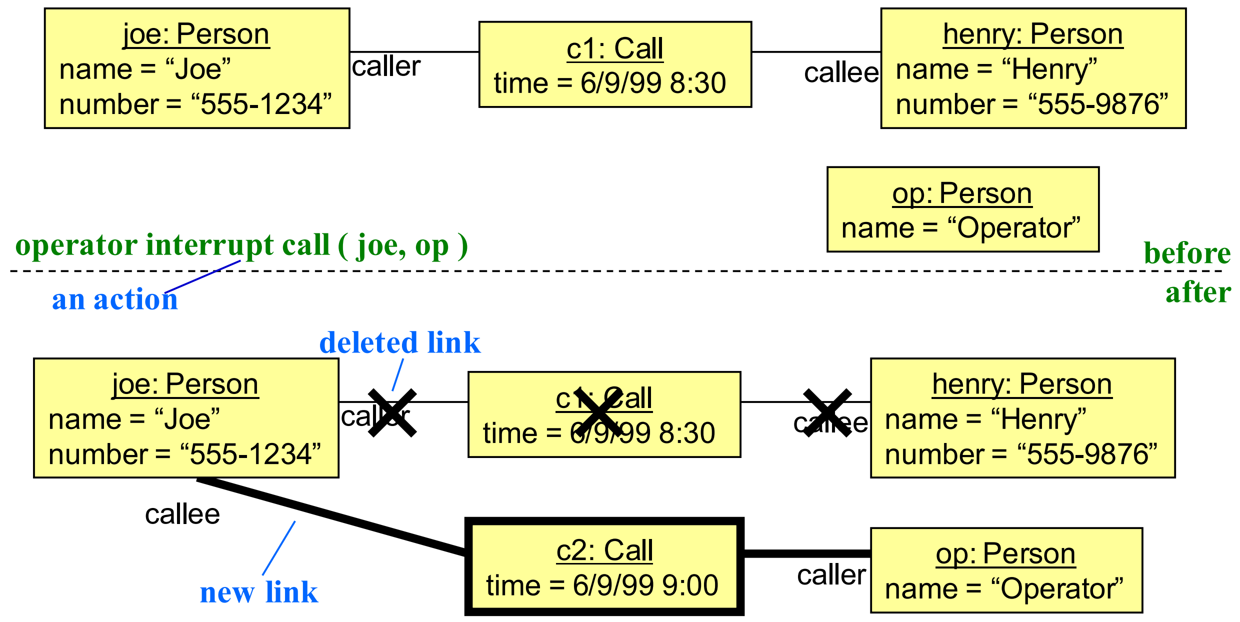
Snapshot Pairs and Actions

This section covers:

- How snapshots pairs illustrate actions
- Generalizing snapshot pairs into an Action Specification
- Formalizing an Action Specification using the Information Model

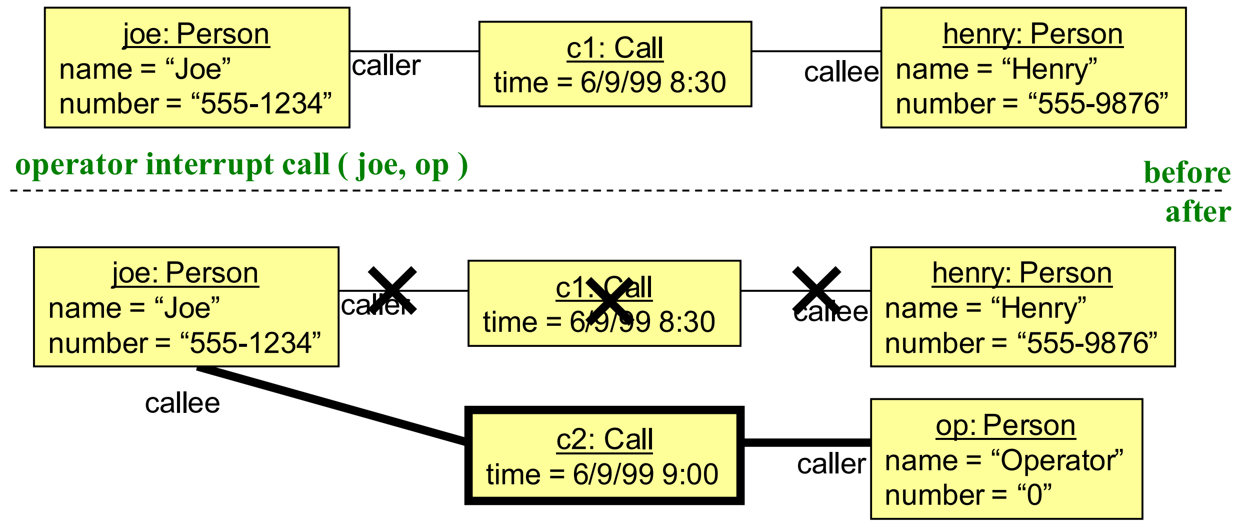
Actions Correspond to Snapshot Changes

- Operator interrupts call between Joe and Henry
  - This is an “action” (strictly, one particular “action occurrence”)



- Convention: simply overlay after “frame” with color change for before → after

How to Read Snapshot Pairs

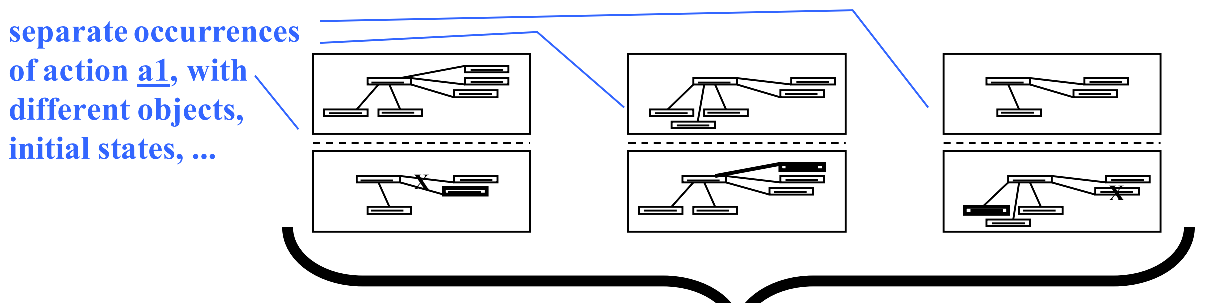


- Before: person joe had a call c1 to person henry
- A **operator interrupt call** happens op interrupting joe
- After: c1 is no more; new call c2 exists with caller op (the operator) and callee joe
- Underlined terms are names of objects (or name of attribute that refers to object)
- “Parameters” are those objects that must be identified or selected for the action
- Note: Action could describe a database update, object method, or manual procedures

Exercise 3.3

Action Specification Generalize Snapshot Pairs

- One action occurrence describes one example event
- An Action Specification describes all allowed action occurrences
  - Action Specification (also called Action Type, analogous to Object Type)


separate occurrences of action a1, with different objects, initial states, ...
X
X

action name

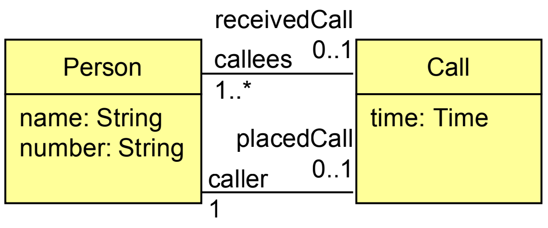
action parameters: the minimal set of objects that must be identified

action specification

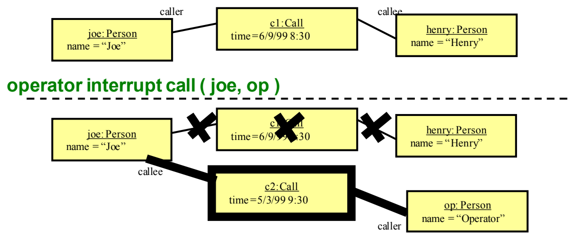
action a1 (t: T, r: R, s: S)
description narrative description of the action
precondition condition that, if true in a "before" snapshot …
postcondition … will result in the "after" snapshot in this condition

informal, or formal OCL (Object Constraint Language)

How to write an Action Spec



- Keep Info Model and snapshot pair handy
- Write an informal post condition
- Identify fewest "parameters" that must be selected from the before snapshot; all other required information can be determined from snapshot
- Formalize the postcondition from parameters and attributes, referring to snapshot and model
- Write a precondition if any



action operator interrupt call (p1 :Person, op: Person)	
description	the telephone operator interrupts a call that a person is on, resulting in the original call being dropped
precondition	-- p1 is caller on a **call**
postcondition	-- the original **call** is dropped (***p1**'s **placedCall** is dropped)*
	-- there is a new **call** with caller op and callee p1
	-- the time of the new **call** is **now**

Underlining is reference to an object, via parameter name, attribute name, etc.

Exercise 3.4

Scenarios and Behavior Specification

This section covers:

- Scenarios, which are a story of a specific sequence of actions
- Activity Diagrams, Use Cases, etc. can specify allowed action sequences

Scenarios : Text Form

- A scenario is a story of a specific sequence of interactions from an initial state
  - Initial state introduces names for objects; first step is usually trigger for the scenario
  - Specific names preferred (call5, Joe); general names (call, person) OK
  - Sequence of actions refers to those names and to new concrete names
  - Ends at a specific final state; intermediate states can be shown if helpful
  - Selected scenarios are documented and maintained; others are throw-away

Use scenarios to elicit models from examples, counter examples, to validate models, …

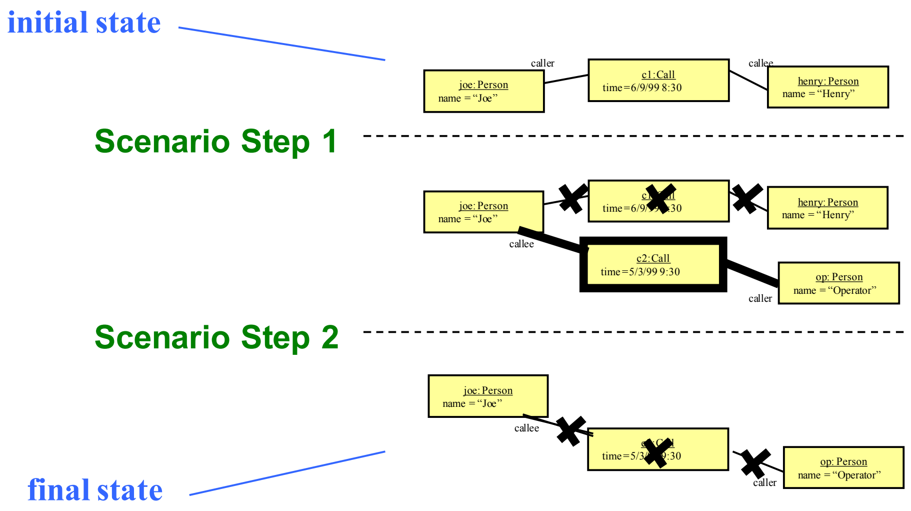
Steps in scenario should generally be written in style: **object** does **action** [to with from…] **object, attributes**

Scenario name: Joe gets emergency message while calling Henry and Bob
Initial state: Joe, Henry, and Bob have telephone service and their bills are paid.
1: Joe initiates a telephone call to Henry, who answers.
2: Bob talks on call-waiting to Henry.
3: Henry ends the call-waiting with Bob.
4: The operator interrupts the call and gives Joe an emergency message.
Final state: Joe and the operator have an active call; Henry and Bob don't.

Note: scenario can also be shown in narrative paragraph form, UML diagrams (e.g. sequence or collaboration diagram), etc. Try to indicate specific objects, values, and states in any scenario.

Scenarios with Snapshots

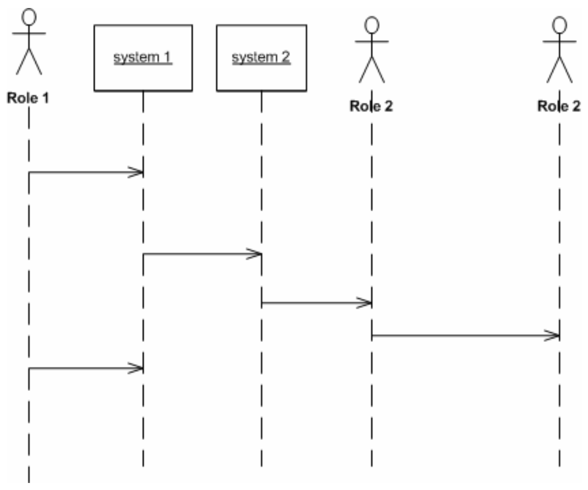
- Any scenario can be explored and understood better with snapshots



- Gives a "filmstrip" view of the changes that happen through the scenario

Scenarios : Diagram Form

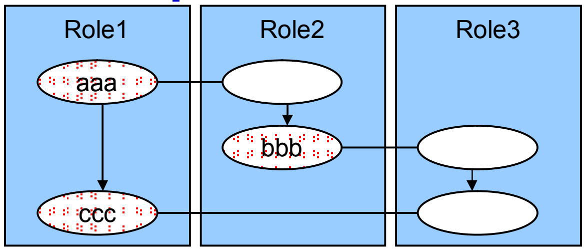
- Sequence, collaboration, and activity diagrams can also be used to show scenarios graphically



Exercise 3.5

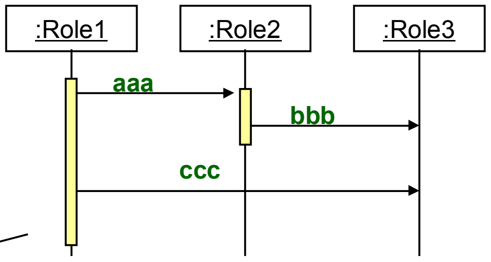
Alternatives to Specify Action Sequences

- Action specifications implicitly define action sequences by pre/post
- (Role) Activity Diagrams – RADs
  - Useful for business process modeling

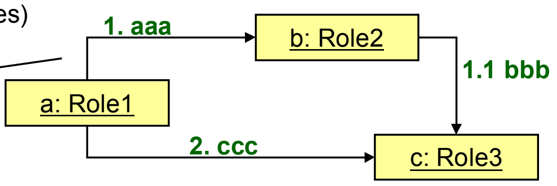


- Use cases
  - Textual and compact

use case name
actors roles that participate
pre condition before
post condition after
1. actor 1 does …to actor 2
2. actor 2 does …
3. actor 1 does …

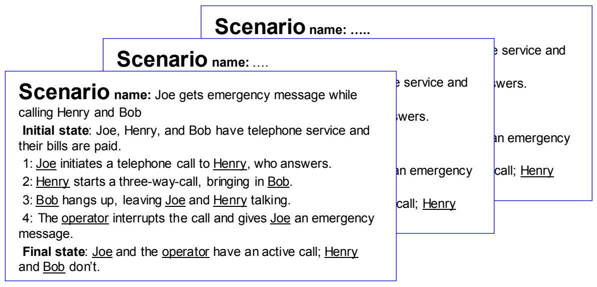


- UML Sequence Diagrams
  - Good for showing time sequence (often examples)
- UML Collaboration/Communication Diagrams
  - Good at showing 2D layout (“The print server is always on the bottom left…”)
  - Often used for examples



Role Activity Diagram Generalizes Scenarios

- A scenario describes just one example sequence of events
- An Activity Diagram describes in what order the actions in scenarios can occur
- It prohibits other sequences of actions
- Looping and parallel actions can be expressed, but may be too complex for taste



Scenario name: .....

Scenario name: ....

Scenario **name:** Joe gets emergency message while calling Henry and Bob

Initial state: Joe, Henry, and Bob have telephone service and their bills are paid.

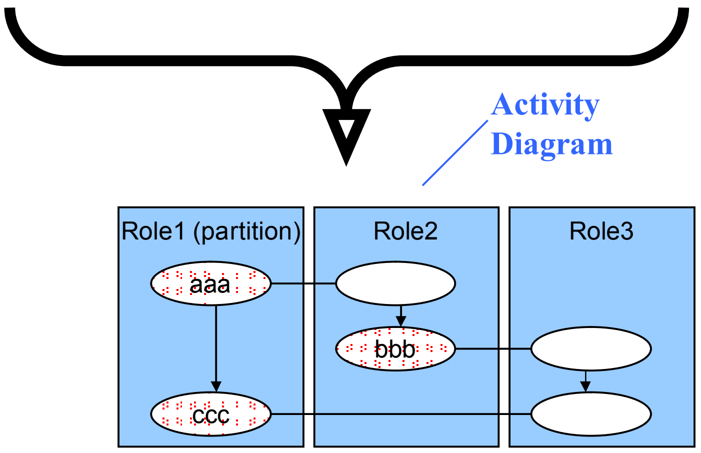
1: Joe initiates a telephone call to Henry, who answers.

2: Henry starts a three-way-call, bringing in Bob.

3: Bob hangs up, leaving Joe and Henry talking.

4: The operator interrupts the call and gives Joe an emergency message.

Final state: Joe and the operator have an active call; Henry and Bob don't.



Example Role Activity Diagram

- ***Scenario name:*** *Joe orders 3-way-calling and DSL*
- ***Initial state****: Joe is already a customer of MegaTel with an active phone line qualified for DSL.*

1: Joe calls the Customer Service Rep and orders 3WC and DSL.

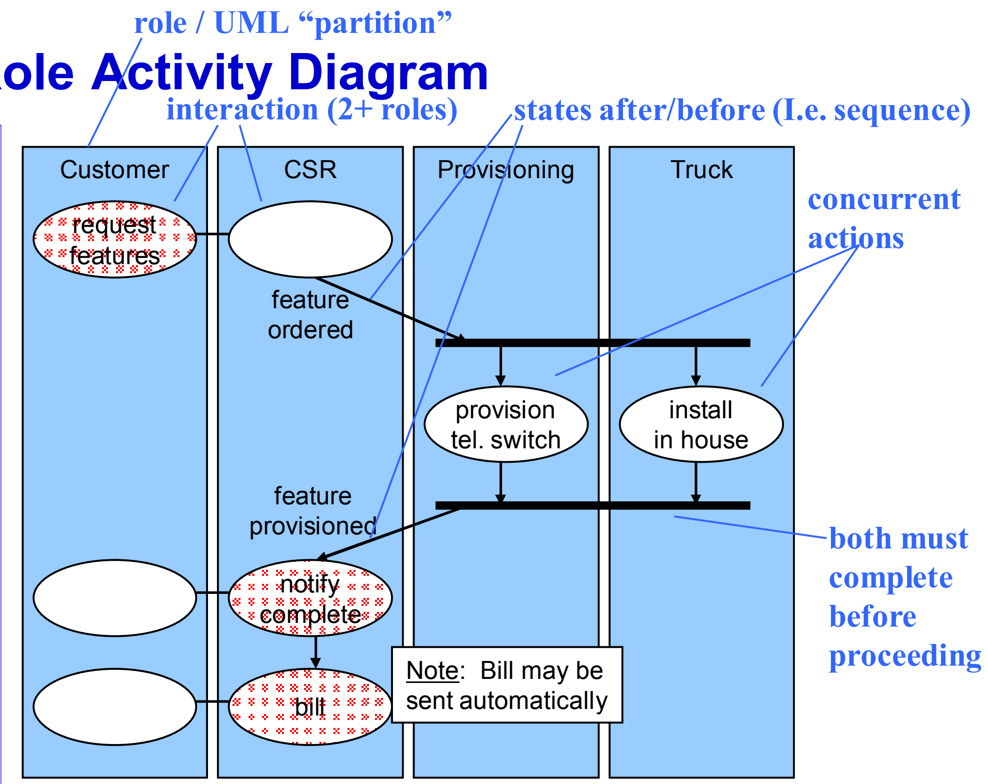
2: The switch provisioning tech activates 3WC

3: The DSL tech installs DSL at Joe's house.

4: The Customer Service Rep calls Joe to tell him his features are active.

5: MegaTel sends Joe a bill for installation.

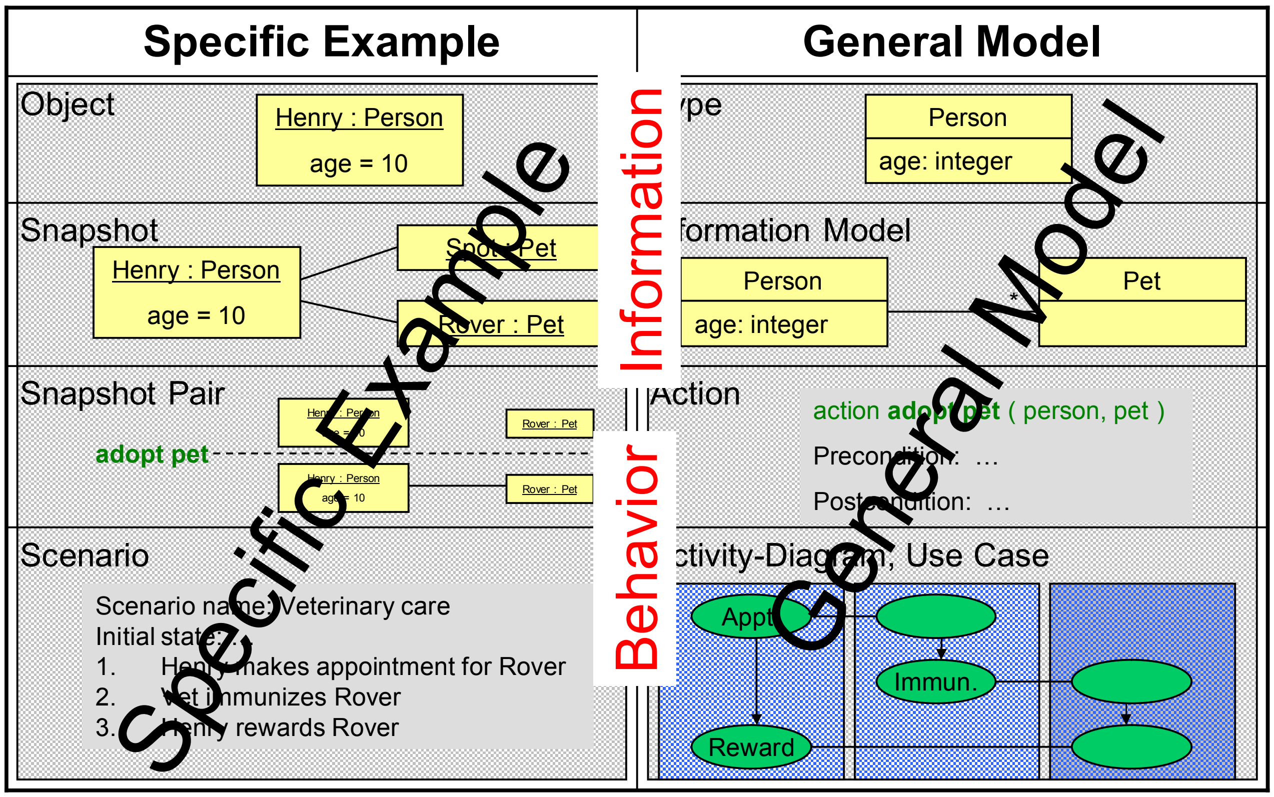
- ***Final state****: Joe has 3WC and DSL active on his phone line and MegaTel remembers to bill him each month.*



- Activity Diagrams can quickly become very complicated
  – For less formal use, looping and special cases you can just add a note

Exercise 3.6

Example vs. Model : Information and Behavior



Naming Guidelines

Model Element	Naming Guideline	Examples
Type	Singular noun	Hotel, Reservation, Product
Attribute	Role, relationship, property to read: "the <*attribute*> of <*object*>" Use plural form for sets.	**owner**, holder, age, subsidiaries e.g. aDog.**owner** (not dog.person)
State or Boolean attribute	A name which fits *object is state*	overdrawn,reserved?, is_open e.g. anAccount . overdrawn or seat . reserved?
Subtype	Qualified noun	Room Preference, Food Preference Sports Car, Family Car
Action	Verb phrase	Open account, withdraw, deposit
Event	Verb phrase, preferably present tense	Account becomes overdrawn, Reservation date has arrived
Process	Verb phrase (one-time or ongoing)	Fulfill An Order, Maintain Inventory Levels

Add a textual definition of each concept or term with
• the same grammatical form as the term i.e. a verb phrase for a verb, a singular noun phrase for a noun
• the common genus class to which the concept belongs plus its distinguishing characteristics

Summary

This chapter showed information and behavior in the specific and general

- Information
  - Objects are and snapshots are specific examples of information state
  - Types and Information Models are general specifications of information
- Behavior
  - Snapshot pairs and scenarios are specific examples of behaviors
  - Action specs, activity diagrams, use cases... are general specifications of behavior
- Behavior Models refer to corresponding Information Models

Chapter 4
Integrated Information and Behavior Modeling

How to use the basic modeling techniques in concert to create precise information and behavior models that are consistent with each other.

Note that we are still building the basic tools in our toolkit. Subsequent chapters will apply these techniques at the business, black box, white box levels.

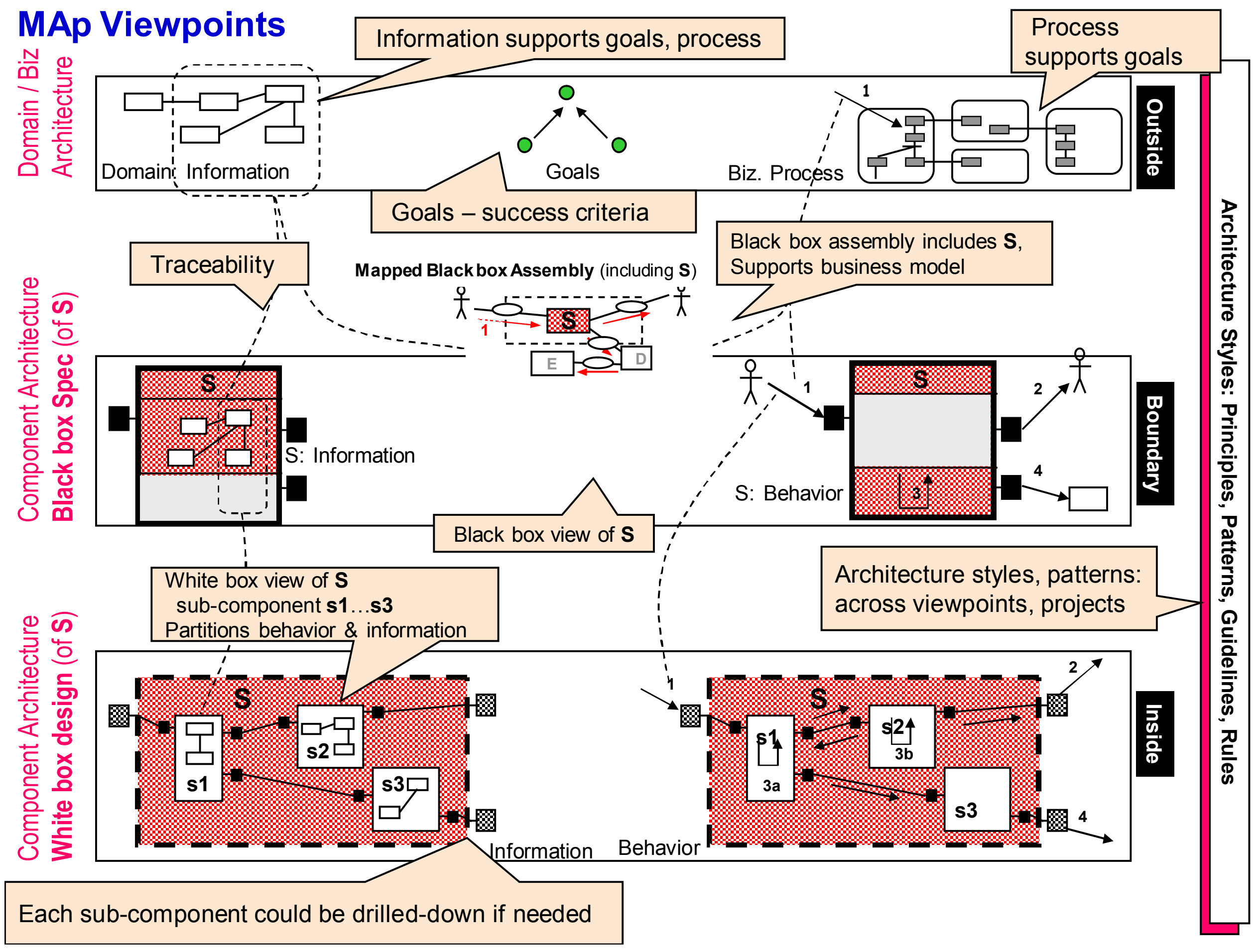
It covers:

- ♦ Integrating behavior and information models
- ♦ Modeling real problems
- ♦ More detail on Types, Scenarios, and Actions

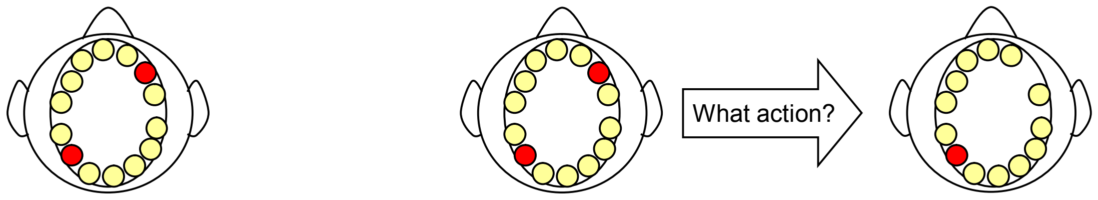
The Dentist Of The Future

- ♦ We will introduce a small example of a Dental Practice
  - – Practice has many dentists and many patients
  - – Patients make appointments, visit, get treated, and pay
  - – Dental Practice manages dentists schedule
- ♦ But this Dental Practice and its use of IT is different …
  - – Track treatment history
  - – Proactively plan treatment prior to appointments
  - – Match dentists to treatment needs
  - – Give cost and insurance information before the fact
  - – …

MAp Viewpoints


Information supports goals, process
Process supports goals
Domain / Biz Architecture
Domain: Information
Goals
Biz. Process
Goals – success criteria
Outside
Traceability
Mapped Black box Assembly (including S)
Black box assembly includes S, Supports business model
Component Architecture Black box Spec (of S)
S: Information
S: Behavior
Black box view of S
Boundary
Architecture Styles: Principles, Patterns, Guidelines, Rules
White box view of S sub-component s1…s3 Partitions behavior & information
Architecture styles, patterns: across viewpoints, projects
Component Architecture White box design (of S)
Information
Behavior
Inside
Each sub-component could be drilled-down if needed

Information and Behavior Models



- Every problem domain has some structure of objects and information
- Dental Practice
  - Patient, Tooth, Treatment, Appointment, Dentist
- Knowledge about the domain is hidden in the terminology and rules
- We build an Information Model to clarify the terminology and relationship between the concepts in that domain or business
- The Information Model describes the structure of the problem domain
  - The state of the world at any point in time

- Most domains also have important dynamics or behaviors
  - The “change” aspects i.e. what actions cause it to go from one state to another
- Dental Practice
  - Make appointment, visit, get tooth extracted, pay, …
- We build a Behavior Model to describe the actions that take place in that domain, the relationships between those actions and the Information Model, and between actions and other actions
- The Behavior Model describes the dynamics of the problem domain
  - Actions / interactions that are possible

Information Model supports Behaviors, Constraints

This slide shows how the information model is built incrementally while understanding behaviors and constraints

Behaviors, constraints, or goals to be captured

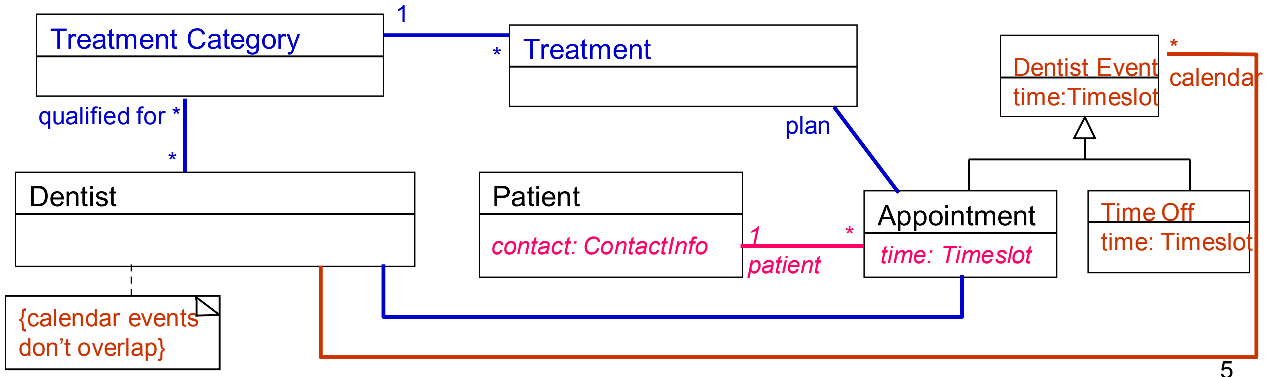
One day before the appointment, the system should contact the patient with a reminder.

Based on the treatment planned for an appointment, different dentists (qualified for different treatment categories) may be assigned to that appointment

A dentist should not be assigned an appointment at a time when he/she has personal time off scheduled

Information enhanced to capture behaviors

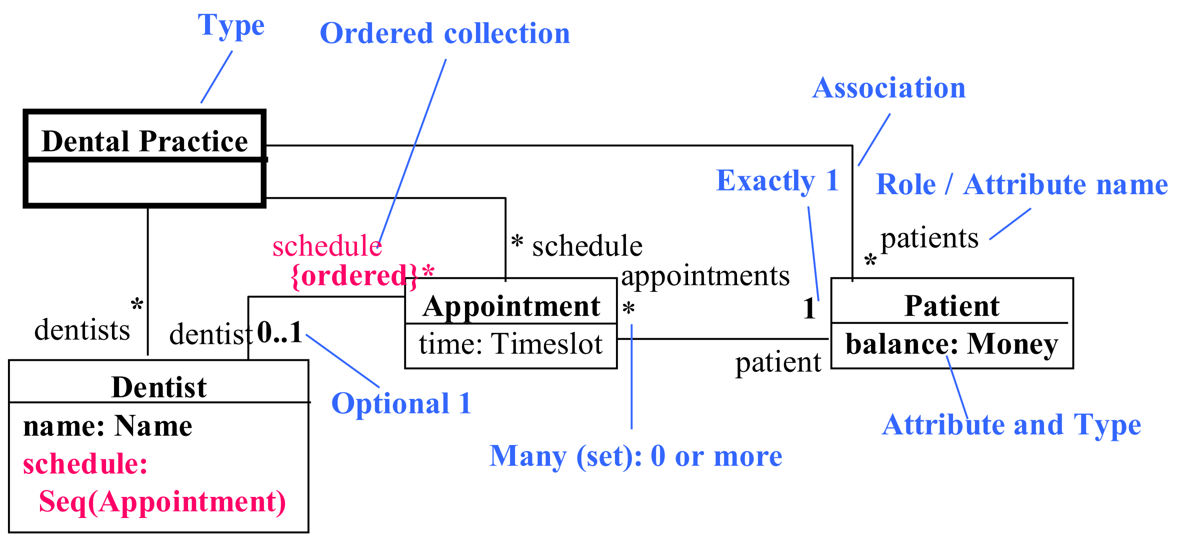
- **Iterate across behavior and information**
- **From action find objects and attributes it affects**
- **From objects, find other actions that affect it**
- **Validate with snapshots and scenarios**



Information Model: More Details

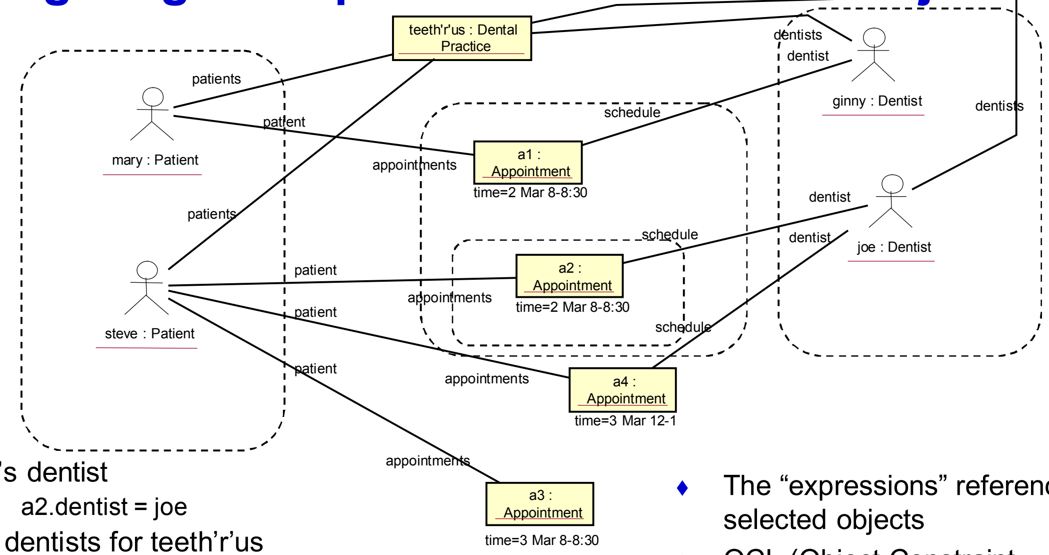
- Navigating
- Formalizing
- Invariants
- Parameterized Attributes
- Subtypes

Information Model (adding "top-level" Object)



- Always include the top-level object
  - It gives a place to define top-level attributes: clientele, dentists, etc.
- Treat associations as equivalent to (single, set, or sequence) attributes

Navigating a Snapshot: How to refer to objects



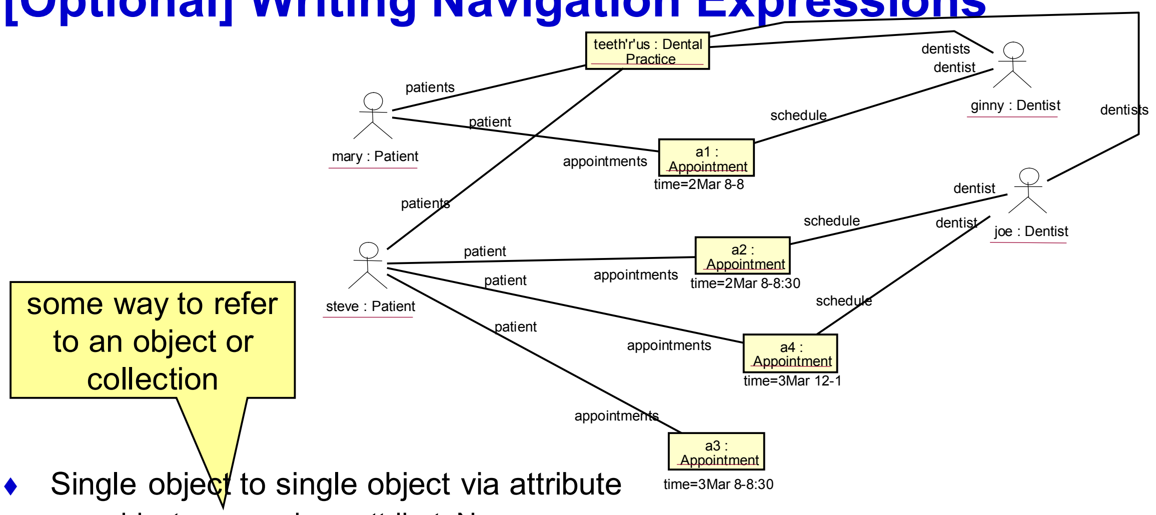
- a2's dentist
  - a2.dentist = joe
- All dentists for teeth'r'us
  - Teeth'r'us.dentists = { ginny, joe }
- All patients for teeth'r'us
  - Teeth'r'us.patients = { mary, steve }
- All appointments with any dentist at teeth'r'us for 2March
  - Teeth'r'us.dentists.schedule→select(time=2Mar) = { a1, a2 }

  Full form: **set_expression → select (iterator | condition_on_iterator)**
- All appointments for joe on 2Mar: joe.schedule→select(a | a.time=2Mar) = { a2 }

- The "expressions" reference selected objects
- OCL (Object Constraint Language): a UML standard

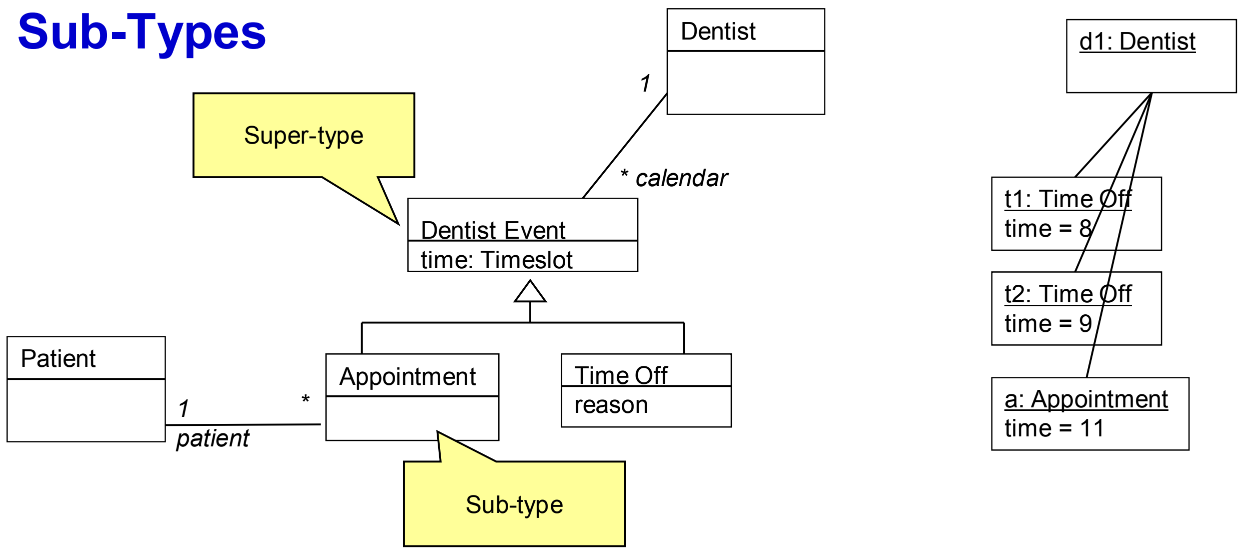
In this course we use OCL to learn clarity in models, although many projects will not use it much in practice.

[Optional] Writing Navigation Expressions



- Single object to single object via attribute
  - object_expression . attributeName
- Single object to set of objects via attribute
  - object_expression . setAttributeName or object_expression . manyAssociationName
- Set of objects to corresponding set of their attributes values
  - object_set_expression . nameOfAttributeOfEach
- Set of objects to some property of the set itself (set size, a subset, etc.)
  - object_set_expression → size
  - object_set_expression → select (condition) or → select (iterator | condition)

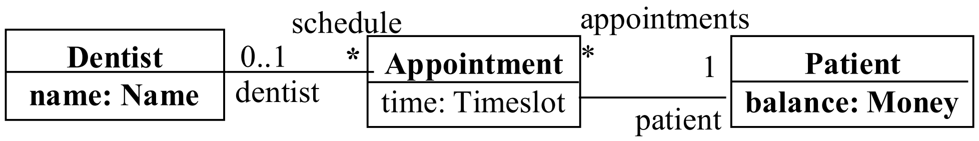
Sub-Types



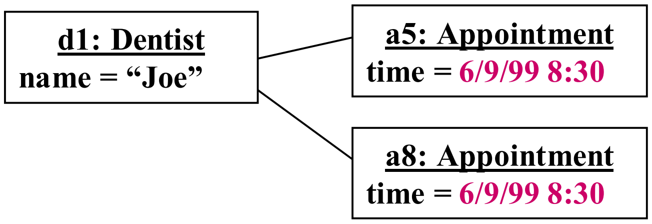
- Super-type defines common attributes and associations
- Sub-type automatically “inherits” those properties
- Sub-type can add its own attributes and associations
- Sub- and super-types do **not** show up as separate objects on snapshots

Exercise 4.1

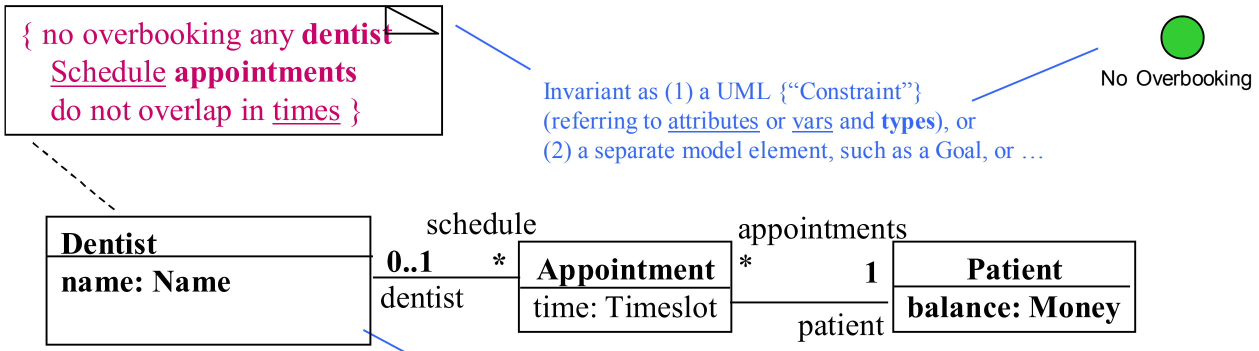
Cannot Capture All Constraints Graphically



- Can a dentist have overlapping appointments?



Invariants are Explicit Constraints on State



Invariant as (1) a UML {“Constraint”} (referring to attributes or vars and **types**), or (2) a separate model element, such as a Goal, or …

Or (3) separate description in text document (referring to attributes or vars and **types**)

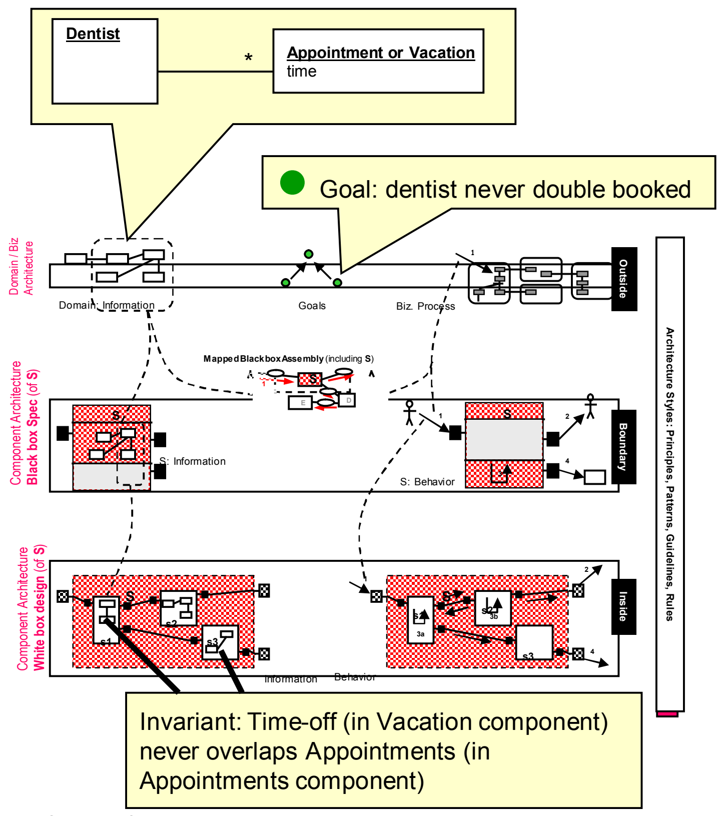
Dentist (invariants or goals)

–No Overbooking — For any **dentist**, there are never two or more **appointments** in its schedule with overlapping times

- These kind of invariants are properly known as “static invariants”

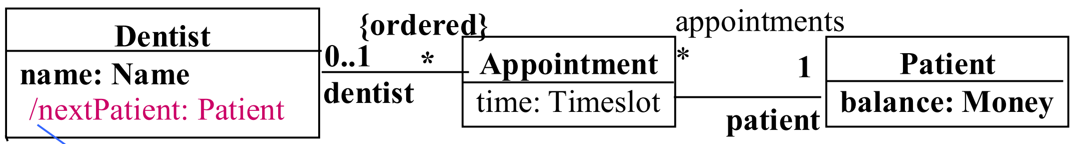
Many different types of invariants can be useful:
- On a single attribute e.g. age < 99
- On multiple attributes: start time < end time
- Derivation rules e.g. age = now – date_of_birth
- On attributes across an association: child.age < child.parent.age
- On a set of objects: dentist.appointments contain no overlapping ones
- Around an association loop: hotel’s guest room is one of the hotel’s rooms

Uses of Invariants: Business, Black-box, White-box


Dentist
*
Appointment or Vacation
time
Goal: dentist never double booked
Domain / Biz Architecture
Domain Information
Goals
Biz. Process
Outside
Mapped Blackbox Assembly (including S)
Component Architecture Black box Spec (of S)
S: Information
S: Behavior
Boundary
Component Architecture White box design (of S)
Information
Behavior
Inside
Architecture Styles: Principles, Patterns, Guidelines, Rules

Invariant: Time-off (in Vacation component) never overlaps Appointments (in Appointments component)

“Convenience” attributes simplify terms

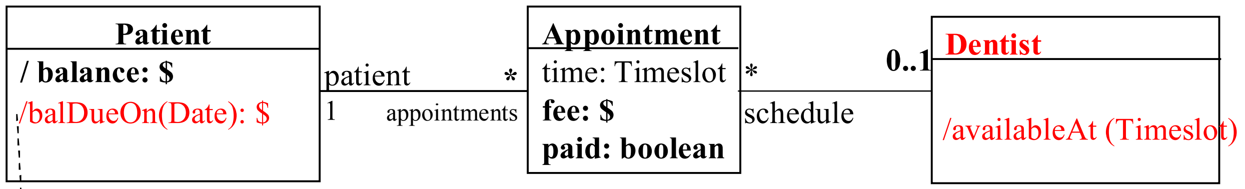


“/” means attribute is for convenience; it can be “derived” from others.
It can have a derivation expression (formal or informal) in a UML “Constraint”

{ nextPatient means the patient in the first scheduled **appointment** in the future
nextPatient = schedule->select (time > now)->first.patient }

- “The next patient for d1” is actually a bit awkward to say
  - d1.appointments->select (time > **now**)->first.patient
- Instead, simply **introduce** a new attribute or association
  - the derivation rule can be separately **defined**, or that can be deferred
- This encapsulates complex navigation into a single attribute
  - Many places can refer to **nextPatient**, single point to define derivation rule

[Optional] Attribute as Query with Parameters

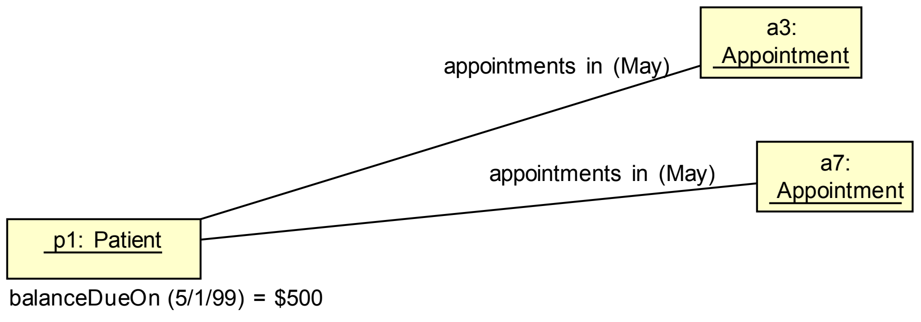


Parameterized attribute is simply a conceptual nested Table[Date->$], with different $ values for different dates. It does not have to be implemented as such, but provides the “vocabulary” to refer to amounts balDueOn a date I.e. query.

balDueOn (d: Date) is the total fees of unpaid appointments 45+ days older than d
balDueOn (d) = appointments->select(not paid and time < d - 45).fee->sum (fee)
(Either a “note” or in text document template)

- A patient owes a total balance (derived from fees for unpaid appointments)
- Patient owes different amounts by different deadlines
  - Patient gets 45 days to make payment for any appointment
- Model as attribute **balDueOn** (Date): $
  - i.e. dueOn parameterized by Date; invariant defines its value for different dates
- How about **availableAt** (Timeslot)?
- We model these as state rather than behavior: entirely a query on current state

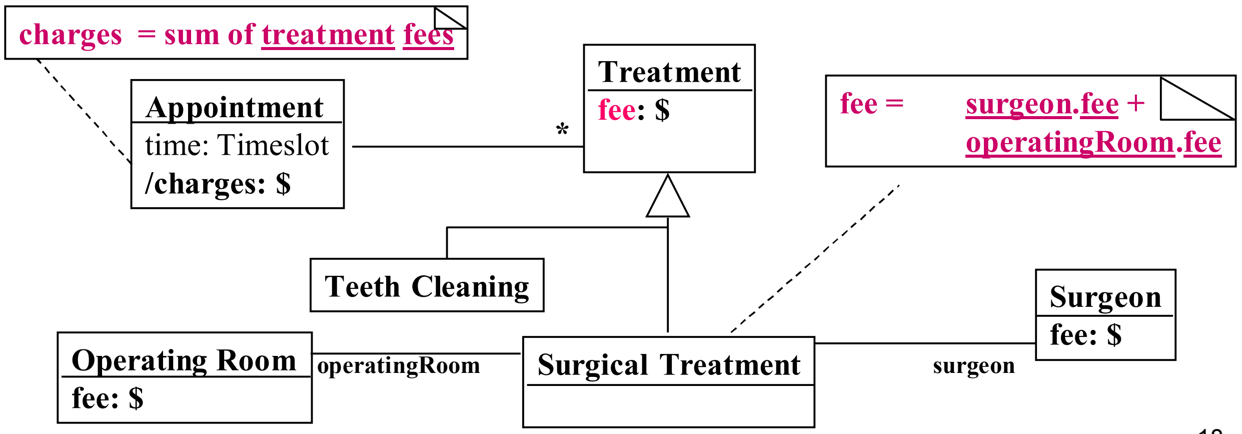
[Optional] Snapshots and Queries



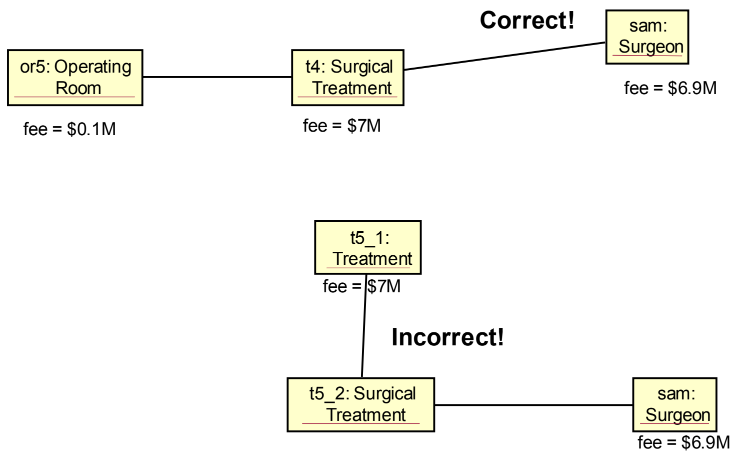
- A snapshot can show values of a parameterized query for any interesting values of the parameters
- Can be used for associations as well

SubTypes and Invariants

- Sub-type is defined as extension of some super-type(s)
  - Sub-type inherits all attributes, invariants, etc. of supertype, adds more info, constraints
- Only use a supertype if its properties are used independent of subtype
- Only add subtypes for variations which elaborate the properties, add new ones
- How would we model
  - Charges for each appointment summed from different kinds of treatments?
  - Dentist having both appointments and time-off?



Snapshots and Sub-Types



- ♦ Instance of the subtype has all the properties of subtype and supertype
  - – No separate subtype instance linked to corresponding supertype instance

Different Uses of Invariants

- ♦ **Definitions** of convenient terms: derived queries / attributes
  - – Dentist: available at (time slot) → no appointment overlapping that slot
  - – These will be used to make any specification simpler and more natural
- ♦ **Goals** to be achieved
  - – Dentist: no overbooking → no appointments assigned with overlapping slots
  - – These will be used for Goals, integrity constraints, etc.
    - • e.g. Portfolio risk is maintained below threshold as market conditions change
- ♦ **Facts** known to be true in the domain
  - – Patients have insurance policies that constrain coverage for specific treatments
- ♦ **States**
  - – Appointment: confirmed → dentist assigned, date > today

Exercise 4.2

Review: Information Model

- A Snapshot is a concrete example of a situation with linked objects
  - Snapshots are crucial to understanding an information model
- Constraints on valid snapshots are called “static invariants”
  - Some are defined by the graphical notation: *, 0..1, {ordered}, etc.
  - Others are written textually, informally or in OCL
- Convenience (derived) attributes help simplify descriptions
- Supertypes and subtypes help extract common but extensible properties
- The Information Model plus explicit static invariants define permitted combinations of attribute / link values

Behavior Model: More Details

- ♦ Actions and Variants
- ♦ Action Granularity
- ♦ Formalizing Actions
- ♦ Actions as Tests

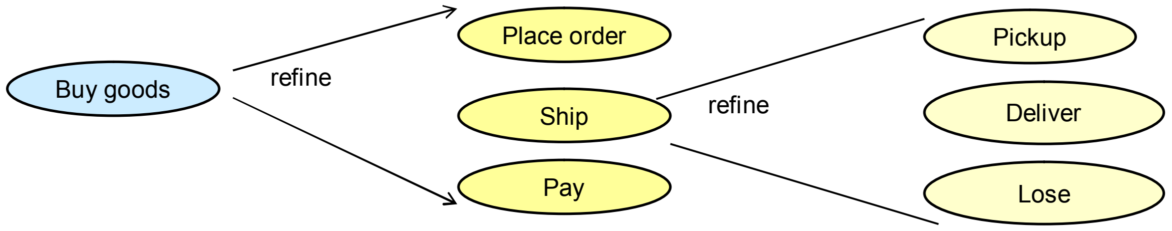
Action and Variants

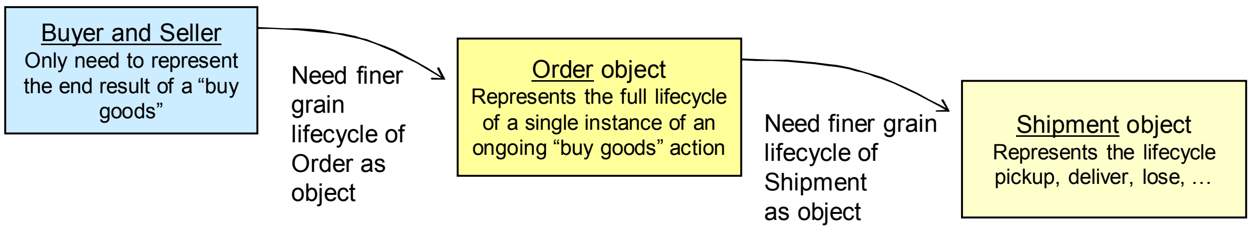
- ♦ MAP uses the term **action** to cover many different variants of behavior
  - An **event** that “happens” in some domain e.g. completion of some activity
  - An entire business process can be considered a long-lived action
  - An **activity** within a business process is an action
  - An **interaction** between business roles is an action
  - A **use case** involving a software system is an action
  - An individual **operation** invocation and execution in software is an action

- ♦ Actions in general have
  - some begin-end granularity, deliberately chosen as a suitable abstraction
  - action name
  - some parameters (for some kinds of actions e.g. use cases and operations, it is useful to distinguish inputs vs. outputs)
  - a start state and an end state written as pre-condition and post-condition
    - Some long-lived processes are better characterized by the state they *maintain*
  - refinements: finer-grained actions which combine into the larger action

Granularity of Actions in MAp

- The granularity of actions depends on the purpose of the model
- Any action can be expanded into finer-grained ones
- Any object can be expanded into finer-grain ones

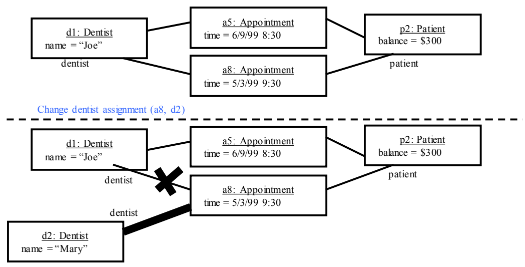


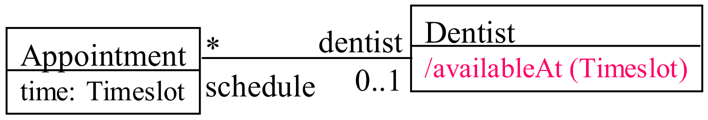

Buyer and Seller
Only need to represent the end result of a "buy goods"
Need finer grain lifecycle of Order as object
Order object
Represents the full lifecycle of a single instance of an ongoing "buy goods" action
Need finer grain lifecycle of Shipment as object
Shipment object
Represents the lifecycle pickup, deliver, lose, …

- If you detail internal sub-actions of an action, model the larger action as an object

Formalizing an Action Specification

- Write informal post condition
- Add action parameters
- Formalize pre and post-condition
- Integrate the behavior and information model

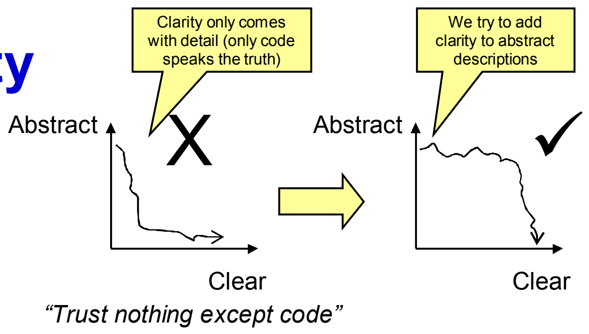




action change dentist assignment (a: Appointment, d: Dentist)
description an existing appointment is assigned a different dentist e.g.
If it was unassigned, or if the previous dentist became unavailable
precondition -- appointment <u>time</u> not passed, and <u>d</u> <u>is available</u> at a's <u>time</u>
a.time > **now** -- formality optional (OCL)
d.availableAt (a.time) -- requires a convenience attribute
postcondition **-- the appointment is now in <u>d</u>'s <u>schedule</u>**
a.dentist = d
-- Can state (redundantly) "d is now unavailable"
-- not d . availableAt (a.time)
-- Can state (redundantly) "previous dentist is now available"
-- a.dentist@**pre** . availableAt (a.time)

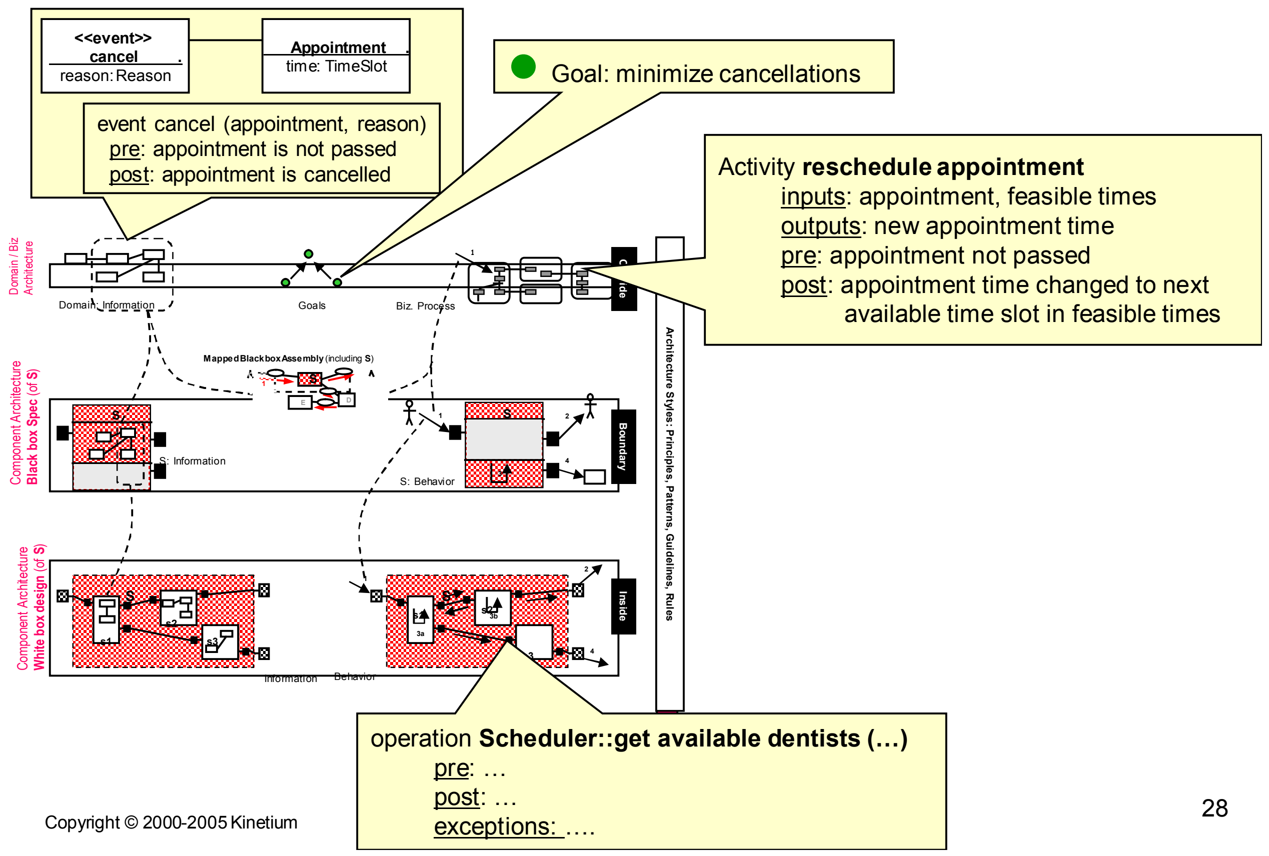
- We will use clear narrative, optionally highlighting <u>attributes</u> and **types**.

A Primary Goal is Clarity


Clarity only comes with detail (only code speaks the truth)
Abstract
X
Clear
"Trust nothing except code"
We try to add clarity to abstract descriptions
Abstract
✓
Clear

1. Abstract ≠ Fuzzy Clear ≠ Detailed
2. **Clear terms** give clear specifications
3. Terms should be **natural** and **simple** for all stakeholders
4. Say anything important **Exactly Once**
5. Good **Information Model** is the key
6. **Avoid** using terms not declared in the model
7. Completeness: models define what must, can, and cannot be

Uses of Actions and Events: Goals, Black-box, White-box


<<event>> cancel
reason: Reason
Appointment
time: TimeSlot
● Goal: minimize cancellations
Domain / Biz Architecture
Domain: Information
Goals
Biz. Process
Mapped Blackbox Assembly (including S)
Component Architecture Black box Spec (of S)
S: Information
S: Behavior
Boundary
Component Architecture White box design (of S)
Information
Behavior
Inside
Architecture Styles: Principles, Patterns, Guidelines, Rules

event cancel (appointment, reason)
pre: appointment is not passed
post: appointment is cancelled

Activity **reschedule appointment**
inputs: appointment, feasible times
outputs: new appointment time
pre: appointment not passed
post: appointment time changed to next available time slot in feasible times

operation **Scheduler::get available dentists (…)**
pre: …
post: …
exceptions: ….

What Does This Have To Do With Architecture?

- A Lot!
- Architecture must always be both abstract and clear
  - Abstract – how to defer selected aspects?
  - Clear – does this implementation conform to that architecture?
- For example: Service Oriented Architecture should define terms such as
  - Service – networked resource with certain characteristics
  - Service Contract – the published specification of the service
  - Component or Agent – the things that implements or uses the service
  - Message – communication between Components
  - Message sender – the Component who sent the message
- Abstract – Multiple options for service contract, message sender, etc.
- Clear – Certain things will definitely not conform
  - A service with no contract
  - A message with missing or forgeable sender information

Language and Vocabulary shape our Architectures

- "We dissect nature along lines laid down by our native language .… Language is not simply a reporting device for experience but a defining framework for it."
  - Benjamin Whorf, Thinking in Primitive Communities in Hoyer (ed.) New Directions in the Study of Language, 1964

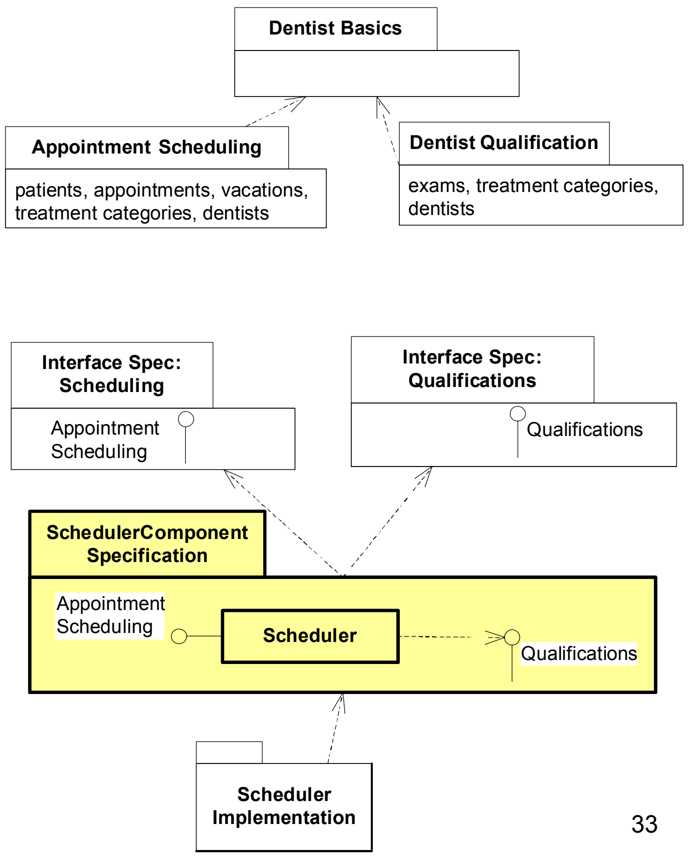
Quality Check on Models

- Models should help all stakeholders to discuss subtle aspects clearly, using a consistent set of terms, and without hand-waving
  - Minimize terminology that is not well defined in the information model + dictionary
- Descriptions in terms of the model should be clear, simple, natural
  - Revise names, introduce convenience attributes to simplify
- The models should be sufficiently consistent
  - All (and only) valid snapshots admitted by the Information Model
  - Scenarios and the state changes are consistent with action specs
  - Action specs are defined consistently with the Information Model
- The models and specs should be concise
  - Eliminate duplicate statements e.g. with convenience attributes
  - Every important domain / client statement or term is stated exactly once
- The models should be sufficiently complete
  - Admits all legal phenomena, prohibits all illegal ones
  - Each object type has creation [and optionally destruction] actions
  - Each attribute and association has creation [and optionally modification] actions
  - For each state of an object, you have considered all relevant actions [including non-occurrence of any action as defined by some form of time-out]
  - End-to-end scenarios adequately exercised against the models

Exercise 4.3

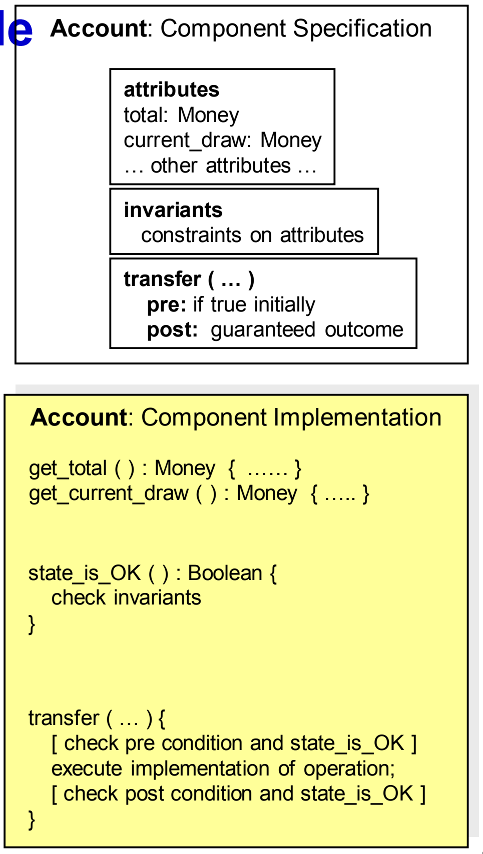
Structuring Models

- A UML **package** can import other packages to use and extend their models
- Packages can separate
  - Subject domains in business
  - Interfaces from component specs
  - Interfaces from implementations
- Where appropriate, a document or document section can be treated like a package


Dentist Basics
Appointment Scheduling
patients, appointments, vacations, treatment categories, dentists
Dentist Qualification
exams, treatment categories, dentists
Interface Spec: Scheduling
Appointment Scheduling
Interface Spec: Qualifications
Qualifications
SchedulerComponent Specification
Appointment Scheduling
Scheduler
Qualifications
Scheduler Implementation

[Optional] Models, Tests, Code

- Models and specifications provide important inputs to coding and to systematic testing with assertions
- Each attribute should be either stored or computed by a query in the implementation of a component
- Each invariant should hold true after every external operation on the component
- Each post-condition should check true after successful completion of every operation on the component (provided it was invoked with the pre-condition true)
- Snapshots and scenarios map to specific test cases and expected outcomes


Account: Component Specification
attributes
total: Money
current_draw: Money
… other attributes …
invariants
constraints on attributes
transfer (…)
pre: if true initially
post: guaranteed outcome
Account: Component Implementation

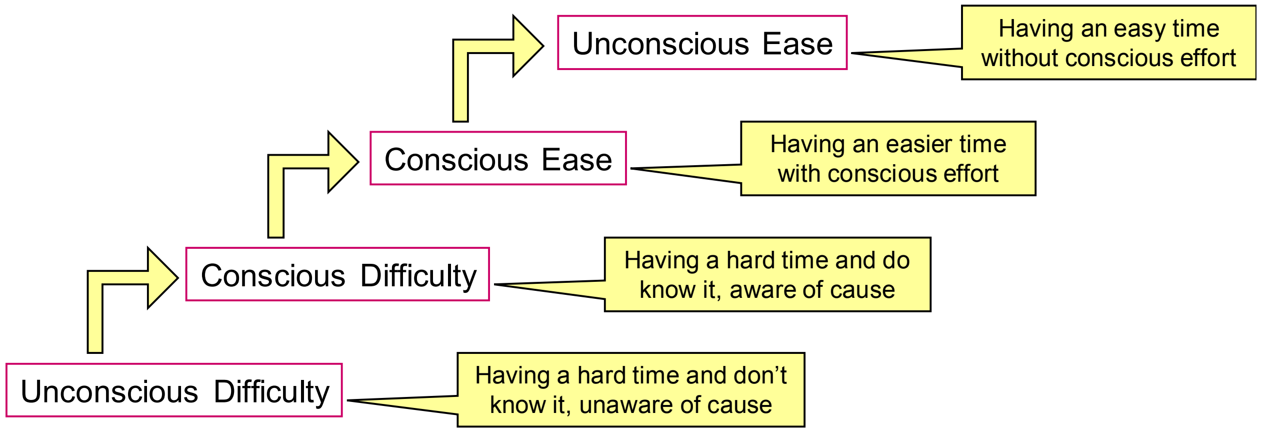
```
get_total ( ) : Money { …… }
get_current_draw ( ) : Money { ….. }

state_is_OK ( ) : Boolean {
   check invariants
}

transfer ( … ) {
   [ check pre condition and state_is_OK ]
   execute implementation of operation;
   [ check post condition and state_is_OK ]
}
```

Migrating to a Model-Driven Approach like MAP

- MAP can be applied as rigorously or lightly as appropriate
- The underlying concepts of MAP are few and simple
- Simple ≠ Easy
- Applying it effectively takes a focused and disciplined mindset and skill improvement
- Classic migration path …


Unconscious Ease
Having an easy time without conscious effort
Conscious Ease
Having an easier time with conscious effort
Conscious Difficulty
Having a hard time and do know it, aware of cause
Unconscious Difficulty
Having a hard time and don’t know it, unaware of cause

Summary

This chapter has shown you how to:

- Integrate your basic modeling skills
- Clearly and precisely specify goals, constraints, and behaviors
- Navigate information models and snapshots
- Use invariants to further constrain the information model
- Use supertypes and subtypes
- Choose action granularity and specify actions
- Check models for quality

Chapter 5

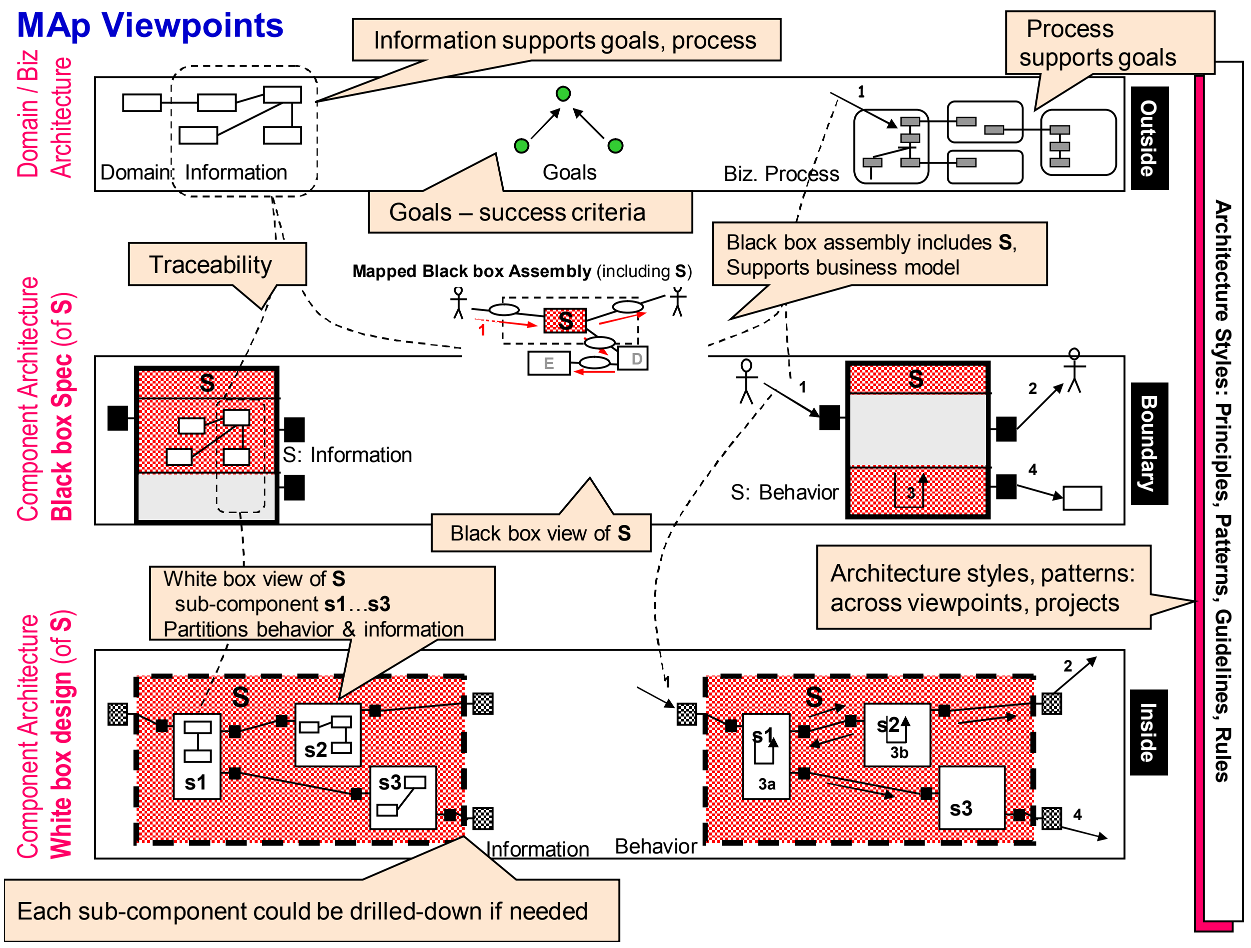
How to Describe a Component to the Business

Business View of <System X>

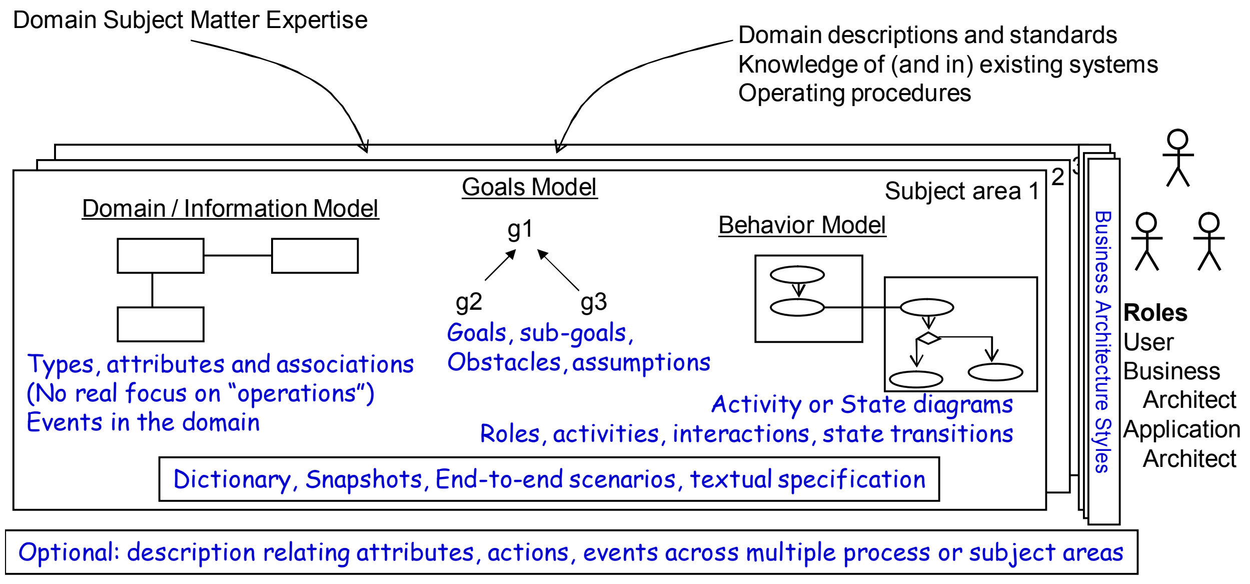
This chapter introduces techniques for modeling business processes with roles, activities, and interactions.

It covers:

- ♦ Business Goals Model
- ♦ Domain / Information Model
- ♦ Business Behavior Model
  - – Activity diagrams
  - – State diagrams
- ♦ [Optional] Defining a reference business process architecture


MAp Viewpoints
Information supports goals, process
Process supports goals
Domain / Biz Architecture
Domain: Information
Goals
Biz. Process
Outside
Goals – success criteria
Traceability
Mapped Black box Assembly (including S)
Black box assembly includes S, Supports business model
Component Architecture Black box Spec (of S)
S
S: Information
S: Behavior
Boundary
Black box view of S
White box view of S sub-component s1…s3 Partitions behavior & information
Architecture styles, patterns: across viewpoints, projects
Architecture Styles: Principles, Patterns, Guidelines, Rules
Component Architecture White box design (of S)
s1
s2
s3
3a
3b
Information
Behavior
Inside
Each sub-component could be drilled-down if needed

Preview of Business Architecture

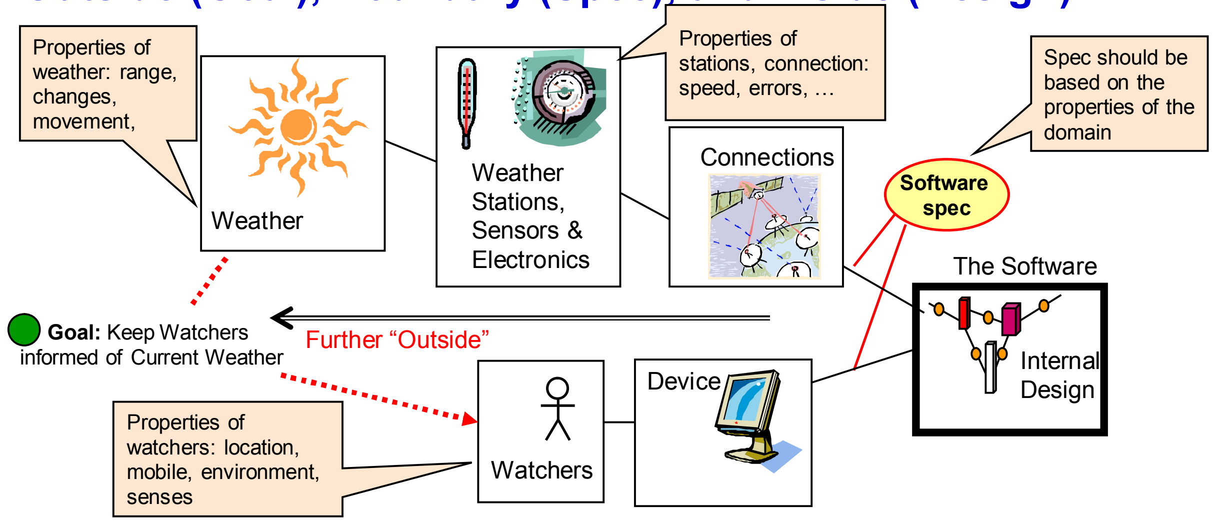


Domain Subject Matter Expertise
Domain descriptions and standards
Knowledge of (and in) existing systems
Operating procedures
Domain / Information Model
Goals Model
Subject area 1
2
3
Behavior Model
g1
g2
g3
Goals, sub-goals,
Obstacles, assumptions
Types, attributes and associations
(No real focus on "operations")
Events in the domain
Activity or State diagrams
Roles, activities, interactions, state transitions
Dictionary, Snapshots, End-to-end scenarios, textual specification
Business Architecture Styles
Roles
User
Business Architect
Application Architect
Optional: description relating attributes, actions, events across multiple process or subject areas

Why do a Business Architecture?

- To get a clear definition of business goals, domain, rules and target business model, define the ultimate requirements for the project, and document that model in a form that serves downstream work.

Outside (Goal), Boundary (Spec), and Inside (Design)



Properties of weather: range, changes, movement,
Weather
Weather Stations, Sensors & Electronics
Properties of stations, connection: speed, errors, …
Connections
Spec should be based on the properties of the domain
Software spec
The Software
Internal Design
Goal: Keep Watchers informed of Current Weather
Further "Outside"
Properties of watchers: location, mobile, environment, senses
Watchers
Device

- A domain is a part of the world with some shared state or interactions with other domains
  - Weather, weather stations, watchers, the machine
- Problem domain is different from Machine domain and intermediate "connection" domains
  - The problem to be solved may be **outside** the machine; the machine + connections help solve it
- **Goals** define what the machine must cause in the **Problem Domain**
  - It is expressed in terms of all relevant aspects of the problem domain(s): weather, watchers
- **Software Specification** defines all interfaces of the software, including technical and UI
  - It is part of the solution, along with other domains, machines, human operating procedures

Steps Toward Business Architecture

- **Frame the Problem**
  1. **Describe the Domain:** Describe relevant properties of the domain that is the subject of goals
     - **Things that exist** – object and attributes
     - **Things that happen** – events and actions affecting objects and attributes
  2. **Define Goals:** Prescribe desired relationships between domain elements that we want to achieve
     - Define stakeholders
     - Explore **why** of goals to get higher-level goals
     - Explore **how** of goals to get more complete sub-goals, assumptions about domain, system-specs
     - Refine domain model and refine goals
  3. **Validate with Scenarios:** Use goals as problem boundary for end-to-end scenarios
     - Use these to validate business goals and process models
  4. **Identify Essential Process Architecture:** Use domain model and scenarios to identify essential processes
     - Case, case-coordination processes
- **[Optional]** Current state analysis for domain knowledge, problems & opportunities
- **Design target-state process model:** using the above goals and as-is models
  - If you design multiple processes, also include an overview of how they interrelate
  - Ensure Information Model supports activities in process and goals model
  - You do not need to decide specific automation or system boundaries
- **Why?**
  - Use goals and understanding of current-state to design future-state business process

Domain Model and Goals

- **Describe the Domain:** Relevant properties of the domain that is the subject of goals
- **Define Goals:** Prescribe desired relationships about domain elements
- Validate with Scenarios: Use goals as problem boundary for end-to-end scenarios

Domain Properties ■ Support Goal Refinements

- **Maximize sustainable revenue from individual patients**
  - ■ ***Assumption**: Sustainable revenue needs perceived & realized value & payment ability*
  - **Maximize perceived value to each patient by providing treatments to meet needs**
    - ■ ***Fact**: Custom treatments cannot be developed from scratch per patient*
    - Have Treatment Portfolio that covers typical Patient Segment Needs
    - Understand Individual Patient Needs
    - Maximize coverage of Individual Needs with Custom Treatments created From Portfolio
    - Keep Patients aware of provided value of treatments
  - **Maximize realized value from patients commensurate with perceived value**
    - ■ ***Fact**: Different patients can return different value, can pay different fees, …*
    - …
  - **Advice patients on insurance options suited to anticipated treatments**

Why does this subgoal help get revenue from customers?

What are some possible subgoals, based on this domain fact?

What is a "Good" Goal Refinement?

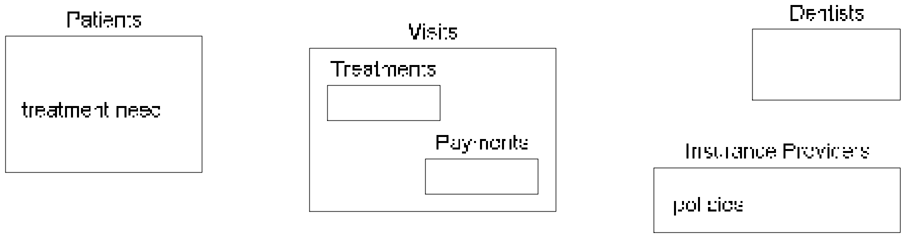
♦ Formally, a set of Goals and Domain Properties {G1, ..., Gn, D1 } refines a goal G if the refinement is:

1. Sufficient / Complete: G1, ..., Gn , and domain properties ➔ G
2. Necessary / Minimal: No subset of G1 … Gn is sufficient to ➔ G
3. Consistent: The goals and domain properties are consistent

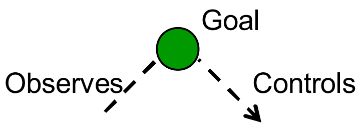
Domain – Patients, Dentists, and Visits

- Patients have treatment needs based on dental conditions. They visit the Dental Practice and are treated by Dentists. Treatments are paid by the Patients and their Insurance Providers.

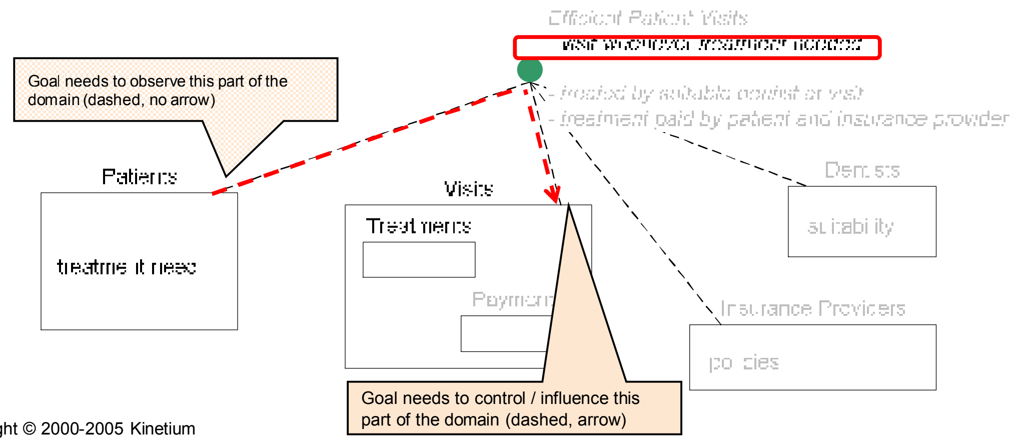
- Sketch of the domain model
  - Drawn informally for now
  - Will be developed into Information Model etc. later



Goals – Patient Visits



- Goal: Efficient Patient Visits:
  - Patients visit whenever treatment is needed
  - Patients are treated by suitable dentists at the visit
  - Treatments are paid for by patients and their insurance providers
- Model indicates which things in the domain are referenced by the goal: can distinguish observe vs. control
- In order to refine and satisfy this goal we need to investigate properties of the domain …
  - What domain properties causes a treatment to "be needed"? What domain properties affect visits?
    - Routine, scheduled follow-ups, and symptomatic or emergency visits
    - Patients forget routine appointments
    - Unexpected calendar conflicts for either patients or dentists
  - What makes a dentist suitable for treating a patient?
  - What are the properties of patient and insurance payment?

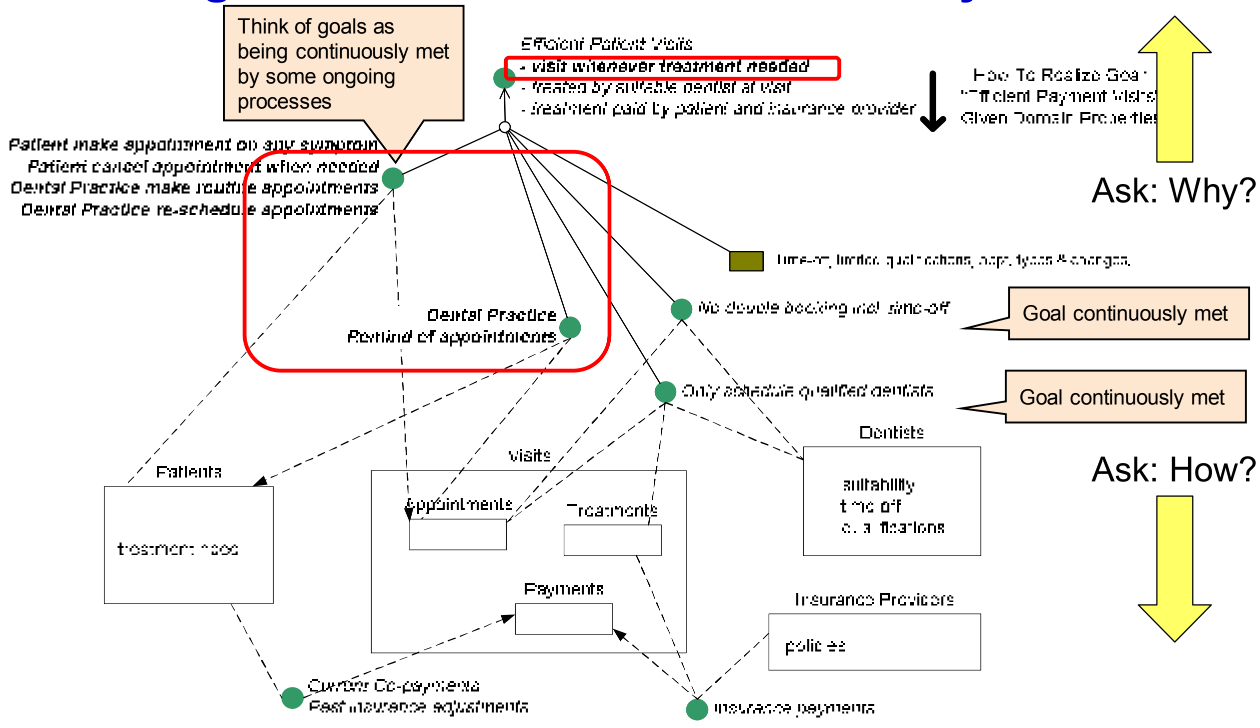

Efficient Patient Visits
visit whenever treatment needed
- treated by suitable dentist at visit
- treatment paid by patient and insurance provider
Goal needs to observe this part of the domain (dashed, no arrow)
Patients
treatment need
Visits
Treatments
Payments
Dentists
suitability
Insurance Providers
policies
Goal needs to control / influence this part of the domain (dashed, arrow)

Domain Properties ■ Guide Sub-Goal Design

- Visits are **needed** for routine checks, follow-up treatments, or based on current symptoms
  - Goal: Both Patients and the Dental Practice make appointments for patient visits as needed.
- A patient's scheduling needs can change due to personal appointments.
  - Goal. Change and Cancel Appointments. Patients can change and cancel their appointments.
- Sometimes there are changes in dentist availability and the Dental Practice needs to re-schedule existing appointments.
  - Goal. Dental Practice Re-schedules Appointments
- Patients sometimes miss appointments.
  - Goal. Remind Patients of Appointments
- Dentist's have time off and can become double booked for appointments or planned time-off.
  - Goal. No double booking of dentists including time-off.
- Not all dentists can perform all treatments.
  - Goal. Only Qualified Dentists Assigned to an Appointment.
- Patients don't like surprises about costs or coverage.
  - Goal. Cost and coverage estimates for appointments known to patients at appointment time.
- Insurance often has co-payments, and actual insurance payments may not match estimates.
  - Goal. Patient pay both co-payments and past-payments

Note: Problem definition includes describing properties **of the problem domain**, and **corresponding** definition of goals and sub-goals, including SWOTs (Strength, Weakness, Opportunity, Threat)

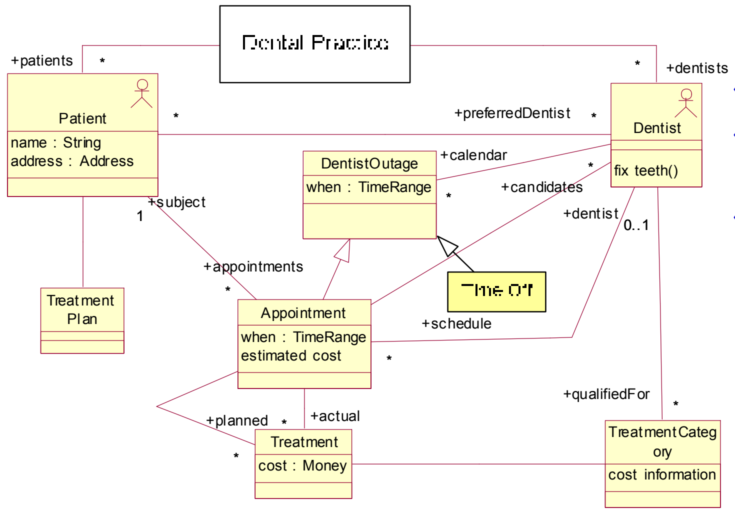
Refining Goals and Domain Model – Why? How?


Think of goals as being continuously met by some ongoing processes
Efficient Patient Visits
- visit whenever treatment needed
- treated by suitable dentist at visit
- treatment paid by patient and insurance provider
How To Realize Goal "Efficient Payment Visits" Given Domain Properties
Patient make appointment on any symptom
Patient cancel appointment when needed
Dental Practice make routine appointments
Dental Practice re-schedule appointments
time-off, limited qualifications, capacity & changes
Dental Practice Remind of appointments
No double booking incl. time-off
Goal continuously met
Only schedule qualified dentists
Goal continuously met
Dentists
suitability
time off
qualifications
Patients
treatment needs
Visits
Appointments
Treatments
Payments
Insurance Providers
policies
Current Co-payments
Past insurance adjustments
insurance payments

Ask: Why?

Ask: How?

Example – Detailed Domain Information Model



- ♦ Goals model gets refined
- ♦ Domain model gets refined
- ♦ Which do we need to add?
  - – time-off
  - – symptoms
  - – insurance
  - – patient's personal calendar
  - – reminders

[Optional] Goal and Goal-Refinement Patterns

- ♦ Where possible use one of the following forms for a goal
  - – Maintain [while condition maintain target-state]
  - – Avoid [while condition avoid target-state]
  - – Achieve [when condition achieve target-state]
  - – Cease [when condition exit target-state]
  - – Min / Max / Optimize / Increase / Decrease [condition to be optimized]
- ♦ There are some patterns of goal refinements as well e.g.
  - – Simple case analysis
    - • Goal: Fill Customer Orders
      - – Domain property: Orders can be for Products or for Services
      - – Sub-goal: Fill Product Orders
      - – Sub-goal: Fill Services Orders
  - – Milestones towards goal
    - • Goal: Deliver Orders From Warehouse On Time
      - – Sub-goal: Package the Order for shipment
      - – Sub-goal: Schedule pickup of Package based on shipment time
      - – Domain property: Scheduled pickups will be delivered within shipment time

Exercise 5.1

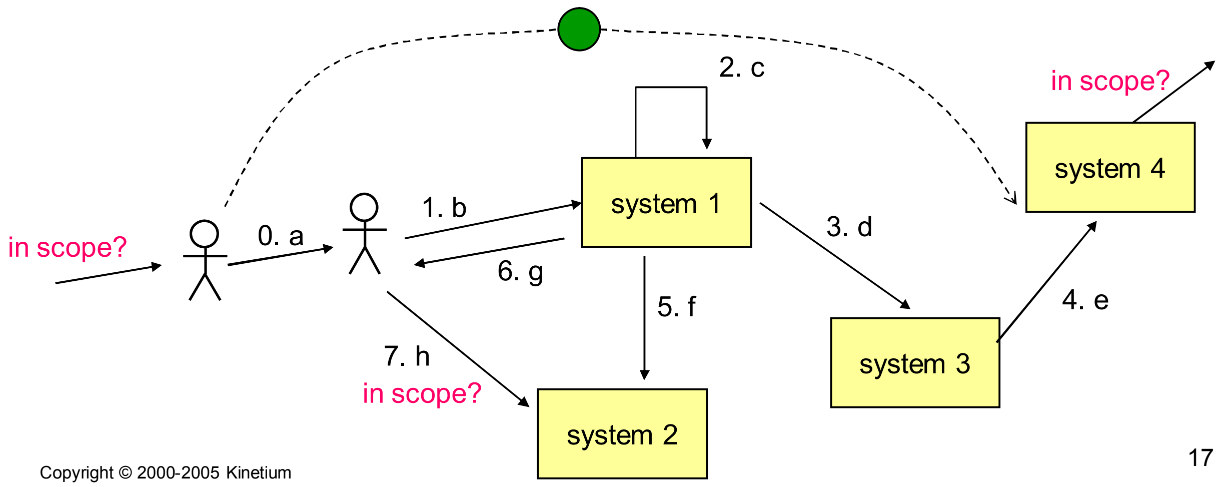
End-to-end Scenarios

Scenarios are sequences of interactions. This section briefly sketches:

- How to select appropriate “size” of scenarios

Finding “End-to-End” scenarios

- **Where** does the scenario start and end, in a “spatial” sense
  - Which software, hardware, people are within scope: phone agent, insurance agent, or policy holder?
  - Start at the **outermost boundary** relevant to the “true” problem i.e. top-most goals controls, observes
- **When** does the scenario start and end, in a “temporal” sense
  - Is the biggest lifecycle “make insurance claim” or “execute full insurance policy lifecycle”?
  - Start with **largest grained actions** at or within this boundary, then finer grain ones as steps in scenarios
- Leave out things that are irrelevant to understanding the problem to be solved
- Once end-to-end scope is decided, choose granularity of objects and size of scenario steps to model
- Explore both success cases and failure paths
- Describe scenario in text, or UML sequence or collaboration diagrams – even early executable test cases!



Example of Boundary, Stakeholders, Goals

- What is the outer boundary for the Dental Practice?
  - Patient? Patient’s family? Patient’s employers?
- What are largest grained actions there? Finer grained ones?
  - Patient lifecycle, visit the dentist, sign in / get treated / pay
  - End to end scenario start from *entire patient lifecycle → complete visit → make appointment*

Sometimes insightful to do careful stakeholder analysis and grouping:
Emergency vs. Routine Patients

- Who are all stakeholders, how to group them, what are their goals, what scenarios apply?
  - *Routine Patients*
    - Want more proactive planning of appointments for dental care and follow-up
    - Would like reminders of appointments with option to change times
    - Need appointments scheduled and completed efficiently, with access to medical history, cost and insurance coverage information, and suitable or preferred dentist or staff assigned
  - *Emergency Patients* - to be seen immediately
  - *Insurers* - justification and audit trail for each treatment
  - *Dental Practice* – Increase patient satisfaction, faster billing, fewer questions from insurance
- Failures, deficiencies, etc.
  - Appointments changed or cancelled due to patient’s personal schedule
  - Appointments changed or cancelled due to dentist becoming unavailable
  - Appointments forgotten by patient
  - Patient care suffers when pre-planned appointments not consolidated with new symptoms
  - Better follow-up planning and scheduling possible

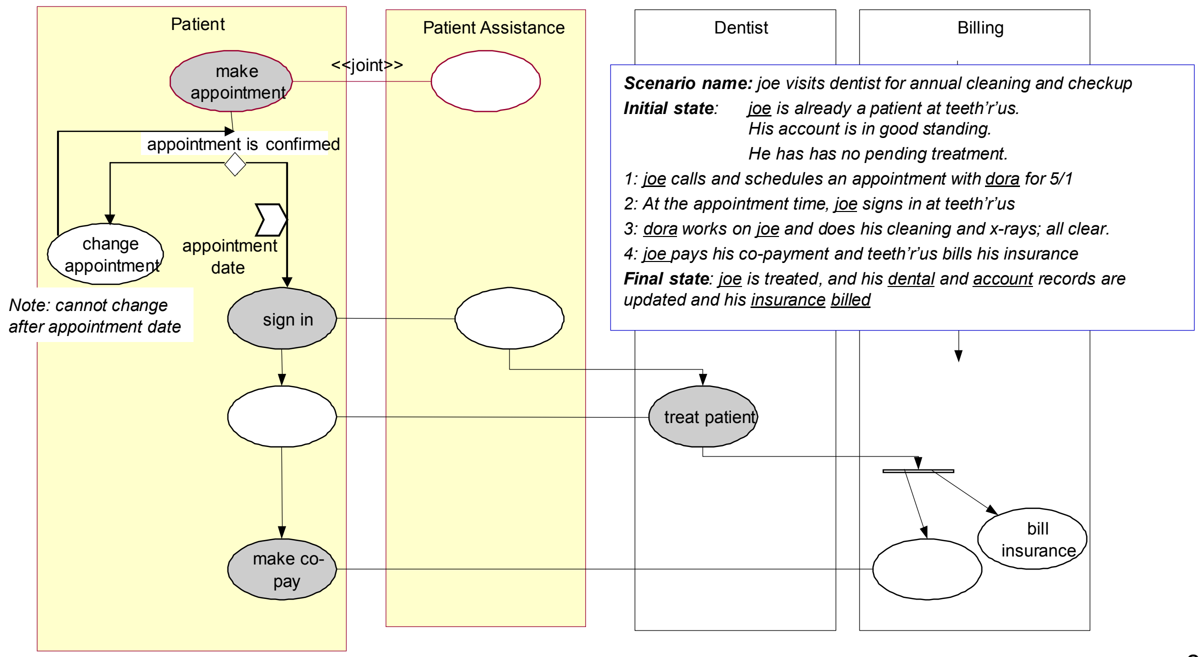
Business Behavior – (Role) Activity Diagrams

A scenario is a concrete story of an interaction sequence. An Activity Diagram specifies required interaction sequences, and generalizes scenarios. This section covers:

- ♦ Describing Business Process with an Activity Diagram
  - – We will include some facilities supported by the new UML 2.0
  - – Alternately, we will use a variation called "Role Activity Diagrams"

Activity diagram Generalizes End-to-End Scenarios

Process:	Patient Visit
Goal:	Patient has made appointment, completed visit, and all charges have been billed



Scenario name: *joe visits dentist for annual cleaning and checkup*

Initial state: *joe is already a patient at teeth'r'us. His account is in good standing. He has has no pending treatment.*

1: joe calls and schedules an appointment with dora for 5/1

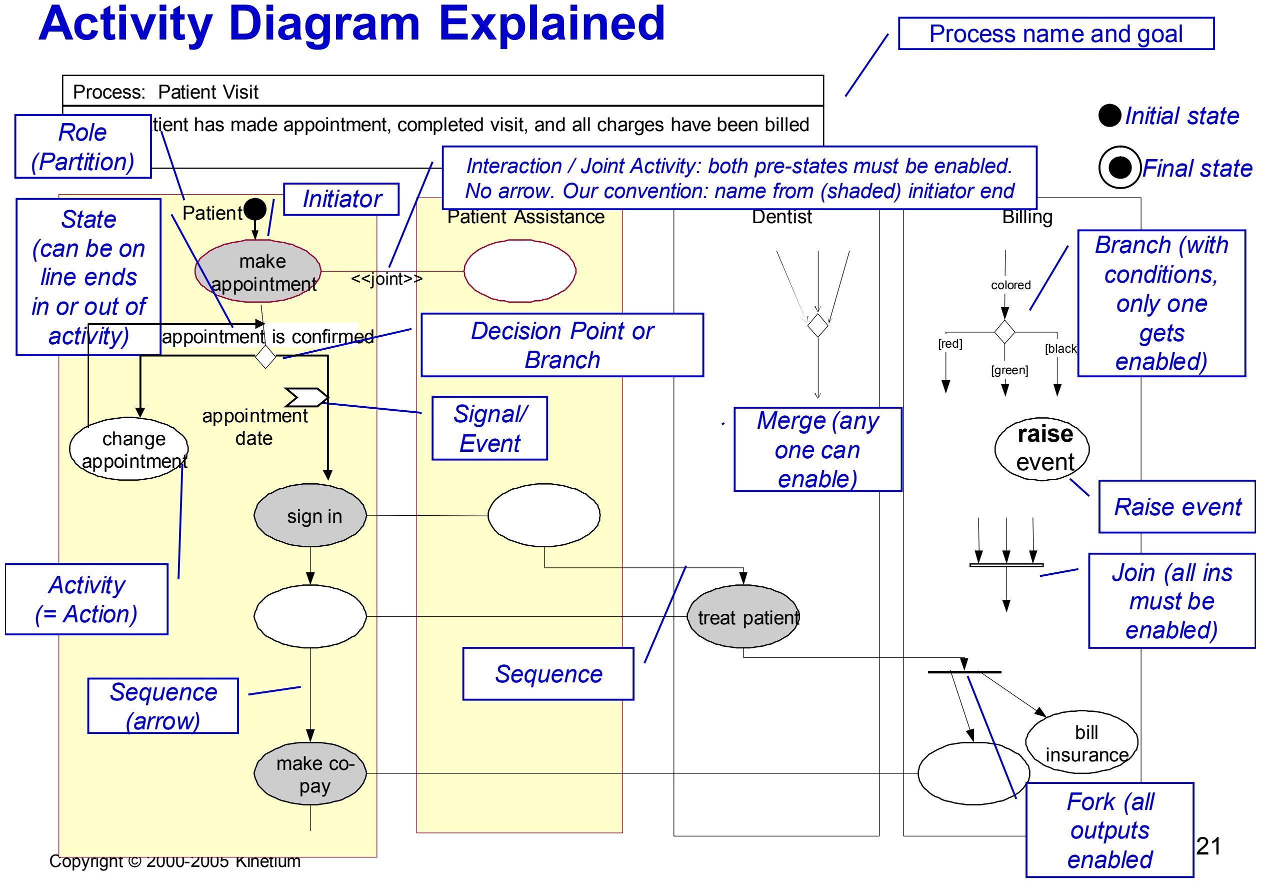
2: At the appointment time, joe signs in at teeth'r'us

3: dora works on joe and does his cleaning and x-rays; all clear.

4: joe pays his co-payment and teeth'r'us bills his insurance

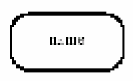
Final state: *joe is treated, and his dental and account records are updated and his insurance billed*

Activity Diagram Explained

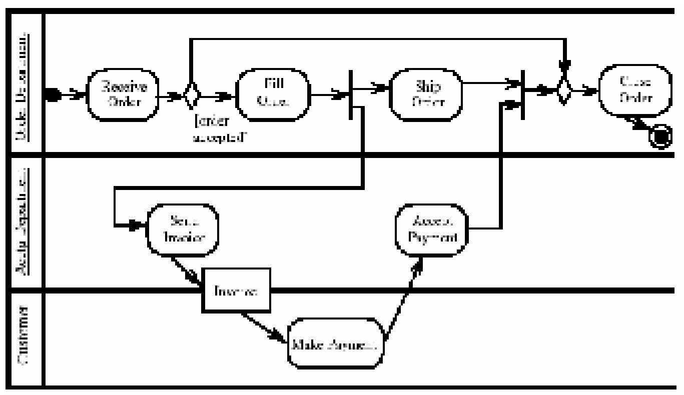


Some UML 2.0 Notes

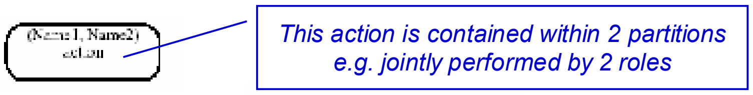
- UML 2.0 uses “Rounded Rectangle” for action and activity



- UML 2.0 “Partition” can indicate role

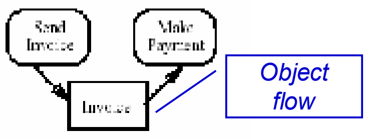


- UML 2.0 does not allow Interactions/Joint Activity on swim-lanes, but 2.0 text version allows “partition” list. Use <<joint>> annotation



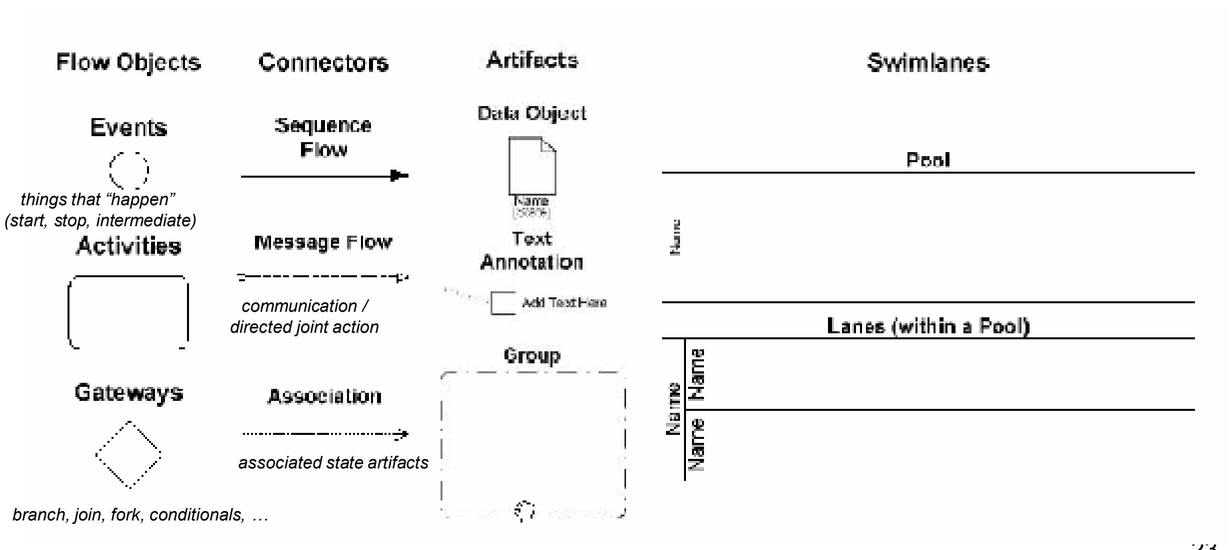
(Name1, Name2) action
This action is contained within 2 partitions e.g. jointly performed by 2 roles

- UML 2.0 has “Object flows” for state, “Interruptible” regions, “pins”, and more



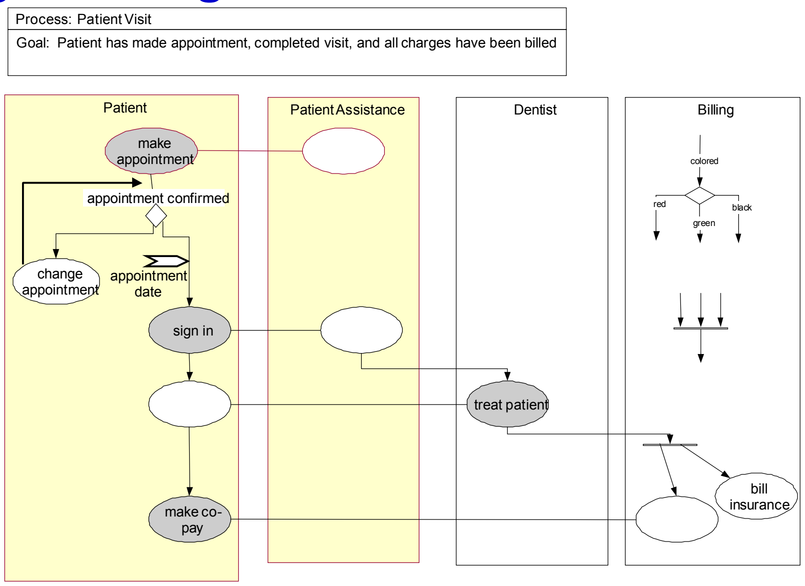
Some BPMN Notes

- Business Process Modeling Notation – Standard
- Constructs and approach consistent with our use of role-activity diagrams



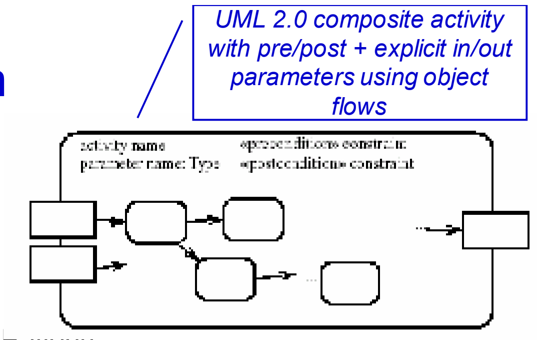
Activity Diagram Pragmatics

Note: the patient can change the appointment anytime many times before appointment date



- Formalizing concurrency, iteration, conditionals… can make diagram detailed
- Just add informal **annotations** to diagram, unless you want executable models
  - Let the diagram focus on being clear about the “normal case”
  - Annotation on diagram, or numbered footnotes
- Compress detailed conditions using names like **“do xyz if appropriate”**

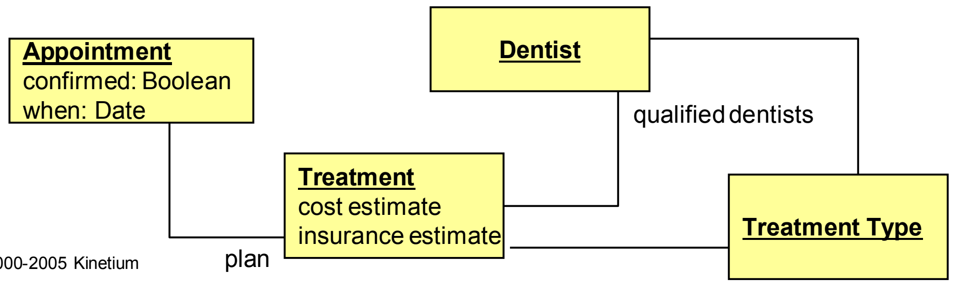
Textual Activity Specification


UML 2.0 composite activity with pre/post + explicit in/out parameters using object flows
activity name
parameter name: Type
«precondition» constraint
«postcondition» constraint

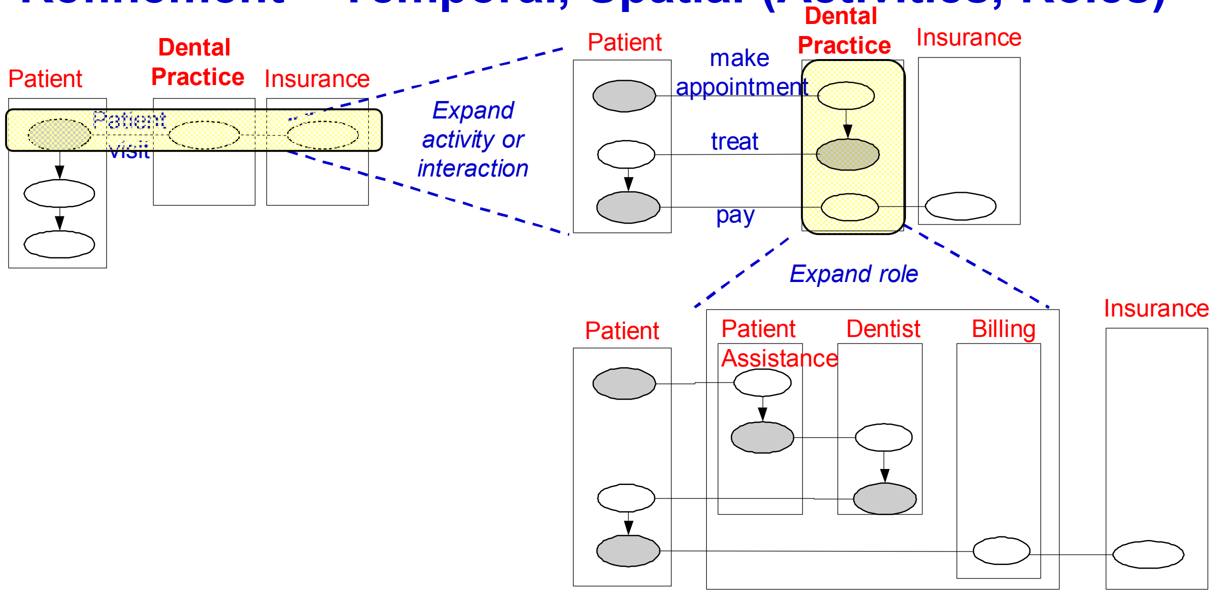
- Actions specified textually

  activity: make appointment
  roles: Patient, Patient Assistance
  inputs: symptoms: Set of Symptom [from Patient]
  outputs: appointment details: Appointment [to Patient]
  goals: references to goals that this activity supports
  Precondition: Patient account is not delinquent
  Postcondition: An appointment has been created for
  patient for date, to cover planned treatments, with dentist qualified for treatment type.
  Insurance and cost estimates recorded.

- As always, this (together with state annotations on diagram) uncovers terminology and states needed in the Information Model



Refinement – Temporal, Spatial (Activities, Roles)



- Any activity can be expanded into finer grained activity or interaction
- Any role can be expanded into a more granular set of roles
- An entire process can be treated as a single action
- **Start with largest grained activities and roles, be willing to zoom in and out**
- **Typically show at most ONE internal action between interactions for an external view**

From Goals to Activities within Processes

- Once goals have been refined adequately we need
  - the actions that are relevant to the goals
  - the requirements on these actions so that the goals are satisfied.
- Identify actions by considering state transitions relevant to those goals
- Goals contribute to specification of actions
  - **action**
    - **must be performed whenever** …
    - **must not be performed unless** …
    - **when performed, must result in** … [success, failure cases]
- **Maintain** [condition → state] goal
  - need constraints on actions that make condition true
  - need constraints on actions that make condition false
- **Achieve** goal …

Specifications on Activities from Goals

- e.g. for a **Maintain** [while condition then state] goal
  - Op 1 to establish state **must** be performed whenever condition becomes true
  - Any Op 2 that exits state **must not** be performed while condition is true
- e.g. for an **Achieve** [when condition then state change] goal
  - Op 1 to establish state change **must** be performed whenever condition becomes true
- Finer analysis possible by making finer-grained distinction of goal types
- A goal influences many actions; an action is influenced by many goals
- One can check that the actions as specified meet the goals

Exercise 5.2

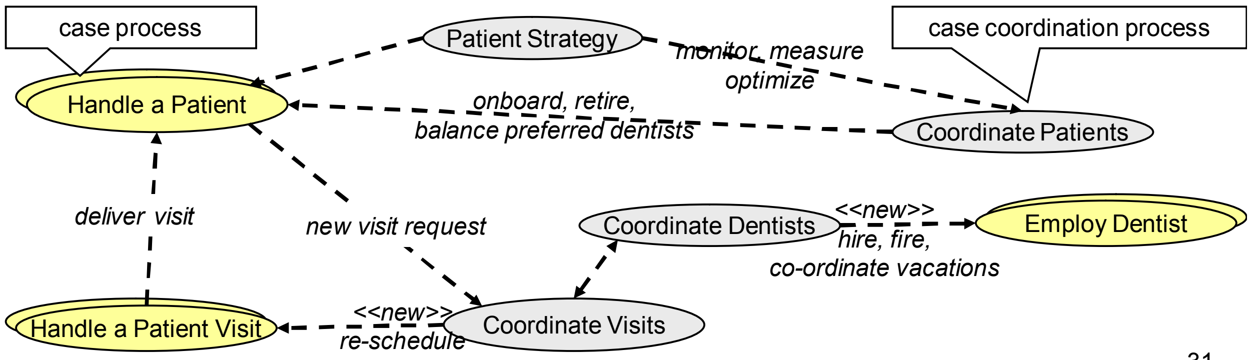
Essential Business Process Architecture

This section covers:

- ♦ Large domains have multiple interrelated business processes
- ♦ Case, Coordination, Strategy gives a clear structure to processes
- ♦ It identifies the essential processes and their key dynamic relationships

Essential Process Architecture

- Find "essential" business entities: things where each instance has a lifecycle you have to manage
  - Patient, Visit, Dentist, Payment
- Define a "case" process: takes each instance through entire lifecycle; focused on a single "case" instance
  - Handle a Patient, Handle a Visit, Handle a Dentist, Handle a Payment
- For each "case", define a "case coordination" process to start, stop, coordinate across multiple "cases"
  - Coordinate Patients, Coordinate Visits, Coordinate Dentists, Coordinate Payments
  - Sometimes "coordination" folds into another "case" process e.g. Treatments are coordinated within a Visit itself.
- Optionally define a "case strategy" process: measure, change, optimize the other processes themselves
- Find dynamic "generate" relations from each case to other entities spawned off, with their own lifecycles
  - Co-ordinate these through the case-coordination process
- Process architecture relates these together with summarized interactions



Pretty Process Pictures – a Reality Check

- An organization operates as a network of processes with dynamics
  - a number of case processes come and go … under the control of
  - case coordination processes … both of which are strategically monitored by
  - some case strategy processes
- Most organizational processes are not neat chains and hierarchies

Business Behavior Model – Lifecycles

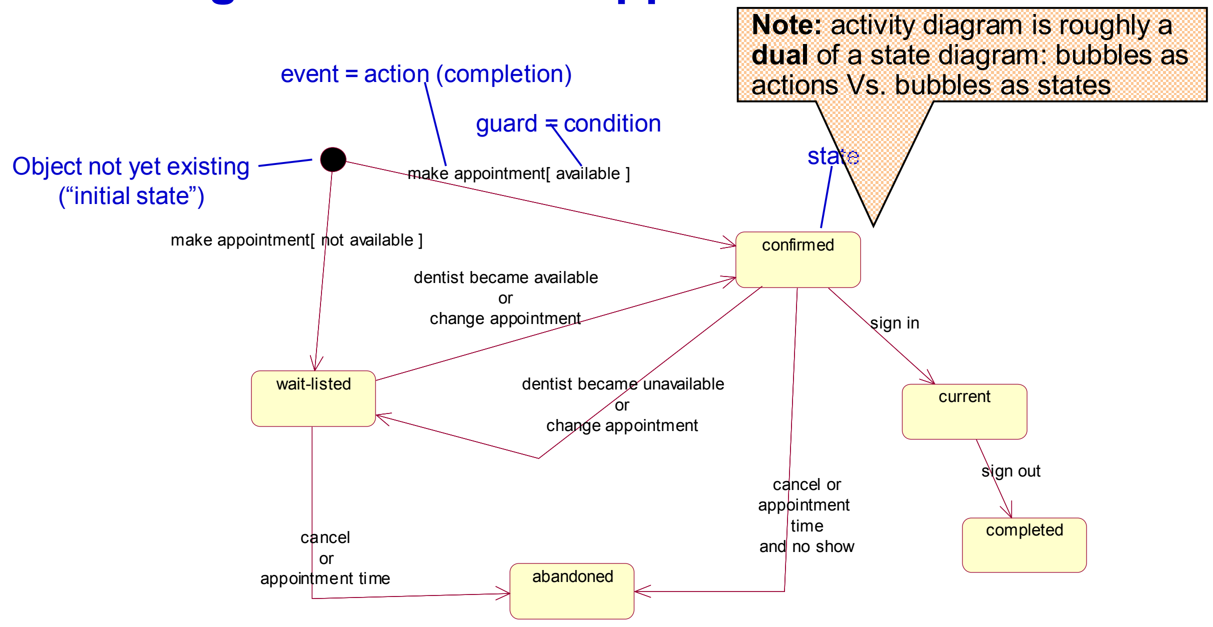
This section covers:

- ♦ Describing Object Lifecycle using a State Diagram and State Matrices

State Model Describes Lifecycle of Object

- ♦ Frequently, some objects in the domain have complex lifecycles
  - – Insurance Company: a Claim - filed, appraised, contested, in court, settled
  - – Pharmaceutical Company: a candidate Drug - lab trials, clinical trials, sample production, FDA approved
- ♦ Question: Which of our activity diagrams represents such a “case” process?
- ♦ **A State Model shows states and transitions between states of such objects**
  - – **Each state is represented by one Boolean attribute (or other status attribute)**
  - – **Each state corresponds to some combination of other attributes and associations**
  - – **The transitions correspond to the completion of actions and activities**

State Diagram of Dentist Appointment


Note: activity diagram is roughly a dual of a state diagram: bubbles as actions Vs. bubbles as states
event = action (completion)
guard = condition
Object not yet existing ("initial state")
state
make appointment[available]
make appointment[not available]
confirmed
dentist became available or change appointment
sign in
wait-listed
dentist became unavailable or change appointment
current
sign out
cancel or appointment time and no show
completed
cancel or appointment time
abandoned

- States: confirmed, wait-listed, current, abandoned, completed
- Events: make appointment, change appointment, sign in, …

State Definition Table and State-Event Table

- Each state is equivalent to a Boolean attribute (or some "status" attribute)
- Each state should be derived from other attributes and associations
  - A simple table (matrix) of states vs. other attributes and associations can be helpful
- Check that states are distinguishable by the other attributes

State	**Derivation Expression**
confirmed	time > now and dentist <> null
wait-listed	time > now and dentist = null
current	time = now and signedIn

- Each transition event should correspond, directly or indirectly, to some action
- All events should be considered in all states
- A simple table of states vs. possible events helps find missing cases

State → **Event ↓**	confirmed	wait-listed	current
Change appointment	Confirmed, wait-listed	Confirmed, wait-listed	NA
Dentist available	Confirmed	Confirmed	NA
Dentist unavailable	Wait-listed	Wait-listed	NA
Time out	Abandoned	Abandoned	NA

Exercise 5.3

Business Architecture – Some Guidelines

- Structure business goals by asking **Why** and **How**
- Always consider relevant domain properties influencing goal refinement
- Focus on end goal and state change for every process or activity
- Focus on states and transitions using state diagram
- Support process behaviors, goals, and states with information model
- Zoom in and out of roles, processes, activities, and goals
- [Optional] Define reference process architecture from object lifecycles

Summary

This chapter has shown that:

- Modeling business goals uncovers assumptions, alternatives, rationale
- A reference business process can be systematically derived
- Activity diagrams and state diagrams provide complementary views

Chapter 6

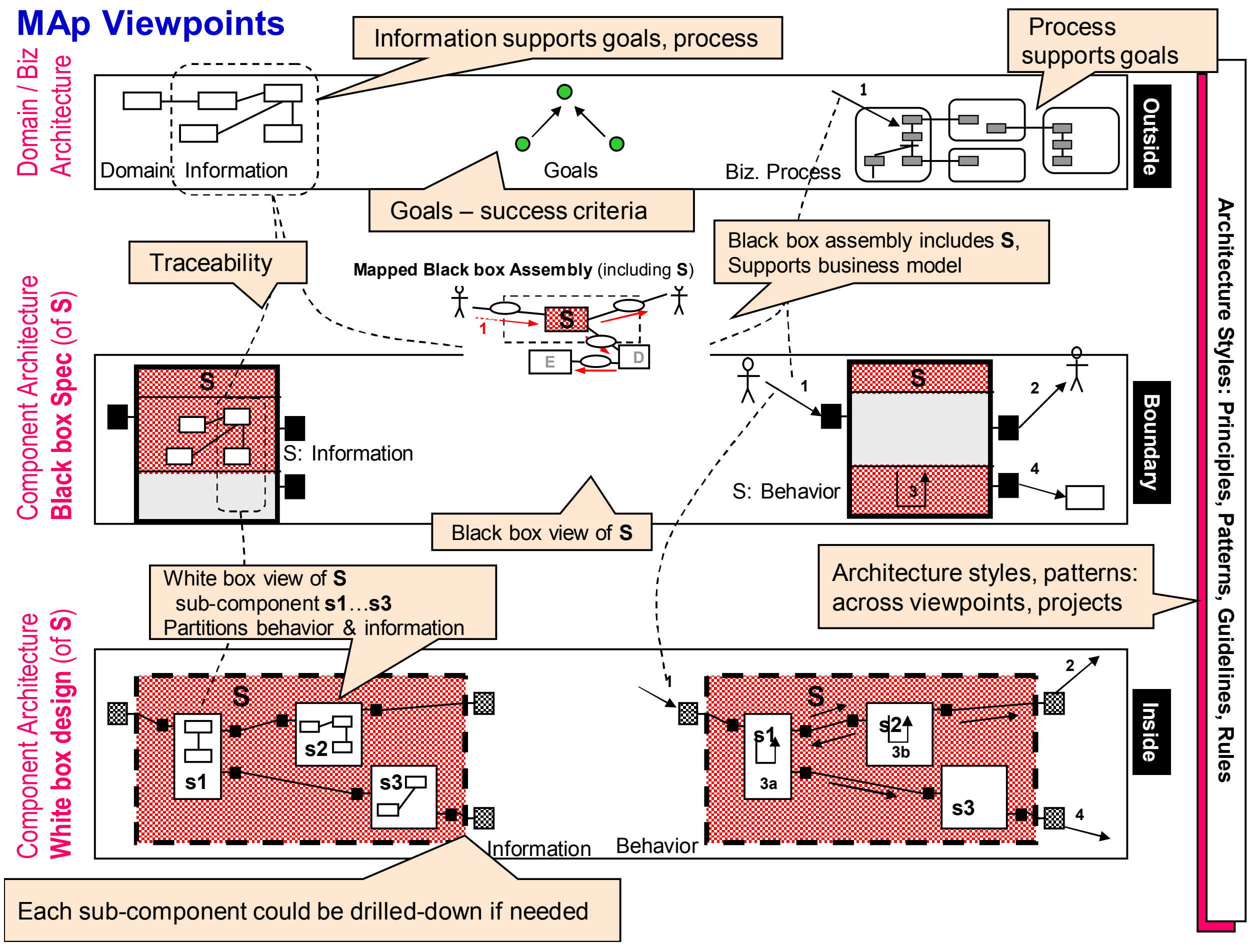
How to Specify a Component's External Properties

Black Box View of <System X>

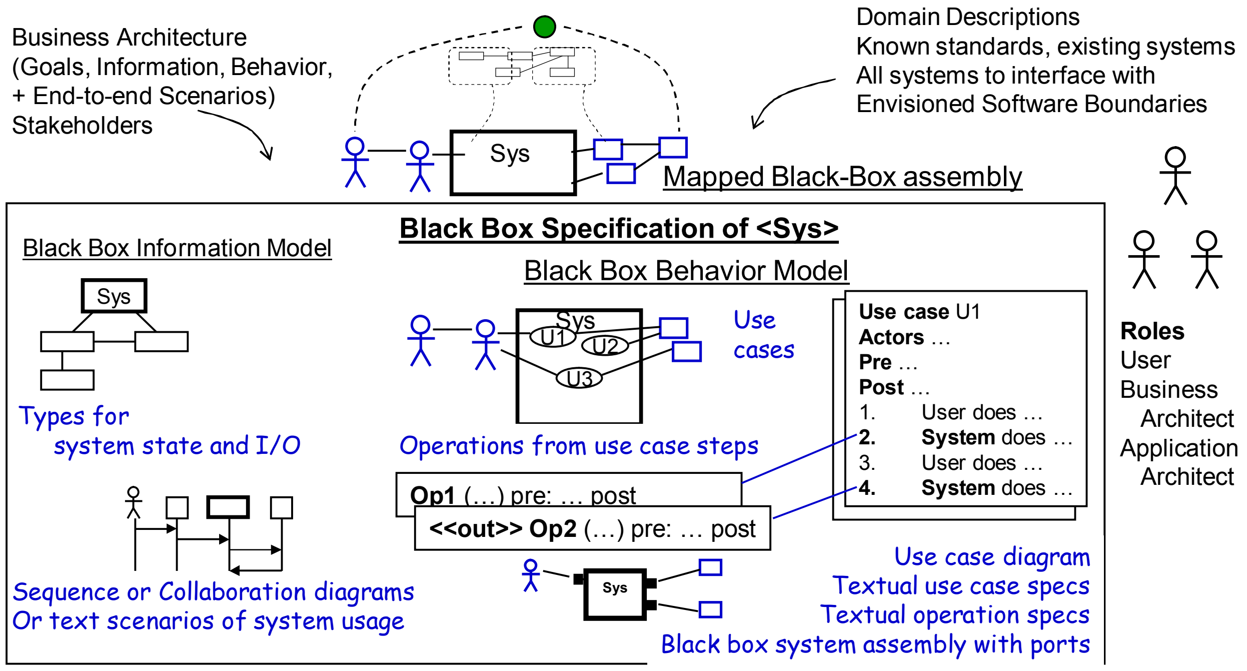
This chapter describes how to specify an application or component from the "black-box" perspective i.e. the boundary of the box and any properties as seen from the outside.

It covers:

- Defining the top-level Mapped Black-Box Assembly
- Specifying Use Cases
  - – Identifying Use Cases from end-to-end scenarios and activity diagrams
  - – Building a Black Box Information Model: persistent state and inputs, outputs
- Designing Use Case Steps
  - – Defining steps within a Use Case
- Defining System Operations
  - – Operations from use case steps
  - – Specifying operations using Information Model
  - – Black box ports and connectors


MAp Viewpoints
Information supports goals, process
Process supports goals
Domain / Biz Architecture
Domain: Information
Goals
Biz. Process
Outside
Goals – success criteria
Traceability
Mapped Black box Assembly (including S)
Black box assembly includes S, Supports business model
Component Architecture Black box Spec (of S)
S: Information
S: Behavior
Boundary
Black box view of S
White box view of S sub-component s1…s3 Partitions behavior & information
Architecture styles, patterns: across viewpoints, projects
Architecture Styles: Principles, Patterns, Guidelines, Rules
Component Architecture White box design (of S)
Inside
Information
Behavior
Each sub-component could be drilled-down if needed

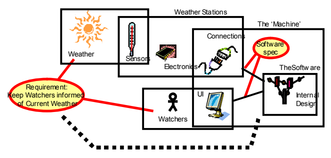
Preview of Black Box Architecture



Why do a Black Box Architecture?

- To envisage a solution to the problem and specify the external behavior of the system in terms of how it interacts with its environment, so as to rationalize its internal architecture.

Steps to Black Box Architecture

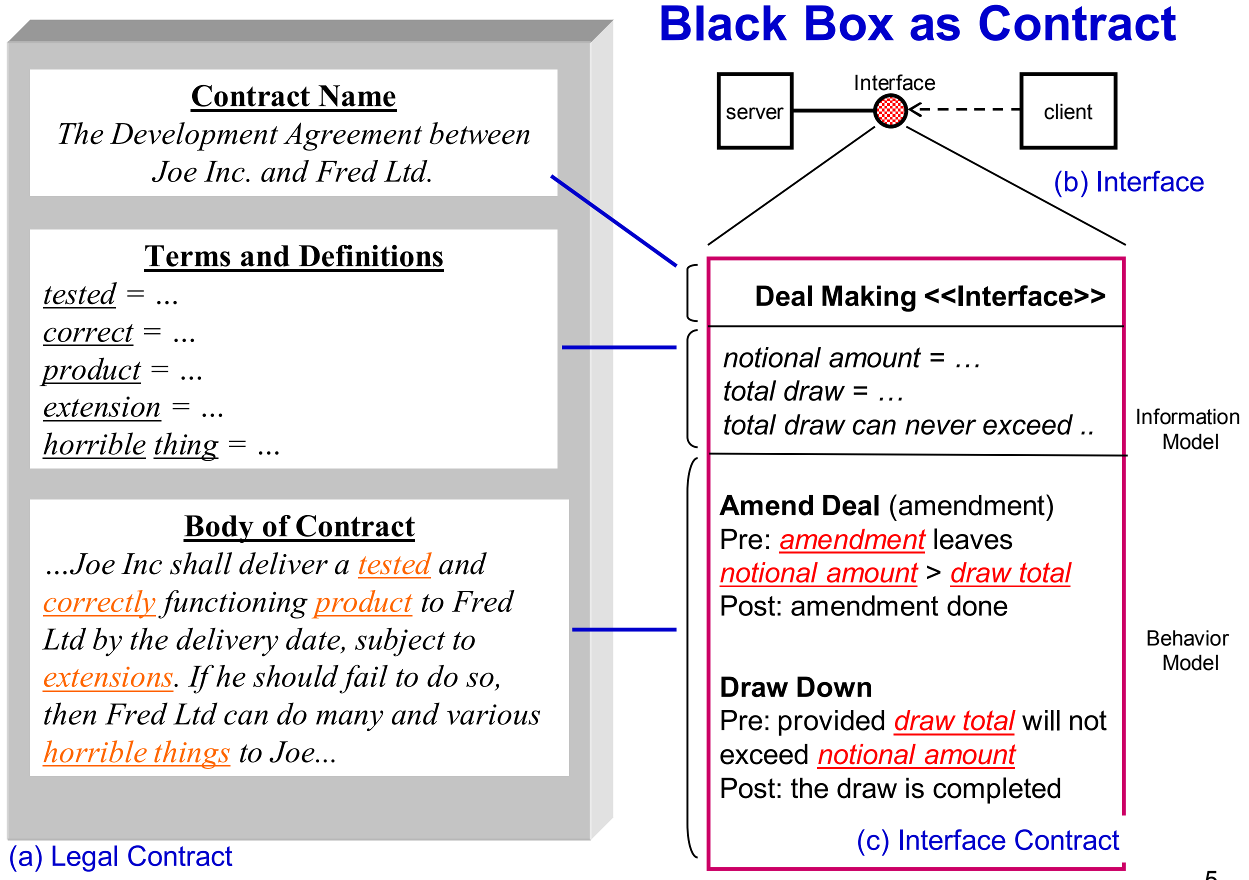


Iteratively do the following validating with end-to-end scenarios

- **Define and validate Mapped Black Box Assembly**
  - Identify large-grained boxes – new and existing
  - Assign responsibilities and needs by ownership of domain model
  - Derive Black-Box assembly
  - Validate with Scenarios and Snapshots

- **Black Box Specification (of one box)**
  - **Define (for your black-box of interest) the Use Cases and Operations**
    - Derive use cases and actors (direct and indirect) from business process, goals, or brainstorm
    - Specify each use case at level of pre/post condition as a single action
    - Design the steps of the use case as lower-level actions
    - Identify the Black Box Operations that will be invoked in those use case steps
    - Define all the black box ports and connectors to outside actors
    - Link use cases, actors from use case to activity and roles from business process
  - **Define the Black Box Information Model**
    - Use the Domain / Business Information Model as a starting point
    - Decide parts in scope for the black box Vs. parts that are managed by other systems
    - Include persistent and I/O types, software-specific types (sessions, preferences, etc.)
    - Validate that the information model supports the use cases and operations

Black Box as Contract



Contract Name

The Development Agreement between Joe Inc. and Fred Ltd.

Terms and Definitions

tested = ...
correct = ...
product = ...
extension = ...
horrible thing = ...

Body of Contract

...Joe Inc shall deliver a tested and correctly functioning product to Fred Ltd by the delivery date, subject to extensions. If he should fail to do so, then Fred Ltd can do many and various horrible things to Joe...

(a) Legal Contract

(b) Interface

Deal Making <<Interface>>

notional amount = …
total draw = …
total draw can never exceed ..

Amend Deal (amendment)
Pre: amendment leaves notional amount > draw total
Post: amendment done

Draw Down
Pre: provided draw total will not exceed notional amount
Post: the draw is completed

(c) Interface Contract

Define and Validate Mapped Black Box Assembly

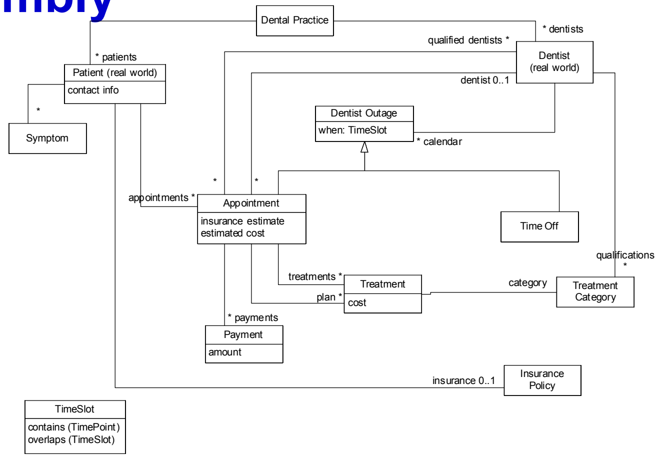
This section covers:

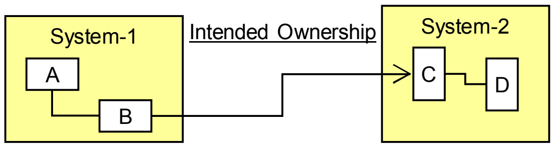
- Defining a Mapped Black Box Assembly

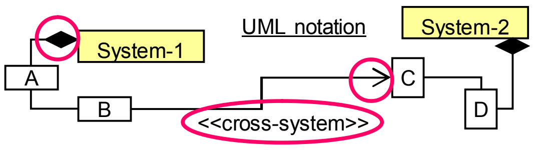
- We will build a Dental System that will work with
  - Insurance System
  - Vacation Calendar application
  - Dentists, Patients, Schedule Operator

Mapped Black-Box Assembly

- **(Work out example on next slide)**
- Which elements of the domain model are owned by (i.e. authoritative source)
  - Patient (real-world)
  - Dentists (real-world)
  - Vacation Calendar
  - Dental System
  - Insurance System
  - (You may need to refine domain model to show ownership)
- If ownership is split at the type or attribute level, split the element and introduce new finer-grained ones
- Mark <<cross-system>> associations
  - Ultimately we will want clear ownership of these as well: **indicate by arrow**
- Which external elements have copies explicitly represented in the Dental System
  - Add accuracy goal between software elements and domain counterparts if needed
- Do not use subtype relations across system boundaries – replace by associations

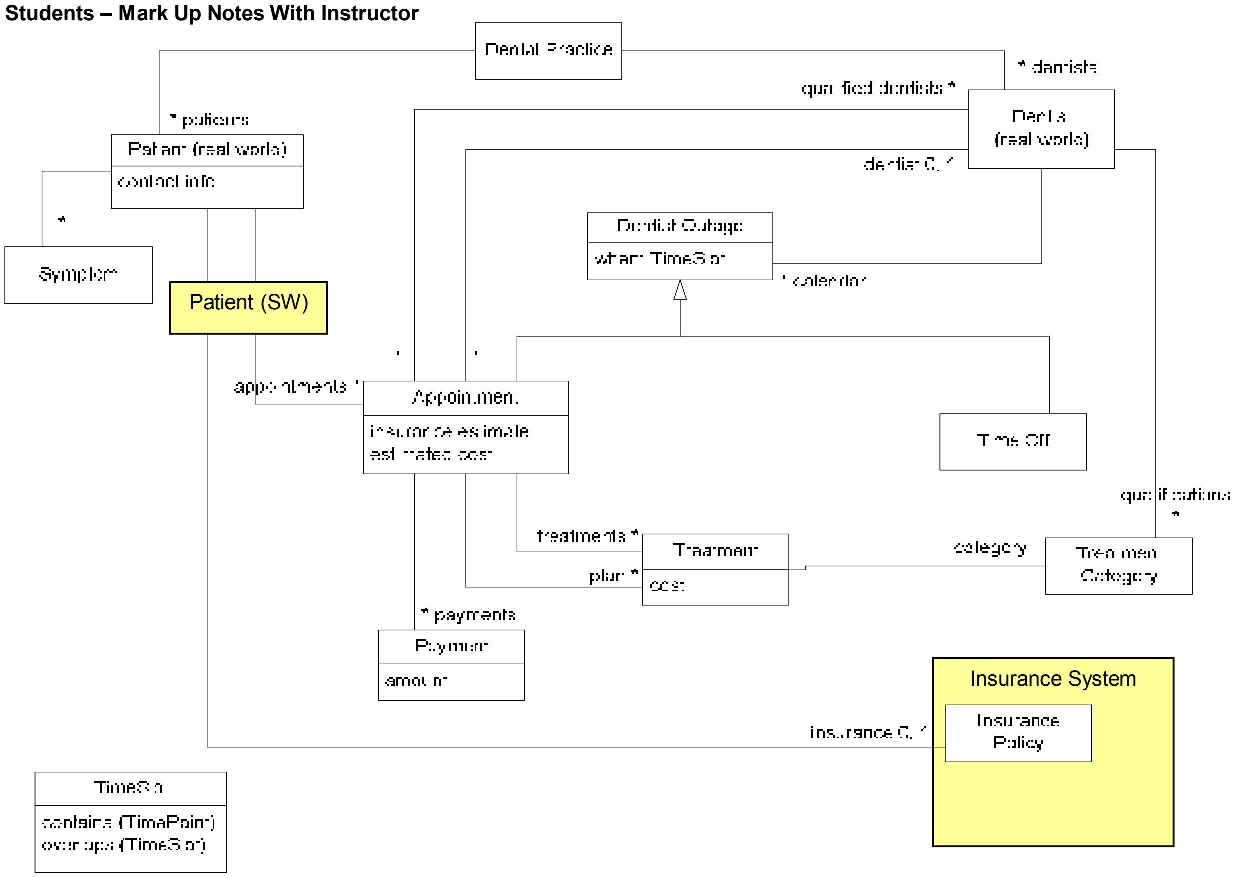






Text / Tabular notation …

Students – Mark Up Notes With Instructor



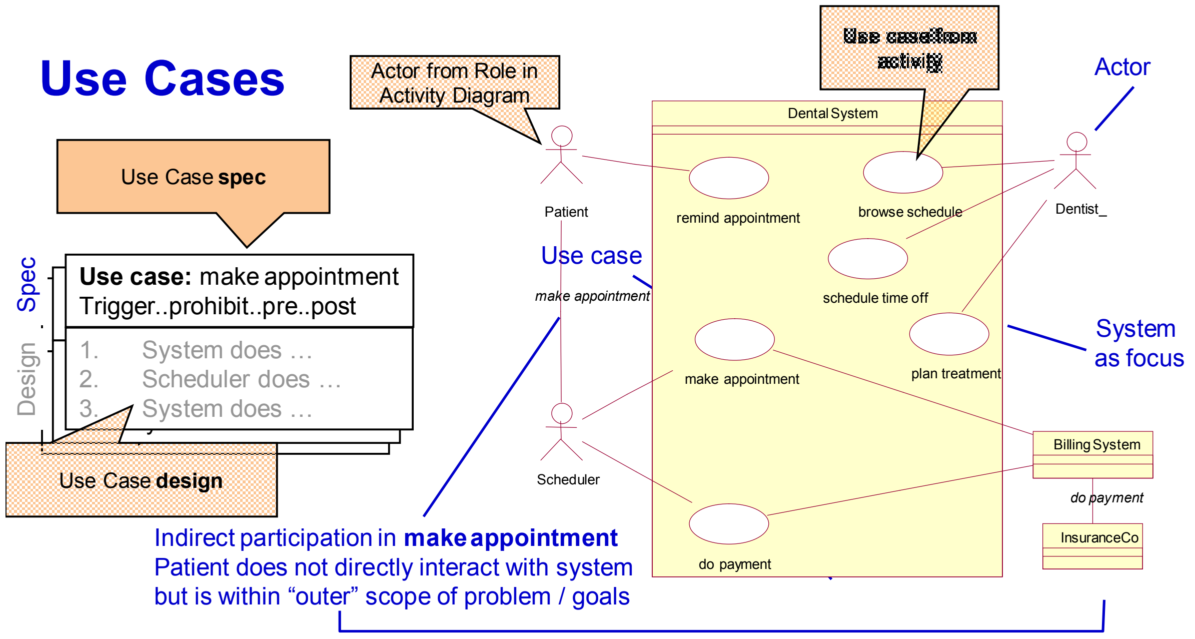
Exercise 6.1

Specifying Use Cases

This section covers:

- ♦ Building an Actor/Use-Case Diagram
- ♦ Specifying a Use Case

Use Cases



- Use case diagram: name, primary and other actors participating in each, included use cases
- Use case textual **specification** adds
  - Inputs and outputs, precondition and postcondition (successful outcome), other information
- Use case step **design** adds (still at the black-box level, and still deferring UI specifics)
  - Steps of use case, alternate and exception paths

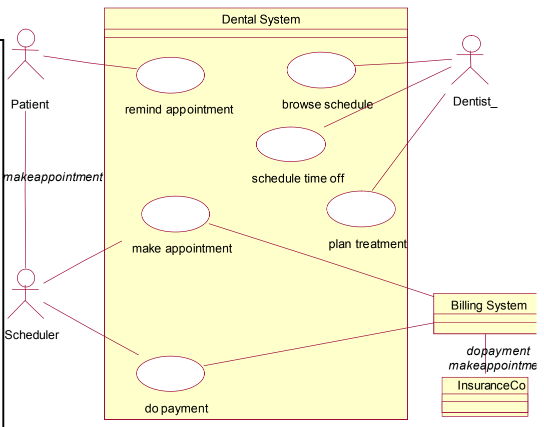
Link from Black Box to Business Model

- Preferably
  - One use case corresponds to one business activity or interaction
    - Use case **<<include>>** (later) can be used to show multiple activities if needed
  - Use case actors include the business roles in that activity or interaction
  - Simply name the use case and actors based directly on the business model
- If not, document in a table
  - For each use case, all business activities that it supports
  - For each actor, how it corresponds to the business roles

Use Case Spec – Expanded

Use case: make appointment (on Dental System) Spec
Actors: Patient, Scheduler, Billing System, InsuranceCo
Inputs: patient info,dates,symptoms (from Patient→Scheduler)
insurance info (from InsuranceCo via Billing System)
Outputs: treatments (to InsuranceCo via Billing System)
appointment info, cost, dentist (to Patient via Scheduler)
Goals: Which goals this use case supports
Trigger: Patient calls for appointment
Precondition: None
Postcondition:
System recorded new appointment on date for patient, suitable dentist for planned treatment and patient preferences. System requested insurance coverage for patient and planned treatment from Billing System and recorded estimated costs based on it. All appointment info was returned to Scheduler.

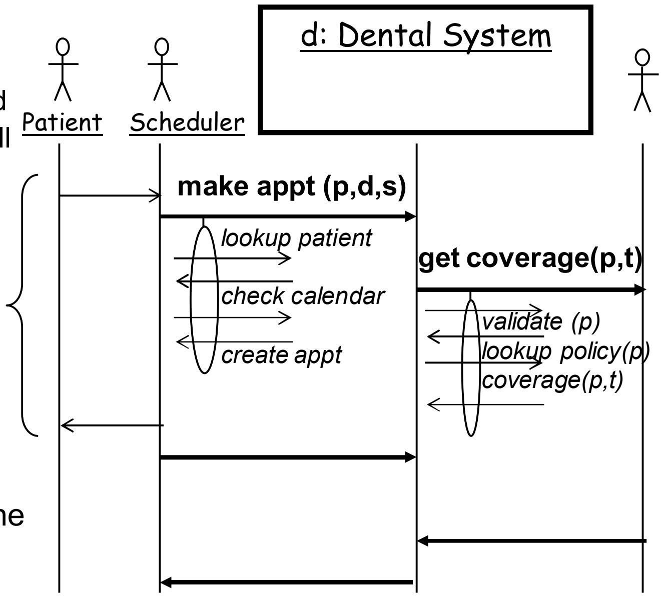
Steps:
1. ...
2. ...



Specification of post-condition should include:
• change of state of the "system"
• any requests made by system to outside actors
• any returned information from the system

Use Case Specification abstracts step details

- Defers interaction dialog of steps
  - Detailed effect of each step deferred
  - Abstracted and rolled-up into overall use case **pre/post**
- Defers protocol of information exchange
  - Each step information exchange deferred
  - Abstracted and rolled-up into use case **ins/outs**
- **Discuss:** What are the ins/outs of the editor "Cut" use-case? What is the detailed protocol of information exchange?

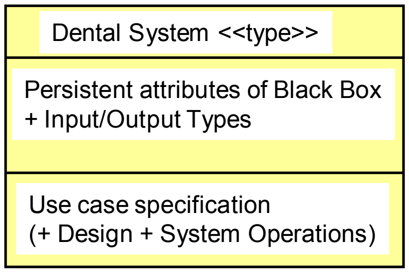


Granularity of Black-Box Application Use Case

- Define lowest level use-cases at the granularity of
  - An interaction which **meets a meaningful goal of the initiating actor**
  - A single short session with the black-box application (minutes to hours)
  - An interaction which is a meaningful business transaction (in that domain)

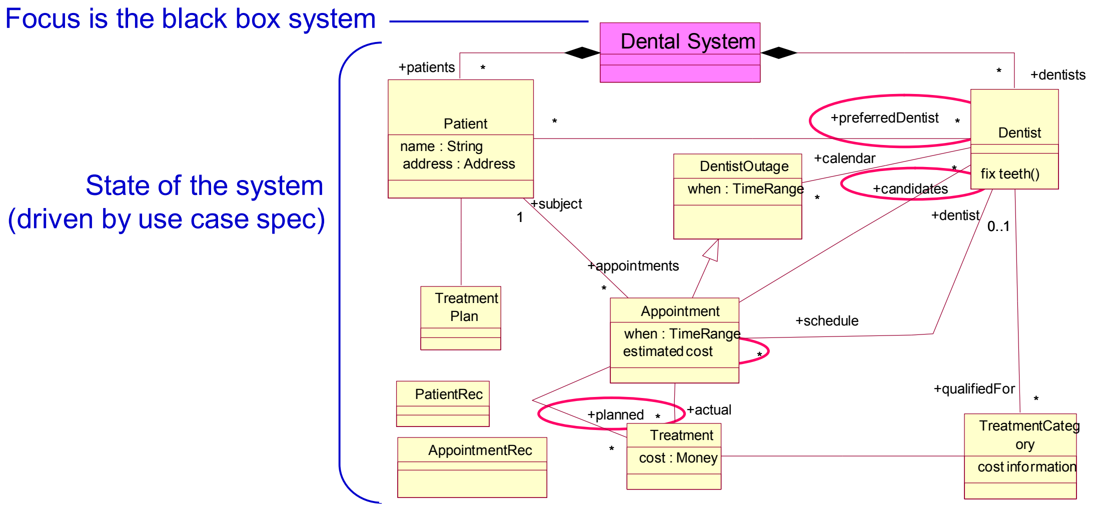
  - **Place an order**: good use-case granularity
  - **Login to system**: not good granularity
    - OK to define it as a use case, but …
    - Include a meaningful higher level use case that includes login
  - **Add line item**: not good granularity for use case, OK for a step
- **But …** Use case spec, step, and system operation are just relative terms
- Explore the steps and alternate paths of the use case, but …
  - Step back and specify the net inputs, outputs, and pre/post conditions
- Typical reasonable range of use case steps is 3 - 12 steps

Formalizing a Use Case Specification


Dental System <<type>>
Persistent attributes of Black Box + Input/Output Types
Use case specification (+ Design + System Operations)

- The use case specification should be expressed using the terms of the Black Box Information Model
  - This is a model of the Black Box System, its persistent attributes, and I/O types
  - The starting point for this is the Domain Model (Information)
- OK to elide some specifics of input/output types at this point
  - Specification of system operations from use case design can be more precise

Focus the Information Model for Use Case Specs



Use case: make appointment (on Dental System)
Actors: Patient, Scheduler, Billing System
Inputs: patient info, date, treatments
Outputs: appointment info with cost and dentist
Precondition: None
Postcondition: An appointment on date created for patient
with suitable dentist for planned treatment and patient preferences.
Insurance and cost estimates agreed with patient and recorded in system.

Exercise 6.2

Designing Use Case Steps

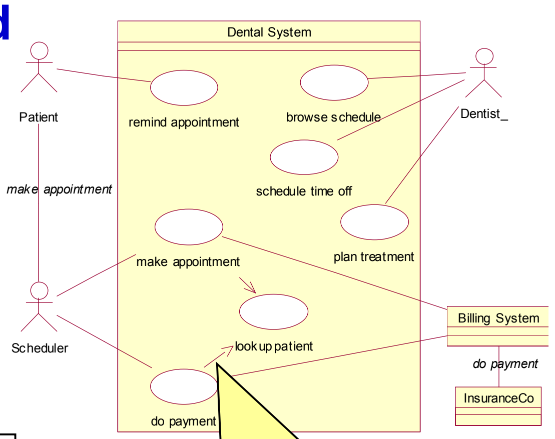
This section covers:

- ♦ Defining use case steps
- ♦ Exception and alternate paths

Use Case Design – Expanded

Use case: make appointment (on Dental System)
Actors: Patient, Scheduler, Billing System
Inputs: patient info,date,symptoms (from Patient via Scheduler)
insurance info (from InsuranceCo via Billing System)
Outputs: treatments (to InsuranceCo via Billing System)
appointment info, cost, dentist (to Patient via Scheduler)
Trigger: Patient calls for appointment
Precondition: None
Postcondition: An appointment created on date for patient, suitable dentist for planned treatment and patient preferences. Insurance and cost estimates agreed and recorded in system.

Spec



one use case "includes" another in its steps (or its "spec")

Main (success) steps:
Design
1. Patient calls Scheduler for appointment
2. Scheduler: *include look up patient*
3. Scheduler enters symptoms and date
4. System queries insurance info from Billing System
5. System: *include estimate patient cost*
6. System provides appointment, plan, staff, insurance, cost
7. Scheduler discusses with patient and confirms

Alternate paths:
2. Patient not found: …

use case lookup patient
Actors: Scheduler
…

use case estimate patient cost
post: …

Included use cases also have specs and/or step-level designs.

Step Details: Iteration, Alternatives, Exceptions

Use case: make appointment
1. Patient calls Scheduler for appointment
2. *Scheduler: Include look up patient*
3. Scheduler enters symptoms and date
4. System queries insurance info from Billing System
5. System provides appointment with plan, staff, cost
6. Scheduler discusses with patient and confirms

- Main use case steps focuses on primary “success path”
  - Iteration, grouping, concurrency … described informally with steps using step numbers:
  - *… Scheduler repeats steps 3-5 until satisfactory dates are found*

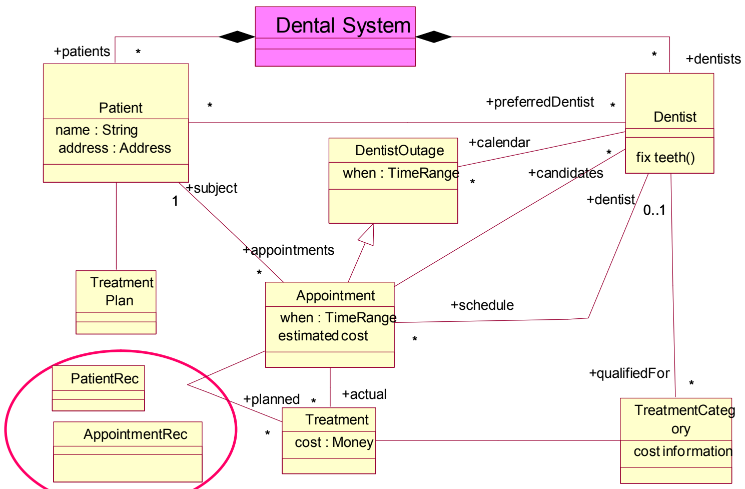
- Alternate paths are named and refer to step numbers in main path
  - **Alternate paths:**

    #2. Patient not found:
    1. Scheduler: include setup new patient before continuing

    #4. Billing System not available:
    1. System detects and signals failure to communicate with Billing System
    2. Scheduler informs patient and manually estimates cost

    **: System Failure (At any time the System fails):*
    1. Scheduler: include restart system, log in, and recover prior state

Elaborate Information Model from Steps



Steps will further clarify input and output types (UI or API will go further)

1. Patient calls Scheduler for appointment
2. Scheduler does lookup patient with System
3. Scheduler enters symptoms and date
4. System queries insurance info from Billing System
5. System provides appointment,plan,candidates,cost
6. Scheduler discusses with patient and confirms

Alternate paths:

2. Patient not found:

Overall Use Case Writing Guidelines

- Use case specification
  - Name a use case by the goal of the primary actor, or the outermost trigger
    - Make appointment
  - List all actors involved
    - Scheduler, Billing System
  - Summarize effective inputs and outputs to/from each actor to the System
    - Patient Info, Dates
  - Write the precondition for this use case to happen (and trigger, prohibition)
    - Order Agent is already logged in
  - Write down the postcondition for on successful use case completion
    - A new Appointment has been …
- Use case steps
  - **Write each step as *actor* … *action* … [with *actor* | other phrase]**
    - System queries insurance information from Billing System
  - Avoid too detailed or intra-actor sequencing
    - Avoid: “System prompts for name, user enters name, system prompts…”
  - Avoid UI specifics
    - User clicks on OK button → User confirms selection

Exercise 6.3

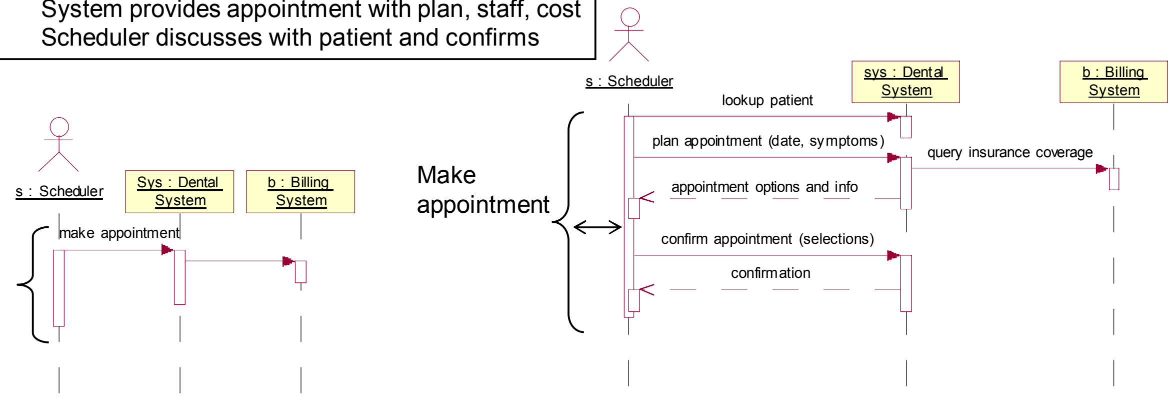
Defining Black Box System Operations

This section covers:

- ♦ Finding System Operations
- ♦ Specifying System Operations

From Use Case to System Operations

Use case: make appointment
1. Patient calls Scheduler for appointment
2. Scheduler *includes look up patient* with System
3. Scheduler enters symptoms and date
4. System queries insurance info from Billing System
5. System provides appointment with plan, staff, cost
6. Scheduler discusses with patient and confirms



- ♦ A use case defines a complete unit of interaction with the system
- ♦ The use case steps hint at "operations" called on the system or by the system
  - – Operations can be API level, or can still abstract more detailed interaction protocol
- ♦ Scenarios and sequence diagrams can be explored at either or both levels
- ♦ A use case or a use case step could correspond to several system operations

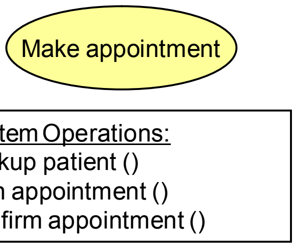
Finding System Operations

- For each step in a use case
  - Identify any operation(s) invoked on the system
    - With any input data provided as parameters and any returned values
  - Identify any **<<out>>** or **<<event>>** interactions out of the system to other actors
    - With any data provided and returned from those calls out
    - A **<<out>>** method call is a service requested, with some expected response
    - An **<<event>>** is an outgoing notification, without any expected response or handling
  - Check that the information model defines the types and attributes required

- Summarize as
  - All operations the system must support: **operation (…)**
  - All operations required from other actors: **<<out>> operation (…)**
  - All events raised by the system to other actors: **<<event>> event (…)**

- Write out the specification of each non-trivial operation
  - Name, Input parameters, Return values
  - Precondition: what should have been true at the start of this operation was invoked
  - Postcondition: what should be the resulting state and any outputs upon completion
    - State-changing **<<out>>** operations might need a model of state of the **external** actor
    - e.g. **<<out>>** put(x) followed by **<<out>>** get():x requires model of state of putter/getter
  - [Exceptions]

Operations Example



System Operations:
Lookup patient ()
Plan appointment ()
Confirm appointment ()

- lookup patient (id: Patient Identification): Patient Info
  find the patient based on patient identification
  post: returns the patient whose identification matches id

- plan appointment (p: PatientID, symptoms: Set(Symptom), dates: Date Range)
  : Set (Appointment)
  return set of Appointment options which meet patient treatment plan, symptoms, and desired dates, with a suitable dentist and cost estimate
  post: returns multiple appointment options
  - with time within dates
  - with a treatment plan based on symptoms and existing treatment for that patient
  - with a dentist who is a candidate for that appointment and is available at that time
  - with cost estimate based on insurance information from the billing system

Exercise 6.4

Finding Supporting or Missing Use Cases

- It is easy to overlook some use cases and actors if domain model and black-box assembly map is not well done
- For each type in the Information Model, ask
  - What event in the domain would cause a new instance of that type to exist?
  - What use case creates a new instance of that type?
  - What use cases remove it?
  - Dental System: what creates patient, dentist, treatment category?
    - Sometimes “hardcoded” into code or configuration files, with no run-time use cases
- For each attribute or association, ask
  - What event in the external world would correspond to a change in that attribute or association?
  - What use cases could set or modify that attribute or association?
- For each use case, ask – What if the use case was abandoned somewhere in between?
- For each actor, ask – What sequence of its use cases would make sense? What would they accomplish?

- What this uncovers
  - missing use cases: there should be a use-case for managing that type, or different use-case path
  - external shared “reference data”: that type ownership belongs elsewhere, we may cache the info
  - initialization or other lifecycle requirements: that type is instantiated at startup

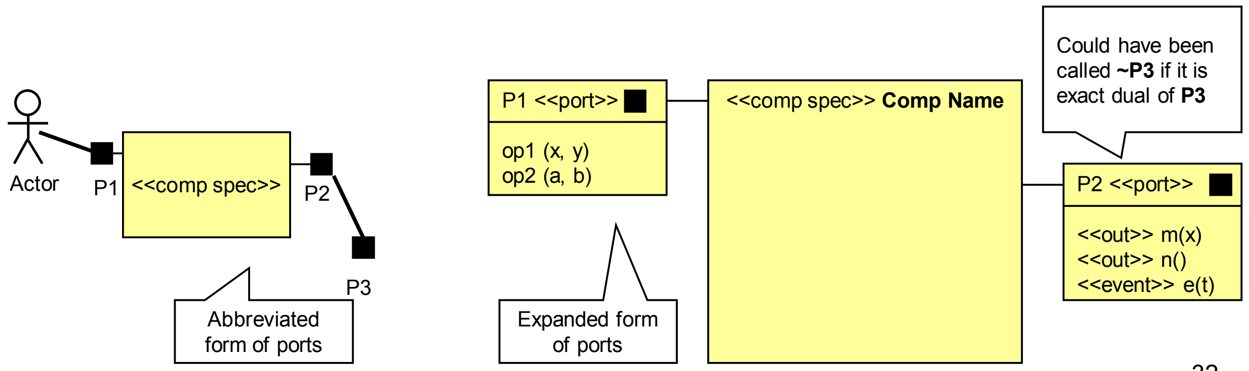
- Do not document purely CRUD use cases in detail
  - If you must mention, simply annotate information model types with <<crud>>

[Optional] Exercise 6.5

31

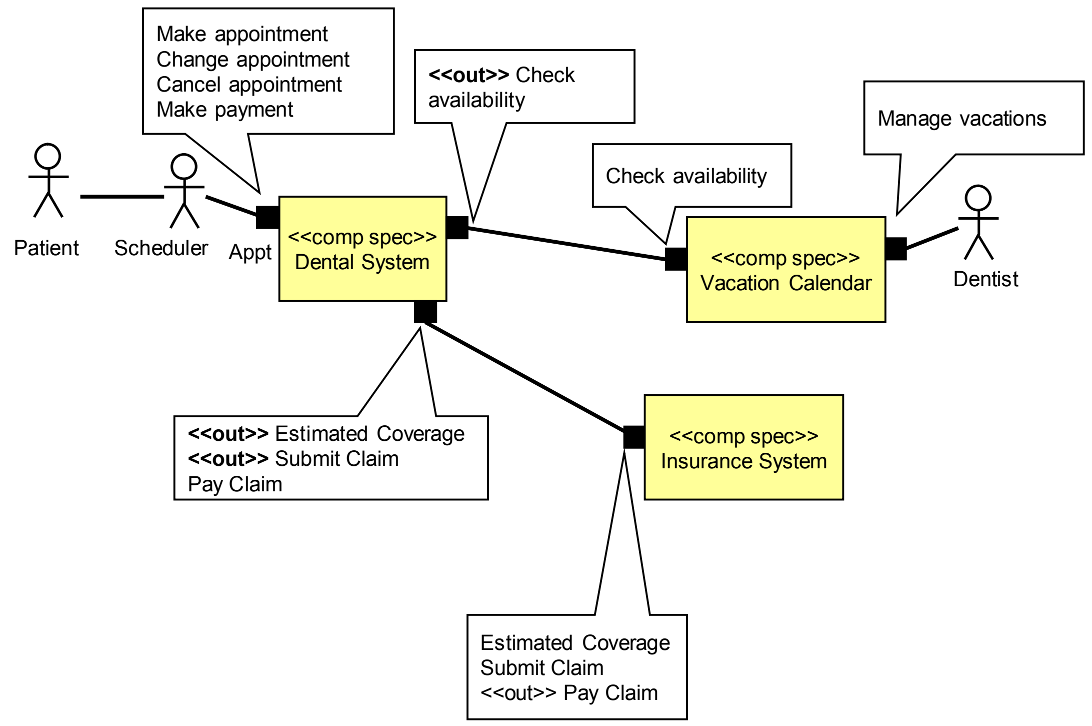
Black Box Assembly with Ports and Connectors

- Each connection to an external actor can be summarized as
  - a "Port" on the black box
    - With a list of incoming operations, **<<out>>** outgoing operations, **<<event>>** events, etc.
  - a "Connector" to the corresponding external actor
    - If another system, connector actually connects to a particular port on that system
- Port is like an Interface
  - Generalizes UML1.x "Interface"; more directly supported in UML 2.0
  - Specified like a Type



32

Dentist Black Box Assembly



Defining "Other Properties" of the Black Box

- There are other properties (including non-functional and technical) that are visible from the outside of the black-box
- MAp allows such other properties to be described on the black box
  - Technical properties of ports and connectors
    - Communication, data formats, specific technical infrastructure elements used and how configured
  - Performance properties
    - Expected timeliness of information and timeliness guarantees
    - Response times, throughputs, transaction rates (see Transactions below) under different load conditions
  - Transactions
    - Transaction scope – existing transaction expected with operation requests? new transaction guaranteed?
    - Isolation levels – visibility of uncommitted data: lost updates, dirty reads, etc.
  - Security
    - Roles, identities, entitlements and access control: information expected, guarantees made
  - Deployment / Location
    - Relevant geographic or network regions and properties for components and actors
    - Relevant properties of connectors

Exercise 6.6

Black Box Architecture – Some Guidelines

- Black-box assembly: include context to end-points of goals
  - E.g. Source of information being entered by Scheduler is the Patient
  - "Real" requirement is to make appointment for patient
  - "End-to-end" scenarios should span the entire assembly
- For a use cases, include at least immediate context of target black-box system
  - E.g. Scheduler entering information into system, other external systems
- Consider longest-lifecycle business activity and then finer-grained ones
  - E.g. Patient does a complete visit, from appointment to final payment
- Define clear responsibilities of system and external actors
  - Include information exchanged, change of state, external communications
- Focus business information to black box information model
  - What information does it provide, require, and remember?
- Validate with selective scenarios

Summary

This chapter has shown that:

- ♦ The definition of software boundary and services is a creative process
- ♦ Black box specification is based on use cases and underlying operations
  - – Use cases are derived from business architecture – goals, processes
  - – Use cases are **specified** by end result, **designed** as sequence of steps
  - – More detailed system operations emerge from the steps of a use case
- ♦ All behavior is described in terms of the Black Box Information Model
  - – Includes persistent attributes of the system and input/output types
- ♦ The Business Information Model forms the basis for the Black Box Info Model
- ♦ The Business Goals & Process Model is basis for the Use Cases

Chapter 7

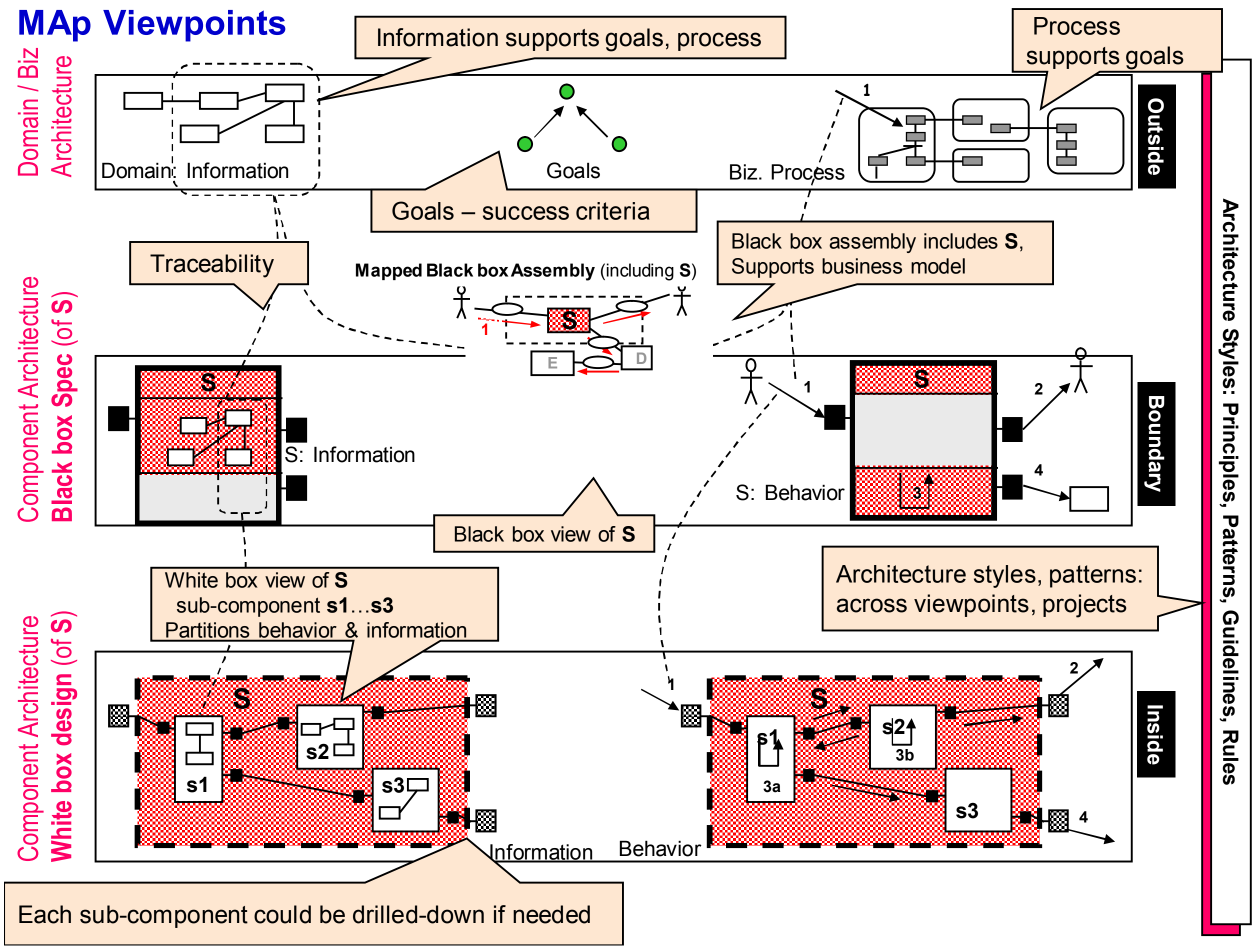
How to Design a Component's Internal Architecture

White box View of <System X>

This chapter describes how to design an assembly of smaller black box sub-components to meet a larger black box component specification, while managing dependencies between the sub-components.

It covers:

- ♦ Draft a partition into sub-components
- ♦ Design sub-component collaborations
- ♦ Define sub-component assembly
- ♦ Specify each sub-component



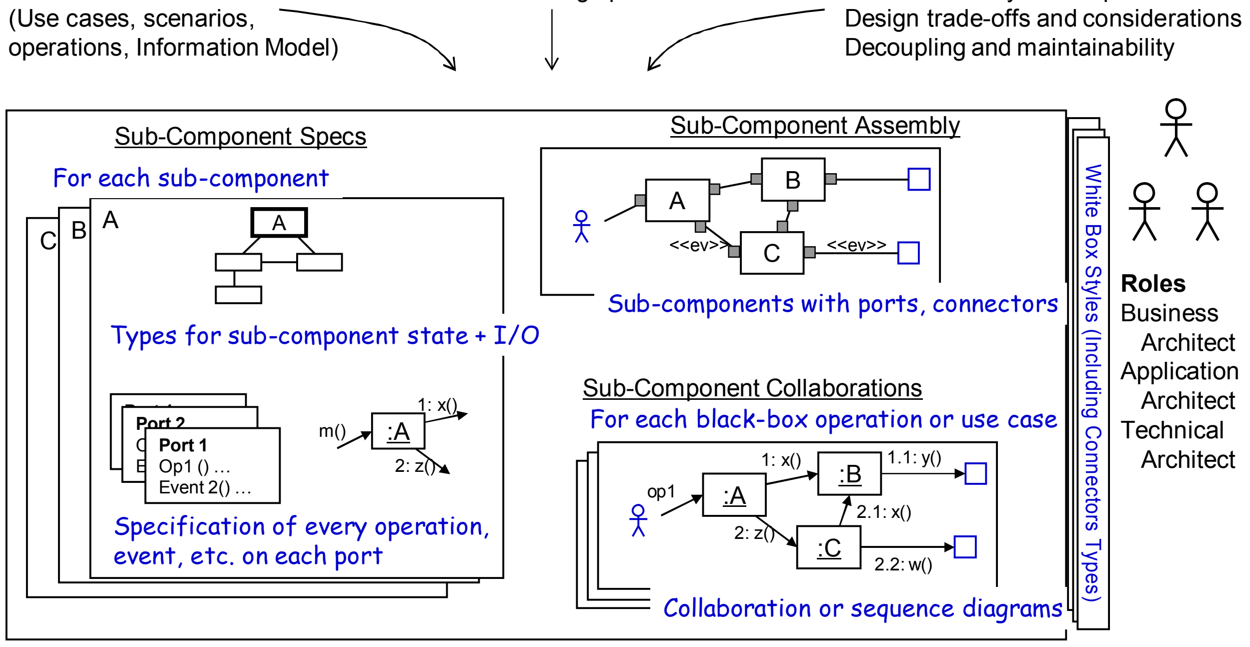
MAp Viewpoints
Domain / Biz Architecture
Component Architecture Black box Spec (of S)
Component Architecture White box design (of S)
Information supports goals, process
Process supports goals
Domain: Information
Goals
Biz. Process
Goals – success criteria
Outside
Traceability
Mapped Black box Assembly (including S)
Black box assembly includes S, Supports business model
S: Information
S: Behavior
Black box view of S
Boundary
White box view of S sub-component s1…s3 Partitions behavior & information
Architecture styles, patterns: across viewpoints, projects
Architecture Styles: Principles, Patterns, Guidelines, Rules
s1
s2
s3
3a
3b
Information
Behavior
Inside
Each sub-component could be drilled-down if needed

Preview of White Box Architecture

Black Box Architecture
(Use cases, scenarios,
operations, Information Model)

Creative design partition

Architectural styles and patterns
Design trade-offs and considerations
Decoupling and maintainability

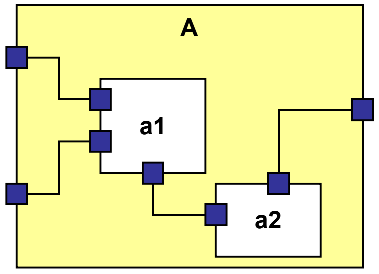


Sub-Component Specs
For each sub-component
C
B
A
A
Types for sub-component state + I/O
Port 2
Port 1
Op1 () …
Event 2() …
m()
:A
1: x()
2: z()
Specification of every operation, event, etc. on each port
Sub-Component Assembly
A
B
C
<<ev>>
<<ev>>
Sub-components with ports, connectors
Sub-Component Collaborations
For each black-box operation or use case
op1
:A
1: x()
:B
1.1: y()
2.1: x()
2: z()
:C
2.2: w()
Collaboration or sequence diagrams
White Box Styles (Including Connectors Types)
Roles
Business Architect
Application Architect
Technical Architect

Why do a White Box Architecture?
- ♦ To design sub-components that collaborate to provide required behaviors
- ♦ To separate logical components and interactions from technical ones

WARNING !! Black-Box and White-Box Dangers!

- ♦ The terms Black-Box and White-Box sometimes lead to confusion
  - – Should I deliver the white-box?
  - – Should I demand the black-box?



- ♦ **Always** be explicit in all usage of the terms
  - – This is the Black-Box View **of Component-A**
    - • Consisting of these ports with behavior specs and information model
  - – This is the White-Box View **of Component-A**
    - • Consisting of these sub-components A1, A2, A3
    - • With these (black-box view) specifications of A1, A2, A3
    - • With these connectors between them

Steps to White Box Architecture

- Choose applicable architecture styles for white box
  - Including connector styles (we assume simple call and event connectors here)
- Draft a partition into candidate sub-components
  - Partition black box information model to get **information-cluster** components
  - Introduce **behavior centric** coordination components for use cases
  - For existing components or vendor packages to be used build "just-enough" models of them
- Design collaborations across sub-components
  - Use the Black Box operations as a starting point
  - Design collaboration between sub-components to meet spec and maintain invariants
  - Validate with scenarios (including performance concerns) and re-factor the partition
  - Use executable tests with stubs where possible
- Define Sub-Component Information Model
  - Derive the information that is managed and exchanged by each component
- Define Sub-Component Assembly
  - Show all sub-components, named ports with operations, events, connectors
- [Optional] Specify each sub-components and connector used in assembly
  - All operations, events, etc. at all ports of each sub-component

Draft Sub-Component Partition

This section covers:

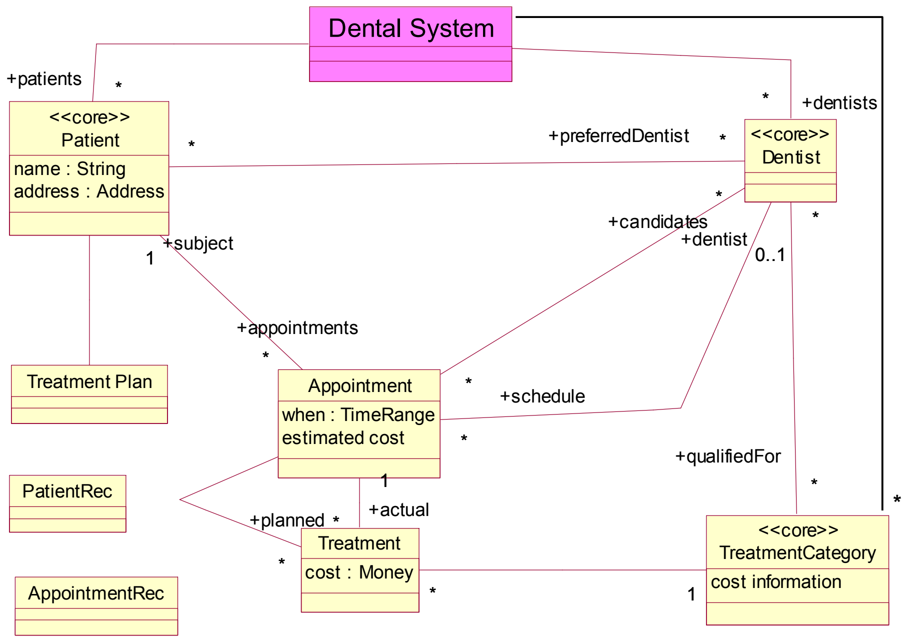
- Information-cluster components with core type analysis
- Behavior/coordinator components from use cases
- Other heuristics

- **Note:**
  - There are many different styles of partitioning sub-components
    - Workflow, n-tiered, co-ordinator based, federated vs. centralized information, …
  - Each is applicable in different situations
    - Nature of problem, non-functional requirements, existing components … all have an impact
  - This course teaches a few specific heuristics and styles, and mentions others

Information Cluster Partition based on “Core” types

- Identify “core” types – they typically satisfy these heuristics
  - Can easily be uniquely identified in business terms and business identifiers
  - Have associations to (owned) detailing types, with constant “1” at core end
  - Often need an association from the “top” type which has just one instance
  - Do not have mandatory (always 1) associations to other types
    - unless they are classifying types: classifying types serve primarily to categorize or group objects of another type e.g. Book Title vs. Book Copy
    - but be prepared to bend this heuristic to further partition and de-couple e.g. distinct lifecycles
- Ignore derived associations and attributes and ignore the “top” system type
  - separately consider whether derived attributes merit a calculation-centric component
- Consider further separating objects that have distinct and complex lifecycles (with their dependent types) and also treat a “core” types
- Consider finer-grained ownership at the attribute level rather than type, if needed
- Mark such types as «core»
- Treat types whose existence is dependent on a core type as ‘detailing’
  - Will typically have a constant “1” association to owning type from dependent type
  - Sometimes might need to break a tie and assign to one core type

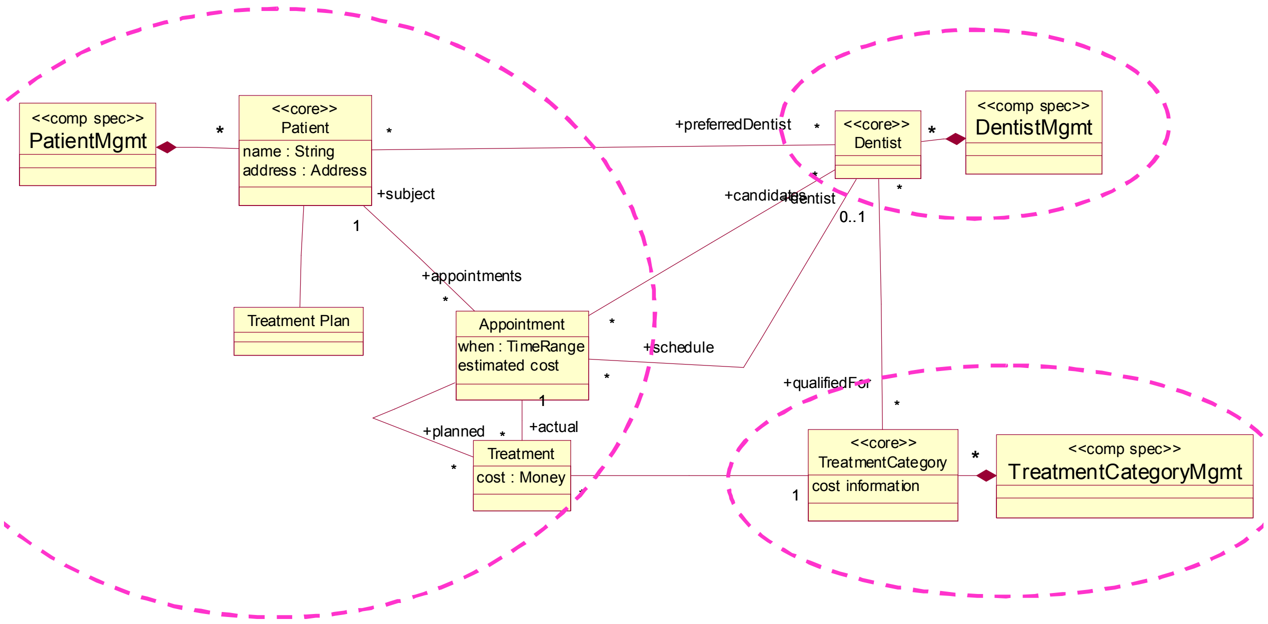
Core Type Analysis - Example



We have simplified the black box model somewhat e.g. removed time off and Dentist Outage supertype

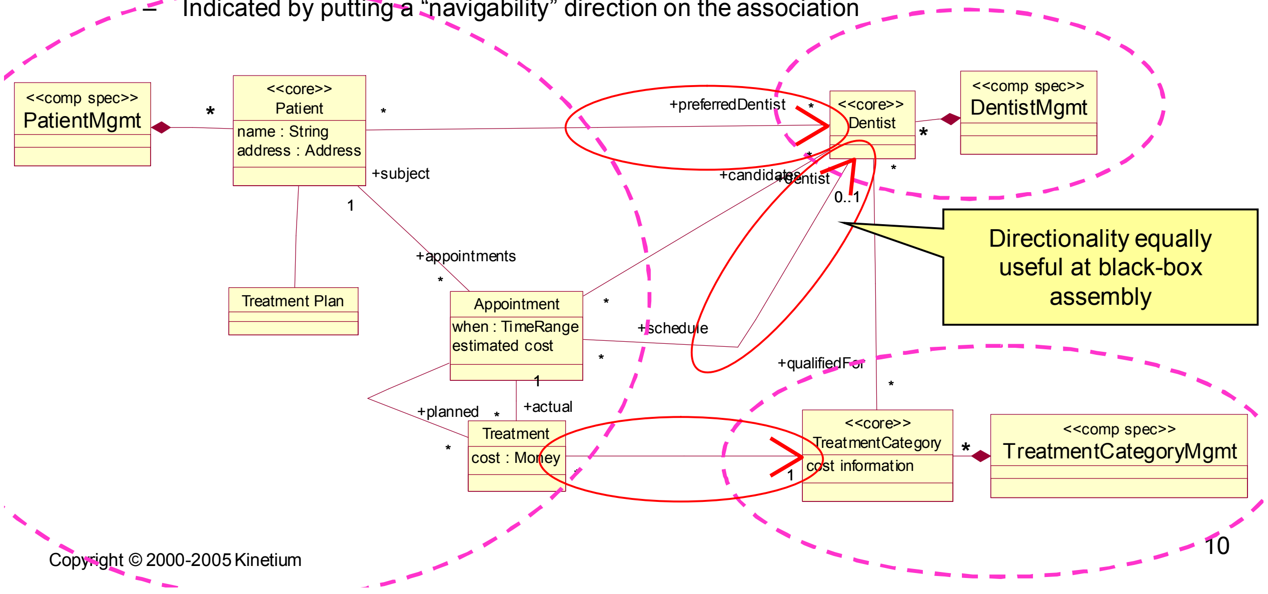
Assign Core Types to Initial Components

- Introduce a component to manage each "core" type (and its detailing types)
  - Components marked with «comp spec» for component specification
  - More advanced: analyze cross-component invariants, split attributes across components
- Show assignment by UML "composition"



Assign Initial Direction to Associations

- When types managed by separate components have cross-component associations
  - We need to decide which side will be the **golden source** of the links
  - Bi-directional dependencies, particularly without clear master, are not encouraged
  - Replication & caching is treated separately, once there is a clear golden source
- Select one side to be the owner of the relationship
  - Anticipate likely direction of frequent navigation (validate later by collaborations)
    - e.g. Derived attributes may need navigate assoc. to "push" changes or to "pull" and re-compute on demand
  - Indicated by putting a "navigability" direction on the association

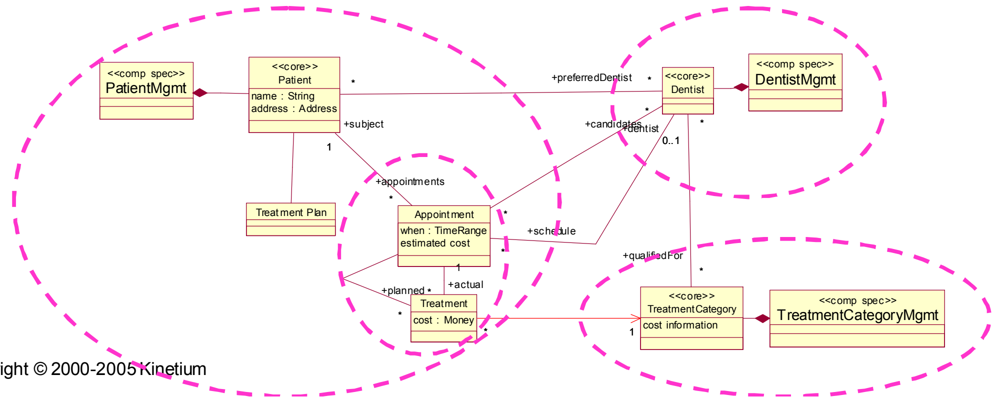


Result of Initial Core Type Partition

- A set of initial components
- The information model managed by each component
  - Core types, detailing types, cross-component association direction

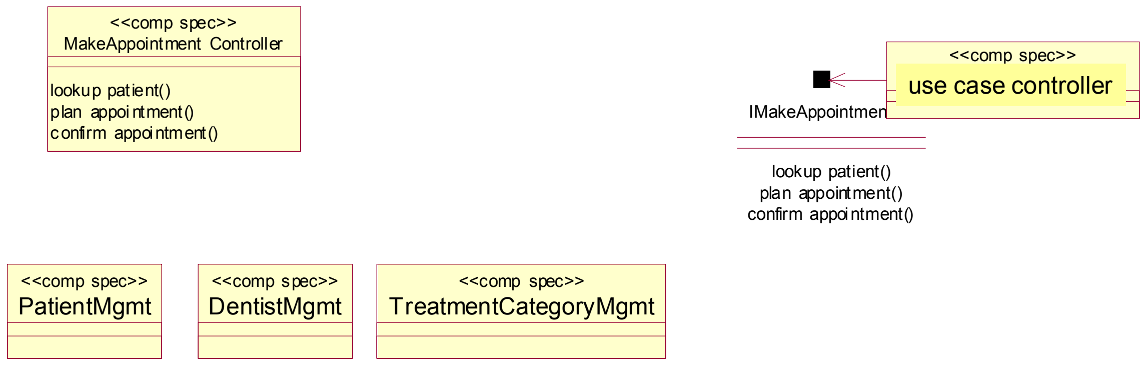
Other partitioning heuristics for Sub-Components

- **Introduce component to co-ordinate the interactions for each use case**
- Introduce a sub-component for each black box port to adapt to that port view, format, etc.
- Finer-grained partition by analysis of the life-cycle of objects
  - If one object lifecycle spawns others with loosely-coupled lifecycles, use separate components
  - E.g. Appointment lifecycle within Patient lifecycle?
- Partition to depend more on stable components than unstable ones
- Introduce computational component e.g. compute derived attributes, associations
- If you find variability in behavior, encapsulate variability within a stable interface
  - Add a component which hides that variability behind an unchanging interface
- Invent generic reusable business service components – tax calculator, calendar
- Goals model: align components with sub-goals and with corresponding domain ownership
- Informal, intuition, experience, educated guesswork

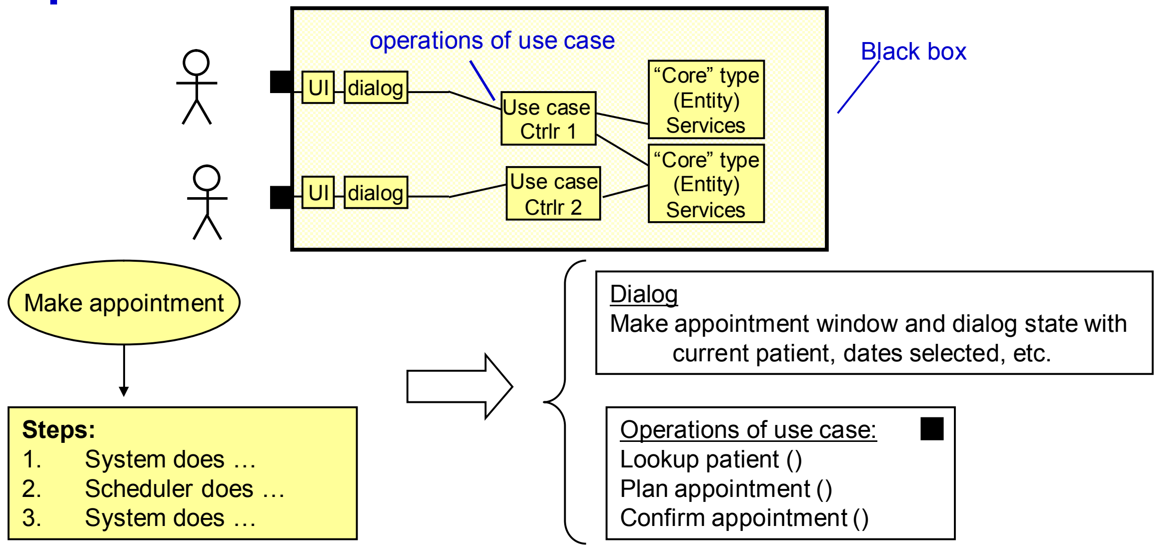


Add Use-Case Supporting Components

- Introduce a component to support system operations of each use case <U>
  - If too many use cases, use a «port» per use case on one single «comp spec»
  - Typically name that component something like <U>Controller, <U>Coordinator
- These components will be the "receivers" of the external requests
  - Via some GUI or other connection to the outside actors …
  - Result in some interaction with and between the other components, internal & external



Component Tiers

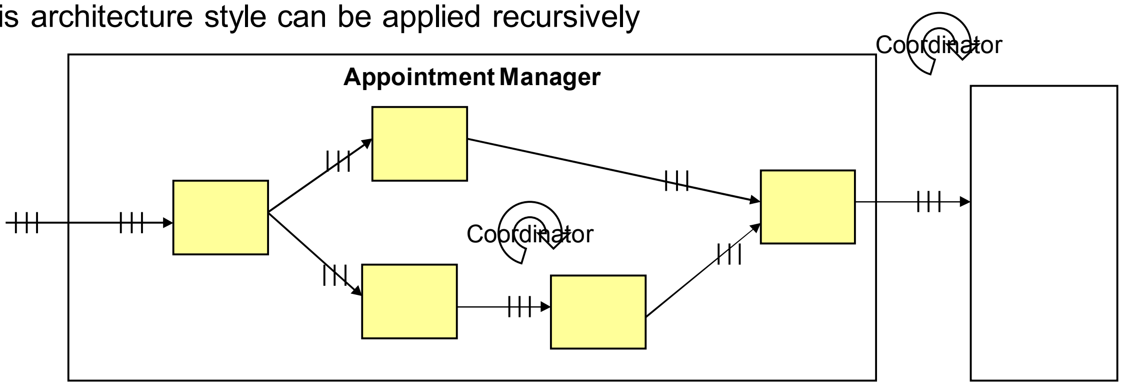


- Use cases steps, interim state, and transactions map to white-box tiers
  - User Interface and dialog state for use case steps, invoking …
  - … System operations required by the steps of the use case, realized by utilizing …
  - … … Further partitioned clusters of business types

Exercise 7.1

[Optional] Workflow Based Component Architecture

- When you have one <<core>> type cluster where many instances of the core type go concurrently through phases of a lifecycle
  - an Appointment goes through a lifecycle
- It can be useful to design the insides of the <<core>> type component with sub-components to perform processes in a workflow, if “flow” is highly variable
  - Ports acts as sources and sinks for stream of uniform data type
  - New connector type represents queue of instances awaiting processing
  - A workflow manager who co-ordinates the flows based on events and state conditions
- One component for each major state transition on the core type
  - Components AppointmentMaker, AppointmentChanger, AppointmentCloser
- This architecture style can be applied recursively



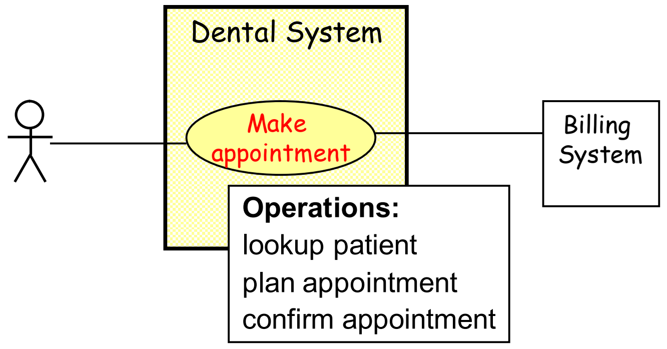
Design Collaborations across Sub-Components

This section covers:

♦ Designing collaborations across sub-components

♦ Collaboration Vs. Sequence Diagrams

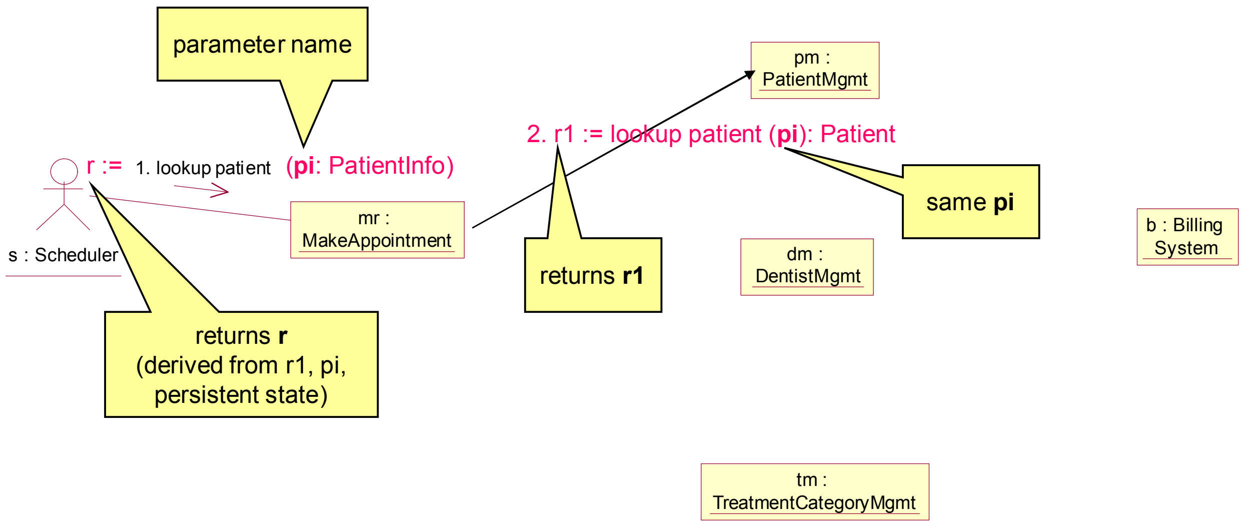
White Box Collaborations – Starting Point



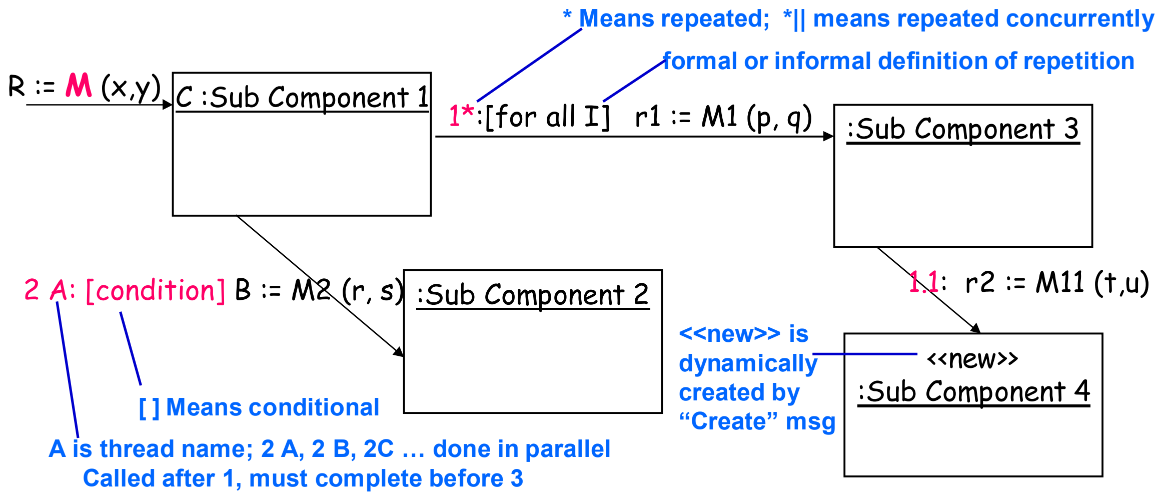
Operations:
lookup patient
plan appointment
confirm appointment

♦ Start with a use case and all the operations from its steps
♦ For the significant operations
  – study the operation specification
  – design end-to-end interactions across sub-components so that
    • operation post-condition is met
    • all external interactions are completed
♦ Cover use case normal and exception paths
♦ Validate end-to-end scenarios from business and black box architectures
♦ Re-factor partitions and responsibility, making interfaces generic

Make Appointment: Lookup Patient (UML Collaboration/Communication Diagram)


parameter name
r := 1. lookup patient (pi: PatientInfo)
s : Scheduler
mr : MakeAppointment
returns r (derived from r1, pi, persistent state)
2. r1 := lookup patient (pi): Patient
pm : PatientMgmt
returns r1
same pi
dm : DentistMgmt
b : Billing System
tm : TreatmentCategoryMgmt

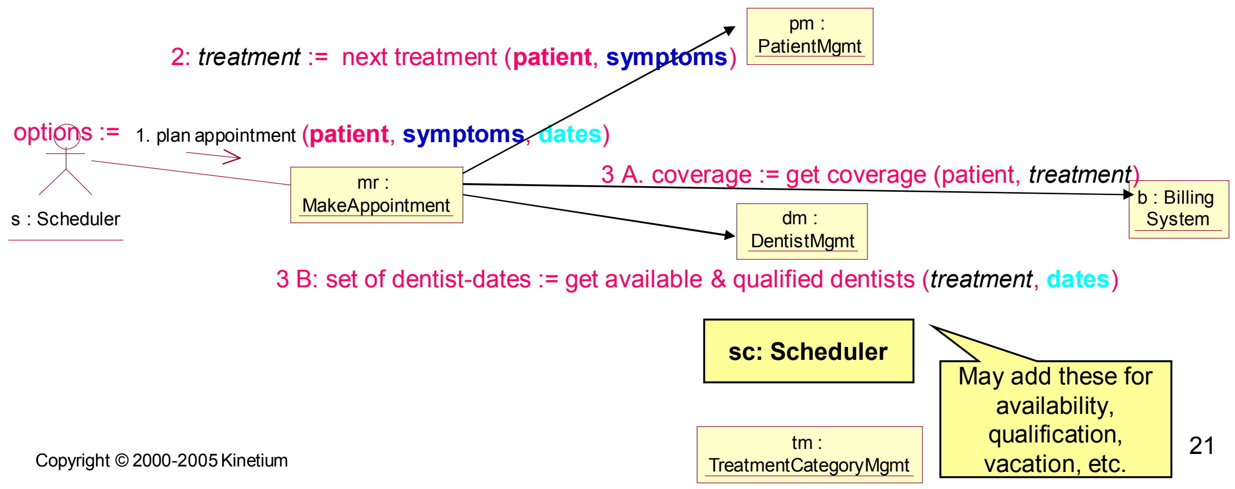
[Optional] UML Collaboration/Communication Diagrams


* Means repeated; *|| means repeated concurrently
formal or informal definition of repetition
R := M (x,y)
C :Sub Component 1
1*:[for all I] r1 := M1 (p, q)
:Sub Component 3
2 A: [condition] B := M2 (r, s)
:Sub Component 2
1.1: r2 := M11 (t,u)
<<new>> is dynamically created by "Create" msg
<<new>> :Sub Component 4
[ ] Means conditional
A is thread name; 2 A, 2 B, 2C … done in parallel
Called after 1, must complete before 3

- Nested numbering for sequence of operation invocations
- Each operation **M** (x, y) on a sub-component **C** specified by:
  - Sequence of invocations from **C** in response to operation invocation **M**
  - Relationship between x, y, and initial state of C → p, q, r, s
  - Final state + final returned values of **C** in terms of x, y, initial state, A, B
- **Note: MAp Interpretation shows sequence, not hard-coded method calls**
  - **We will separately choose types of connectors and ports on components**

Make Appointment: Plan Appointment

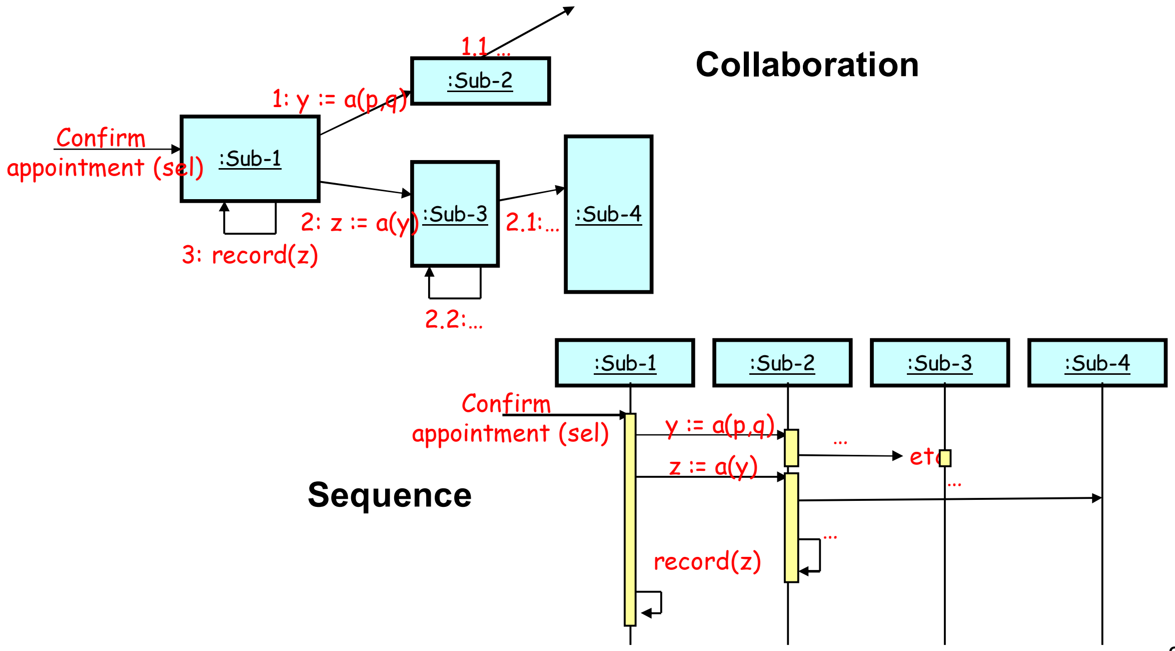
- plan appointment (p: PatientID, symptoms: Set(Symptom), dates: Date Range)
  : Set (Appointment)
  return set of Appointment options which meet patient treatment plan, symptoms, and desired dates, with a suitable dentist and cost estimate
  post: returns multiple appointment options
  - with time within dates
  - with a treatment plan based on symptoms and existing treatment for that patient
  - with a dentist who is a candidate for that appointment and is available at that time
  - with cost estimate based on insurance information from the billing system


pm : PatientMgmt
2: *treatment* := next treatment (**patient**, **symptoms**)
options :=
1. plan appointment (**patient**, **symptoms**, **dates**)
s : Scheduler
mr : MakeAppointment
3 A. coverage := get coverage (patient, *treatment*)
b : Billing System
dm : DentistMgmt
3 B: set of dentist-dates := get available & qualified dentists (*treatment*, **dates**)
sc: Scheduler
May add these for availability, qualification, vacation, etc.
tm : TreatmentCategoryMgmt

Exercise 7.2

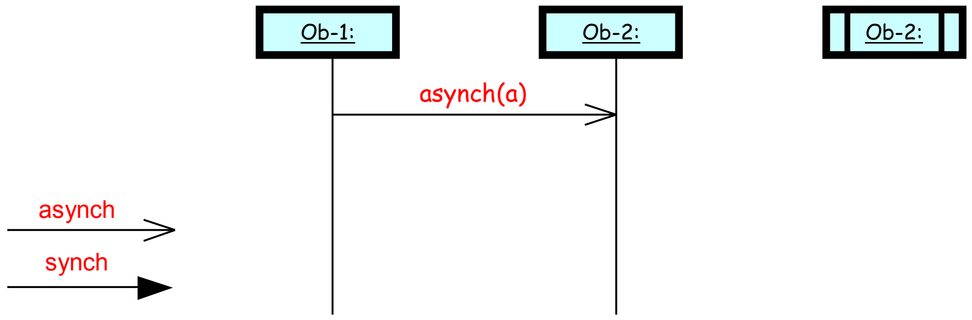
Collaboration versus Sequence Diagrams

- Collaboration and sequence diagrams are alternate diagrams of interactions
- Collaboration style helps build up “overlays” to get to the assembly definition



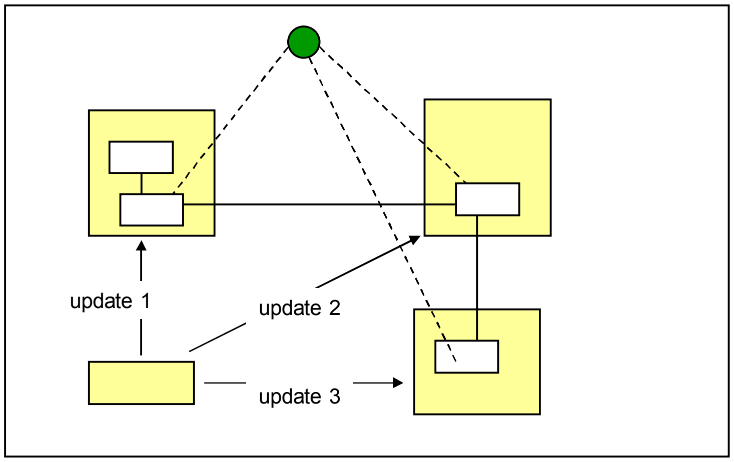
[Optional] Active Objects and Asynchronous Messages

- UML supports asynchronous messaging
  - Does not wait for a return
  - Shown in interactions with one-sided arrowhead (1.x) or stick arrow (2.0)
  - Both passive and active objects can send asynchronous messages
- UML has “active objects” with **dark** borders (1.x) or double **bars** (2.0)
  - Maintain their own thread of control
  - Only active objects can receive asynchronous messages

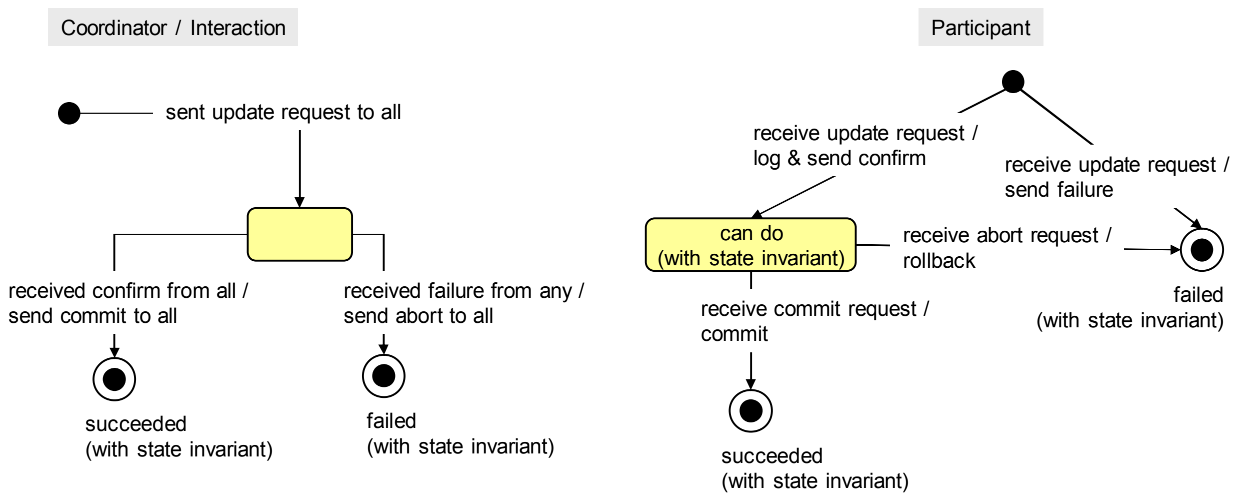


Cross-Component Invariants and Interaction State

- Consider invariants or consistency goals that cut across sub-component boundaries at the white-box level e.g. consistency constraints desired at the black-box level that are now partitioned across sub-components
- Handle with care to ensure consistency across multi-step updates, specially in the presence of failures. Consider a range of potential solution approaches:
  - Distributed cross-component transactions – transaction coordinator with 2-phase commit of local atomic transactions
  - Retry of failed portions of transactions, or retry-all semantics (assuming duplicate of a previously successful request are harmless)
  - Compensating transactions – simple local transactions with compensating (“undo”) transactions
  - Acceptable tolerances for incomplete or imperfect ACID guarantees
  - Consistent creation and propagation of transaction context on all calls, including nested calls
- A state model (or activity or sequence diagram) of the interaction as a unit and/or each participant, with state changes and invariants for key states, can help to understand the interactions and failure modes and to design an appropriate protocol.



Modeling Interaction Transaction State – Example


Coordinator / Interaction
sent update request to all
received confirm from all / send commit to all
received failure from any / send abort to all
succeeded (with state invariant)
failed (with state invariant)
Participant
receive update request / log & send confirm
receive update request / send failure
can do (with state invariant)
receive abort request / rollback
failed (with state invariant)
receive commit request / commit
succeeded (with state invariant)

- With suitable state changes and state invariants = 2-phase commit protocol
- Other variants might include re-try, compensating (‘undo’) transactions, etc.

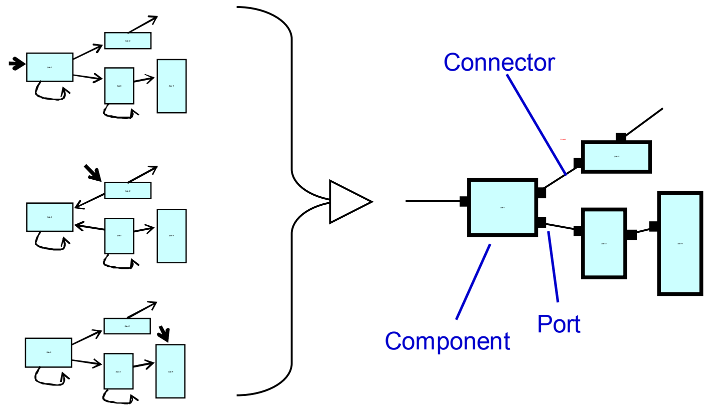
Define Sub-Component Assembly

This section covers:

- ♦ Basic Connectors
- ♦ Building an Assembly Model

Assembly Combines Collaborations

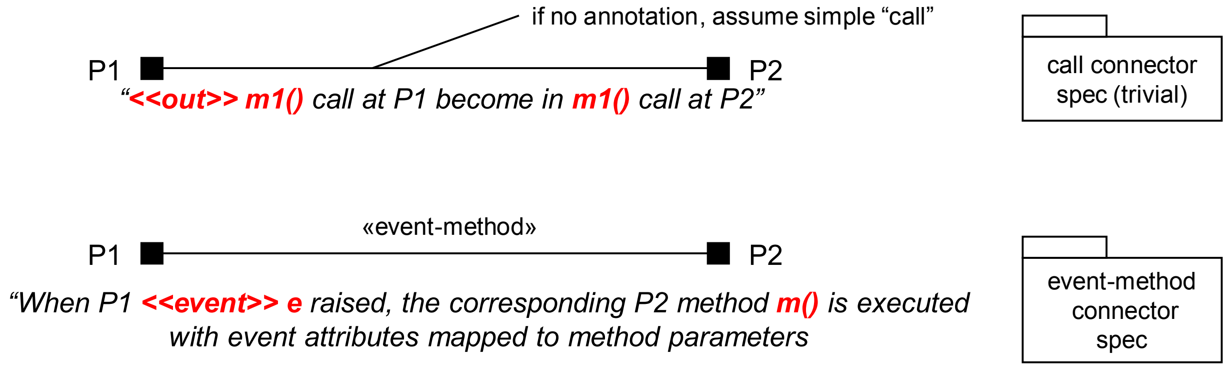
- ♦ One collaboration shows the interaction resulting from one trigger
- ♦ An assembly is a summary of component interconnections
- ♦ Combines all collaborations into ports and connectors, elides sequencing



- ♦ Why do we need "connectors"?
  - – Each component need be specified only from port to port i.e. it is encapsulated
  - – For collaboration, we rely on at least a basic "call" connector for communication
  - – We ultimately would like a library of connectors: call, events, adapting, replication, …

Connectors and their Specifications

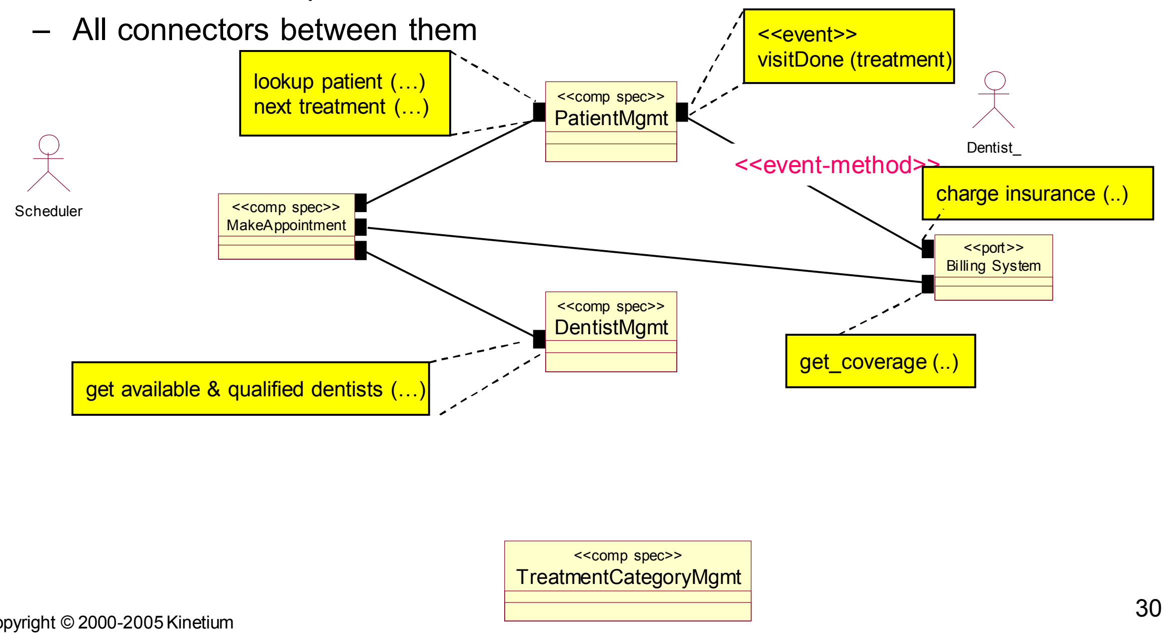
- To use a connector in the white box assembly, we must know it's spec
  - Documented in a connector specification and used as a stereotype or pattern`


if no annotation, assume simple "call"
P1
P2
"<<out>> m1() call at P1 become in m1() call at P2"
call connector spec (trivial)
«event-method»
P1
P2
"When P1 <<event>> e raised, the corresponding P2 method m() is executed with event attributes mapped to method parameters
event-method connector spec

- Using a connector means:
  - connecting it to appropriate ports
  - configuring it if necessary: e.g. which events to which methods, how to map attributes to parameters
- It is possible to have richer connector types: synchronizing, replicating, etc.
- Each connector type could have many different platform realizations
- Platform realizations are described in Technical Architecture

Assembly Example

- Assembly shows
  - External actors / ports
  - Internal sub-components
  - All connectors between them


<<event>> visitDone (treatment)
lookup patient (…)
next treatment (…)
<<comp spec>> PatientMgmt
Dentist_
Scheduler
<<event-method>>
charge insurance (..)
<<comp spec>> MakeAppointment
<<port>> Billing System
<<comp spec>> DentistMgmt
get_coverage (..)
get available & qualified dentists (…)
<<comp spec>> TreatmentCategoryMgmt

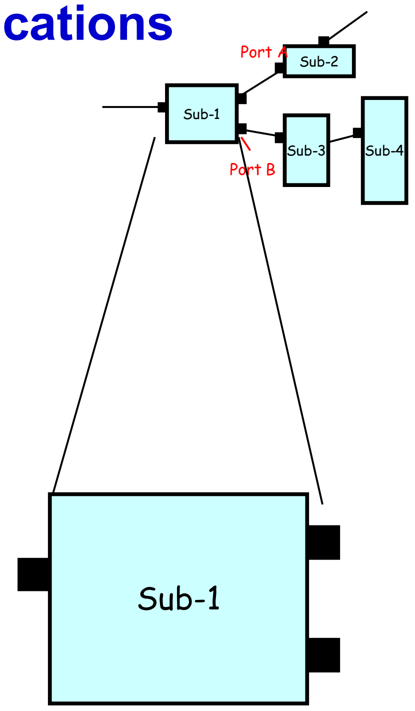
Exercise 7.3

Specify Sub-Components

This section covers:

- Specifying each sub-component

[Optional] Sub-Component Specifications



- ♦ A sub-component spec does not refer to other sub-components
  - – It just refers to its ports
- ♦ For each sub-component
  - – What is all information exchanged [at each port]
  - – What is all information remembered [at each port]
  - – What are logical specs for all behaviors [at each port]
    - • Include required interfaces/ports for operations required
    - • Include inputs, outputs, state changes, exceptions

[Optional] Operations and Event Specifications

Operations

- ♦ Simple operations – change component state and return result
  - – Specified by operation signature and pre-condition / post-condition
- ♦ Complex operations – require interaction sequences with other components
  - – Specified by
    - • Operation signature
    - • Collaboration diagram with component as main "focus"
    - • Pre-condition / post-condition that includes variables used in interaction sequence
  - – Will often require analysis of transaction properties
- ♦ Both cases can include separate exception specifications

Events

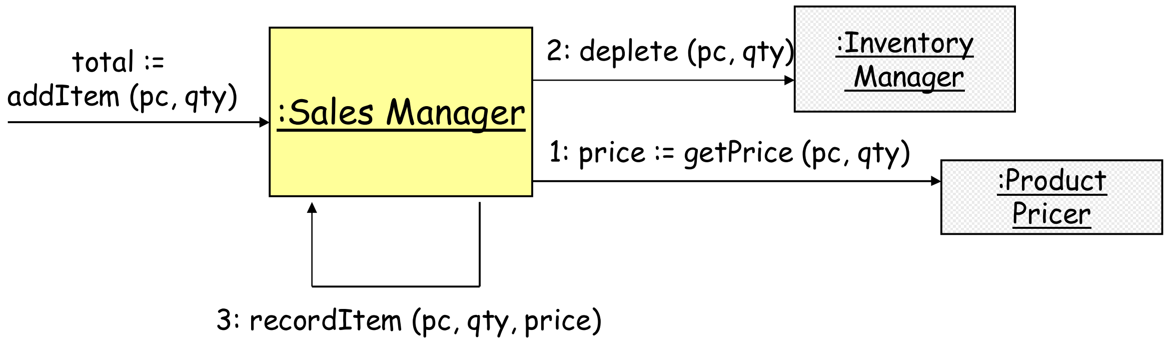
- ♦ Events raised: specified by
  - – explicitly including [**raise** eventName] in all applicable operation specs, or
  - – implicitly specifying **event** eventName using
    - • pre / post condition
    - • to define the state change that will raise that event
    - • implicitly specifying all operations that cause that state change

[Optional] Specifying Simple Operations

- Simple operations – change component state and return result
  - Operation signature and pre-condition / post-condition
- A Sales Manager component with startSale, addItem, endSale
  - startSale ()
    pre: there is no current sale
    post: a new **Sale** is created and becomes the current sale
  - addItem (pc: ProductCode, qty: Integer): Money
    pre: there is a current sale, s; pc is code for catalog **Product**, p
    post: a new **SaleItem** is added to s,
    with item cost = p.costFor(qty) and product = p
    the total of s is increased by item cost -- *can be replaced by invariant*
    Item cost is returned
  - endSale (): Money
    pre: there is a current sale
    post: …. Is returned
- As always, all operations are specified in terms of a single information model including persistent state and all data exchanged
  - What is the information model of Sale Manager?

[Optional] Specifying Complex Operations

- Complex operations require sequences of interactions with others
  - Operation signature, collaboration diagram centered on the component as main "focus", post-condition only adds missing information
  - Often require careful analysis of transactional properties



- Sales Manager::
  addItem (pc: ProductCode, qty: Integer) : Money
  pre: there is a current sale, s
  post: 1: product pricer's price retrieved, price
  2: inventory manager's stock has been depleted
  3: a new SaleItem has been added to s
  4: sale item's total has been returned

- (separately)
  Inventory Manager::
  deplete (…)
  pre: …
  post: …

Exercise 7.4

Summary

This chapter has shown that:

- A black box specification can be realized by an assembly
- There are useful heuristics for partitioning into sub-components
- Collaborations between sub-components achieve the black-box spec
- The white box assembly defines how the sub-components are connected
- The sub-component specs specify each sub-component

Chapter 8

How to specify a component's external technical properties?

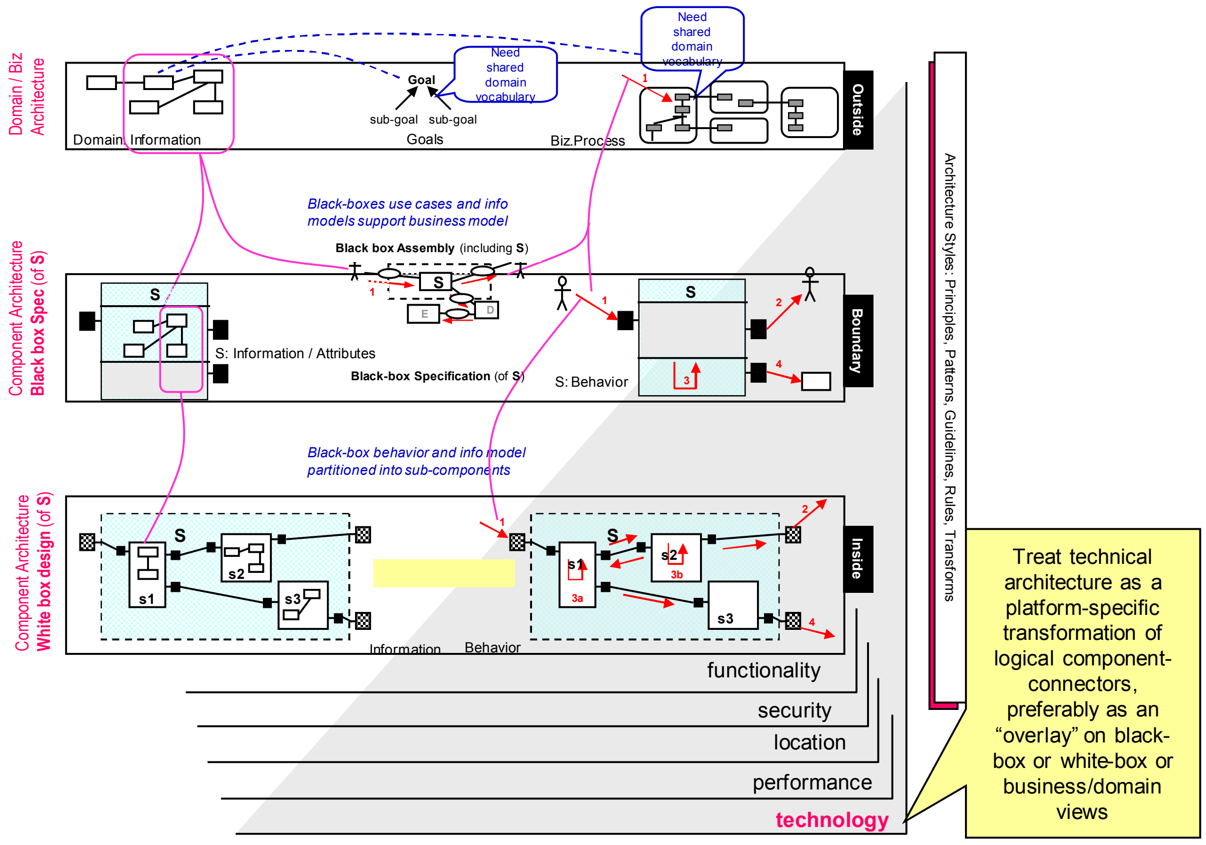
Technical View of <System X>

Technical architecture defines how the logical component and connector are realized in platform-specific terms.

It covers:

- ♦ Defining and Using (Technical) Styles, including mappings
- ♦ Technical Realization of Connectors
- ♦ Technical Realization of Components

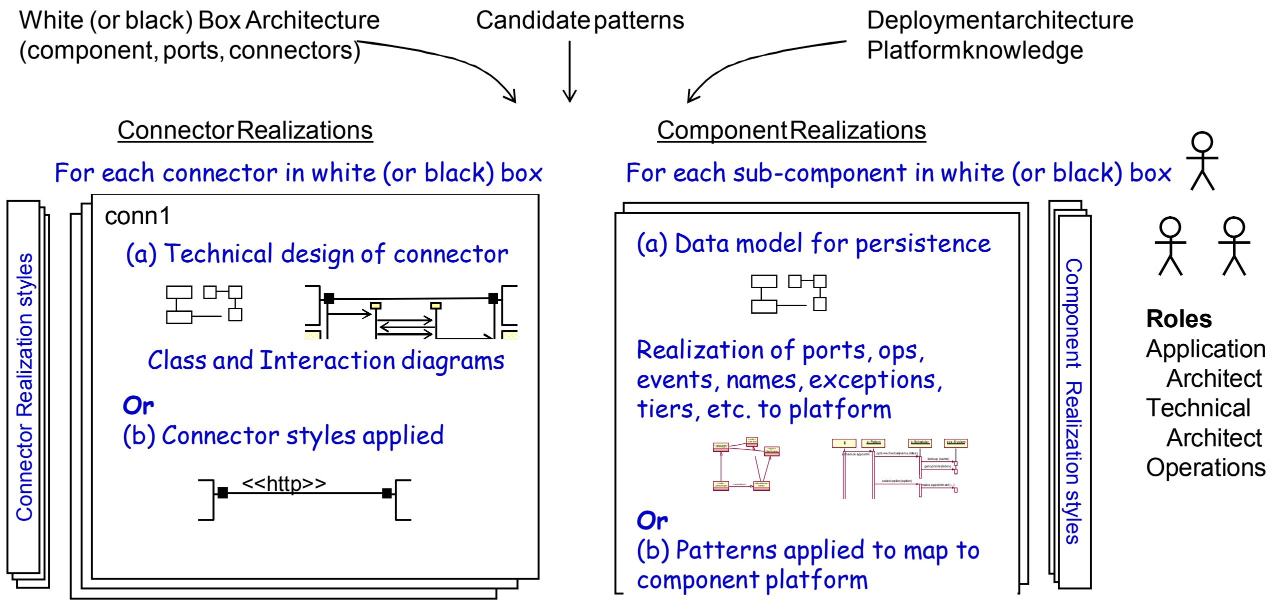
MAp Viewpoints


Domain / Biz Architecture
Component Architecture Black box Spec (of S)
Component Architecture White box design (of S)
Need shared domain vocabulary
Need shared domain vocabulary
Goal
sub-goal
sub-goal
Domain Information
Goals
Biz.Process
Outside
Black-boxes use cases and info models support business model
Black box Assembly (including S)
S
E
D
S: Information / Attributes
Black-box Specification (of S)
S: Behavior
Boundary
Black-box behavior and info model partitioned into sub-components
S
s1
s2
s3
s1
s2
s3
3a
3b
Information
Behavior
Inside
functionality
security
location
performance
technology
Architecture Styles: Principles, Patterns, Guidelines, Rules, Transforms

Treat technical architecture as a platform-specific transformation of logical component-connectors, preferably as an "overlay" on black-box or white-box or business/domain views

2

Preview of Technical Architecture


White (or black) Box Architecture (component, ports, connectors)
Candidate patterns
Deployment architecture
Platform knowledge
Connector Realizations
For each connector in white (or black) box
Connector Realization styles
conn1
(a) Technical design of connector
Class and Interaction diagrams
Or
(b) Connector styles applied
<<http>>
Component Realizations
For each sub-component in white (or black) box
(a) Data model for persistence
Realization of ports, ops, events, names, exceptions, tiers, etc. to platform
Or
(b) Patterns applied to map to component platform
Component Realization styles
Roles
Application Architect
Technical Architect
Operations

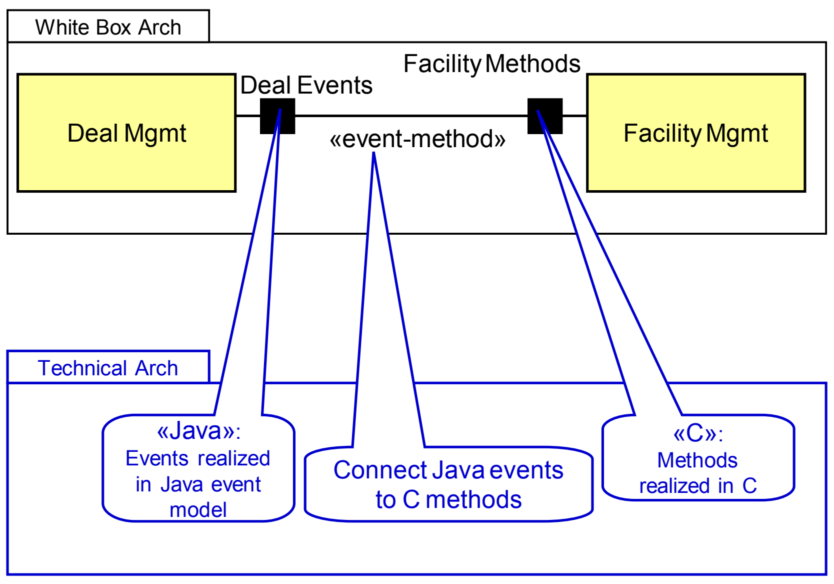
Why do a Technical Architecture with connector styles?

- Connectors repeatedly address similar connectivity and adaptation concerns. We want a consistent connector design and implementation. Sophisticated connectors decrease coupling between components. Consistent mappings from component specs to platforms help decrease needless variation.

Steps to Technical Architecture

- Technical Realization of Connectors
  - Choice of communication protocol
  - Encoding of data formats for communication
  - Adapt protocols with intermediate components
  - Encourage or enforce use of standardized co-ordination infrastructure
- Technical Realization of Components
  - Design Database Models
  - Map components, ports, methods, events, etc. to platform specifics

Technical Realization of Logical Components, Connectors


White Box Arch
Deal Mgmt
Deal Events
Facility Methods
«event-method»
Facility Mgmt
Technical Arch
«Java»: Events realized in Java event model
Connect Java events to C methods
«C»: Methods realized in C

White Box Arch Styles
An **event** is specified by:
- event name and event information
- condition for event to be raised

An **event method** connector:
- takes a raised event **e (T)**
- takes a corresponding method **m(x,y)**
- takes a mapping from **T** to **x, y**
- ensures **m** invoked when **e** is raised

Technical Arch Styles
To realize event **S::e (T)** in Java:
- method **S**::add_**e**_Listener (x)
- method **S**::remove_**e**_Listener (x)
- call **S**::notify_**e** (T)

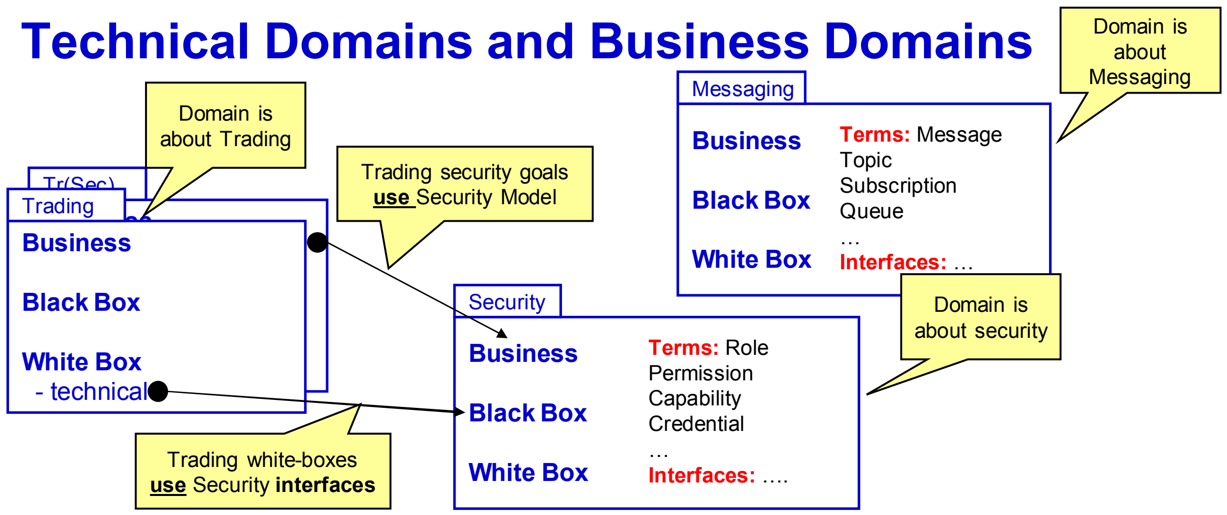
To realize method **A::m (x, y)** in C:
- procedure **m** (**A**_id, **x, y**)
- Map ids, parameter types: …

To connect Java event to C method:
- add appropriate listeners
- connect Java event to Java method
- call C via Java Native Interface

- Map component spec to platform: methods → procedures? transactions? relational data?
- Map operations, parameters, failures → platform naming, parameter types, exceptions

Exercise 8.1

Technical Domains and Business Domains



- Infrastructure or technical domains (e.g. Security, Messaging) have their own business (or “domain”), including goals, black box, and white box models
- These can be developed, to some extent, in parallel with the primary business domain (e.g. Trading)
- When Trading is defined in technical or platform terms (e.g. white-box/technical), that platform itself is defined in part by the technical domains and their black-box or white-box interfaces.

- A domain is some (portion of a) world usefully treated as a unit in problem solving and goal analysis and in goal dependencies. A sub-domain is a subset of a domain
- A domain service is a way to offer access to the implemented technology or conceptual terminology of a domain from dependent domains. (related to a “Concern”)

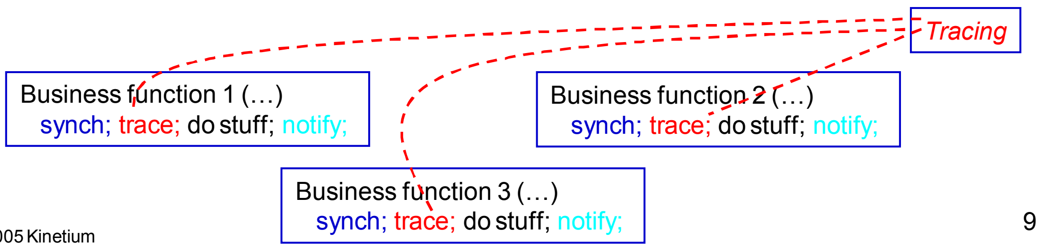
Defining and Using Patterns

This section covers:

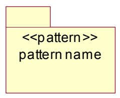
- Concerns tangle and scatter across designs
- Patterns separate design concerns
- Patterns are defined by adding parameters to a package
- The same pattern is applied to many parts of a design

[Optional] Concerns Tangle and Scatter across designs

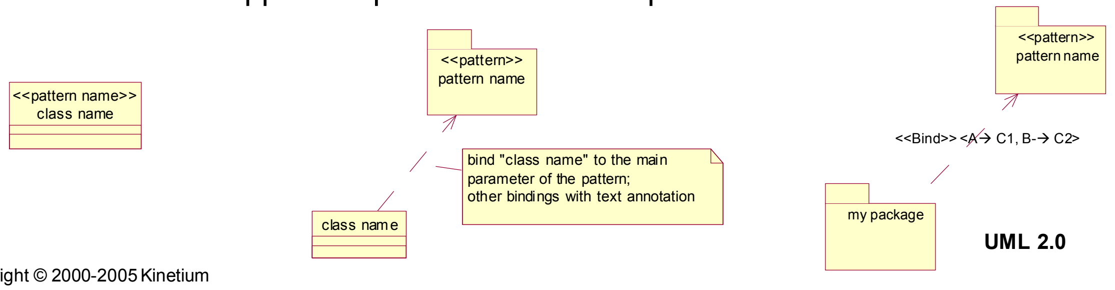
- An implementation has to deal with a myriad of different concerns
  - Primary business functional logic
  - Tracing and logging
  - Synchronization under concurrent access
  - Notification of events to unknown external parties
- What happens to the design and code of simple interacting objects when you add
  - Tracing
  - Synchronization
  - Notification
- Designs and code will
  - Tangle all these issues together – primary functionality is mixes with many other issues
  - Scatter a given issue – no central architectural control over trace, synchronize, …


Tracing
Business function 1 (…)
synch; trace; do stuff; notify;
Business function 2 (…)
synch; trace; do stuff; notify;
Business function 3 (…)
synch; trace; do stuff; notify;

MAp Usage of Patterns

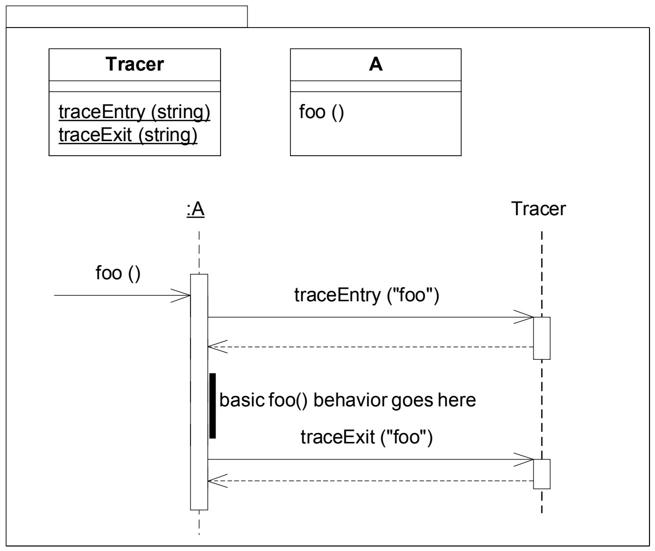


- Pattern: a recurring model structure that appears in multiple places
  - Defined in a package
    - With parameters of the pattern
    - As part of the Architectural Style
    - Formally described with a model, or
    - Informally described with a note, example, code fragment, etc.
- Pattern application uses that pattern in a particular context
  - It is shown by
    - A stereotype with the pattern name: bind the element to the placeholder in pattern
    - A package with an arrow for the primary parameter binding if any, text annotation for other parameter bindings
- Patterns apply in all viewpoints and mappings across viewpoints
- **Note:** UML 2.0 support for patterns is much improved



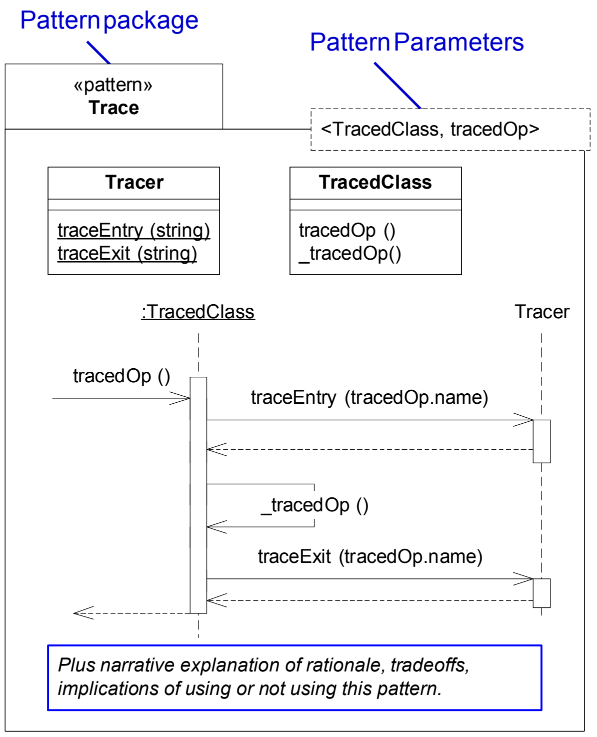
Example: Logging or Tracing in a Specific Case

- A class **A** has a method **foo** that needs to be traced
- A **Tracer** class provides the basic entry and exit calls
- **A::foo** makes calls to do the traces

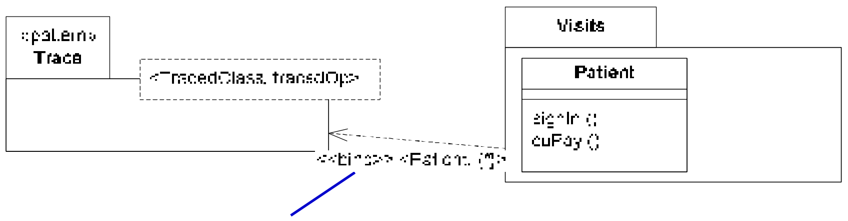


Example: Defining Trace as a Pattern

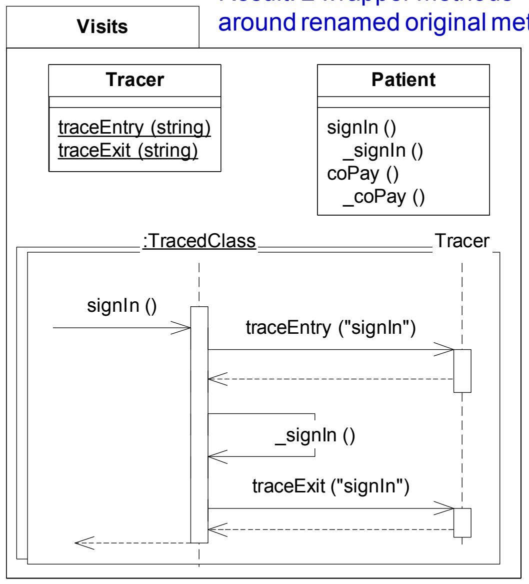
- Define a package for the pattern
- Decide "parameters" of package
- <TracedClass, tracedOp(…)>
  - These are pattern parameters
  - Any TracedClass will do
  - Any tracedOp
- Define the expanded version generated by the pattern
  - Just like a normal model
  - Insert **Tracer** class
  - Rename original method to **_tracedOp**
  - Insert "wrapper" method **tracedOp**
    - Has same name as pattern parameter
    - Call **_tracedOp ( )**
  - Access to things like tracedOp.name



Example: Apply Pattern to all Methods of a Class



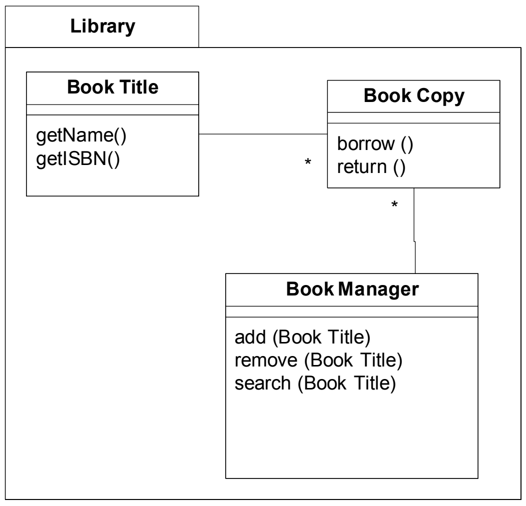
Parameter Binding
trace all methods of **Patient** class



Result: 2 wrapper methods
around renamed original methods

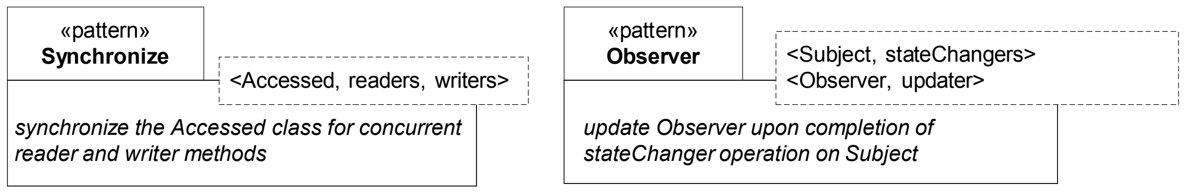
- Apply the pattern by “binding” its parameters
  - Can include one or several bindings
  - This example binds all methods of Patient
- This needs an extension to UML patterns
  - UML permits just 1-to-1 substitution
- The resulting design has all the right traces in the right methods

Designs Apply Multiple Patterns



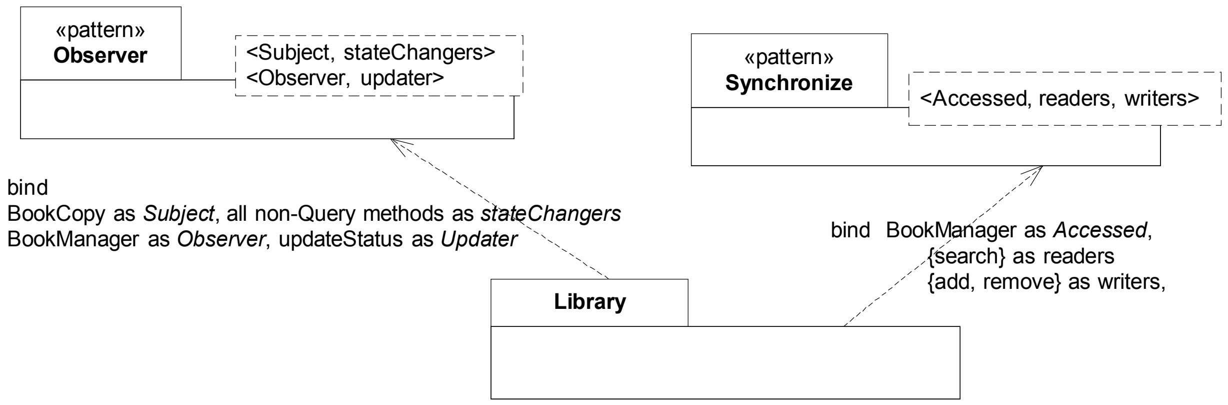
- This is an example of a basic design model
- Its focus is main functionality
  - Borrow and Return Book Copies
  - Add, Remove, Search Book Titles
- But it will also need
  - Observation/Notification of Copy changes
  - Synchronization of concurrent access
- These could tangle up the design
  - Unless defined separately as patterns
  - And applied to the basic design model

Patterns: Synchronization, Notification


«pattern»
Synchronize
<Accessed, readers, writers>
synchronize the Accessed class for concurrent reader and writer methods
«pattern»
Observer
<Subject, stateChangers>
<Observer, updater>
update Observer upon completion of stateChanger operation on Subject

- Each pattern is defined with its parameters
- Synchronize: protect reader and writer methods from each other
- Observer: generate notifications of state changes

Apply Observer and Notification Pattern


«pattern»
Observer
<Subject, stateChangers>
<Observer, updater>
«pattern»
Synchronize
<Accessed, readers, writers>
bind
BookCopy as Subject, all non-Query methods as stateChangers
BookManager as Observer, updateStatus as Updater
bind BookManager as Accessed,
{search} as readers
{add, remove} as writers,
Library

- Apply Observer Pattern
  - Subject = BookCopy
    - stateChangers = { all BookCopy methods without a “read-only” tag }
  - Observer = BookManager
    - updater = updateStatus
- Apply Synchronize pattern
  - Accessed = BookManager
    - readers = { search }
    - writers = { add, remove }

Other Examples of Patterns

- Composite
  - An composite object contains other objects that can themselves be either composites or simple; both offer similar interfaces
- Publisher-Subscriber Hub
  - Keep the state of cooperating components synchronized by enabling one-way propagation of changes. One publisher notifies the hub of changes; the hub propagates changes to all active subscribers
- Adaptive Propagator
  - Define a network of objects so that when one changes state, dependent objects are marked as affected but actual update can be dealt with separately
- Sliding Window
  - A set of types for dealing with sliding windows of time and how various properties change across those windows
- Lazy Object Merge
  - Sometimes two distinct objects are discovered to be the same. Mark the first as superseded, link it to the second, and have the first behave by delegating to the second
- MQ Series Queue Local Naming
  - For a source S of events and a destination D that will be co-located on a single machine, use an MQ Series queue named <S><D><…>, configured as <….>, and set up the S and D ends of the queue as <….>

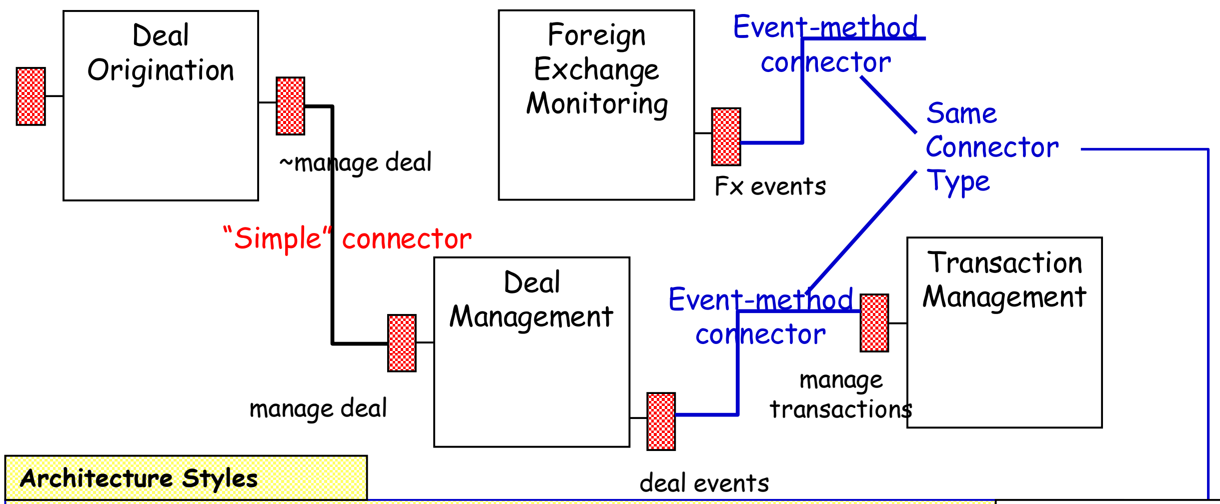
Exercise 8.2

Technical Architecture for Connectors

This section covers

- Connector specification Vs. realization
- Example of Event → Method Connector
- Example of Simple and Sophisticated Batch Connector
- Connectors and Integration Style
- User-Interfaces as Connectors

Connectors Assemble Components



Architecture Styles

1-Way Property Connector

Propagate source property change
- to the destination property

Change Veto Connector Type

Given a source port property
-Require a "propose-change" event
-Add veto protocol from listeners

Event-Method Connector

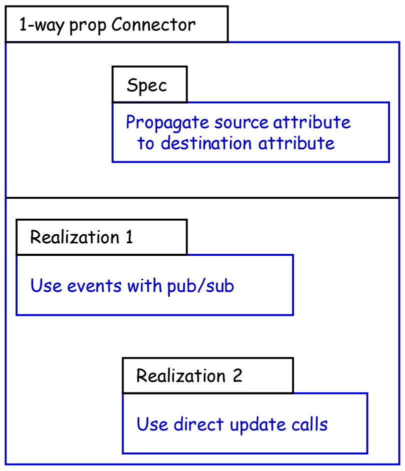
Given a source port
-with a set of events and event data
And given some number of destination ports
-with methods, input args, data mappings
"Connect" the events to the methods
-**Logical view**: business protocol
-**Technical view**: platform protocol

e.g. FX_rate_change event calls Recalc_portfolio_value

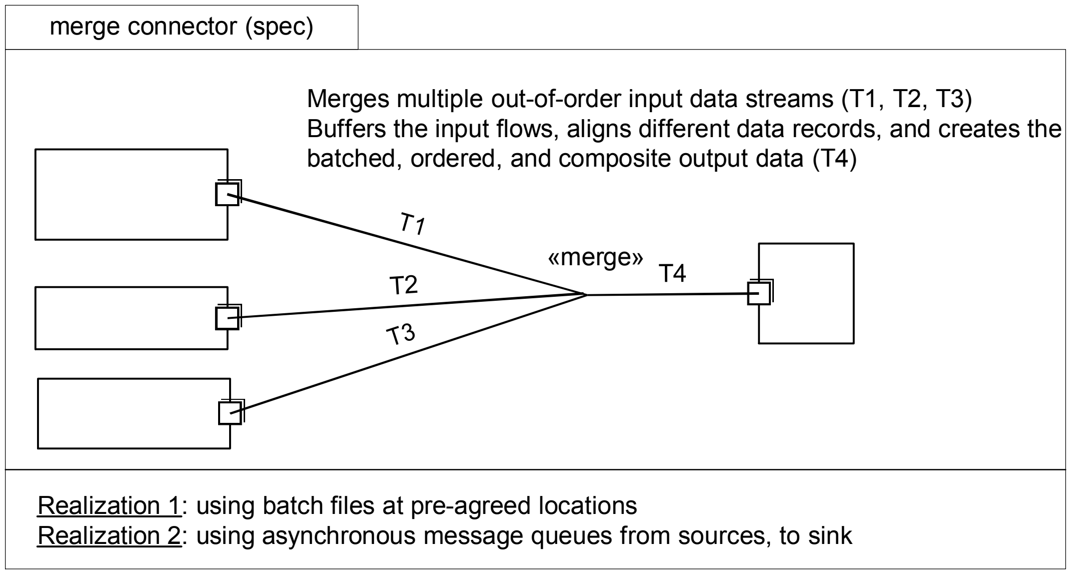
e.g. MQSeries.Send() operation is invoked, which puts a msg in…

Connectors Have Specification and Realization

- Each connector is specified as follows
  - Set of ports
  - Properties, guarantees, and protocols between ports
- Each connector has one or more realizations
  - Realization of properties, guarantees, protocols on some platform
- Connector definition can be layered
  - One specification
  - Multiple realizations
- Example: Different ways to update destination attribute with source attribute changes
  - Loosely coupled publish / subscribe mechanism
  - Direct update calls from source to destination



Sophisticated Connectors – Data Merge (Informal)



Merges multiple out-of-order input data streams (T1, T2, T3)
Buffers the input flows, aligns different data records, and creates the batched, ordered, and composite output data (T4)

Realization 1: using batch files at pre-agreed locations
Realization 2: using asynchronous message queues from sources, to sink

- Note: sophisticated connectors enable us to use simpler approaches in white-box architecture e.g.
  - Equality invariants across components → use property synchronization connectors
  - Merge relationships across data collections or streams → use merge connector
  - Event-trigger relationships across components → use event-method connector

Exercise 8.3

Technical Architecture for Components

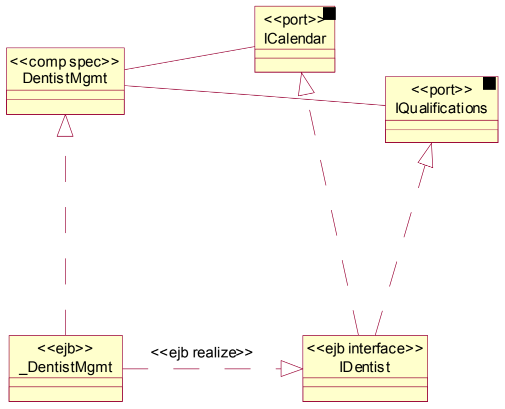
This section covers

- ♦ Component Realization
- ♦ Mapping to Platform Architecture
- ♦ Information Model to Data Models
- ♦ Patterns for object-relational mapping

Component Realization

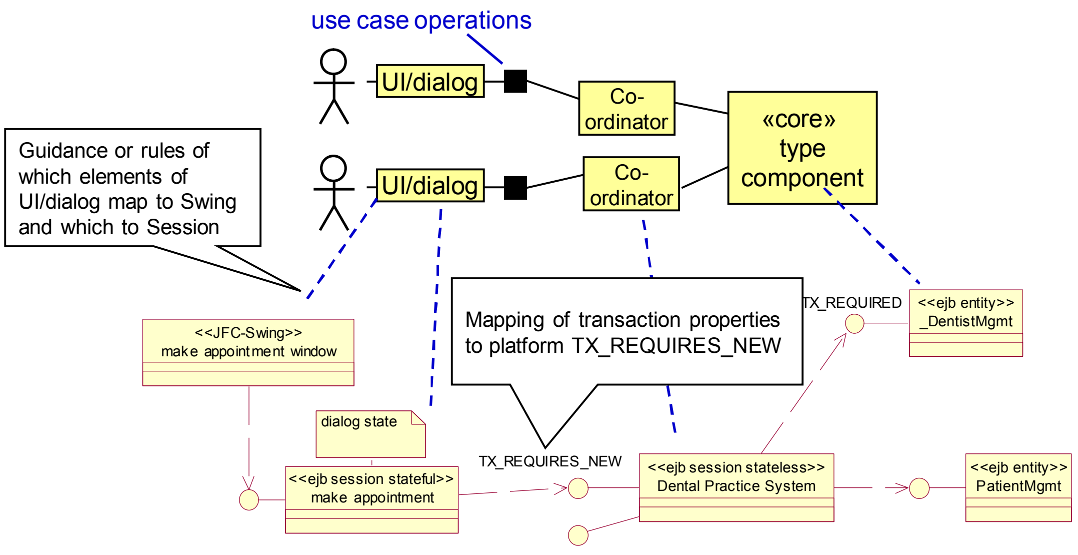
- Logical component must be designed to full technical detail
  - Platform
    - OS, data base, language, …
  - Platform-specific data model
    - Typically expressed as relational model
      - Data model, degree of (de)normalization, indices, etc.
  - Nested levels of sub-components if needed
    - Platform may offer specific approaches to sub-components in different tiers
    - Optionally just done in code without explicit models
  - Platform implementation code
    - Exploit platform facilities e.g. synchronization, notification, …
- Again we prefer to define uniform rules or policies to map to the platform

Component Realizations Examples



- Mapping patterns for components include
  - Mapping of ports: single interface per component? Map ports accordingly
    - E.g. Java allows multiple inheritance of interfaces; EJB has no outgoing ports
  - Operation, parameters, and errors names and types
    - e.g. Java exceptions
  - Standard platform patterns
    - e.g. home interfaces in EJB
  - Component managers vs. Objects
    - EJB requires a home interface which can serve some manager functions

Mapping to Platform Architecture

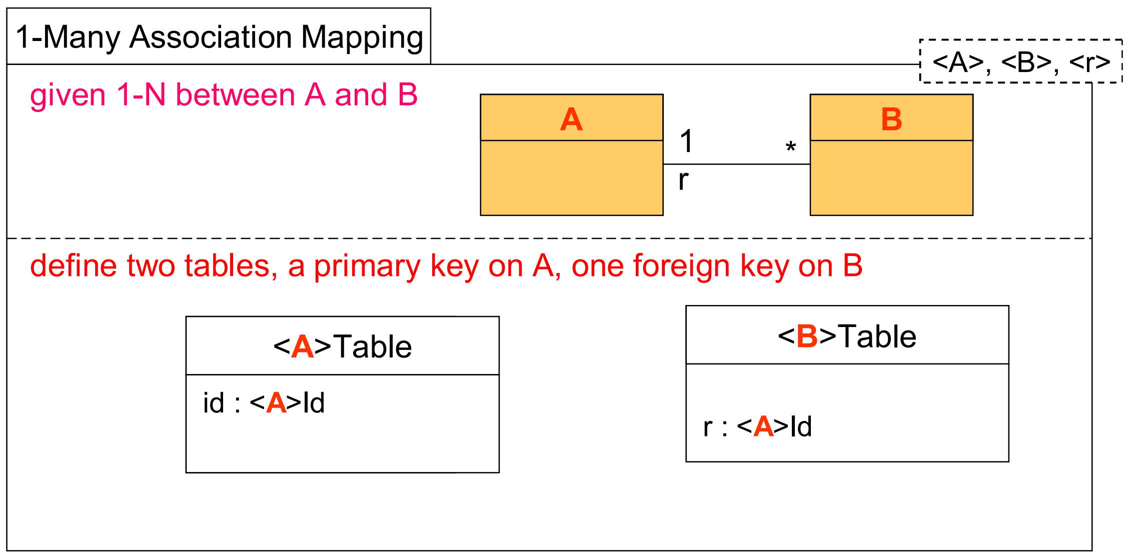

use case operations
UI/dialog
UI/dialog
Co-ordinator
Co-ordinator
«core» type component
Guidance or rules of which elements of UI/dialog map to Swing and which to Session
<<JFC-Swing>> make appointment window
dialog state
<<ejb session stateful>> make appointment
Mapping of transaction properties to platform TX_REQUIRES_NEW
TX_REQUIRES_NEW
<<ejb session stateless>> Dental Practice System
TX_REQUIRED
<<ejb entity>> _DentistMgmt
<<ejb entity>> PatientMgmt

- ♦ Can define patterns to map to platform supported architecture
  - – EJB Session beans (stateful for dialog; preferably stateless for server)
  - – EJB Entity beans
  - – Servelets and JSPs

Information Model to Data Models

- ♦ Persistent information must be mapped to selected data model
- ♦ Information model has
  - – implicit object identities
  - – platform neutral attribute types
  - – collections, sequences permitted
  - – parameterized attributes
  - – derived attributes
  - – Invariants
- ♦ Data model might have
  - – Tables
  - – Columns with pre-defined types
  - – Primary keys
  - – Foreign keys
  - – Indices

Patterns for object-relational mapping


1-Many Association Mapping
<A>, <B>, <r>
given 1-N between A and B
A
B
1
*
r
define two tables, a primary key on A, one foreign key on B
<A>Table
id : <A>Id
<B>Table
r : <A>Id

- Many other object-relational mapping patterns apply
  - Subtypes → single table vs. multiple tables
  - Identities → generate and lookup identifiers
  - Rows → Active Objects: wrap object smarts around individual row data structure
  - Tables → Table Gateway: provide access to an entire table and encapsulate SQL

Exercise 8.4

Summary

This chapter has shown that:

- ♦ Separation of white box logical from technical views has good benefits
- ♦ Systematic mapping between white box and technical based on
  - – Patterns: many of which can be defined as parameterized packages
  - – Pattern application: binding selected parts of your design using patterns
  - – Separation: specification vs. realization for connectors and components
- ♦ Connectors
  - – Specifications define logical dependency between ports of components
  - – Realizations define protocols, data formats, and adaptors
- ♦ Components
  - – Specifications define logical behavior at all ports
  - – Realizations map to platform architecture, appropriate data models, etc.

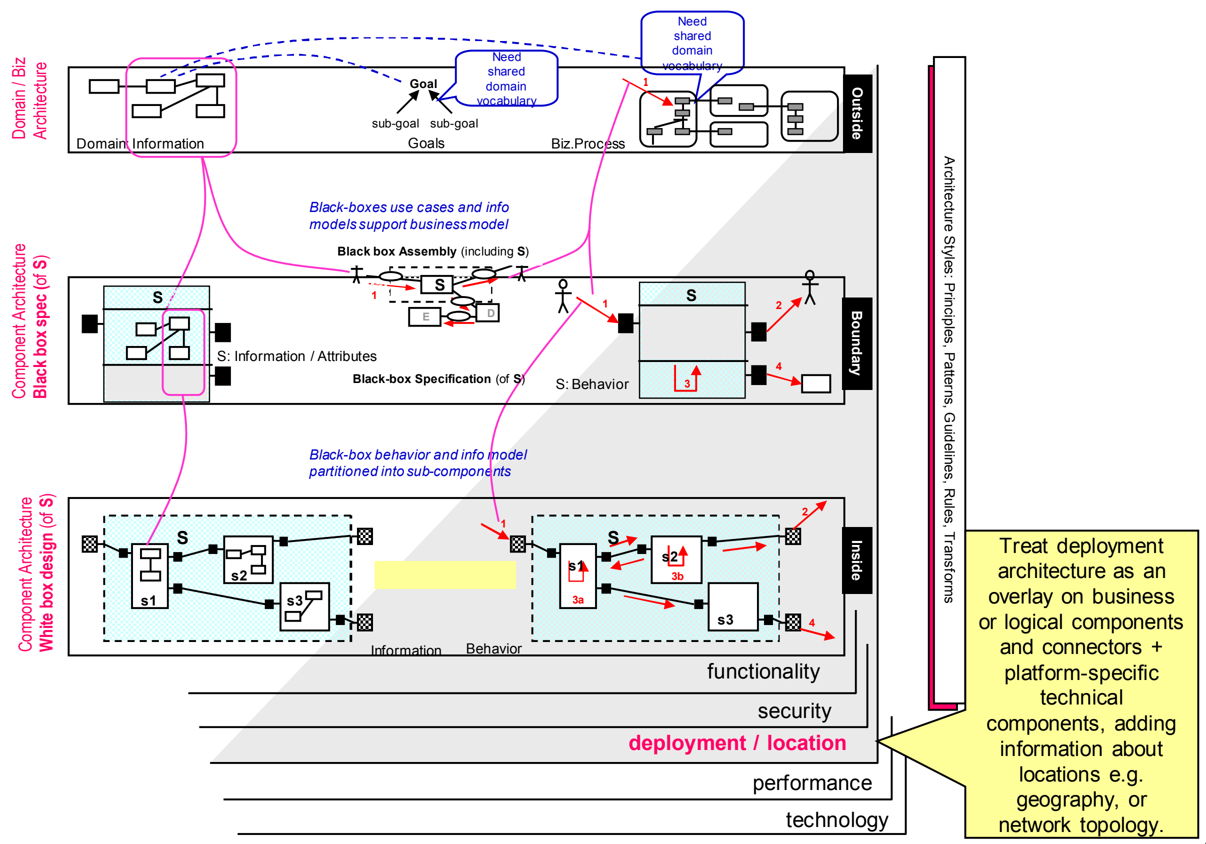
Chapter 9
Deployment View of <System X>

Deployment Architecture describes the system topology needed to install, configure, and support the system. It describes locations (geographic and network), types of hardware nodes, network connections, and the application and platform software that is deployed on them, and constraints on all of the above. It can be considered early in the lifecycle.

This chapter covers:

- ♦ Specify node types
- ♦ Identify deployment locations and constraints
- ♦ Map nodes and devices to locations
- ♦ Specify connections between nodes or locations
- ♦ Specify software and network configuration

MAp Viewpoints

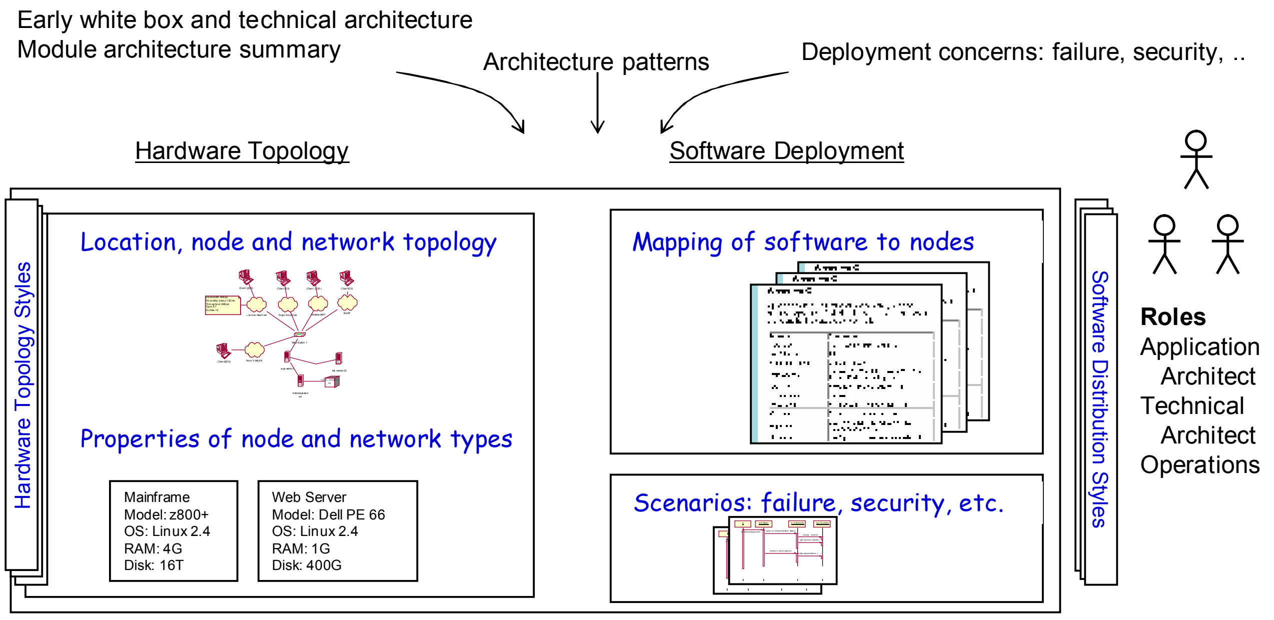


Domain / Biz Architecture
Domain Information
Goal
sub-goal
sub-goal
Goals
Need shared domain vocabulary
Need shared domain vocabulary
Biz.Process
Outside
Black-boxes use cases and info models support business model
Black box Assembly (including S)
S
E
D
Component Architecture Black box spec (of S)
S
S: Information / Attributes
Black-box Specification (of S)
S: Behavior
S
Boundary
Black-box behavior and info model partitioned into sub-components
Component Architecture White box design (of S)
S
s1
s2
s3
Information
Behavior
S
s1
s2
s3
Inside
functionality
security
deployment / location
performance
technology
Architecture Styles, Principles, Patterns, Guidelines, Rules, Transforms

Treat deployment architecture as an overlay on business or logical components and connectors + platform-specific technical components, adding information about locations e.g. geography, or network topology.

2

Preview of Deployment Architecture


Early white box and technical architecture
Module architecture summary
Architecture patterns
Deployment concerns: failure, security, ..
Hardware Topology
Software Deployment
Hardware Topology Styles
Location, node and network topology
Properties of node and network types
Mainframe
Model: z800+
OS: Linux 2.4
RAM: 4G
Disk: 16T
Web Server
Model: Dell PE 66
OS: Linux 2.4
RAM: 1G
Disk: 400G
Mapping of software to nodes
Scenarios: failure, security, etc.
Software Distribution Styles
Roles
Application Architect
Technical Architect
Operations

Why do a Deployment Architecture?

- ♦ Deployment decisions impact availability, reliability, performance, scalability, manageability, operations, and more. We want to leverage deployment patterns consistent with the application and technical architectures, and to define deployable software modules to be built.

Many Concerns Influence Deployment Decisions

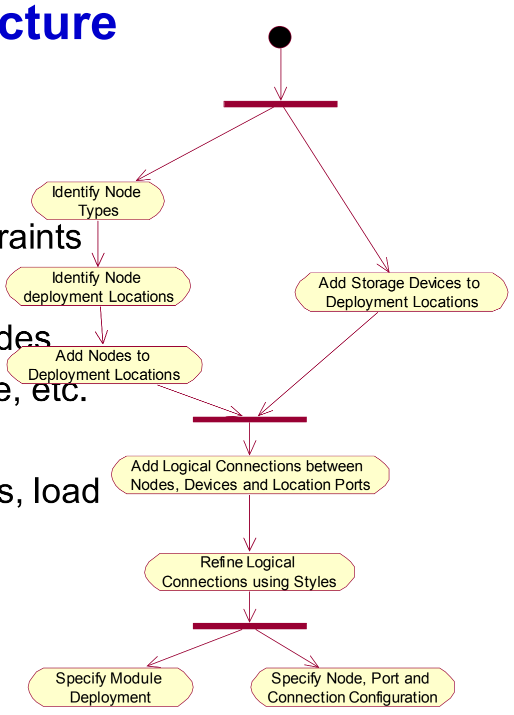
- ♦ What existing infrastructure do we have?
- ♦ What guidelines or policies exist for deployment?
- ♦ What are security risks for the nature of information and connections?
- ♦ How available must the system be in the presence of failures?
- ♦ Where are the other systems we need to interact with?
- ♦ How will you monitor and manage the system once you deploy it?
- ♦ What hardware, network, software, and configurations need to be deployed?

- ♦ MAp currently describes the resulting deployment model, not all the analysis.

Steps to Deployment Architecture

- The following are not a strict sequence

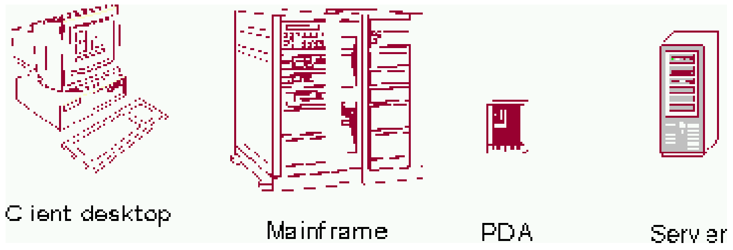
- Specify node types
- Identify deployment locations and constraints
- Add nodes to locations
- Specify logical connections between nodes
- Add additional devices: network, storage, etc.
- Map deployment units to network
- Iterate with scenarios: failures, intrusions, load spikes, etc.


Identify Node Types
Identify Node deployment Locations
Add Storage Devices to Deployment Locations
Add Nodes to Deployment Locations
Add Logical Connections between Nodes, Devices and Location Ports
Refine Logical Connections using Styles
Specify Module Deployment
Specify Node, Port and Connection Configuration

- Alternatively, specify all relevant visible properties in an SLA

Specify Node Types

- Define the types of processing nodes and their key characteristics
  - Thin clients: OS, processor, memory, disk space, …
  - Servers: OS, processors, memory, disk space, …
- Find or introduce icons or stereotypes for these nodes



Client desktop	Mainframe	PDA	Server
Model: Any	z800 +	Any	Sunfire 12K
OS: Any browser	Linux	Palm 3.5 +	Solaris 8 OE
RAM: 128M +	2 GB	8M +	1 GB
Disk: 40G +	4TB +	N/A	1 TB

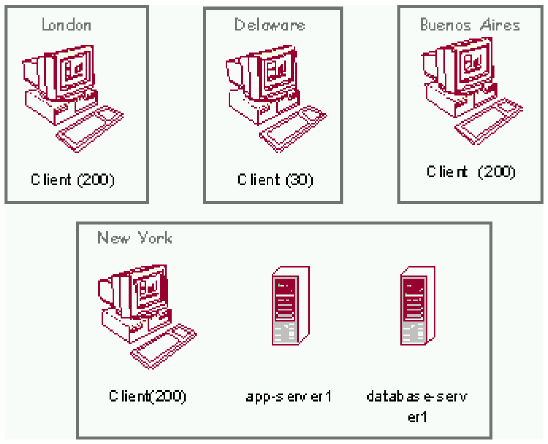
Not an exhaustive attribute list

Identify Deployment Locations

- Name (and define if needed) the locations where nodes will be located
  - A location can be geographical or physical area, network location, or both
  - Chosen at granularity appropriate to need, and to define connectivity
  - Define constraints on nodes and locations
- Examples
  - Toronto Teeth'R'Us
  - New York Dental Schedulers' desks
  - New York Dental Data Center at 45 Tooth Tower
  - New York Dental Security Zone 3
    - Definition: the network zone between the internet firewall and the database servers

Add Nodes and other Devices

- Place appropriate node types and numbers at each location

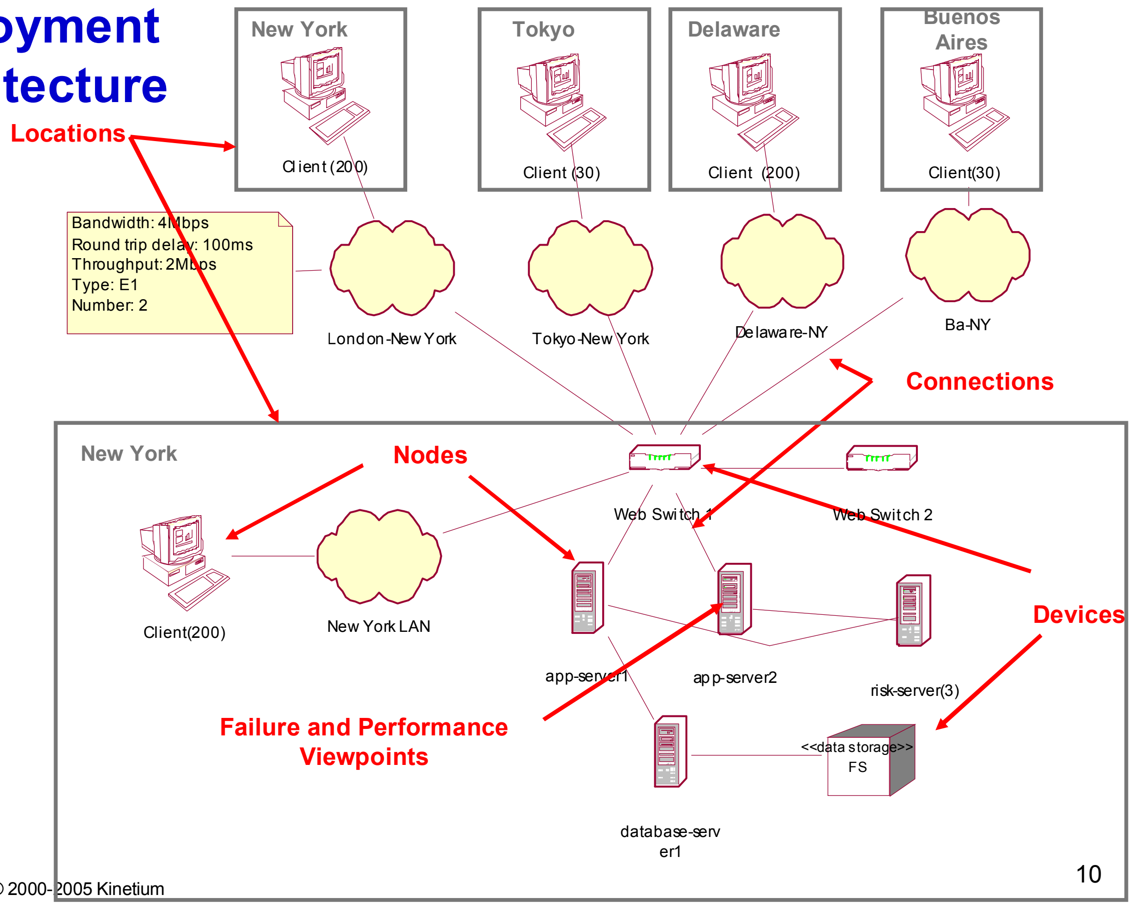


- Add other devices such as web-switches, firewalls, storage devices, etc.

Add Connections and Connection Attributes

- Network connections support the communications needed by white box and technical architecture
- Network attributes include:
  - Bandwidth: Measures the total data transfer capacity of the network connection as (mega- or kilo-) bits per second, e.g. 4 Mbits/sec
  - Roundtrip delay: Measures time between two nodes to queue a request at switching nodes in the network plus time to propagate data (packets). This varies by distance and bandwidth. This is measured in milliseconds, e.g. 100ms
  - Throughput: Measures the available data transfer capacity of the network connection, considering connection sharing with other applications, e.g. 2Mbits/sec
  - Connection type: The type of network connection, e.g. private line, metropolitan area network (MAN), local area network (LAN), E1.
  - Number of connections: The number of network connections available.
- Optionally name and define new types of network connections and their attributes

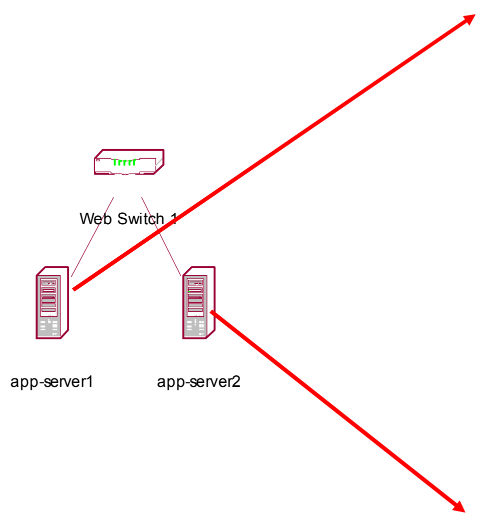
Deployment Architecture

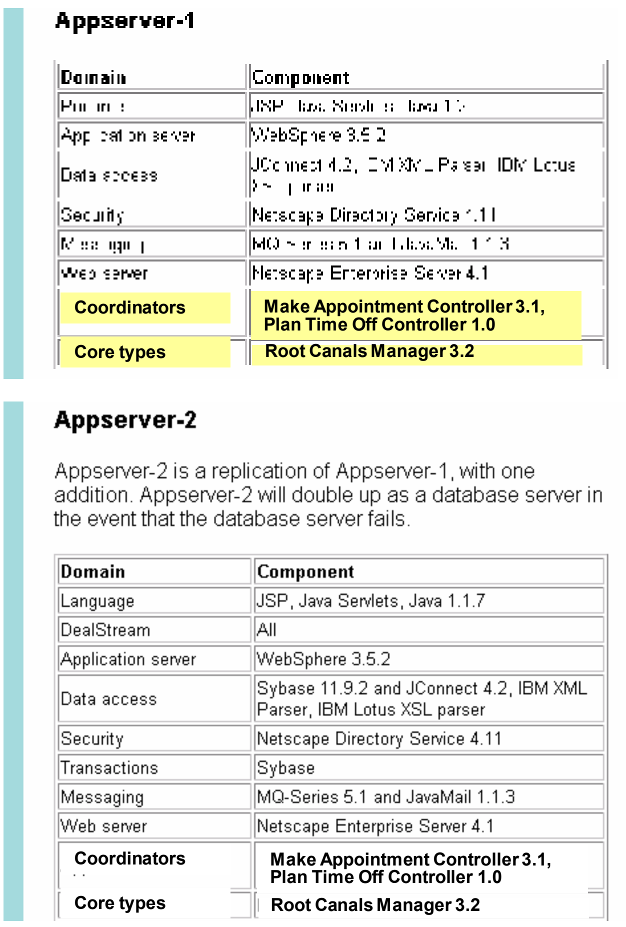


Map Software Modules to Network

- Map deployable software units onto the network
  - These units must be the targets of “builds” in the module architecture
- Ensure all modules they depend upon are also suitably mapped
  - Modules that realize other application components
  - Modules that realize technical components
- Can include very specific platform details and configuration settings
  - Configuration of users, directories, database accounts
  - Setup of communication ports on servers
  - Product specific configuration e.g. for Oracle, Veritas, WebSphere, …
  - Configuration of IP network usage, router and web switch programming

Software Deployment




Appserver-1

Domain	Component
[illegible]	JSP, Java Servlets, Java 1.1.7
Application server	WebSphere 3.5.2
Data access	JConnect 4.2, IBM XML Parser, IBM Lotus XSL parser
Security	Netscape Directory Service 4.11
Messaging	MQ-Series 5.1 and JavaMail 1.1.3
Web server	Netscape Enterprise Server 4.1
Coordinators	Make Appointment Controller 3.1, Plan Time Off Controller 1.0
Core types	Root Canals Manager 3.2

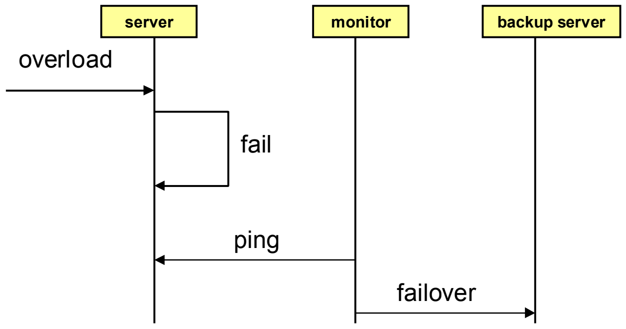
Appserver-2

Appserver-2 is a replication of Appserver-1, with one addition. Appserver-2 will double up as a database server in the event that the database server fails.

Domain	Component
Language	JSP, Java Servlets, Java 1.1.7
DealStream	All
Application server	WebSphere 3.5.2
Data access	Sybase 11.9.2 and JConnect 4.2, IBM XML Parser, IBM Lotus XSL parser
Security	Netscape Directory Service 4.11
Transactions	Sybase
Messaging	MQ-Series 5.1 and JavaMail 1.1.3
Web server	Netscape Enterprise Server 4.1
Coordinators	Make Appointment Controller 3.1, Plan Time Off Controller 1.0
Core types	Root Canals Manager 3.2

Iterate with Deployment-Related Scenarios

- Evaluate for load changes, failures, security scenarios
- Some scenarios are best described with scenario diagrams
  - The objects are deployed application and platform software components, and hardware nodes and networks
  - Interactions or events correspond to load changes, failures, security breaches



Exercise 9.1

Summary

This chapter has shown that:

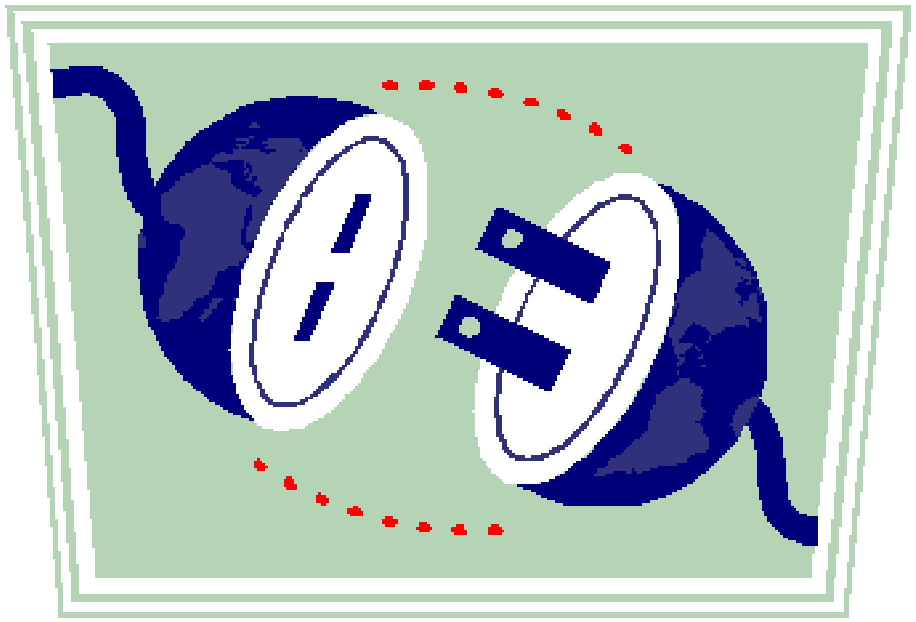
- Deployment architecture defines the topology of hardware and software
- It is influenced by many concerns of the deployed system
- It describes
  - Locations, hardware nodes and network connections
  - Application and platform software that is deployed on them
  - Optionally, how the architecture handles scenarios of failure, security, etc.

Appendix: References

1. MAP web site
   Available for licensed users at www.kinetium.com
2. “UML Distilled: A Brief Guide to the Standard Object Modeling Language”, Martin Fowler and Kendall Scott
3. “The Object Constraint Language : Precise Modeling With UML”, Jos B. Warmer, Anneke G. Kleppe
4. “UML 2.0 Specification”, www.omg.org
5. “Business Processes : Modeling and Analysis for Re-Engineering and Improvement”, Martyn A. Ould
6. “Business Process Management – A Rigorous Approach”, Martyn Ould, Meghan-Kiffer Press 2005
7. “Objects, Components, and Frameworks with UML: the Catalysis Approach”, D. D’Souza and A. Wills, Addison Wesley 1998
8. “UML Components: A Simple Process for Specifying Component-Based Software”, John Cheesman, John Daniels, Addison Wesley 2000
9. “Problem Frames”, Michael Jackson, Addison Wesley, 2001
10. “Practical Software Requirements”, Benjamin Kovitz
11. Goal Modeling: several papers on KAOS, NFR, and I*
12. J. Mylopoulos, L. Chung and B. Nixon, “Representing and Using Nonfunctional Requirements: A Process-Oriented Approach”, IEEE Trans. on Software. Engineering, Vol. 18 No. 6, June 1992, 483-497.
13. “Design Patterns”, Ralph Johnson et al
14. “Analysis Patterns”, Martin Fowler
15. “Writing Effective Use Cases”, Alistair Cockburn
16. “Enterprise Distributed Object Computing: EDOC”, www.omg.org

MAP

Exercises



Notes for Exercises

Lab exercises will be worked first with an initial **Instructor** section and then with a **Student** section. The former is for the instructor to work through, in part or in whole, with the students, to explain the problem and illustrate the approach. The latter is for the students to work out in teams. For long exercises with multiple sections, the instructor might work out the instructor example just one section at a time, letting students work on the corresponding section of their student labs.

The **[estimated times]** are shown for each lab student section, not including the time for the instructor to work out the instructor example section.

Some labs have a starting template for the solution. Please begin that lab using that template. If solutions to the previous exercises are available, please use that solution as the starting point for the subsequent exercise so we are always working on a converging solution.

Work on the exercises in a **team** of 2-4 people. Work around one end of a table if possible. We recommend that you work on a single solution using just one exercise workbook; you can always make photocopies of your solution for the team at the end of each day.

The purpose of many of these labs is to learn to use models to produce clear descriptions with reduced ambiguities and errors. In some cases there will be many ways you can define the problem itself, or the solution to the problem. Remember that our main goal is to focus on applying MAP.

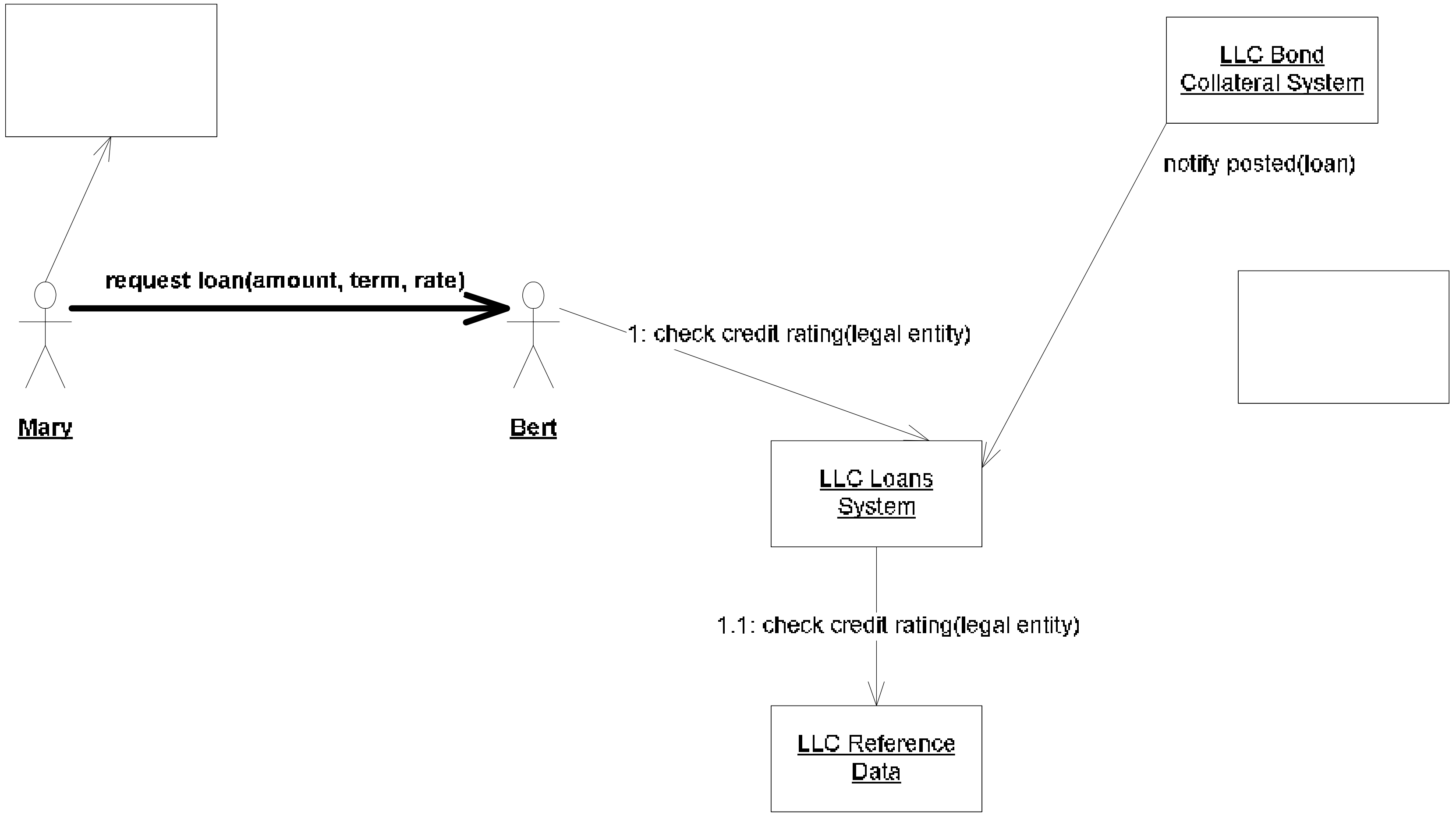
Exercise 1 Describe Objects [30]

Consider the scenario in the table below, then work through the subsequent questions 1(a), 1(b), ..etc.:

1. Mary, an employee of Plastic Novelties Inc (PNI), approaches Bert, a loan officer at LotsALoans Corporation (LLC) for a corporate loan to PNI. She specifies the loan amount and preferred term (duration).
2. Bert checks PNI's credit rating, and those of any parent companies of PNI, in the LLC Loans System.
3. The LLC Loans System obtains the legal entity credit rating and hierarchy information for PNI from the LLC Reference Data System, and returns it to Bert.
4. Bert then checks the total credit exposure of LLC to PNI within the LLC Loans System.
5. Since these initial checks were successful, Bert initiates a loan record within the LLC Loans System.
6. In accordance with LLC policy, Bert stipulates to Mary the amount and acceptable types of collateral that PNI must post to secure the requested loan.
7. Mary arranges via PNI's Borrower Collateral Management System for the collateral to be posted.
8. Some time later, PNI's Borrower Collateral Management System posts bond and real estate collateral by interacting directly with the LLC's Bond System and LLC's Real Estate Collateral Management Systems.
9. Once the required amount of collateral has been posted, the LLC Loans System notifies Bert.

1 a) Show Interacting Objects

- Draw the individual objects **participating actively** in the interactions, without showing interaction. Only show objects that participate actively in the interactions (e.g. an agent or a system), not objects that are part of the information exchanged between them (e.g. a reservation record). Use any icon for each object, writing the object name on or below the object. Include both physical objects and software systems. Don't show "internal" objects at your current level of description.
- Draw the interactions involved. Use an arrow for any request from object A to object B, or to itself. Number the interactions to show sequencing, optionally using nested numbering to distinguish trigger (3) from responses (3.1, 3.2, …), and label the interaction to name the interaction and to indicate information passed or returned in that interaction.


LLC Bond Collateral System
notify posted(loan)
request loan(amount, term, rate)
1: check credit rating(legal entity)
Mary
Bert
LLC Loans System
1.1: check credit rating(legal entity)
LLC Reference Data

1 b) List Information Owners

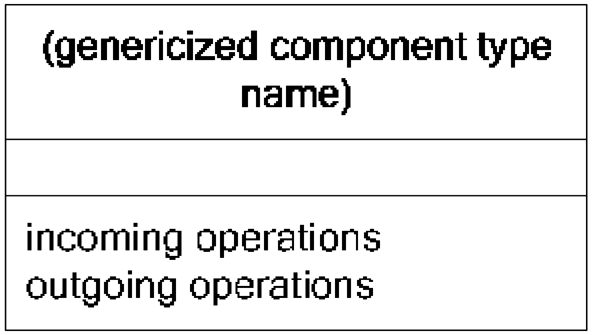
List information ownership: Make a list of the objects and attributes that are owned and managed by each software system in your diagram.

Loans System	loan portfolio (including total credit exposure), loans (including principal, rate, term), reference to collateral posted
Reference Data System	

Note that you have described distinct kinds of objects: those that interact with others, those that are stored persistently, and those passed around.

1 c) Polymorphic Component Types

Draw a “Polymorphic” component type (one with two different implementations of the same interface). List the operations on that type, both incoming and outgoing.



1 d) Find Multiple Types

In the description below, consider the behaviors of the described object, which can be viewed as having two different types by two different applications or components. Name and define those types with some operations and/or attributes.

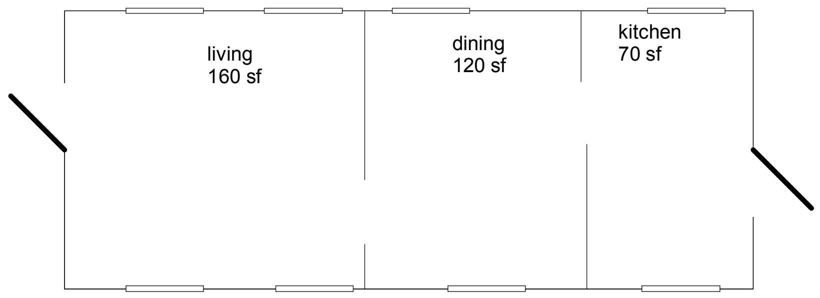
Discuss a problem that might arise if an object with two types is represented in 2 different software systems as 2 disconnected objects.

LotsALoans Corporation (LLC) Example

- The business event of PNI's first repayment against its newest loan with LLC appears in the loan system as a repayment type, with a link to its associated loan, an amount and a date of payment, together with a due date and amount due.
- This repayment also appears in the Loans subledger system as a journal entry associated with a debit and a credit transaction against the appropriate subledger accounts.

Exercise 3.1 Objects, Snapshots [15]

You are given the following floor plan of your house that was built in 1950 in Squirrel Hill.



Draw one object representing your residence that tells how many **exterior** doors and windows it has, when it was built, and what neighborhood it is in. Draw objects representing each room and its approximate size (ignore wall thickness).

Make this collection of objects into a snapshot by drawing links between the objects: Draw links between each pair of rooms that shares a wall and between each room and the residence that encloses it. Label each link.

Exercise 3.2 Information Models [15]

Use the snapshot from the previous lab and create an information model from it.

How would your snapshot change if you were to represent windows and doors as objects? Make this change to your information model (and to your snapshot if you have time).

Exercise 3.3 Snapshot Pairs [15]

Starting with the previous lab, and ignoring doors and windows as objects (simply modeling counts of doors and windows instead):

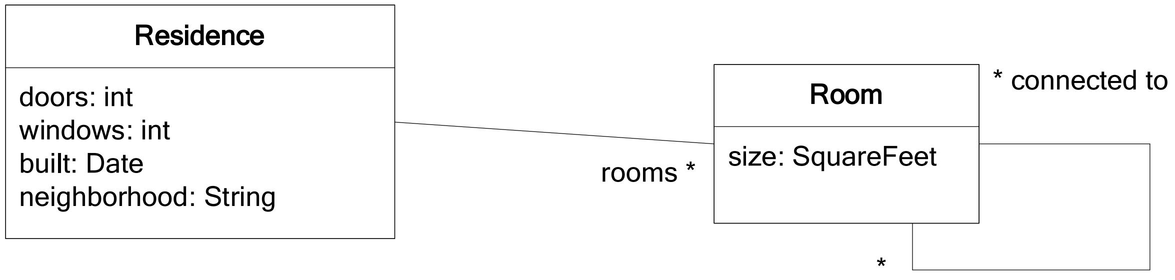
Draw a before and after snapshot pair for what happens when a window is added to the house.

Draw a before and after snapshot pair for what happens when a wall is removed to join the living and dining rooms of the house.

Draw a before and after snapshot pair for what happens when you remodel the interior of a room (no changes to windows, doors, or room connections). Do you see any changes in your snapshots? Why or why not? Does this mean that your model is “wrong”? Generalizing from this, what should guide the things that should appear in a model?

Exercise 3.4 Actions [15]

Write an informal action specification for the two snapshot pairs from the previous exercise, underlining terms which refer to parameters or attributes in your information model. Choose parameters to the actions to correspond to some object types (or attribute types) in your model.



action add window ( )

description

pre

post

action remove wall ( )

description

pre

post

Exercise 3.5 Scenarios with text and snapshots [15]

You are putting a new wing on your mansion, which consists of laying a foundation, adding walls, roofing, adding windows, adding doors, and putting down finished flooring. Windows, doors, and floors you can do yourself; for all the others you sign a contract with a contractor, and he constructs those parts before you do a final walkthrough.

Write a textual scenario of your specific series of renovations to add a wing. Use the scenario format, with scenario title, initial state and final state. Create snapshots for the before and after states of at least 3 steps in the scenario. Add new types of objects or attributes to your information model if needed. Use simple attributes (such as roof_done: True / False) where it is adequate to describe your scenario.

Scenario name: new wing on mansion

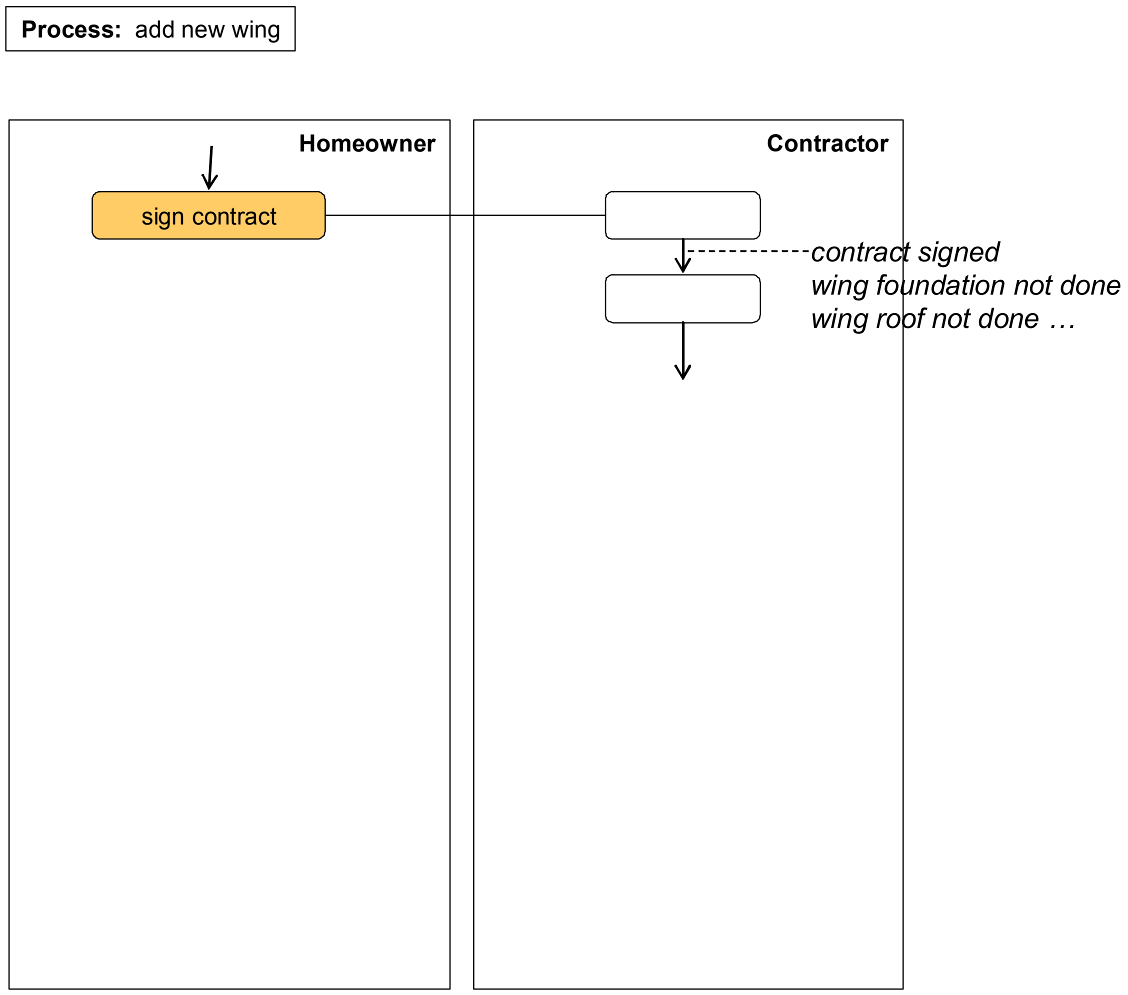
Initial state: Titus is a contractor, Jake the homeowner

1. titus and jake sign the contract
2.
3.
4.
5.
6.
7.
8.
9.
10. …

Final state:

Exercise 3.6. Role Activity Diagrams [15]

Using the scenario from the last exercise as a guide, draw a Role Activity Diagram (RAD) that ensures that those same actions are done in an appropriate sequence and collaboration occurs when necessary.



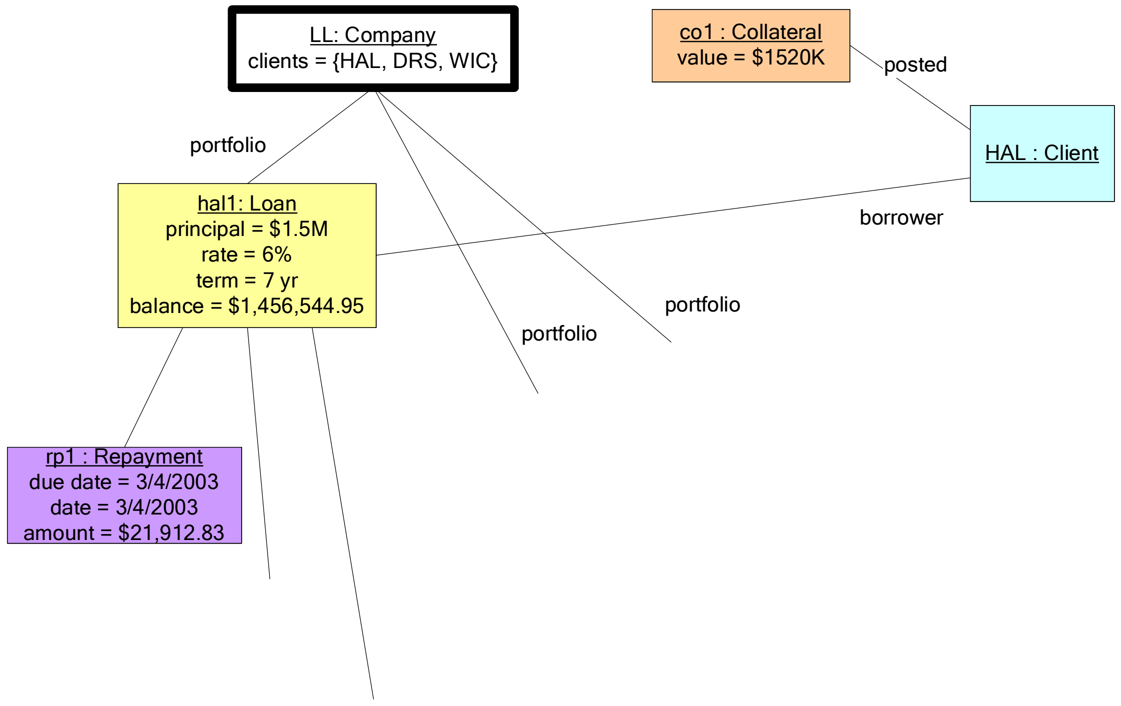
Exercise 4.1 Draw Object Snapshots, Build Information Model from Snapshots. Understand Navigation Expressions. [60 total. Instructor may interleave instructor example with corresponding student lab]

(a) Snapshots [& Optional User Interface]

Read the business state description provided below, and then complete the following steps. (Assume USD currency throughout.)

1. Draw a snapshot describing the business state in terms of objects, attributes and links. Your snapshot will show real-world objects, not just software records. If the links between objects gets cluttered, use attr = value as an alternative.
2. For each **type** of object, write down when a new object of that type comes into existence.
3. [Optional] Sketch either a user-interface, or a domain-specific drawing, for this snapshot.

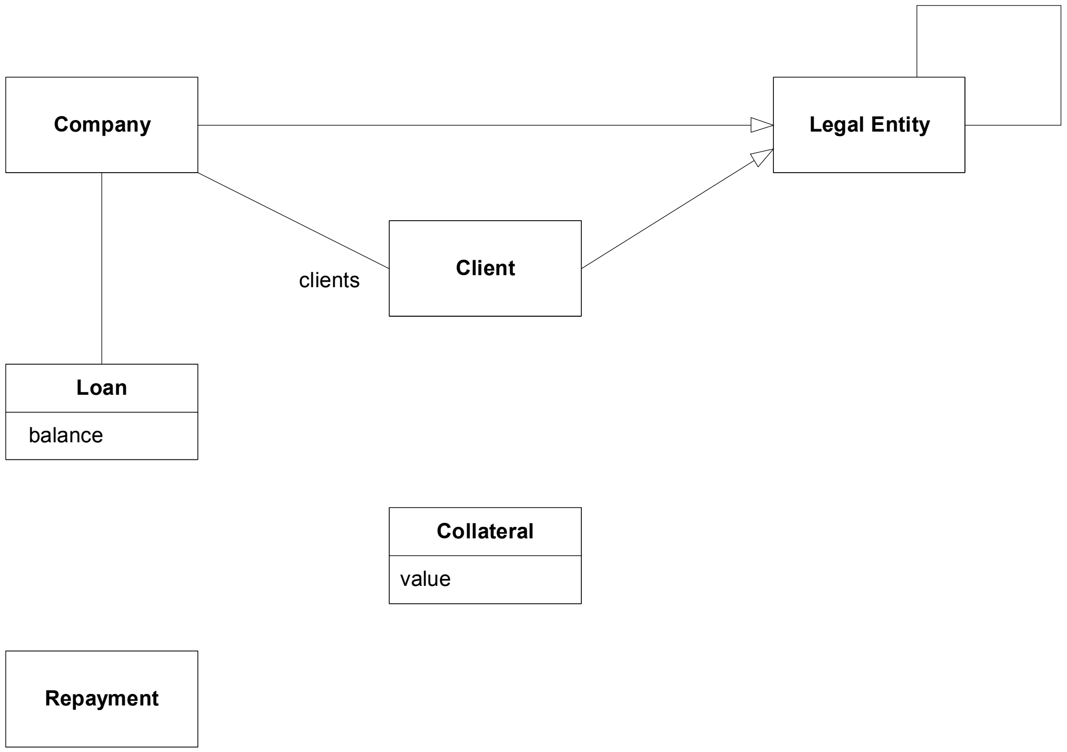
1. LotsALoans corporation (LL) is a company that provides loans to corporate clients.
2. Three of its current clients are Hotels Austerity Limited (HAL), Widget International Corp (WIC), and Daylight Robbery Security Systems Inc (DRS).
3. WIC and DRS are both subsidiaries of the Legal Entity called Variegated Holdings, Inc (VHI).
4. HAL currently has a loan (hal1) with a principal of $1.5M, an interest rate of 6% and a term of 7 years.
5. DRS has a loan (drs1) with a principal of $2M at an interest rate of 7% and a term of 10 years.
6. WIC has one loan (wic1) with a principal of $500K and another (wic2) with a principal of $750K. Both are at an interest rate of 5.8%, and extend over a term of 4 years.
7. HAL has made three on-time monthly repayments against loan hal1, of $21, 912.83 each, on 3/4/2003, 4/4/2003 and 5/4/2003. The due-dates for these repayments were the same as the repayment dates. Since interest is compounded monthly, the balance on hal1 is now $1,456,544.95.
8. HAL has posted collateral for hal1 worth $1,520,000.
9. DRS has posted collateral for drs1 worth $2,144,750.



Type of object	**Created when …**
Client	LLC receives the first ever loan application from that client.
Legal Entity	
Loan	
Collateral	

(b) Build Information Model from Snapshots

Build an information model from the snapshots you have developed, using the solution to the previous lab. You may need to use your general knowledge about the domain.



(c) Interpret Navigation Expressions

Identify the **result** sets of objects the following expressions refer to in the solution snapshot. (Update attributes and association if needed). [Optional] For the informal expressions, write out the **formal** version. For the formal expressions, re-state informally, underlining attribute references.

1. All borrowers. **Result:** { ….. }

Formal:

2. All the clients who have loans with principal greater than $750K.

Formal:

3. The loans for which no collateral has yet been posted.

Formal:

4. The amount of the interest element of repayments rp1, rp2 and rp3.

Formal:

5. LL.clients->select(c | c.parent->NotEmpty)

Informal:

6. LL.clients→exclude(LL.portfolio.borrower)

Informal:

(d) Write Navigation Expressions

Write the expressions (both informally and formally) to refer to the following objects in your snapshot, starting from the top-level object. Note that this is a little bit like saying: A is the answer, what is the question? {x, y} means the set of objects x and y.

{ rp1, rp2, rp3 }

{ VHI }

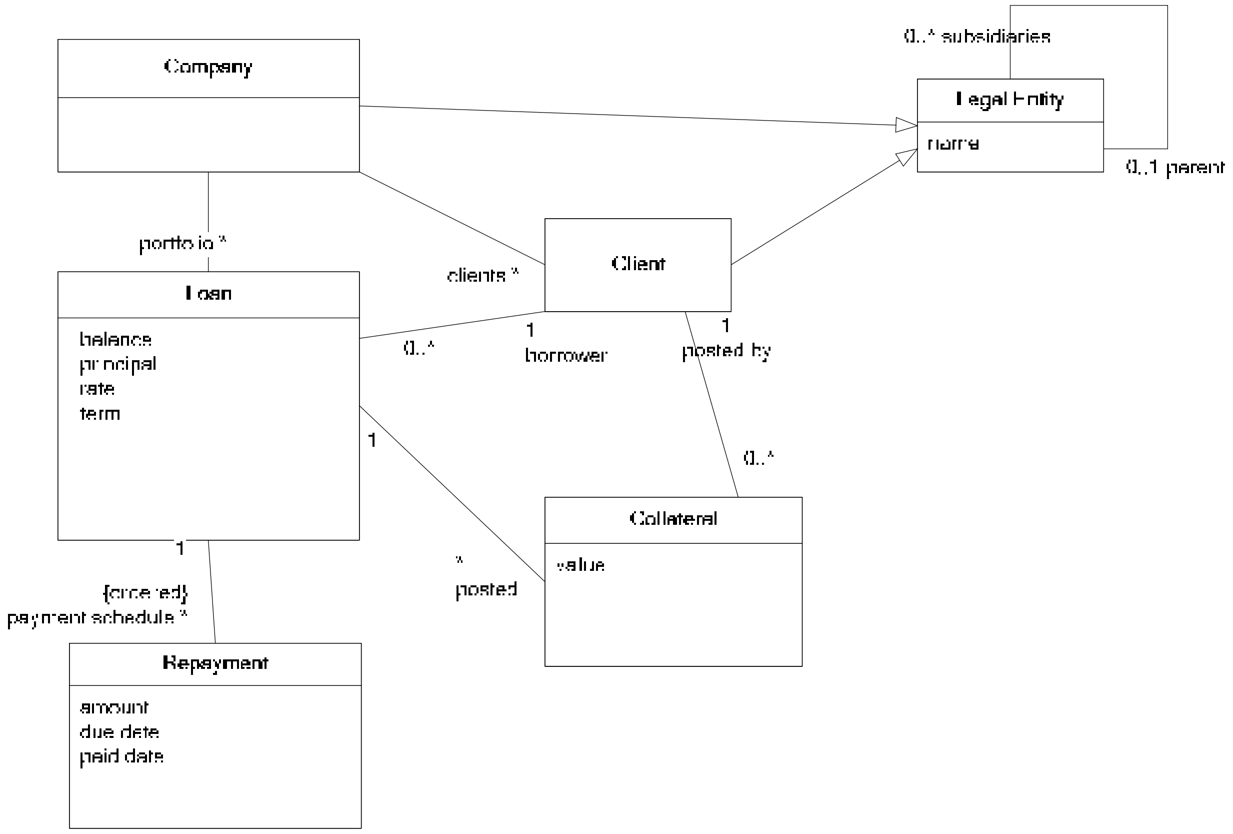
{ 5.8%, 6%, 7% }

Exercise 4.2 Introduce Convenience Attributes. Write Invariants. Utilize Supertypes. [45]

(a) Introduce Convenience Attributes

Introduce a single convenience derived attribute (or association) into your Model for each of the following. Do **not** describe how that attribute might be derived from others. Use a suitable type for the attribute.

- The total value of the collateral posted for a Loan
  Loan: totalCollateral : USD
- The total credit exposure for a Company (sum of loan balances), across its entire portfolio of Loans.
- The number of days by which a Repayment on a Loan is overdue at the present time (assuming a "due date" attribute on Repayment)
- The Repayment amount currently overdue on a Loan, across all overdue Repayments for that Loan. (When doing part (b) for this attribute, assume the existence of an "amount" attribute on Repayment showing the amount due in that repayment.)
- The amount of principal repaid for a Loan.



(b) Write Invariants

In this exercise, you might find that requirements can be ambiguous. Working out precise definitions fleshes out many underlying questions.

i. Derivation Invariants for Convenience Attributes

For each of the convenience attributes created in part (a) above:

- o Look at (or sketch) a snapshot to understand the values of your derived attributes on a snapshot, and how you derived those values from others.
- o Write an informal (and, optionally, a formal) invariant that defines the derivation rule for each derived attribute. On the informal version, underline every term that corresponds to an attribute you have introduced in your information model. For complex cases, break them down into multiple simpler attributes that you put together. You can write these in a simple tabular list; in practice they could be shown as "Constraints" on a UML diagram.

Company: credit exposure	The sum of the <u>balance</u> of all loans in the <u>portfolio</u>
Loan: …	

ii. [Optional] Extra Model Constraint

Write an invariant to stipulate that a <u>borrower</u> from a **Company** is necessarily a **Client** of that **Company**. Be careful not to say that every **Client** is necessarily a <u>borrower</u>.

(c) Utilize Supertypes

Add supertypes and invariants to your model to describe the following by considering these questions:

What object type best encapsulates the variation?
What attributes can be described generically on a supertype across the variations?
Where can those generic attributes be used e.g. to define other derived attributes?
What subtypes would capture the different variations? What attributes do the subtypes need? How do they derive the supertype attributes?

- Loans are secured using different types of collateral holdings: stock holdings, bond holdings, or real estate holdings. Each type of collateral has somewhat different characteristics.
- Stocks and bonds are securities, which are identified by a standard identifier called a CUSIP (Committee on Uniform Security Identification Procedures) number. Every security is issued by a unique Legal Entity called the issuer.
- Bonds have a rating associated with them, which reflects the credit rating of the issuer. They also have a par value, a duration, an interest rate, and a price, which is determined by bond specialists based on current interest rates and other market characteristics.
- Stocks have a price, which is determined by obtaining a quote. (Ignore the bid/ask spread for current purposes.) The exact details of the quoting procedure will not be described further.
- A real estate holding has (minimally) a property value, which is obtained by invoking the services of a real estate valuation expert.

Exercise 4.3 Define Scenarios and Snapshot Pairs. [45]

(a) Scenarios and Snapshot Pairs

1. Name a long-lived activity (hint: possibly several years long) that customer Hotel Austerity Limited (HAL) participates in with LotsALoans Corporation (LLC). When does it start, and when does it end?
   Answer:

2. List the set of finer grained actions that realize the long-lived activity. Introduce and name the long-lived object type ***if*** it is missing. What would be states of an object representing one instance of this type progressing through its lifecycle?
   Answer:

3. Define a scenario which represents a "normal path" through this entire long-lived activity for HAL and LLC. Let each step correspond to one finer-grained action, and pick actions that are conceptually at a consistent level of granularity or abstraction (i.e. agreeAmount Vs. selectLegalEntity). Define your scenario so that one of the states in the scenario is our earlier snapshot from 4.1(a). Consider the states after each step of the scenario, referring to attributes in your information model. Sketch the snapshot after *HAL draws down full loan amount.*

   Scenario: Hal's loan with full collateral, single draw-down, all on-time repayments
   initial state: HAL is not a subsidiary of any other legal entity. It has a credit rating of BBB.
   scenario steps

   1. HAL requests a loan for $1.5M over 8 years from LLC.
   2. LLC offers a rate of 6%, and counter-proposes a term of 7 years with 84 monthly repayments.
   3. HAL agrees to the amount, rate, and term.
   4. LLC stipulates to HAL that collateral must be ….
   5. Hal agrees and posts the entire collateral of …
      Snapshot: the loan is now active
   6.
   7.
   8.
   9.
   10.

   91. HAL makes the 84th monthly repayment of ….

   final state: HAL's loan with LLC has a balance of $0, and is closed.

4. Sketch some other key snapshots for your scenario.

5. List some exceptions that might arise at steps along your scenario.
   Answer:
   At step (…) :
   At step (…) :

(b) Specify an Action

Write an action specification for the actions as outlined below:

- o Start with an informal description of the **post-condition** of the action.
- o Identify the parameters of the action, and re-write the post-condition informally, referring to the parameters and navigating from those parameters using attributes from the Information Model. To help identify parameters, look at a before snapshot and decide which object/value (or objects / values) would need to be identified for that snapshot to change. Use the underline convention to show references to attributes and to the parameters. Note that the actions are not independent if they refer to some common attributes.
  - o Consider making the specification more concise and in terms a client might use to describe it, by introducing convenience attributes and invariants. Move parts of the pre/post into invariants, if they represent constraints on every valid snapshot (not just to the current action).
  - o Add the pre-condition if any, for which that post-condition applies.
  - o Formalize parts or all of your specification with formal expressions.

drawDown **(loan: Loan)**
-- Increase balance by drawing down the loan amount: assume total principal is always drawn down in a single operation
pre: *-- what needs to be true for drawdown to be done? Refer to information model attributes*

post: *-- what is changed after a drawdown?*

scheduleRepayments **( )**
-- set up a series of repayments that will pay down the balance over the term
pre: *-- refer to information model: what needs to be true before repayments are scheduled e.g. what terms agreed?*

post:

makeRepayment **( )**
-- pay interest and repay some of the principal

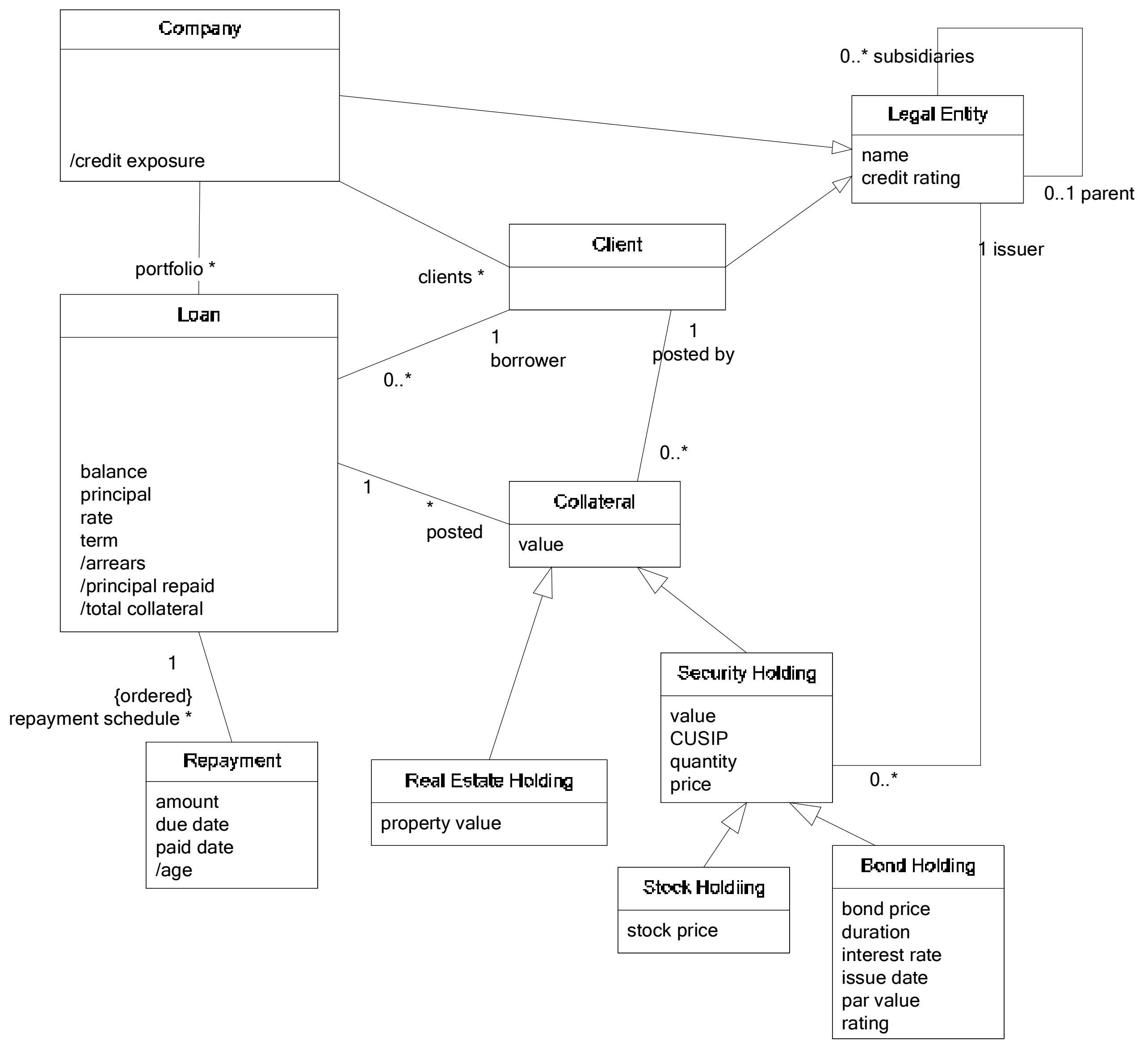
Exercise 5.1 Goals and Domain Modeling [60]

Start with the information model from the last solution. Read through the following rough requirements notes for the LLC Loans System and its associated business processes. Then answer the questions that follow.

a. **Revenue & Risk:** LLC generates **revenue** by charging interest on the loans that it extends to its (corporate) customers. Lending entails **credit risk**, which must be monitored and controlled to ensure that the **projected revenue** is not outbalanced by unacceptable risk. Risk can be controlled by adjusting factors such as to whom LLC makes loans, require collateral, borrower credit ratings, etc.
b. Risk in relation to projected revenue must be monitored by loan officers by taking into account:
    - the credit ratings of borrowers;
    - the principals and interest rates on loans;
    - the **regularity** of the borrowers' repayments;
    - the rating of any bonds used as collateral;
    - the value of any assets used as collateral, and
    - the credit exposure entailed by loans.
c. **[Optional]** Currently, the only available measure of credit risk is credit exposure. The credit exposure for a set of loans is computed by summing the book values (balances outstanding) on those loans.
d. **[Optional]** Whenever any risk driver mentioned above changes, the loan officers for the affected loan(s) must be notified immediately.
e. **[Optional]** Risk must be controlled by constraining lending activities and by adjusting attributes such as borrower credit rating and required collateral from time to time. These constraints can be varied over time, and be based the parameters mentioned in b, above, and at any meaningful level of aggregation (i.e., all loans, loans for a particular client, etc.).
f. **[Optional]** Loan officers should optimize lending activities by trying out various "what if" scenarios involving hypothetical changes in any of the parameters that can affect projected revenue or risk.
g. **[Optional]** The information that can be made available to the Loans System typically varies in currency as follows:
    - stock price quotes are delayed by 20 minutes;
    - bond valuations and ratings are available each day;
    - real estate valuations are available on request;
    - repayment activities are recorded each day;
    - client credit ratings are available daily;
    - legal entity hierarchy changes are available daily;
    - client credit downgrades may occur at any time intra-day, and must be taken into account immediately.

Questions

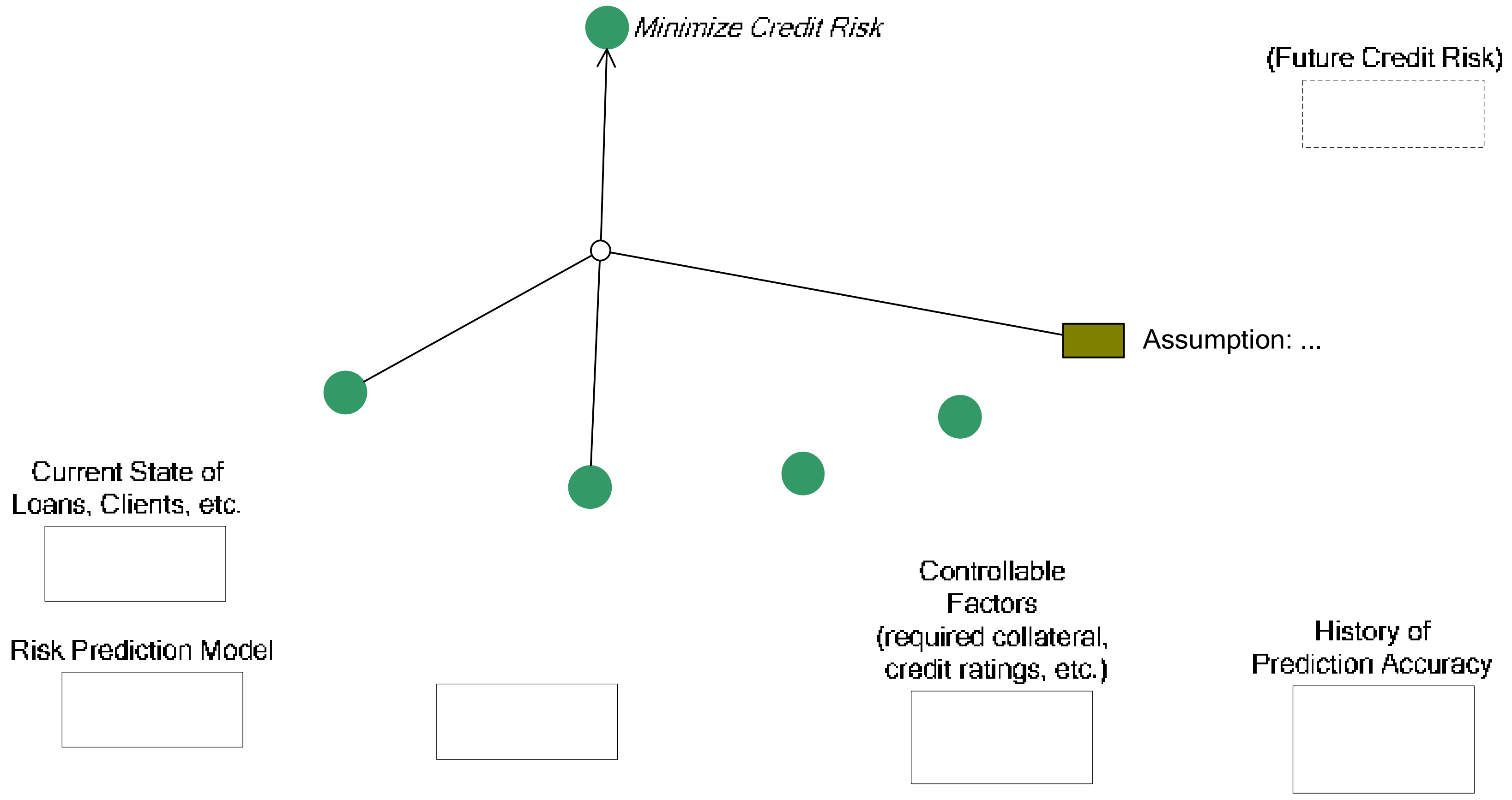
1. Locate "revenue", "projected revenue", "credit risk", and "repayment regularity" in the information model and understand their derivations.



2. Statement (b) suggests a high level goal implicit in the information provided above. State that high level goal.

3. Consider the goal "Minimize credit risk" which, at all times, minimizes the risk of future credit loss. This goal is fundamentally about the risk of future credit events, and can at best be approximately measured and met. Devise a goal model. Start with identifying what the top-level goal wants to "control". Then devise a suitable set of sub-goals for this goal, showing which parts of the domain they control/observe. When considering any one sub-goal, you can assume that other sub-goals are met. If there are some addition domain properties (e.g. assumptions you make about risk, or prediction, or history) add them to the goal refinement. Each sub-goal should "at all times" maintain some relationships, based on the following hints:
   a. Assuming the existence of some "risk prediction model" that helps predict credit risk
   b. A set of risk-related parameters that can be tweaked e.g. collateral demands and credit-ratings
   c. Recognizing that no risk model is perfect
   d. A desire to improve the risk model over time

   Informally list some events that **<u>must</u>** be handled for your subgoals to be met i.e. which events could perturb the sub-goals.



Minimize Credit Risk
(Future Credit Risk)
Assumption: ...
Current State of Loans, Clients, etc.
Risk Prediction Model
Controllable Factors (required collateral, credit ratings, etc.)
History of Prediction Accuracy

4. **[Optional]** Requirement note (e) mentions “lending activities and attributes” in connection with constraint definition and enforcement. At what point in the specification and development of the Loans System will we have to know exactly what these activities and attributes are? Which models will document them?
5. **[Optional]** In requirement note (f), which is the higher level goal: to support “what if” scenarios, or to optimize lending activities? (Hint: Ask “why” to get to higher level goals; ask “how” to get to lower level ones.)
6. **[Optional]** Propose a definition of “optimal” in the concept of optimizing lending activities in (f).
7. **[Optional]** Describe / discuss some use cases that could provide a refinement of requirement note (f).
8. **[Optional]** Could there be a conflict between the timeliness properties implied in requirement note (g) and the timeliness stated in (d)?

Exercise 5.2 Build a Role Activity Diagram, Activity Specs, Info. Model [30]

Starting with the process and activity descriptions in the table below:

1. Build a Role Activity Diagram (RAD):
   a. First name the overall process and write a short postcondition for the overall (successful) change of state for the process.
   b. Use the list of finer-grained activities below, and write down the main post condition of each. This clarifies the start/end of that activity.
   c. Place the activities on a RAD. Make this a "black-box" RAD i.e. minimize "internal" structure or sequencing of actions unless they are right before, after, or directly involved with external interactions. Initially do not connect the activities.
   d. Annotate the line ends in and out of the activity with the name of the state before / after the activity.
   e. Connect outgoing lines of one activity to incoming lines of another activity if the state annotations are the same. Revise state names.
   f. Add other flows and connections, including branches, forks, etc. Use annotations if the formal notation gets complex. If you get hung up on how to graphically show iteration, exceptions, etc. simply annotate the diagram with footnotes or comments e.g. *"Activities A, B, C repeated until condition X holds", or "Activities X, Y, Z are interrupted or aborted if event E happens"*
   g. Check that your RAD works by walking through your earlier scenario (from Exercise 4.3(a)).

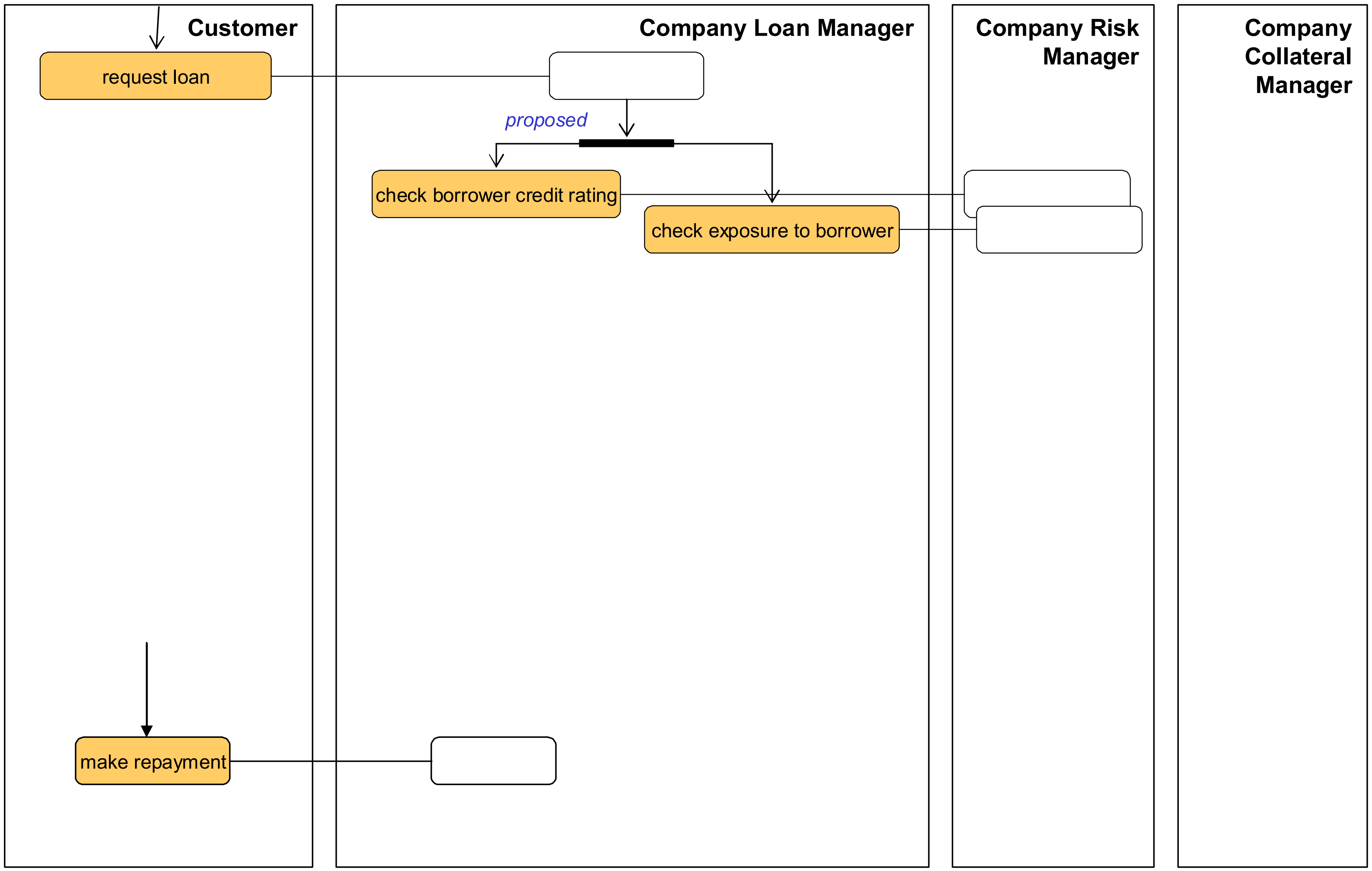
2. Write an activity specification for **draw down** and **post collateral**. You can expect your specification to be very similar to the generic action spec you wrote earlier, but with a RAD you also indicate which role provides each input required for the activity, and if any other roles are involved.

3. Update your information model to make sure it supports your specification. Write a short description of the types *Loan* and *Customer*, indicating whether, in your business model, these represent types in a software system or not.

The process of obtaining and paying off a loan:
- Request a loan
- Check borrower credit rating
- Check exposure to borrower
- Stipulate collateral
- Agree principal, rate, & term
- Post collateral (assume all collateral must be posted as one unit)
- Activate loan
- Draw down
- Make repayment

5.2(1) Answer:

Process: **Goal:**



5.2 (2). Activity Specification

activity: post collateral
roles:
inputs: from
precondition:

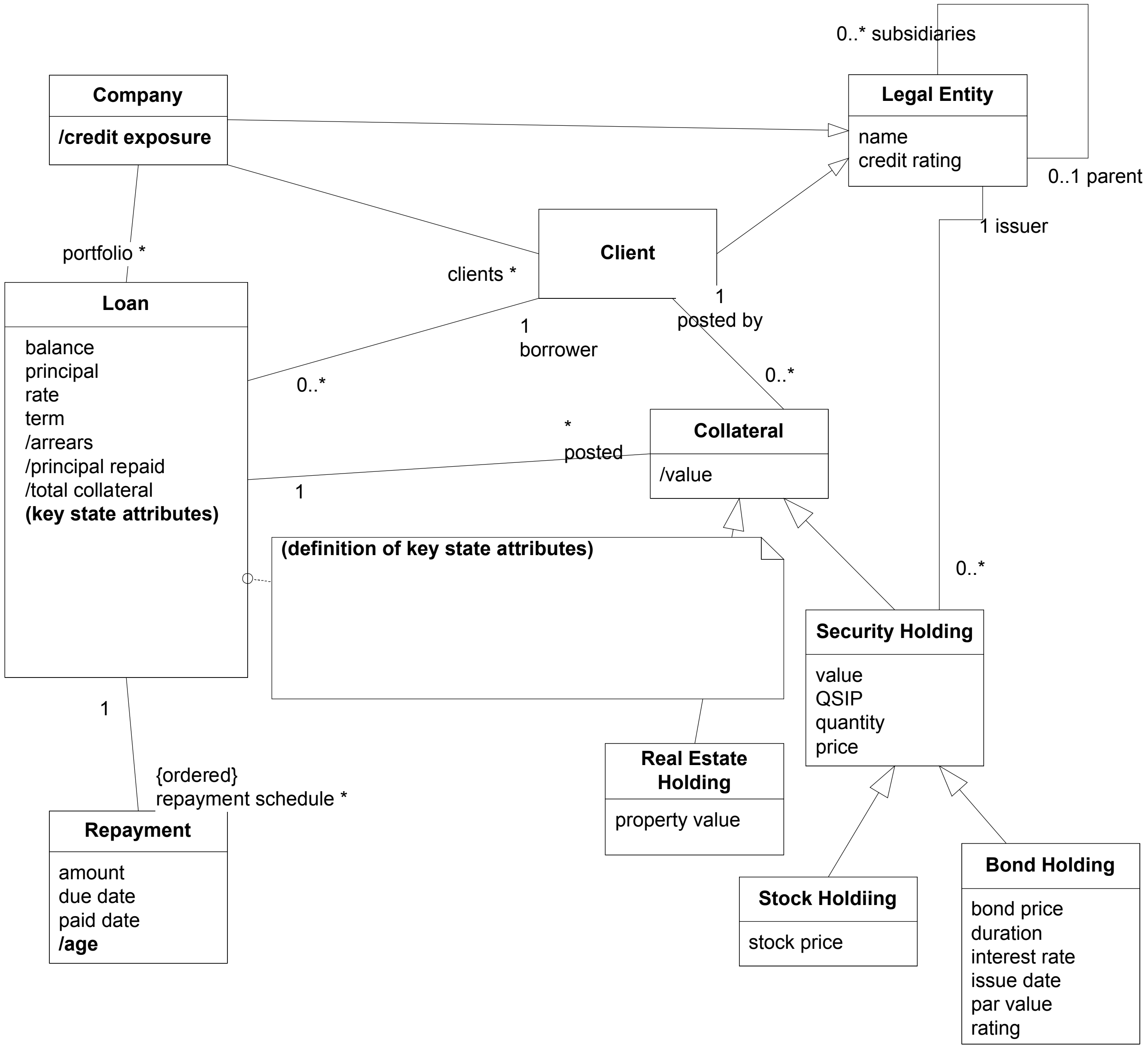
postcondition:

activity: draw down
roles:
inputs: from
precondition:

postcondition:

5.2 (3) Updated Information Model

Since this is a (case) process for handling a Loan, most of the changes should be centered on the Loan type.



Exercise 5.3 Build a State Diagram and State Definition Matrix.

Based on the RAD you developed, identify the object that has an interesting lifecycle through the entire process.
Sketch a state model for that type. Transitions should correspond to actions in your RAD, and states should include key states on the RAD.

Construct a matrix showing each state and how it is derived from the attributes in your information model.

[Optional] Construct a matrix of states and actions to ensure that you have covered all state transition cases.

State	**Derivation Expression**

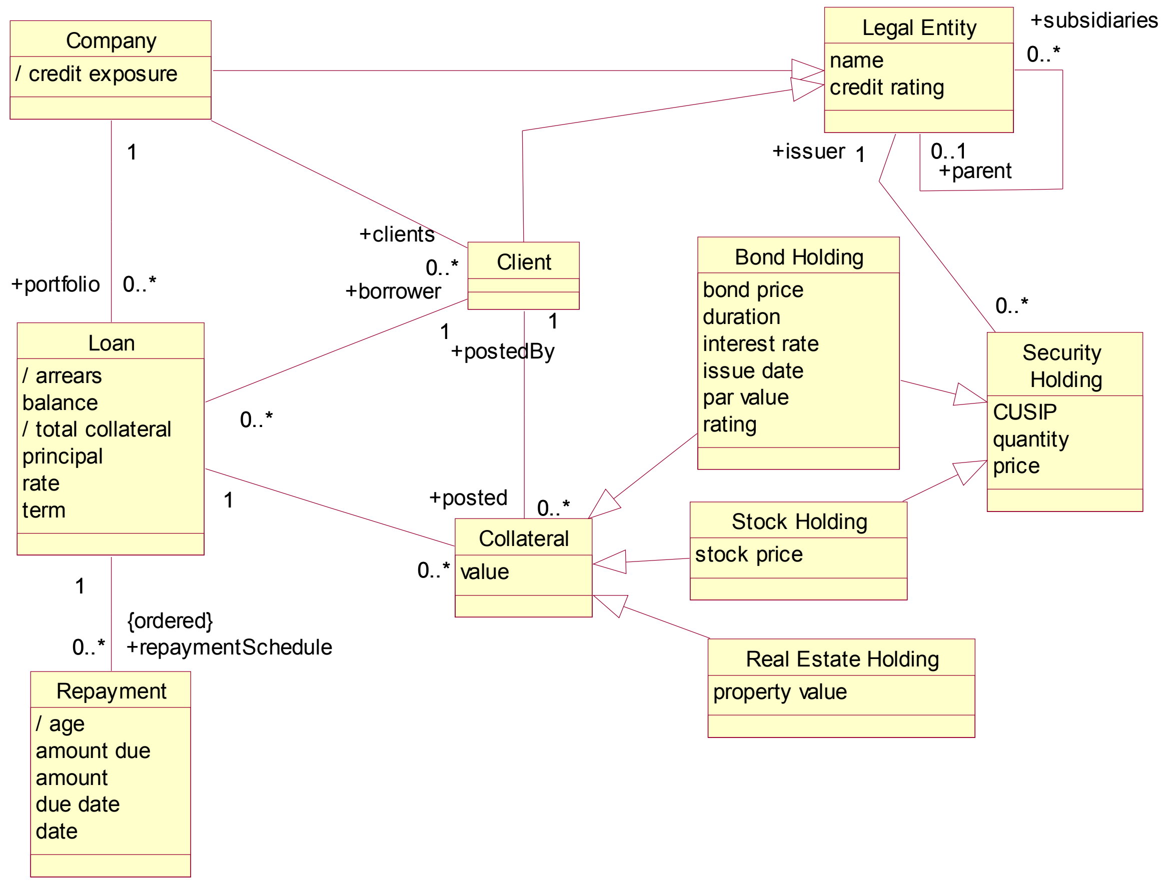
Exercise 6.1 Mapped Black Box Assembly

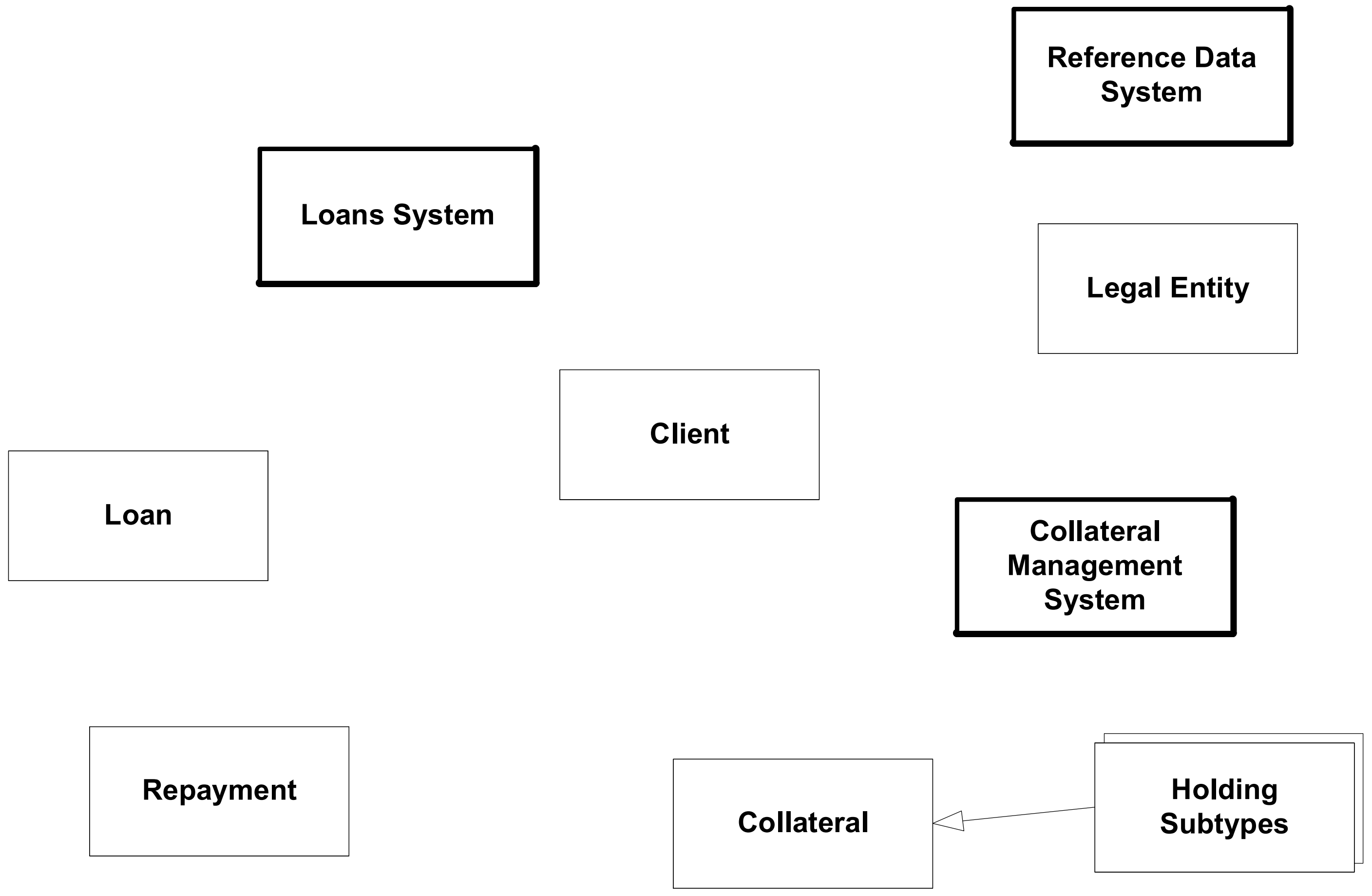
Your black-box assembly will consist of the Loans System, a Reference Data System, and a Collateral Management System.

Starting with the business information model:

a. Add these systems to the model, with UML-composition associations (black diamond) to the types they directly own.
b. Trace through the model until you reach a type which is either (a) not owned, or (b) partly owned. Assume that the Loans System owns and tracks the credit rating of its Clients, and in order to do this has to also own and track the credit ratings of its entire legal entity structure. The Reference Data system owns the core Legal Entity structure itself, with the actual parent-subsidiary relations. Model this as split objects, with the credit ratings on the Loans System side, and the parent-subsidiary association on the Reference Data side.
c. In the case of a partly owned type, or where portions of the attributes are owned by different systems, introduce a new type name (e.g. "Collateral Management View of XYZ") with the owned attributes and associations, and introduce an association with the other part of the type. This is a <<cross-system>> association and will typically have some consistency requirements associated with it.
d. In case you split a type that has a sub-type relation, break into two separate types (with inherited properties assigned to one or the other) with an association between them i.e. the subtype is represented by two different objects in two different systems, and the association represents the pair-wise correspondence.
e. Do any goals or invariants cross the systems? Mark them on your diagram with references into the parts of the model they reference.

Answer. Initial information model is below. Template to show the partition across the black-box assembly is on the next page.
Note: Consider information split across systems, and associations/invariants/goals between them.)




Reference Data System
Loans System
Legal Entity
Client
Loan
Collateral Management System
Repayment
Collateral
Holding Subtypes

Exercise 6.2 Use Case Specification

1. Draw a use case diagram showing the following use cases:
    - **post collateral** (assume all collateral must be posted as one unit)
    - **draw down**
    - **update LE info.** (***Just for this use case :*** assume that, for performance reasons, Loans System will keep a copy of the parent/subsidiary relation from the Reference Data system in order to manage it's credit ratings properly and efficiently. Such information would typically not be mixed into a purely logical view of the architecture, which should typically just describe the authoritative sources and not performance optimizations.).

    Just for the **update LE info** use case: assume this is a daily batch synchronization of replicated LE hierarchy information across the systems, and that the Loans System stores an extra credit rating attribute on each LE instance that it maintains.

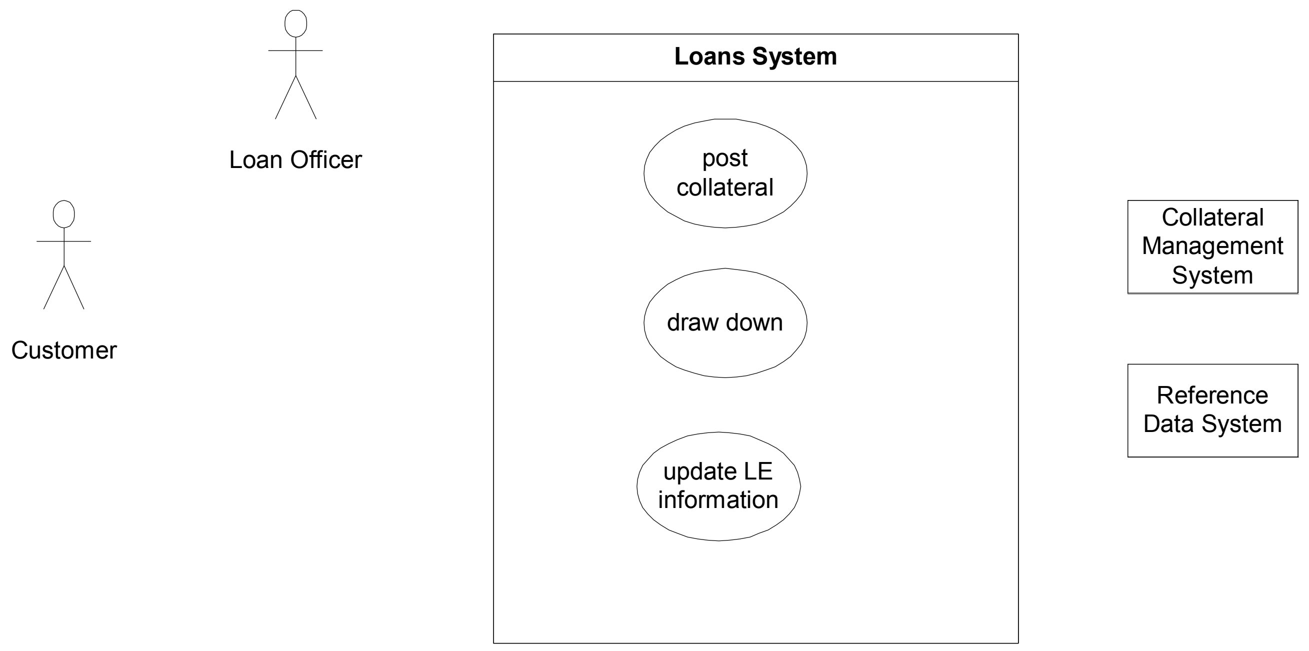
    On your use case diagrams, include the following external human roles (actors) and systems:

    i. **Customer (actor)**
    ii. **Loans Officer (actor)**
    iii. **Collateral Management System**
    iv. **Reference Data System.**

    Add links from use cases to actors that immediately interact with the loans system in that use case. In the case of "draw down", assume that there is a direct way for the customer to draw down funds without involving the loan officer, e.g., via electronic funds transfer.

2. Specify the three use cases in more detail as shown below. Include the net state change in the "system", as well as all communications with other actors. Reference and underline attributes or inputs or outputs, remembering that you are now talking about the state of **the system**, and inputs/outputs to that system. Your inputs and outputs can be simple values (dates, quantities) or objects (e.g. Order), without details about how those objects are identified. Understand how the use cases interact via these attributes.

1: Answer:


Loan Officer
Customer
Loans System
post collateral
draw down
update LE information
Collateral Management System
Reference Data System

Use case: **draw down**
Actors:
Inputs: ___________ [**from** ____________]

Outputs: ___________ [**to** ____________]

Trigger:
Precondition:

Postcondition:

Use case: **post-collateral**
Actors:
Inputs: ___________ [**from** ____________]

Outputs: ___________ [**to** ____________]

Trigger:
Precondition:

Postcondition:

Use case: **update LE info**
-- synchronize LE information between systems
Actors:
Inputs: ___________ [**from** ____________]

Outputs: ___________ [**to** ____________]

Trigger:
Precondition:

Postcondition:
For every instance of .... In ... There is ....

Exercise 6.3 Describe Steps and Alternate Paths.

Elaborate the “post collateral” use case with steps of the main "success" scenario. Underline attributes or *include otherUseCases* in your steps. Check that your use case specification abstracts the detailed step-level information exchanged via the inputs/outputs of the specification. Write at least one successful alternate path description, and one failure alternate path.

Use case: **post collateral**
Actors: Loan Officer, Loans System
Inputs: Loan, Client, list of assets (obtained from Client representative) to be posted as collateral for Loan
Outputs: Confirmation of posting (to Loan Officer), **Collateral Update (to Collateral Management System)**
Trigger: Client contacts LLC offering assets as collateral against loan
Precondition: Loan is “proposed”; client offering collateral is client for Loan; collateral fulfils general criteria for suitability.
Postcondition:
If the collateral offered is sufficient, the offered collateral extends the set of holdings (**in Collateral Management System)** already in use as collateral against loan and the loan is funded. If collateral is insufficient, there is no change to the system.

Main (success) steps:
1.
2.

Alternate paths:
#…

#…

use case <...> (only required if the above use case “includes” other use cases)
post: ...

Exercise 6.4 Identify and Specify Use Case Operations

Look through the steps in your use case solution from the previous exercise, and identify (a) operations that the system provides that are invoked during that use case; (b) <<**out**>> operations that other actors provide that are invoked by the system; (c) raised <<**events**>> or other behaviors that the system needs to exhibit. You will sometimes need to consciously choose the granularity and level of abstraction you want to call "one operation" e.g. placing an entire order Vs. adding one line item at a time.

Identify all inputs and outputs for each such operation. **You should be able to describe what the operation does and its outputs in terms of its inputs and persistent state.**

Specify at least 2 operations with at least its names, signature of all parameter types, and informal description of its postcondition. Reference and underline attributes in the postcondition. Remember that all the terms referred to in a postcondition should be one of (a) an input parameter provided to that operation; (b) an attribute of that input parameter; (c) an attribute that was part of the system state.

Use case **post collateral**

Exercise 6.5 User Interface Specification

Using the steps of your use case and the information model:

1. Draft a list of named windows, and describe each.
2. Sketch an information model with the main UI state, including the information model of each view and its relation to the core information model
3. Define any application-specific UI events if they correspond to multiple detailed UI alternatives
4. Show state diagram of main UI windows, labeled by all UI events
5. Describe operating procedures for users using terms from the information model and use cases.

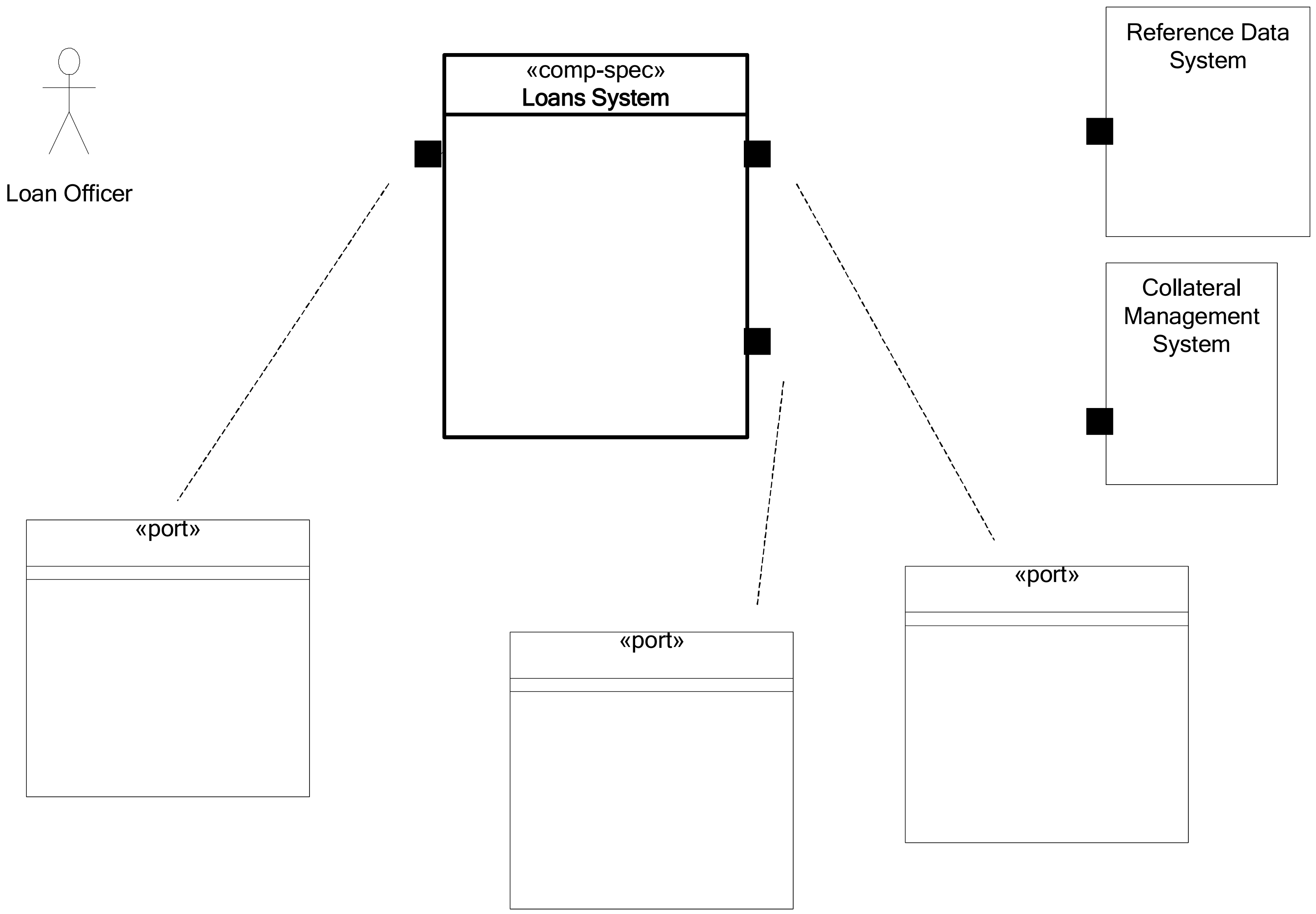
We will skip this lab. It is included for outside class use.

use case **post collateral**

Exercise 6.6 Specify Black Box Ports and Assembly

In this exercise we will set up our system description for consistent and recursive treatment between black box and white box views.
Note: This exercise can be scoped down to the ports and operations required by the "post collateral" use case for the purposes of focus and continuity with the previous exercises.

1. Draw the ports on your "system", with the system having a «comp spec» stereotype and icons for each port that provide or require services from other actors. Name each port.
2. Separately show each port as a type, listing the system operations in/out of each port including incoming and <<out>> or <<event>>.
3. Show the connectors from these ports to the actors or external systems.
4. For each black-box involved, list the elements (types, attributes) of the information model it owns.


Loan Officer
«comp-spec»
Loans System
Reference Data System
Collateral Management System
«port»
«port»
«port»

Exercise 7.1 Draft Core Type Components and Use-Case Coordinator Components.

(a) Find Core Type Components

Using your black box information (adapted from the business information model) from 6.1:

Identify the «core» types. These types tend to have a unique identification in the business, 1-to-many associations to other types that can be considered "detailing" types on them, associations to some type that serves primarily to "classify" them, an association to the "top", and do not have mandatory ("1") associations to other types. Defer analysis of any derived associations you might have. Optionally, if you have objects with a complex lifecycle that is quite different from that of the core type, consider partitioning that (with its detailing types) into a separate component i.e. treat it as its own «core» type by bending the above rules a bit.

Assign each core type <X>, with its detailing types, to its own component. Typically name that component something like <X>Mgr, <X>Manager, or <X>Mgmt. Show assignment by the UML black-diamond (called "composition" in UML).

Check your model and its associations to see if all core and detailing types are unambiguously associated with just one component. You may need to associate a direction with some associations to reduce bi-directional dependencies across components. Think about likely navigation paths across components in making this decision (it will be validated when designing end-to-end collaborations across components).

Lab – LotsALoans Corp (LLC)

(b) Add Use-Case Based Components

Using your initial component partition and the use case model:

Introduce a component for each large-grained use case on the system, with the operations from the steps of that use case. This component will act as the primary initial "receiver" of the external operation requests (e.g. via a UI) from the steps of that use case, and will also typically act as coordinator across the «core» type components. It does not hold session-specific dialog logic or state; that is considered part of the UI/dialog tiers. Typically name component for use case <U> something like <U> Controller or <U>Coordinator, or a role-name version of the use case.

Lab – LotsALoans Corp (LLC)

(c) [Optional] Add UI / Dialog Components

If your system has a user interface, add a UI component (pure presentation) and a corresponding dialog component (remembers the state of the dialog, with preceding selections and the results of preceding requests).

Exercise 7.2 Design Collaborations across Sub-Components

Starting with your candidate set of sub-components, including those based on «core» types and those that support use cases. Keep your model of **white-box information ownership** front of you. Pay special attention to the directions on cross-component associations.

Draw an instance of each sub-component, and each external system you interact with (layout the sub-components in a consistent and intuitive way). For each "architecturally significant" operation (likely to uncover new paths/interaction), draw a collaboration diagram as follows:

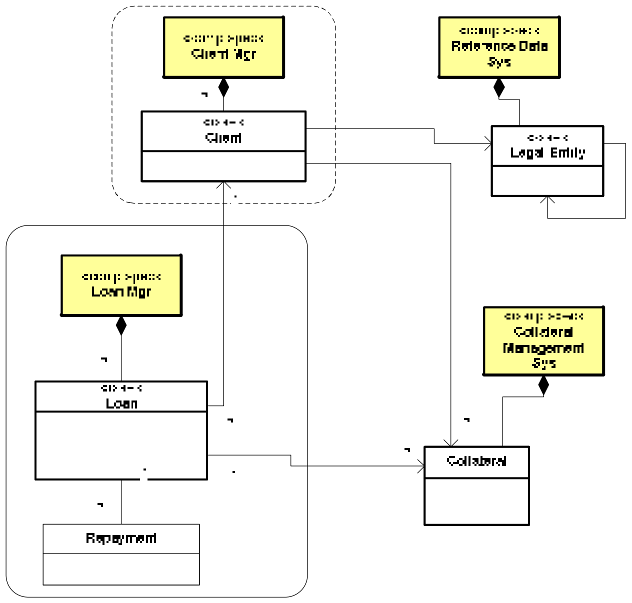
i. Select the use-case component that is the "receiver" of that operation.

ii. Draw the request and label it: **x := request (a, b)**, where x is the result expected back (if any), and **a,b** are the objects or values passed in with the request. Note that **x, a, b** are variables, not types. Remember that a given sub-component can process a request using (a) any input parameters it received; (b) any information it receives from other components it interacts with; (c) anything it "knows" as part of its own information model.

iii. Follow through the rest of the interactions that ensue, using the names **x, a, b**, names of attributes defined in the information model of each sub- component, and introducing new names ("local" variables) as needed. Note that an arrow with **1.1 x := r(a,b)** simply means that at step 1.1, the method r is executed, using parameters **a, b**; it does not mean a hard-coded invocation from one sub-component to another. You can add annotations and/or UML icons as information explanation of iteration, conditions, etc. if you need to.

Check that you have accomplished all of the post conditions specified for that use case, and completed all external communications.

Lab – LotsALoans Corp (LLC)
use cases: **draw down, post collateral**



Exercise 7.3 Define Sub-Component Assembly

Assume that you have the following two kinds of connectors available to connect your sub-components:

A default “call” connector: it requires an outgoing method call from one end to exactly match the invoked method on the other end. It provides synchronous call-and-return behavior.

An «event» connector: it takes several pairs of:

an event to be raised at one end, with some name and event information type.

a method to be invoked at the other end, with some method parameter types.

a definition of the mapping from event information to method parameters

It provides asynchronous communication of the event, resulting in the invocation of the method.

Build your sub-component assembly as follows:

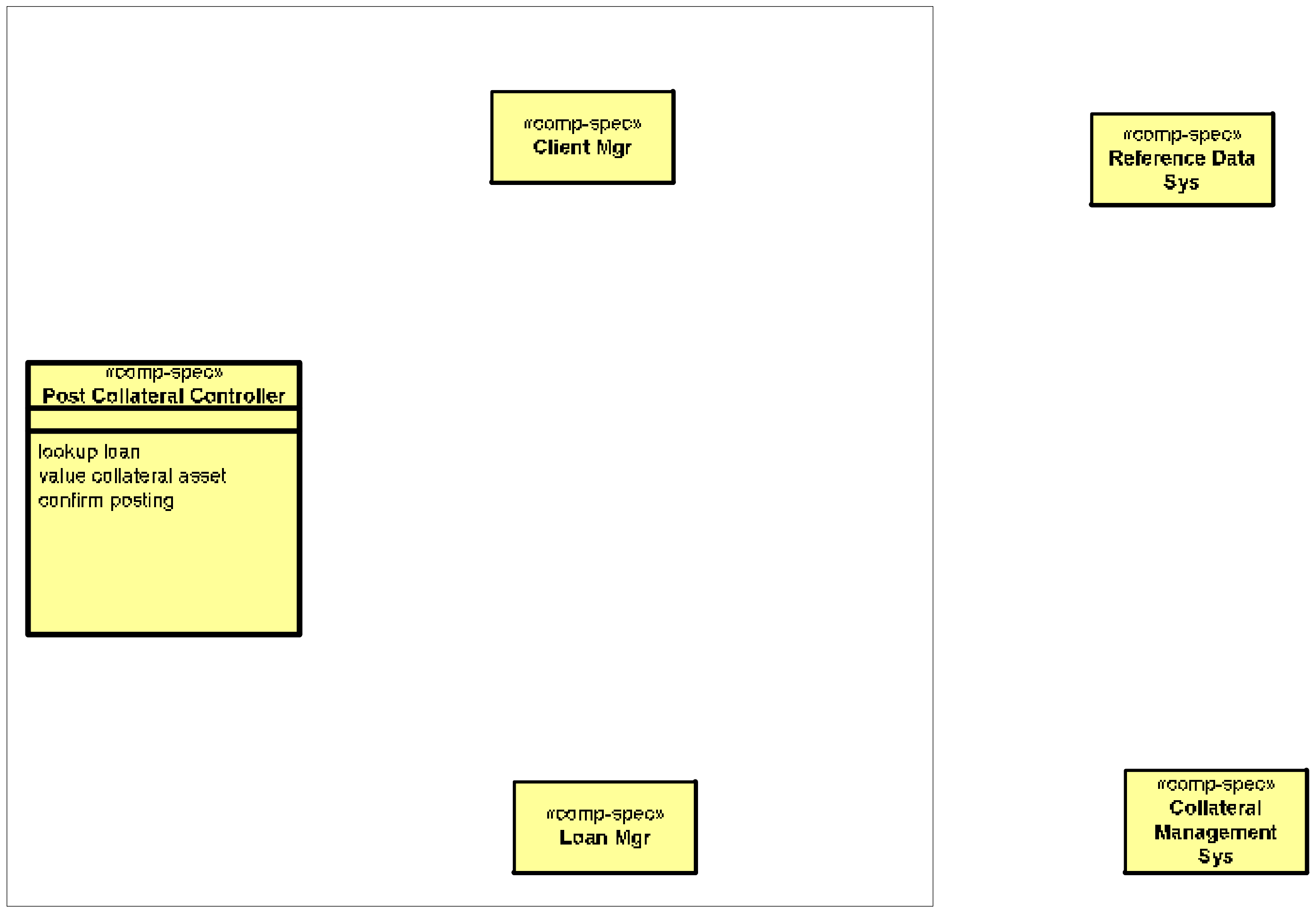
Draw a line connecting sub-components that interact in your collaboration diagrams, and connecting them to external actors.

Decide what connector type you will use and annotate the connector with «connector type», unless using “call” connector.

Define a «port» at each end of the connectors. Name the port ~A if it is purely a client port to a port A for a set of client-server interactions across a default “call” connector. Name the port something like <X>Events for a port that is nothing more than a source of events.

Define at least two ports with some of their operations, <<out>> calls, and <<events>>, including parameters and their types.

Lab – LotsALoans Corp (LLC)


«comp-spec»
Client Mgr
«comp-spec»
Reference Data Sys
«comp-spec»
Post Collateral Controller
lookup loan
value collateral asset
confirm posting
«comp-spec»
Loan Mgr
«comp-spec»
Collateral Management Sys

Exercise 7.4 [Optional] Specify Sub-Components

Specify each sub-component as follows:

Draw a «comp spec» with associations to a set of «port»s.

For a server port **A**, or an event port: list the incoming operations or outgoing events with names and signatures. For a client port, the name **~A** can suffice as a reference to the corresponding server port **A**, without **~A** having to list operations or events.

Draw the information model of the «comp spec». Use UML "composition" from the «comp spec» to show the relationship.

Add a «data type» class for any input or output parameters referred to in the ports. There will typically be some attributes in the persistent information model and in the data types that "link" them together.

Write a short post-condition specification for at least two of the operations on one of the component ports. Underline terms that refer to attributes in the information model.

Exercise 8.1 Apply a Pattern

You are given the following technical architecture style, mapping from logical components, ports, events, and methods to the Java platform:

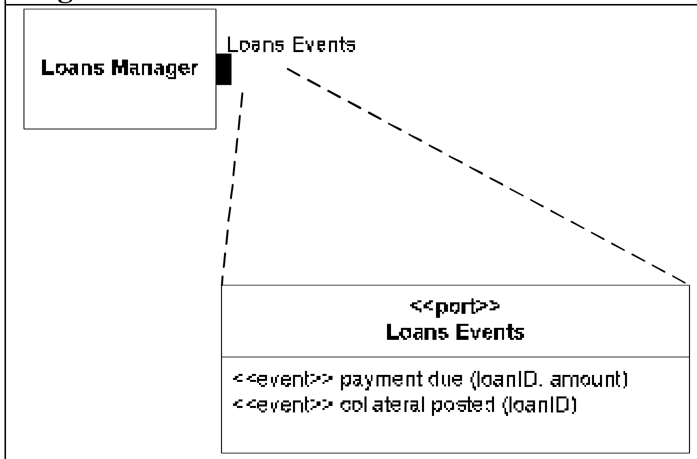
Logical Model	**Platform Mapping**
Any element name	Spaces replaced with "_"
Component	Java package
Port	A public Java class in that package

<<event>> e (x, y)	An interface:
	    interface E_Listener {
	        void notify_e (T) ; // where T is a struct-like object combining event parameters x and y
	    }
	A collection of 3 methods and 1 data member on the port:
	    o add_e_listener (E_Listener o): adds o to a list of E-Listeners
	    o remove_e_listener (E_Listener o): removes o from the list of E-Listeners
	    o notify (): calls notify_e on each listener in the list

Event-method connector	[Optional] If you are familiar with dynamic proxies in Java, sketch a generic event-method connector class which could take a pair of ports and multiple event-method pairs, and do the right event registration and adaptation between the ports.

Create the result of applying this style to the following logical white-box model:

Logical Model	**Platform Mapping**

 Loans Manager
Loans Events
<<port>>
Loans Events
<<event>> payment due (loanID, amount)
<<event>> collateral posted (loanID)
	package Loans_Manager;
	
	interface payment_due_listener {
	  ….;
	}
	
	interface collateral_posted_listener {
	  ….;
	}
	
	public class ….. {
	
	}

Exercise 8.2 Extract a Pattern

Identify a named pattern in your information model, as suggested below.

Define the "parameters" of that pattern as types and/or their attributes. Simply write the parameters as **<Type, attribute, ...> <Type, attribute, ...>**. Other parts of the pattern can be "generated" from those parameters. If you need to "generate" a name for an element in your model from the pattern write a compound name e.g. <Resource>_Consumer becomes Room_Consumer if <Resource> = Room.

Define the "expansion" of that pattern that should result when it is applied to given parameters.

Apply the pattern to your model by showing a dependency arrow from your package to the pattern package, annotated with **bind** [pattern parameters] i.e. the bindings of pattern parameters.

Lab – LotsALoans Corp (LLC)
We will skip this student lab. It is included for outside class use. Do one or more of the following: Loans may have a start and end date. Purchase bids might have a minimum and maximum dollar amount. Extract a pattern for a Range of <X>, including some common attributes on Ranges and <X>s. There is an underlying pattern for resource allocation. Define this pattern using things like <Resource>, <Capability>, <Resource Owner>, <Job>, etc. Include the main attributes and associations, and an invariant for "no double booking". Check that it can cover such uses as: o dentists to appointments, or o rooms to hotel guests, or o cars to rental requests. Each Legal Entity can have any number of subsidiaries. In other contexts, a product might have a corresponding bill-of-materials. When you got an order, or when inventory ran low, you had to order more from suppliers based on the bill of materials. Hence, each product is comprised of parts; and some parts (excepting the "atomic" ones) are comprised of sub-parts. Find, extract, and apply the pattern "Composite" to your model.

Exercise 8.3 Design a Connector Spec + Realization

For the connector described below:

Sketch one connector specification pattern. This will include enough information that someone can use it in a white-box assembly and understand its effect, without knowing about the protocol or technology detail that underlies it. Create one example of "applying" that pattern, and work out the "expanded" result of that application.

Sketch one possible connector realization for that same connector on a platform such as .NET or J2EE.

On your existing case study model, or inventing a small fragment for the purpose, show how you might apply the connector spec in your white-box architecture, and its realization in the technical architecture. Work out the "expanded" result of applying those patterns.

Instructor Example - Hotel Austerity Limited (HAL)	**Lab – LotsALoans Corp (LLC)**
A connector that: **Spec:** Keeps a property at a "sink" end updated with any change in a corresponding property at the "source" end. The properties may be simple (numbers, strings), or complex (structured objects, or entire sets of objects). **Realization:** Does the above by: o Requiring the "source" end to offer an outgoing event with data about changed values o Requiring the connector itself to offer event subscribers an interface via which different sets of published properties can be referred to by a single subscriber, in order to subscribe to their changes: **subscribe(PropertyName, Subscriber)**, with an unsubscribe() operation. o Requiring every interested "sink" end to subscribe with the connector itself, and to provide an operation to be invoked when a change has taken place.	We will skip this student lab.

Exercise 8.4 Design a Component Realization

Consider a single target technology for (some of) your components e.g. J2EE, .NET, a proprietary component platform, or a legacy environment, perhaps choosing something that is relevant to your project. Discuss any regular mappings you can find between the logical view of components and their interactions, and how those might map to the target technology e.g. names of components, how they are located by others, what the operations correspond to in the technology, how in or out parameters are communicated, etc.

Instructor Example - Hotel Austerity Limited (HAL)	**Lab – LotsALoans Corp (LLC)**
	We will skip this student lab.

Exercise 8.5 Define a Data Model

Define a pattern for dealing with one (simplified) part of an object-relational mapping. Find one place to apply this pattern to your information model Work out the expanded result of applying that pattern.

Instructor Example - Hotel Austerity Limited (HAL)	**Lab – LotsALoans Corp (LLC)**
Mapping a subtype to separate tables	We will skip this student lab. Mapping a N-M association to an associative table

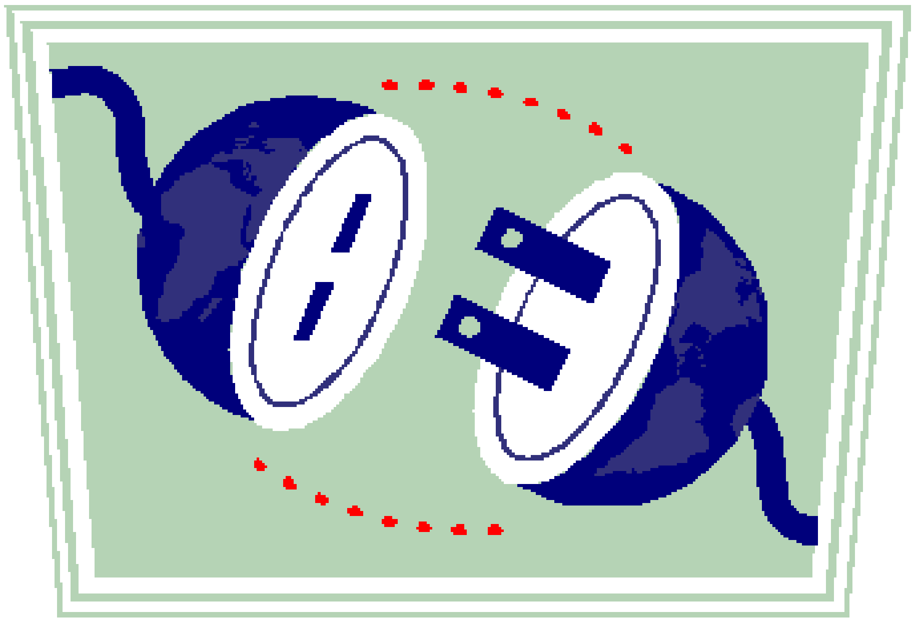
Exercise 9.1 Document a Given Deployment Design

Given the deployment decisions below, build a deployment architecture with the diagrams and text descriptions.

Instructor Example - Hotel Austerity Limited (HAL)	**Lab – LotsALoans Corp (LLC)**
HAL is a **hotel chain** with 2 hotels in Chicago (Chic1 and Chic2), and 1 in New York (Man1). The IT headquarters is in Denver. Each hotel has three machines for reservations and one back office server. Each reservation machine is a Macintosh G4 and the office server is an Athalon Linux server with a RAID array. The IT headquarters has a round-robin load balancing router, five transaction-processing machines, and one DB2 database. The TP machines are rack mounted Athalon Linux servers and the DB2 database runs on a mainframe. Each hotel has a 100mb LAN and a T1 connection to IT headquarters. There is also a backup dial-up connection. The hotel machines run OS X and the Opera web browser for reservations. The back office server runs the client version of a custom J2EE application, and the IT headquarters runs the server version of the same application. What happens when a reservation machine crashes? When the T1 link to the IT headquarters is down? When a transaction-processing machine fails?	Construct a plausible deployment design for LLC. Be sure to cover all of the steps in the deployment design process. Identify scenarios for failure, overload, and security violation cases. Evaluate your deployment with respect to these scenarios. Does your deployment assume anything not specified about the connectors? For example, guaranteed delivery of messages after outage, in-order delivery of messages, timely delivery.

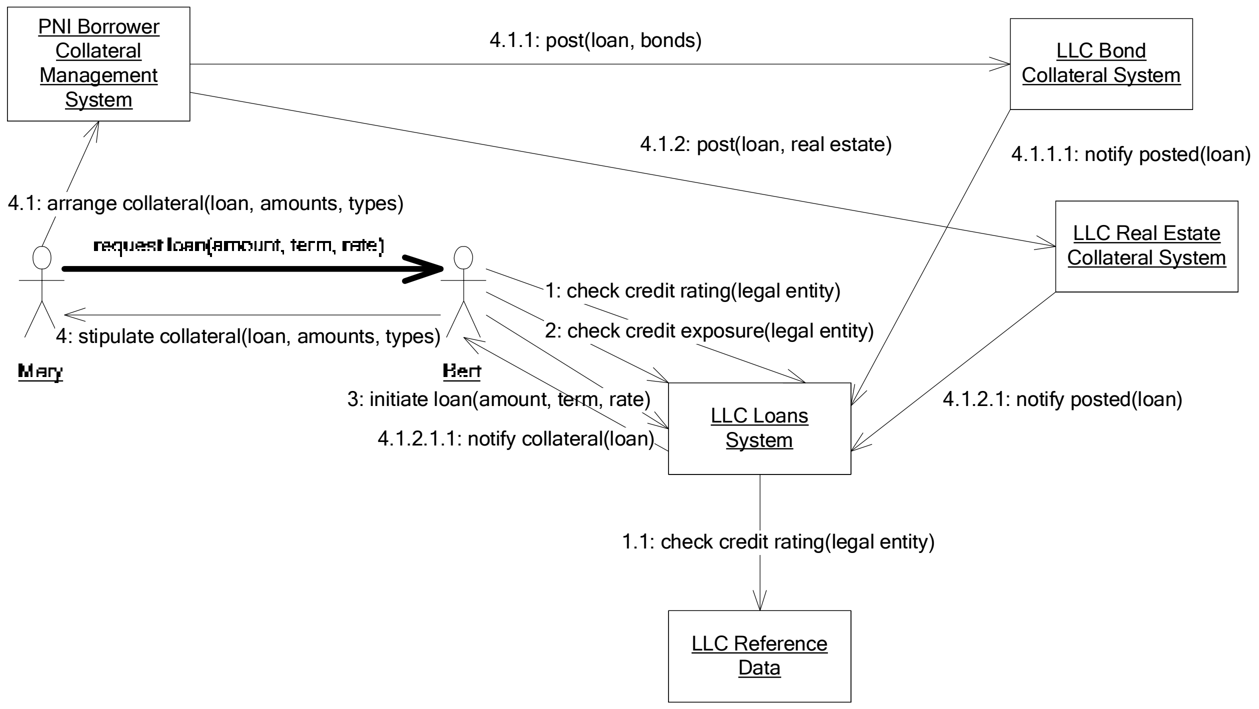
MAP

Possible Solutions to Exercises



1 Describe Objects

1.a Show Interacting Objects

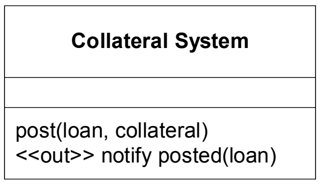

PNI Borrower Collateral Management System
4.1.1: post(loan, bonds)
LLC Bond Collateral System
4.1.2: post(loan, real estate)
4.1.1.1: notify posted(loan)
4.1: arrange collateral(loan, amounts, types)
LLC Real Estate Collateral System
request loan(amount, term, rate)
1: check credit rating(legal entity)
2: check credit exposure(legal entity)
4: stipulate collateral(loan, amounts, types)
Mary
Bert
3: initiate loan(amount, term, rate)
4.1.2.1: notify posted(loan)
LLC Loans System
4.1.2.1.1: notify collateral(loan)
1.1: check credit rating(legal entity)
LLC Reference Data

1 b. Kinds of Objects and Attributes Managed

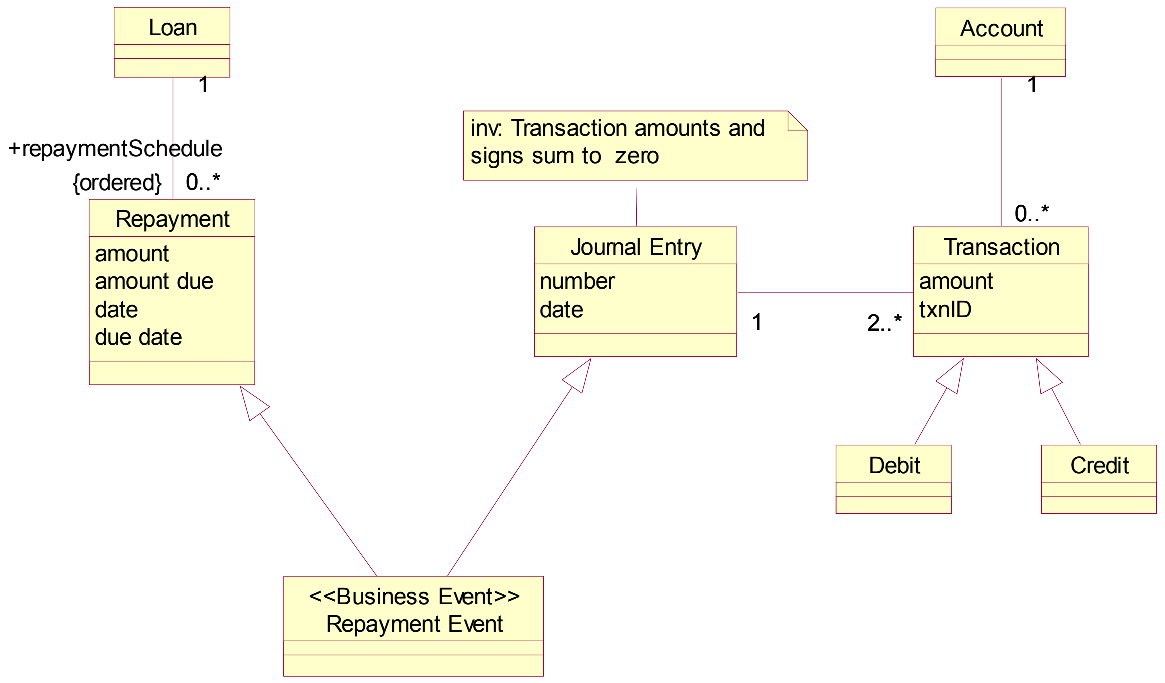
Loans System	loan portfolio (including total credit exposure), loans (including principal, rate, term), reference to collateral posted
Reference Data System	legal entity hierarchy (parents and subsidiaries), credit ratings;
Borrower Collateral Management System	obligations to provide collateral, and details of assets posted as collateral, together with references to the relevant loans
Bond Collateral System	details of bonds posted as collateral
Real Estate Collateral System	details of real estate posted as collateral

1 c. Polymorphic Type

Detailed handling of collateral types varies, but same operations are supported.

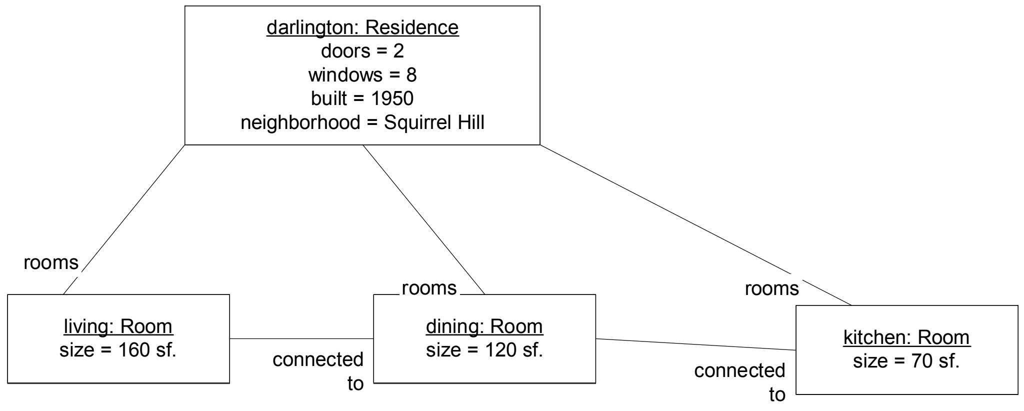


1 d) Find Multiple Types

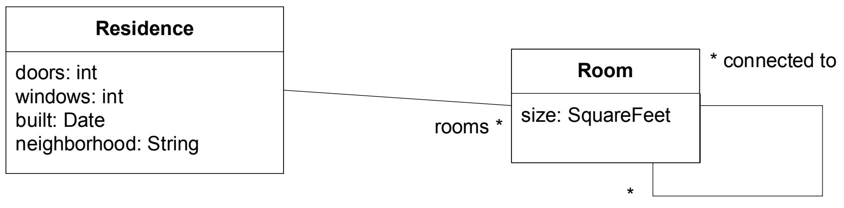


If a two or more aspects (types) of the behavior of a single underlying object are tracked by separate computer systems as though they are totally separate objects, serious inconsistencies typically arise. For example, in this case, the asset corresponding to the unpaid balance on a loan should be correctly reduced over time as repayments take place. There must therefore be communication between the loans system and the subledger system whereby the business event of repayment is converted into an appropriate journal entry and posted to the ledgers.

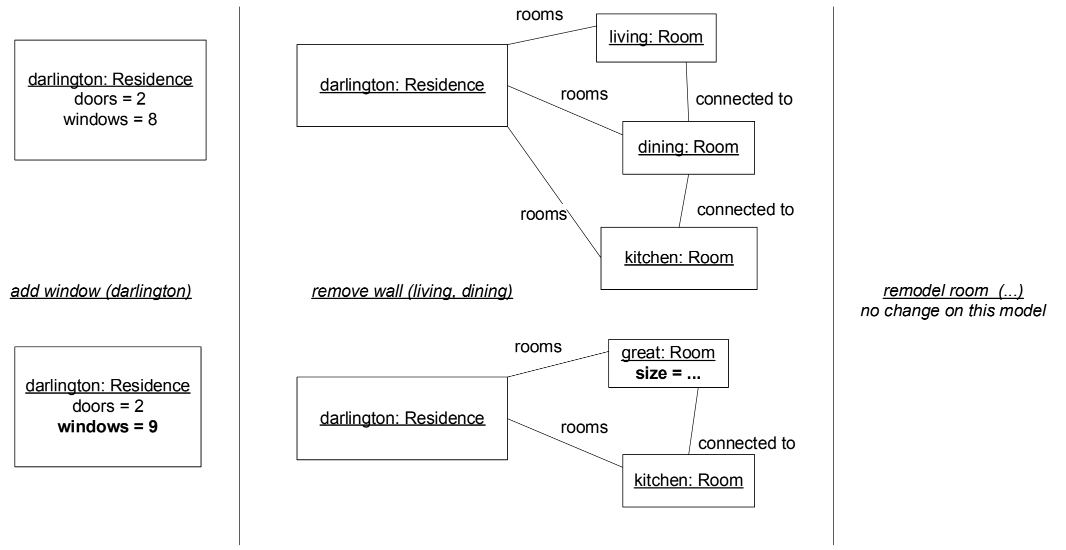
3.1 Objects, Snapshots



3.2 Information Model



3.3 Snapshot Pairs



The fact that remodel_room has no visible effect on our model of the residence means either:

1. That action and its effect is not relevant to our purpose, or
2. Our model needs to be extended to allow a description of what (relevant) changes take place when a room is re-modeled.

3.4 Actions

action	add Window (r : Residence)
description:	the house is remodeled to add a new window to a residence
precondition:	none
postcondition:	the number of windows in the residence r is one greater than before

action	remove Wall (r1: Room, r2: Room)
description:	the house is remodeled to join two contiguous rooms into one
precondition:	the two rooms r1 and r2 are connected to each other
postcondition:	A new room, r3, is created and added to house rooms.
	r3 is connected to all the rooms that r1 and r2 were connected to.
	the size of r3 is the sum of the sizes of r1 and r2.
	the rooms r1, r2 are gone.

3.5 Scenarios with text and snapshots

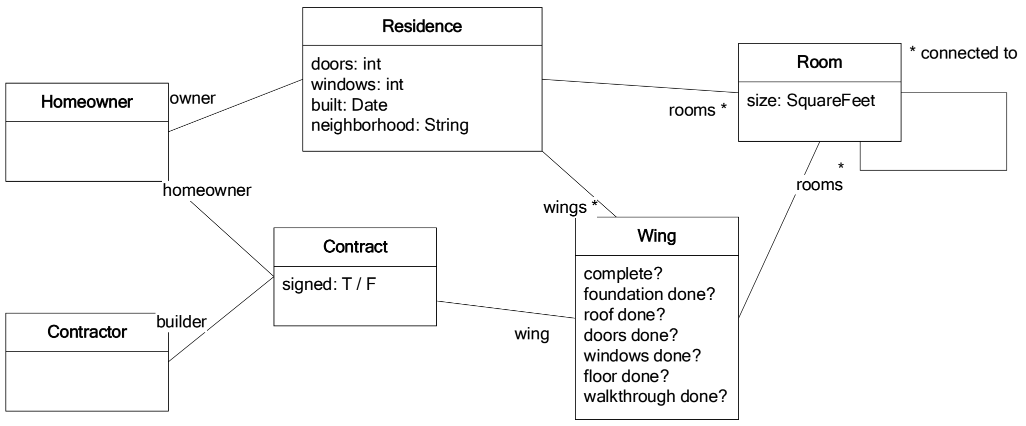
Scenario name: new wing on mansion

Initial state: Titus is a contractor, Jake the homeowner

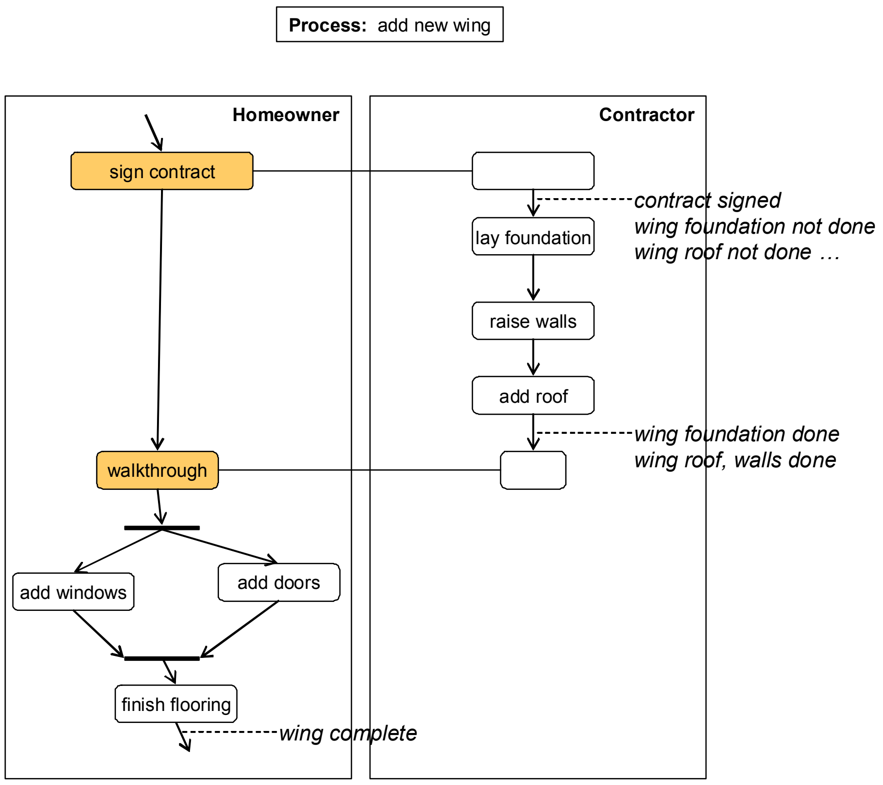
1. titus and jake sign the contract
2. titus lays the foundation
3. titus raises the walls
4. titus adds the roof
5. titus and jake do the walkthrough
6. jake adds the windows
7. jake adds the doors
8. jake adds the finished flooring

Final state: the new wing is complete

Snapshots: This would probably require new types of objects for Contractor, Homeowner, Wing, Contract, and possibly Foundation, Roof, Wall (although these could be modeled with simple T/F attributes on the wing to indicate if the foundation, roof, walls etc. were completed or not, assuming there was no need to capture deeper details of the domain).



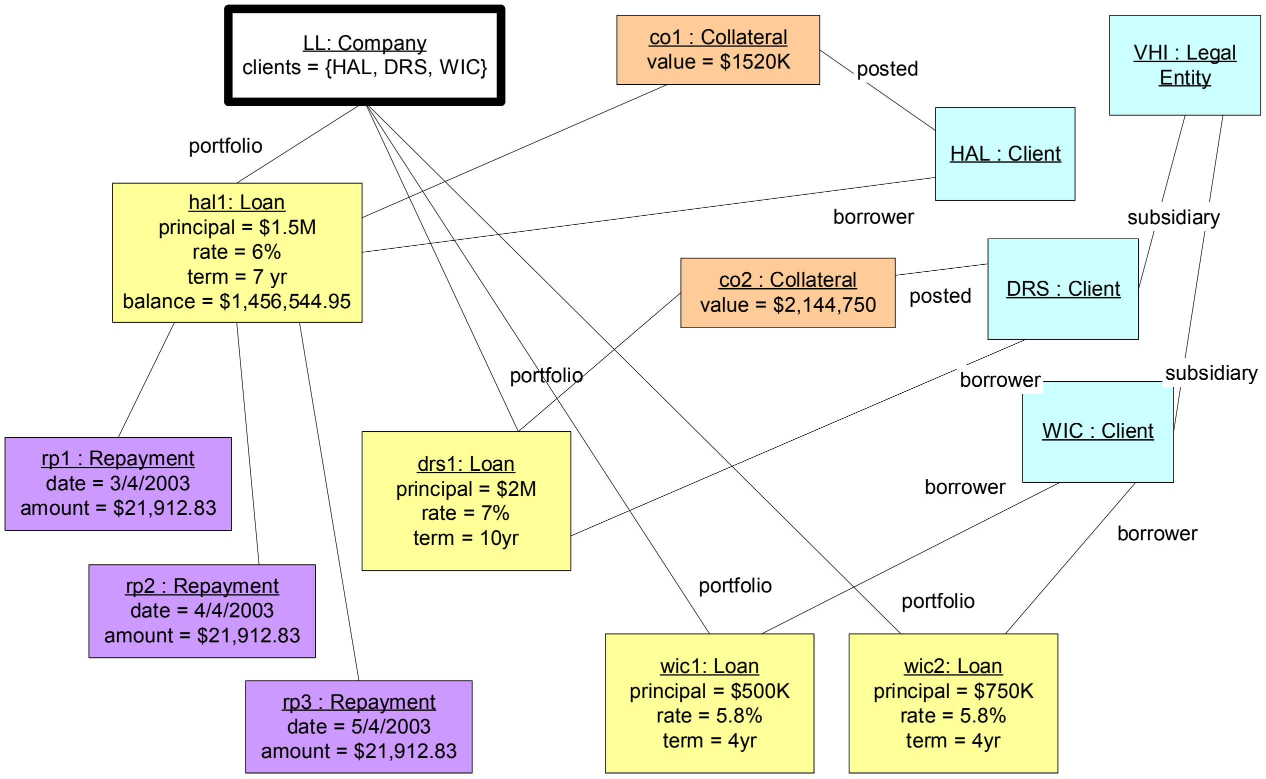
3.6 Role Activity Diagrams



4.1 Draw Object Snapshots, Build Information Model from Snapshots. Understand Navigation Expressions.

(a) Snapshots & User Interface

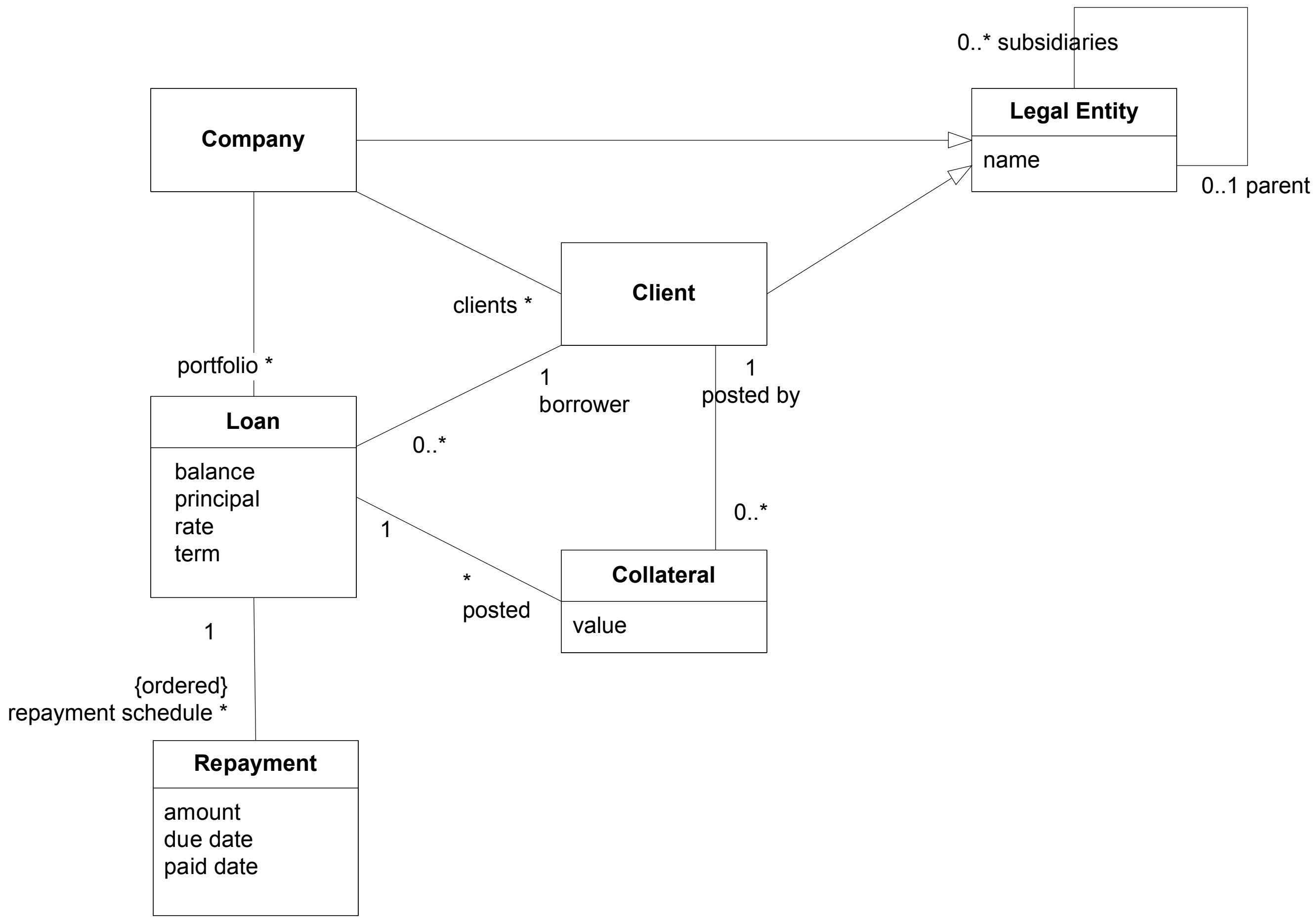
Note: information not shown in snapshot is simply omitted, rather than lost.



Creation:

Type of object	**Created when …**
Client	LLC receives the first ever loan application from that client.
Legal Entity	When any kind of legal entity becomes known to the company in any capacity.
Loan	When a new loan is requested between the company and a client.
Collateral	When collateral is posted against a loan.
Repayment	As soon as the repayment is scheduled.

(b) Build Information Model from Snapshots



(c) Interpret Navigation Expressions

1. LL.portfolio.borrower = {HAL, DRS, WIC}
2. LL.portfolio → select(principal > $750K).borrower = {HAL, DRS}
3. LL.portfolio → select(l | l.collateral->IsEmpty) = { wic1, wic2 }
4. hal1.repaymentSchedule → sum(amount) – (hal1.principal – hal1.balance) = $22,283.45
5. The borrowers who are subsidiaries of some parent LE = { DRS, WIC }
6. The clients who do not have any loans = {}

(d) Write Navigation Expressions

1. **Informal:** All repayments made to LL.

   Formal: LL.portfolio.repaymentSchedule

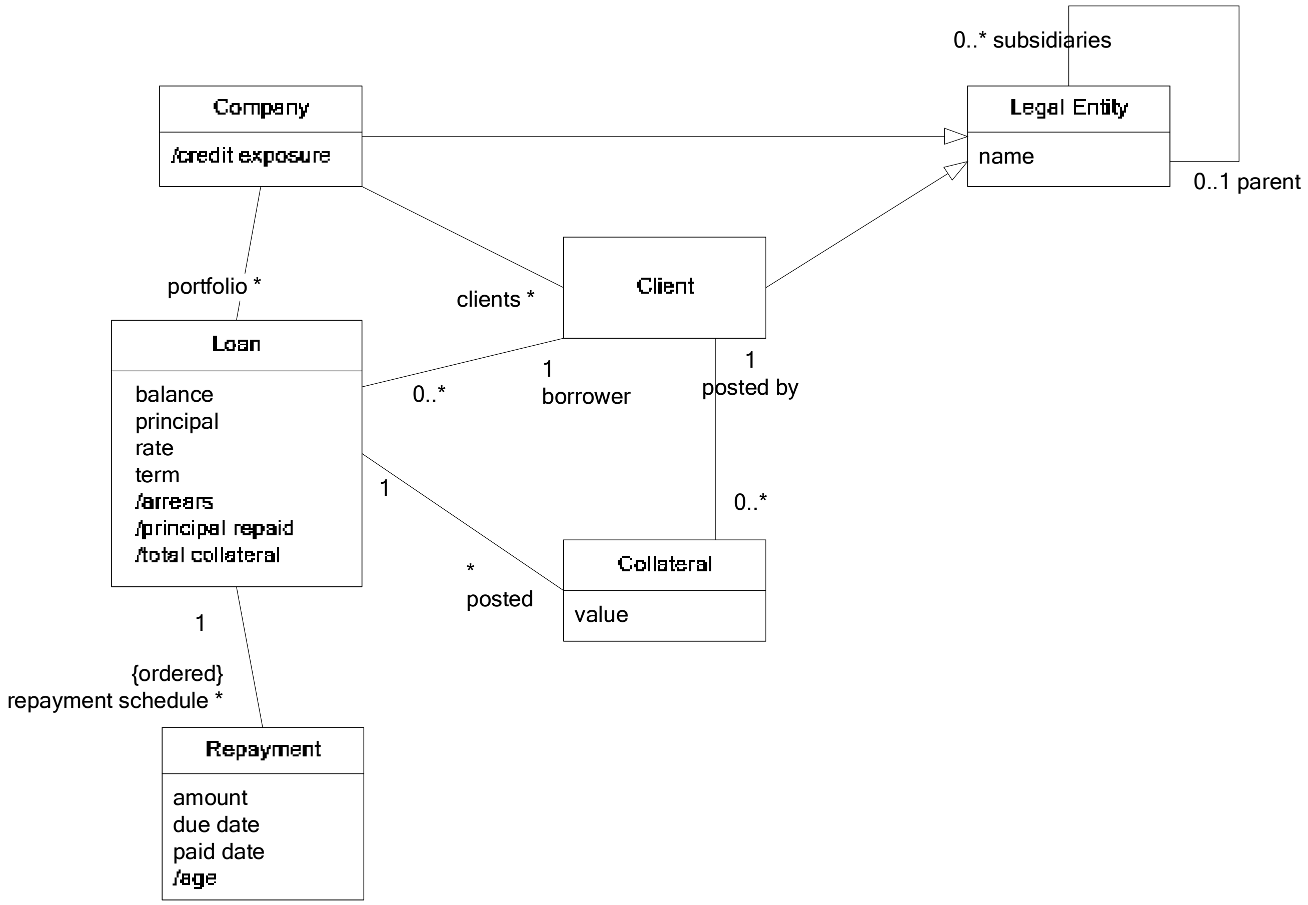
2. **Informal:** Legal Entities who are parents of borrowers.

   Formal: LL.portfolio.borrower.parent
3. **Informal:** The interest rates on loans in the portfolio

   Formal: LL.portfolio.rate

4.2 Convenience Attributes. Invariants. Supertypes.

a) Introduce convenience attributes



b) Write Invariants

i. Derivation Invariants for Convenience Attributes

1. The total value of the collateral posted for a Loan: sum of the value of all the loan's posted collateral
   Loan: totalCollateral = posted → sum(value)

2. The total credit exposure for a Company, across its entire portfolio of Loans: the sum of the balance on all loans in the portfolio.

   Company: creditExposure = portfolio → sum(balance)

3. Assuming a "due date" attribute on Repayment, the number of days by which a Repayment on a Loan is overdue at the present time: age = current date – due date (roughly)

   Repayment:
   if (paid_date = null & now > due date)
   then age = now – due date
   else age = 0

4. The Repayment amount outstanding on a Loan, across all overdue Repayments for that Loan. Assume the existence of an "amount" attribute showing the amount due on a Repayment.

   Loan: arrears = repayment schedule → select(age > 0) → sum(amount)

5. The amount of principal repaid for a Loan.

   Loan: principal repaid = principal – balance

ii. Extra Model Constraint

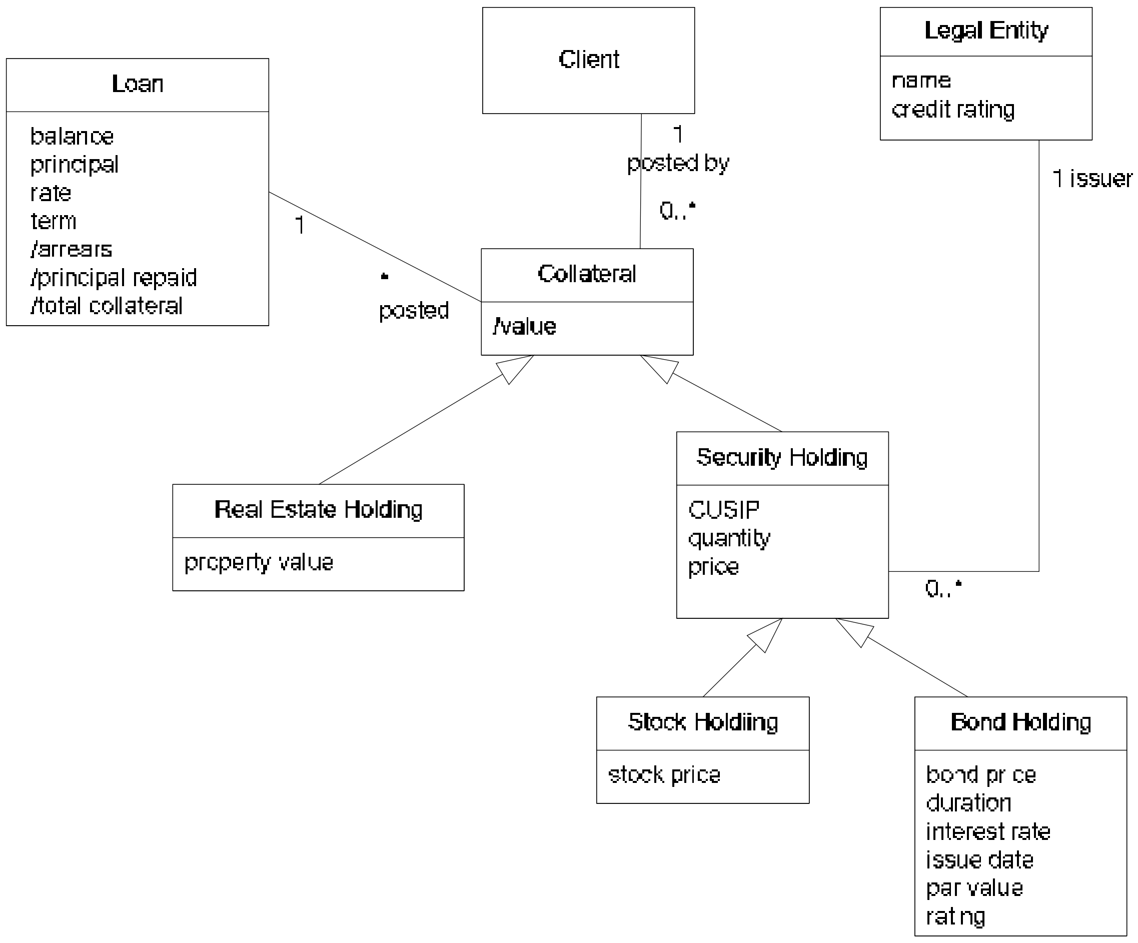
Informal: A Borrower from a **Company** is necessarily a **Client** of that **Company**.

Formal: Company inv : self.clients → includesAll(self.portfolio.borrower)

Equivalent set expression: self.portfolio.borrower   self.clients.

Note: this invariant restricts the cyclic associations between **Company**, **Client** and **Loan**.

c) Utilize Supertypes



Invariant on **Bond Holding**: rating is the same as the issuer's credit rating

There are different types of collateral: **Stock Holding**, **Bond Holding** and **Real Estate Holding**.

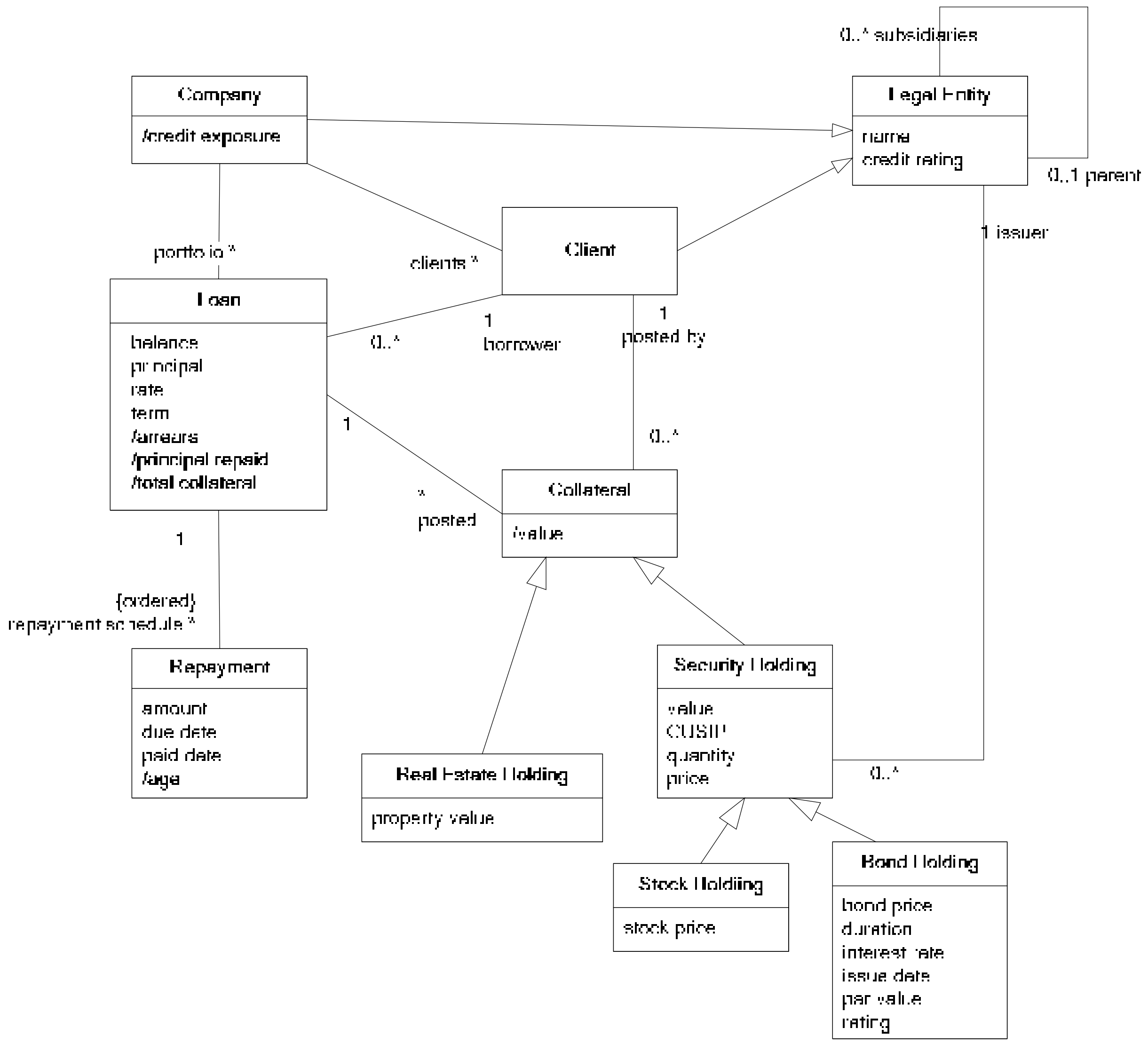
The supertype is **Collateral**.

In addition, **Stock Holding** and **Bond Holding** are both types of **Security Holding**.

The generic attribute Collateral::value puts a USD value on a collateral holding. Each subtype has its own valuation attribute: BondHolding::bondPrice; StockHolding::stockPrice and RealEstateHolding::propertyValue, whose value is determined using a procedure (whether manual, semi-automatic or full automatic) specific to that type of holding. For SecurityHoldings, the Collateral::value requires further computation to multiply the respective price attribute by the SecurityHolding::quantity attribute.

Resulting Information Model

/



4.3 Define Scenarios and Snapshot Pairs.

(a) Scenarios and Snapshot Pairs

1. the long-lived activity could be: **complete loan**

   starts – when HAL requests a loan from LLC.

   ends – when HAL has completed all the scheduled repayments on the loan.

2. the finer grained actions could be: request loan, agree amount, offer term, agree term, offer rate, agree rate, post collateral, activate loan, draw down, make repayment.

   the type could be: Loan.

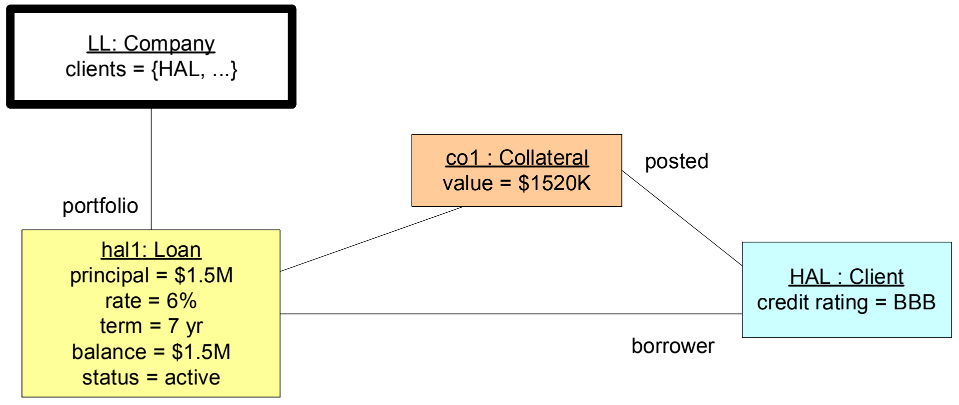
   the states could be: proposed, active, closed.

3. scenario: HAL borrows from LLC and then pays off the loan

   initial state: HAL is not a subsidiary of any other legal entity. It has a credit rating of BBB.

   scenario steps

   1. HAL requests a loan for $1.5M over 8 years from LLC.
   2. LLC offers a rate of 6%, and counter-proposes a term of 7 years.
   3. HAL agrees to the amount, rate, and term.
   4. LLC stipulates to HAL that collateral must always be greater than or equal to the balance outstanding on the loan at any given time.
   5. HAL agrees and posts initial collateral whose value is $1520K.
   6. LLC schedules all 84 repayments and activates the loan.
   7. HAL draws down the full loan amount.



   8. HAL makes the first monthly repayment of $21, 912.83 on 3/4/2003.
   9. HAL makes the second monthly repayment of $21, 912.83 on 4/4/2003.

10. HAL makes the third monthly repayment of $21, 912.83 on 5/4/2003…

91. HAL makes the 84th monthly repayment of $21, 912.83 on 2/4/2009.

final state: HAL's loan with LLC has a balance of $0, and is closed.

4. after step 3 – hal1 has a principal of $1.5M, a rate of 6% and a term of 7 years, the balance is $0 and status is "proposed".
   after step 7 – collateral of $1520K has a new link to hal1 and a new link to HAL, the balance equals the principal at $1.5M, and status is "active".
   after step 91 – the full set of 84 repayments is in place and linked to hal1; the balance is reduced to $0 and status is "closed".

5. possible exceptions
   HAL's credit rating is downgraded, resulting in a requirement for more collateral and/or adjustments to other loan parameters; e.g., interest rate, term.
   HAL fails to make timely repayments, loan is foreclosed, and collateral is seized by LLC
   Collateral asset value falls, resulting in need for posting of further collateral
   HAL files for bankruptcy, increasing the risk that the loan will become a bad debt.

(b) Specify an Action

drawdown (ln: Loan)

```
    precondition:
        -- the loan has been designated as active by the lender
        -- and the repayment scheduled has been established
        ln.active and ln.repaymentSchedule → NotEmpty
    postcondition:
        -- the balance on the loan equals the principal (which remains unchanged)
        ln.balance = ln.principal and ln.principal = ln.principal@pre
```

scheduleRepayments (ln: Loan, firstRepaymentDate : Date)

```
    precondition:
        -- Principal, rate and term have been agreed
        ln.principal <> 0 and ln.rate <> 0 and ln.term <> 0
    postcondition:
        -- A set of monthly scheduled repayments amortizing the principal over the
        -- term at the agreed rate is created with due-dates, and associated with the loan.
        -- (Assume existence of necessary amortization and date functions.)
        let amountDue be amortizedRepayment(ln.principal, ln.rate, ln.term) and
            numRepayments be 12 * ln.term in
            ln.repaymentSchedule→size == numRepayments and
            ln.repaymentSchedule→forall(r | r.amountDue = amountDue) and
                ln.repaymentSchedule→dueDate =
                    dateSequence(firstRepaymentDate, Month,
                        numRepayments)
```

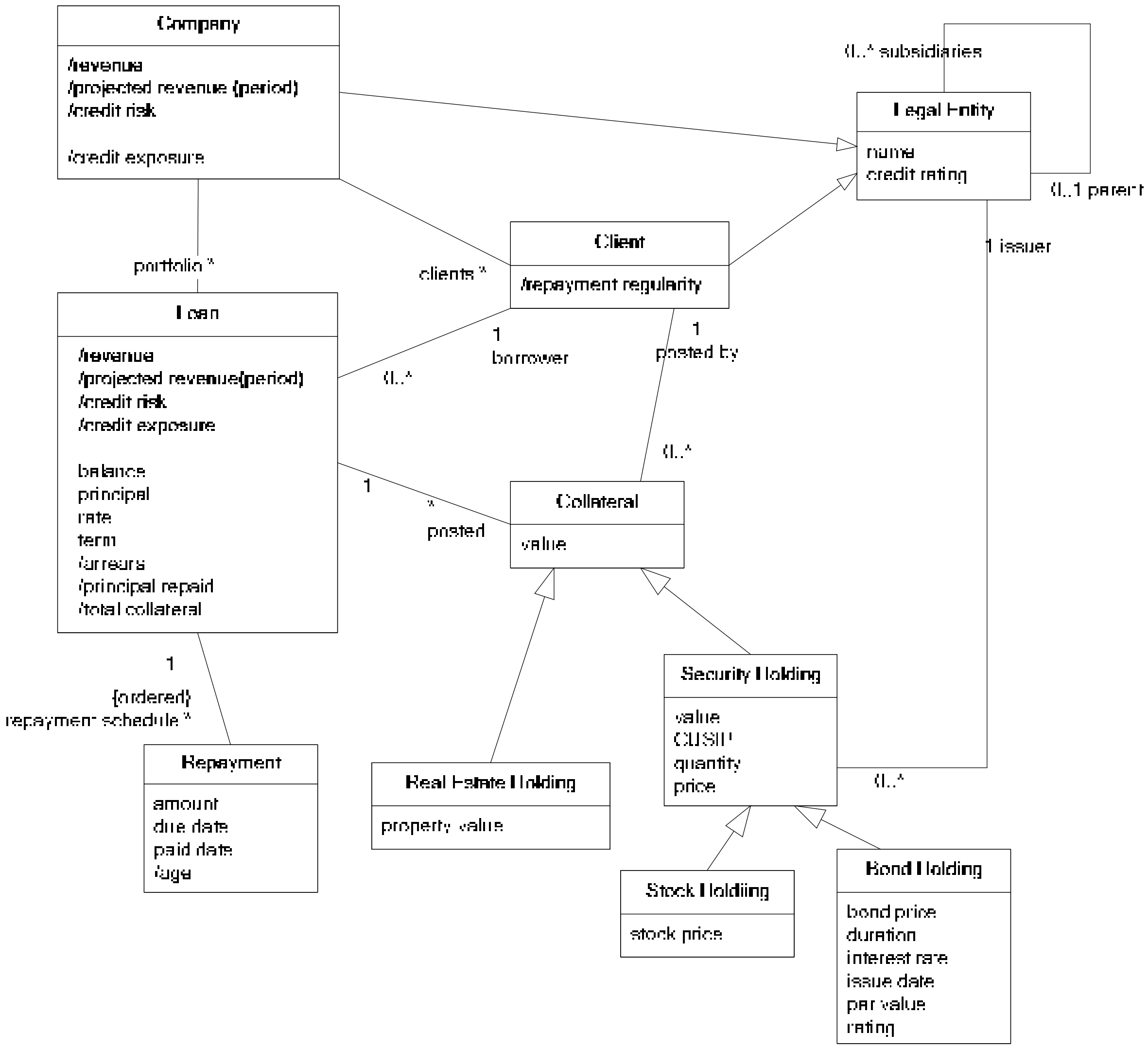
makeRepayment (r: Repayment)

```
    precondition: Loan has a balance
    postcondition:
        -- The repayment amount and date are filled in and the balance on the loan
        -- is reduced by the amount determined by the amortization schedule.
```

```
-- Assume existence of a "principalRepaid" convenience attribute on
-- Repayment.
r.amount = r.amountDue and r.date = now and
        r.loan.balance -= r.principalRepaid
-- Note that this post condition could have been simplified by defining an invariant
-- forcing the balance to be in line with the principal repaid by all repayments
```

5.1 Define Business Goals and Architecture

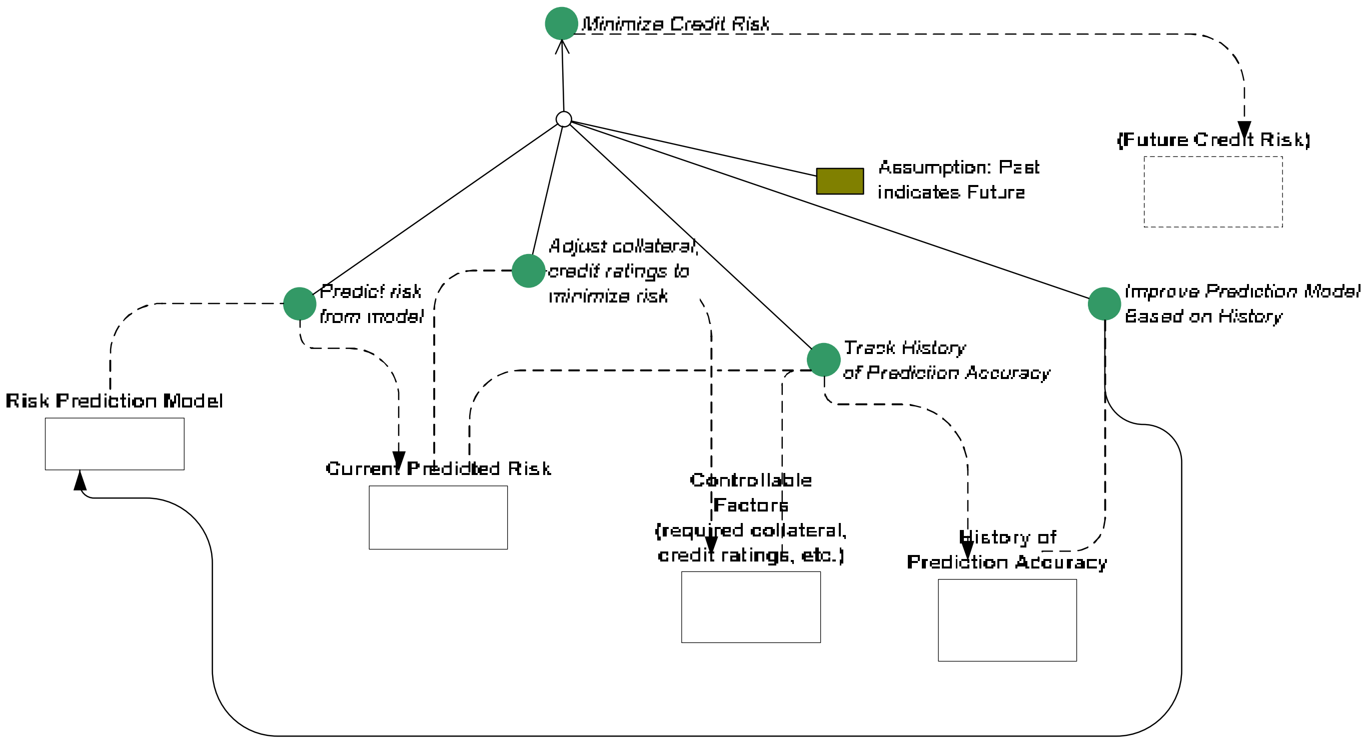
1. These concepts are mentioned in the goals/requirements, and so must be formalized. This can be done in the business information model that accompanies the goal model.
   - o The interest paid per loan to date can be formalized as a convenience attribute on Loan, by summing over the interest repaid on all the repayments made so far.
   - o Projected interest can be computed for a given future time period for a loan by summing over the interest element of the relevant scheduled repayments, perhaps present valued at some rate.
   - o Credit risk could be a (complex, future-valued) attribute on both Loan and Company.
   - o Repayment regularity could be an attribute on Client, determined by it's repayment history.



2. The top level goal may be stated in one of the following similar ways:

- Minimize credit risk while maximizing projected revenue;
- Maximize projected revenue/credit risk ratio (or some other function of revenue,risk)
- Minimize the credit risk/projected revenue ratio.

3. If we interpret "credit risk" as being specifically about the future (as opposed to being about the past, or even about our best guess about that future), then it cannot be objectively measured. While the credit exposure itself can be defined and objectively measured, the minima that might be achieved within any given future time frame cannot be predicted with certainty. Consequently, "minimize" and "maximize" goals are often called "soft goals". Despite their "softness", they have an important place in goal modeling. A reasonable way to refine into sub-goals would be:
   i. Predict Credit Risk Based On Risk Prediction Model
   ii. Modify Controllable Credit Risk Parameters (collateral demanded, credit ratings, black-listed clients, etc.) To Limit Predicted Risk
   iii. Track Historical Performance of Risk Prediction Model
   iv. Improve Risk Prediction Model Over Time


Minimize Credit Risk
(Future Credit Risk)
Assumption: Past indicates Future
Adjust collateral, credit ratings to minimize risk
Predict risk from model
Improve Prediction Model Based on History
Track History of Prediction Accuracy
Risk Prediction Model
Current Predicted Risk
Controllable Factors (required collateral, credit ratings, etc.)
History of Prediction Accuracy

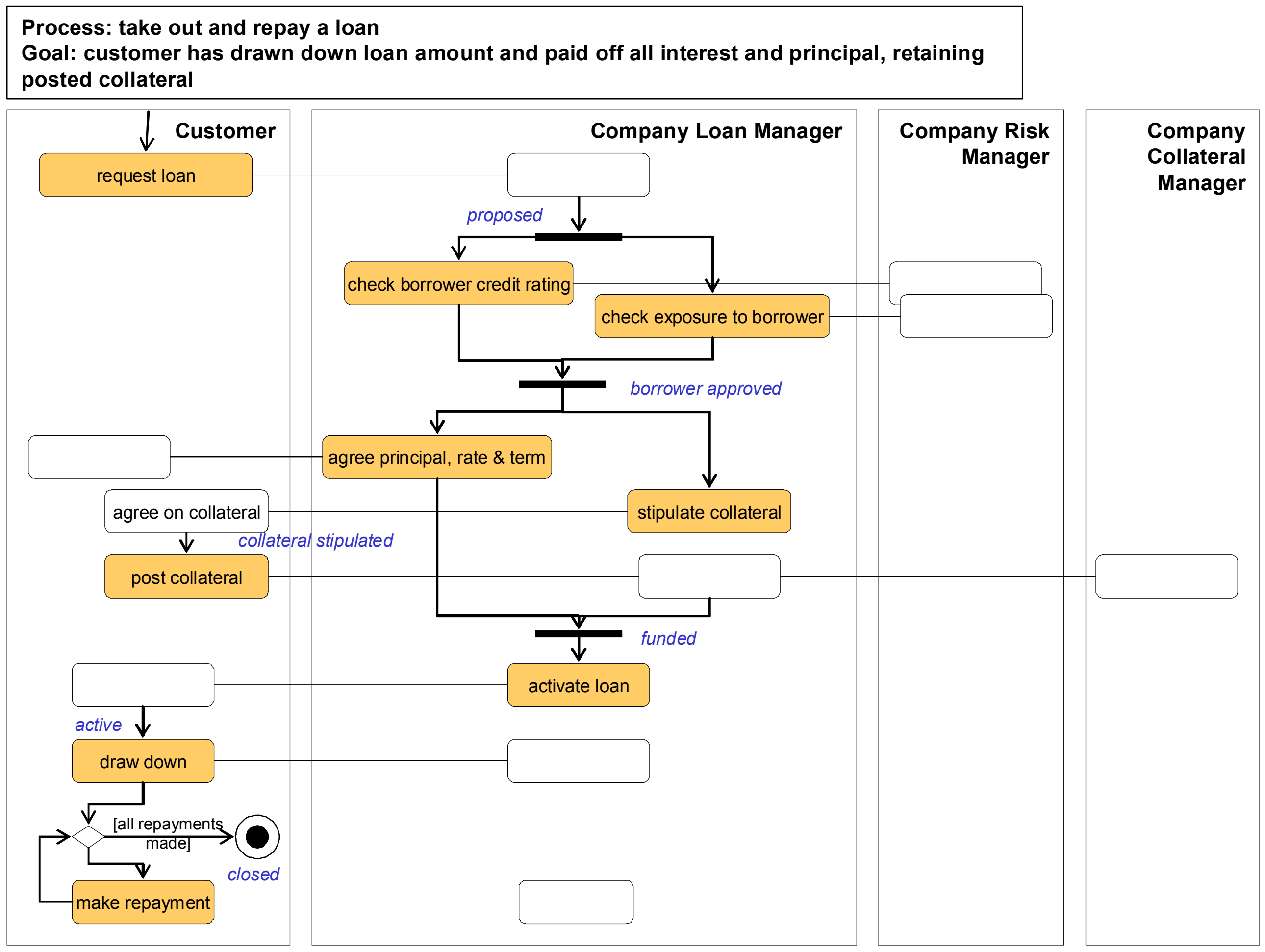
Events in the domain that we will have to react to in order to meet these subgoals "at all times": changes in credit ratings (including due to parent/subsidiary structure), collateral value change, etc.

4. The lending activities and attributes will be fully defined once the Business Architecture (processes, roles, activities, information model) has been completed. Sometimes further detail

is added in the Application Black Box Architecture (use cases and information model). Remember that models should be developed iteratively, and the activities that will be constrained from a risk perspective and the attributes cited in the constraints can be defined in a second pass over the Business and Application Black Box Architectures once both are somewhat stable.

5. The goal of optimizing lending activities is the higher level goal.
6. Answer #2 above offers a more precise statement of what might constitute optimization of lending activities: optimization of credit risk vs. projected revenue.
7. These could be reporting use cases to allow drilling up/down the loan, client and legal entity hierarchies looking at total credit exposure, revenue (actual and projected). Drill-down into collateral assets and values would also help. The use cases should support plugging in of different hypothetical values to drive results.
8. There is no magic bullet, and it may well be a true conflict. Making things clear makes it easier to recognize true conflicts, and to evaluate solutions to the conflict. The following process would yield a rational solution.
    - Find out if there really is a conflict. What does “notify immediately” mean in quantitative terms?
    - If more timely information truly is not available, assess the risk that would result from a less-than-immediate availability of information.
    - Fashion a compromise. For example, provide daily reports that include notice of client credit downgrades, but also arrange to trigger extra reports intra-day if a downgrade occurs more than a certain amount of time earlier than end of day.
    - Get explicit agreement from stakeholders to any such compromise, and document it with the original conflict within the goals model.
    - Provide traceability back from use cases and information model to the conflict within the goals model so the rationale for the compromise functionality can be reviewed and validated.

5.2 Role Activity Diagram


Process: take out and repay a loan
Goal: customer has drawn down loan amount and paid off all interest and principal, retaining posted collateral
Customer
Company Loan Manager
Company Risk Manager
Company Collateral Manager
request loan
proposed
check borrower credit rating
check exposure to borrower
borrower approved
agree principal, rate & term
agree on collateral
stipulate collateral
collateral stipulated
post collateral
funded
activate loan
active
draw down
[all repayments made]
closed
make repayment

Notes: the RAD above focuses on main flow, and makes some simplifying assumptions as follows.

- The full loan amount (principal) is always drawn down in a single operation.
- The case of late or absent repayments leading to foreclosure is not shown.
- Numerous other contingencies, such as variation in collateral value, loan interest rate changes, etc., have been omitted.
- This model does not deal with the lending of further amounts followed by further drawings and consequent adjustments in repayment schedule, except in the form of taking out an entirely new loan.

2. Activity Specifications

activity: post collateral

roles: Company, Client

inputs: loan: Loan, collateral: Collateral from Client outputs: none

precondition: loan is proposed, and collateral amount has been stipulated by lender

postcondition: stipulated collateral has been posted against the loan, loan is funded

activity: draw down

roles: Company, Client

inputs: loan from Client – *loan from which to draw down*

outputs: cash: USD for Client

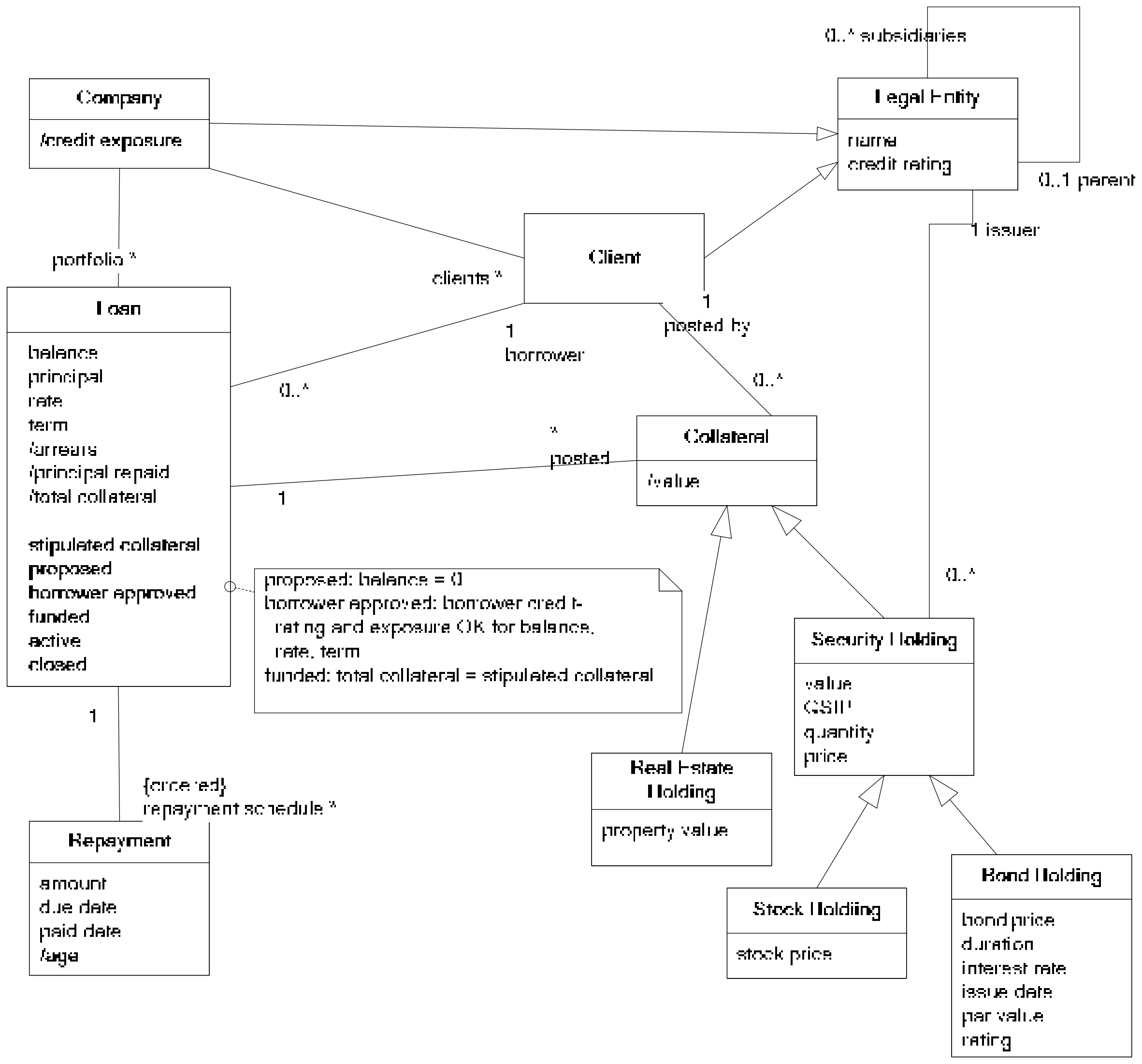
precondition: the loan is active.

postcondition: the balance equals the principal, which is unchanged from before

1. Information Model

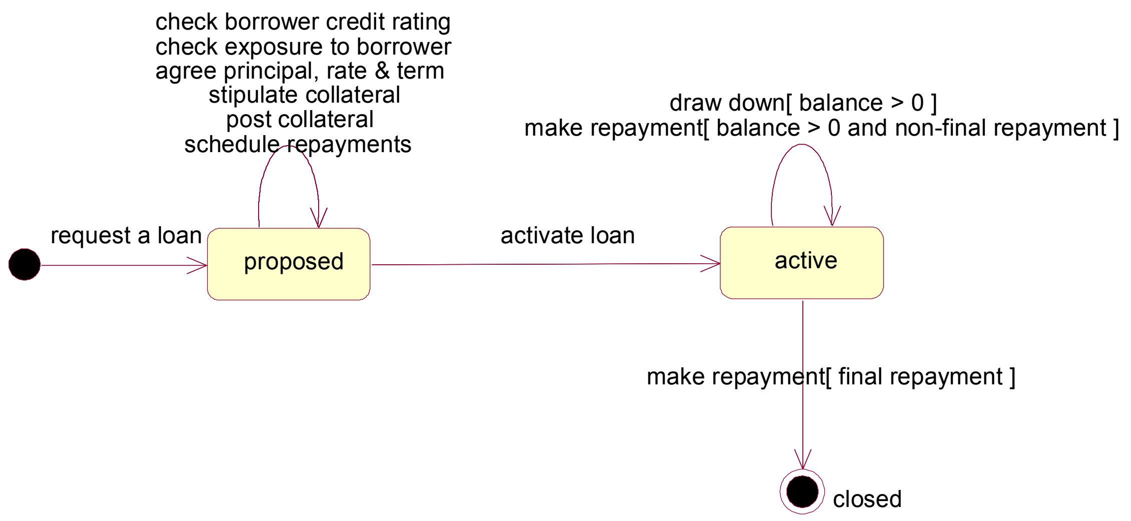
The types and attributes have business meaning and are not specific to a software system.

Client	*The party who borrows.*
Collateral	*Assets belonging to the Client, which can be transferred in whole or in part to the Company in the event that a Loan is foreclosed.*
Company	*The party who lends.*
Legal Entity	*Any legally incorporated enterprise.*
Loan	*The record of the funds advanced to the borrower and the history of repayment.*
Repayment	*One of the scheduled transfers of funds from Client to Company to reduce outstanding balance and interest*


0..* subsidiaries
Company
/credit exposure
Legal Entity
name
credit rating
0..1 parent
1 issuer
portfolio *
clients *
Client
1
posted by
1
borrower
Loan
balance
principal
rate
term
/arrears
/principal repaid
/total collateral
stipulated collateral
proposed
borrower approved
funded
active
closed
0..*
0..*
*
posted
1
Collateral
/value
proposed: balance = 0
borrower approved: borrower credit rating and exposure OK for balance, rate, term
funded: total collateral = stipulated collateral
0..*
Security Holding
value
CSIP
quantity
price
1
{ordered}
repayment schedule *
Repayment
amount
due date
paid date
/age
Real Estate Holding
property value
Stock Holding
stock price
Bond Holding
bond price
duration
interest rate
issue date
par value
rating

.

5.3 Build a State Diagram & Definition Matrix

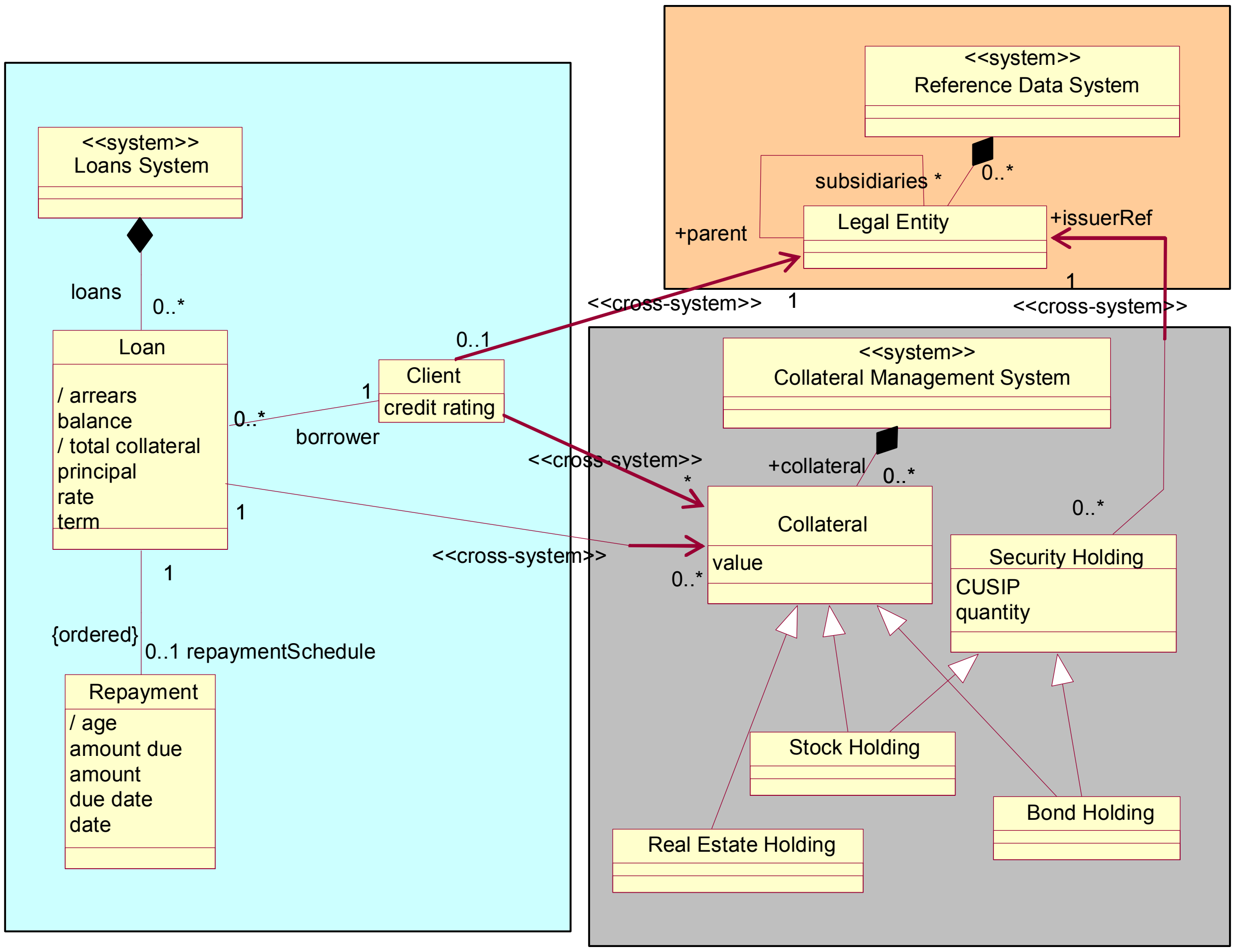

check borrower credit rating
check exposure to borrower
agree principal, rate & term
stipulate collateral
post collateral
schedule repayments
draw down[balance > 0]
make repayment[balance > 0 and non-final repayment]
request a loan
proposed
activate loan
active
make repayment[final repayment]
closed

Note1: Since there are many activities that take place within state “proposed” it might be worth considering some interesting sub-states of that state e.g. borrower approved, funded, etc. This solution does not name and define those finer-grained states, but they would be straightforward.

Note2: This front-loaded complexity is typical of loans; there is a lot more work to do before a loan is activated than there is once funds are drawn down and repayments begin.

State	Definition
proposed	balance = 0 (no draw down made yet) borrower approved: borrower rating & exposure OK for term, principal, rate funded: total collateral = stipulated collateral
active	balance > 0 (draw down made and loan activated)
closed	balance = 0 and one or more repayments made

6.1 Mapped Black Box Assembly



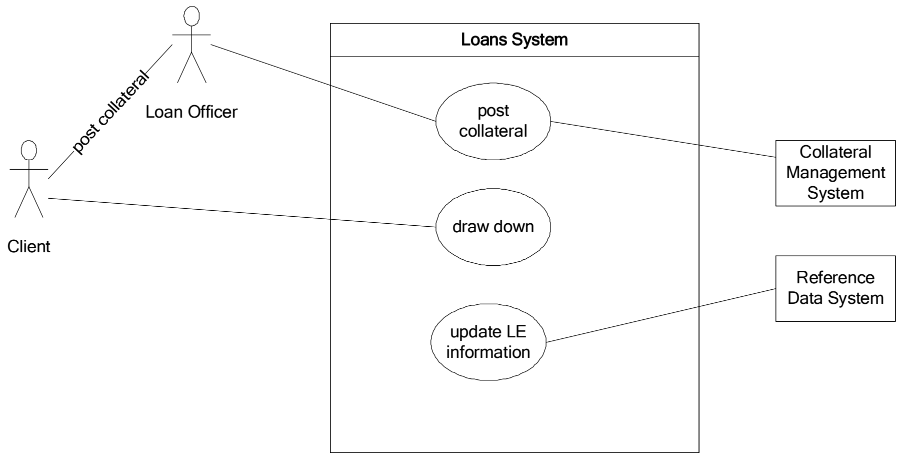
Notes:

- Some cross-system associations are necessary to relate different types, or differing views (projections) of the "same" types between systems, to ensure that they remain consistent.
- Examples given here are:
  - Loans and Reference Data occurrences of Legal Entity/Client to be kept in synch;
  - We might be able to dispense with LE and the parent-subsidiary relationship altogether in Loans and maintain only Client, synchronizing it directly with LE within Reference Data regularly; or maintain a cache for performance reasons.
  - Every Security Holding in Collateral Management to refer to an Issuer Legal Entity within Reference Data;

- Collateral occurrences within Loans and Collateral Management to be kept in synch. (An alternative would have been to treat Collateral as being the responsibility entirely of Collateral Management.)

6.2 Use Cases

1) use case diagram



2) use case specifications

Use case: post collateral

Actors: Loan Officer, Loans System, Client, Collateral Management System

Inputs: Loan ID, Client ID, list of assets (obtained from Client) to be posted as <u>collateral</u> for <u>Loan</u>

Outputs: Confirmation of posting to Client via Loan Officer, Collateral record to CMS

Trigger: <u>Client</u> contacts LLC offering assets as <u>collateral</u> against <u>loan</u>

Precondition: <u>Loan</u> is "<u>proposed</u>"; <u>Client</u> offering <u>collateral</u> is <u>borrower</u> for <u>Loan</u>; <u>collateral</u> fulfils criteria for suitability.

Postcondition:

If the collateral offered is sufficient, the offered collateral extends the set of holdings already in use as collateral against loan (in the Collateral Management System) and the loan is funded. If collateral is insufficient, there is no change to the system.

Use case: draw down

Actors: Client, Loans System

Inputs: <u>Loan</u>, specified by <u>Client</u>

Outputs: Notice of balance [to Client], funds transfer to [Client? Missing Actor?…]

Trigger: Client initiates automated draw-down process

Precondition: Loan is "active" and balance is zero

Postcondition: Loan balance equals principal, which remains unchanged, and funds transfer has been initiated

Use case: update LE info

Actors: Loans System, Reference Data System

Inputs: Changes in reference data structure from Reference Data System.

Outputs: None.

Trigger: Daily synchronization schedule.

Precondition: true

Postcondition: Systems are synchronized such that

- every instance of Legal Entity in the Loans System has a corresponding instance in the Reference Data System.
- every instance of a parent/child link in one system has a corresponding link in the other system.

6.3 Describe Steps and Alternate Paths

Use case: post collateral

Actors: Loan Officer, Loans System, Client, Collateral Management System

Inputs: Loan, Client, list of assets (obtained from Client) to be posted as collateral for Loan

Outputs: Confirmation of posting to Client, Collateral posting to CMS

Trigger: Client contacts LLC offering assets as collateral against loan

Precondition: Loan is "proposed"; Client offering collateral is borrower for Loan; collateral fulfils general criteria for suitability.

Postcondition:

If the collateral offered is sufficient, the offered collateral extends the set of holdings already in use as collateral against loan (in the Collateral Management System) and the loan is funded. If collateral is insufficient, there is no change to the system.

Main (success) steps:

1. Client sends collateral asset list to loan officer.
2. Officer looks up loan in system to validate:
   - i. That it exists and has been proposed;
   - ii. that client offering collateral is the client for the loan.
3. System remembers loan number for use as input parameter on posting.
4. For each item on the list of assets:
   - i. officer looks up value of asset in loan system;
   - ii. loan system obtains asset valuation from collateral management system;
   - iii. loan system adds value of asset to posting total.
5. Officer checks that posting total is not less than total valuation claimed by client.
6. Officer OKs posting of assets to the system.
7. Loans system informs Collateral Management System that collateral has been posted.
8. Loans System confirms posting.

Alternate paths:

1a: Collateral list fails general suitability criteria.

1a1. Officer contacts client and explains problem.

2a: Loan in incorrect state.

2a1. Officer attempts to diagnose how error arose.

2b: Client mismatched.

2a1. Officer attempts to diagnose how error arose.

4iia: Asset cannot immediately be valued (e.g., real estate, or system unavailable).

4iia1. Asset valuation request is put in queue in collateral management system.

4iia2. Use case suspended pending valuation.

4iia3. Collateral management system reports back when asset valued.

4iia4. Resume at 4iii.

5a: Collateral insufficient.

5a1. Officer saves entered data, and contacts client to explain problem.

5a2. Client offers further collateral.

5a3. Resume at 5.

6.4 Identify & Specify Use Case Operations

Use case: post collateral

Operations from steps

1. ….
2. Officer looks up loan in system to validate status and client

   lookup loan (loan_id, client_id): Loan Record…
   returns the loan record from the loan portfolio that matches loan_id, checking that the borrower has client_id
3. ….
4. For each item on the list of assets:
   i. officer looks up value of asset in loan system;

      asset_value (loan_id, asset_info): $
      returns the value of the described assets

   ii. <<out>> loan system obtains asset valuation from collateral management system;

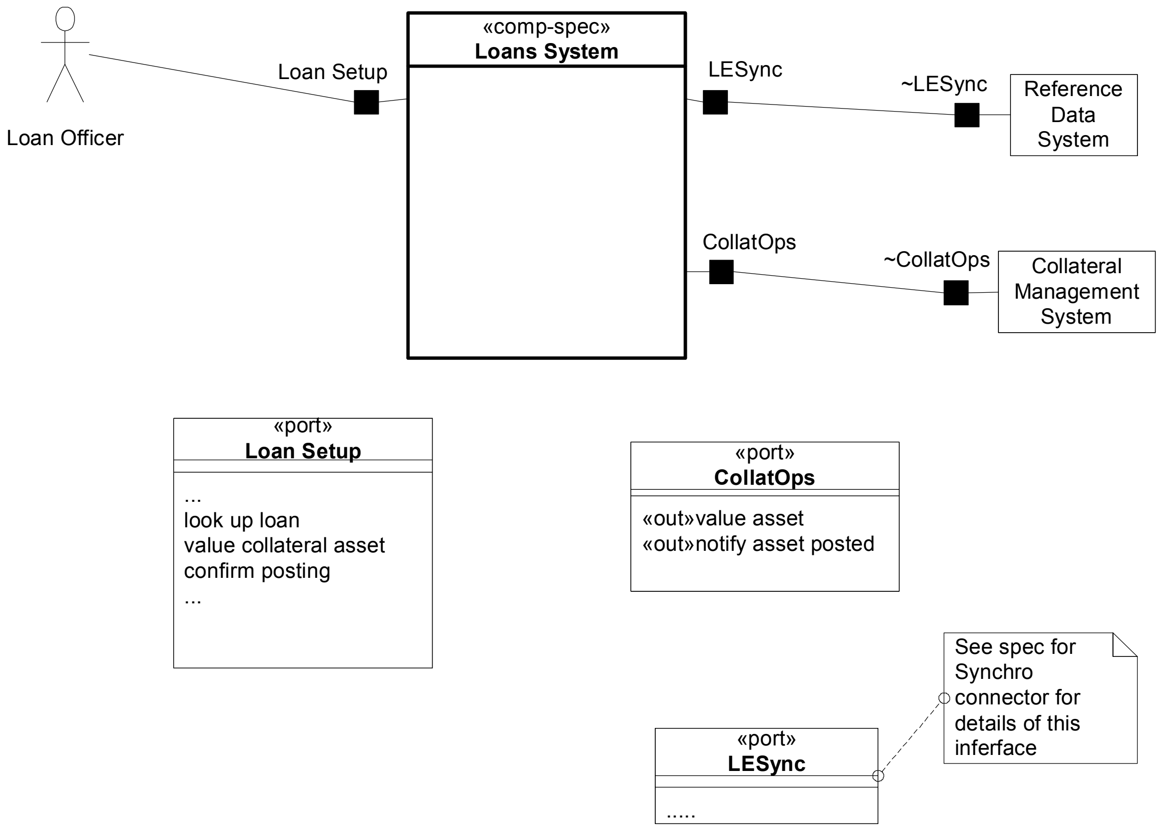
      <<out>> asset_value (asset_info): $

   iii. ….

      accumulate_asset_value (…)
      adds the asset value to the collateral for the loan
5. ….
6. Officer OKs posting of assets to the system.
7. <<out>> Loans system informs collateral management system that collateral has been posted.

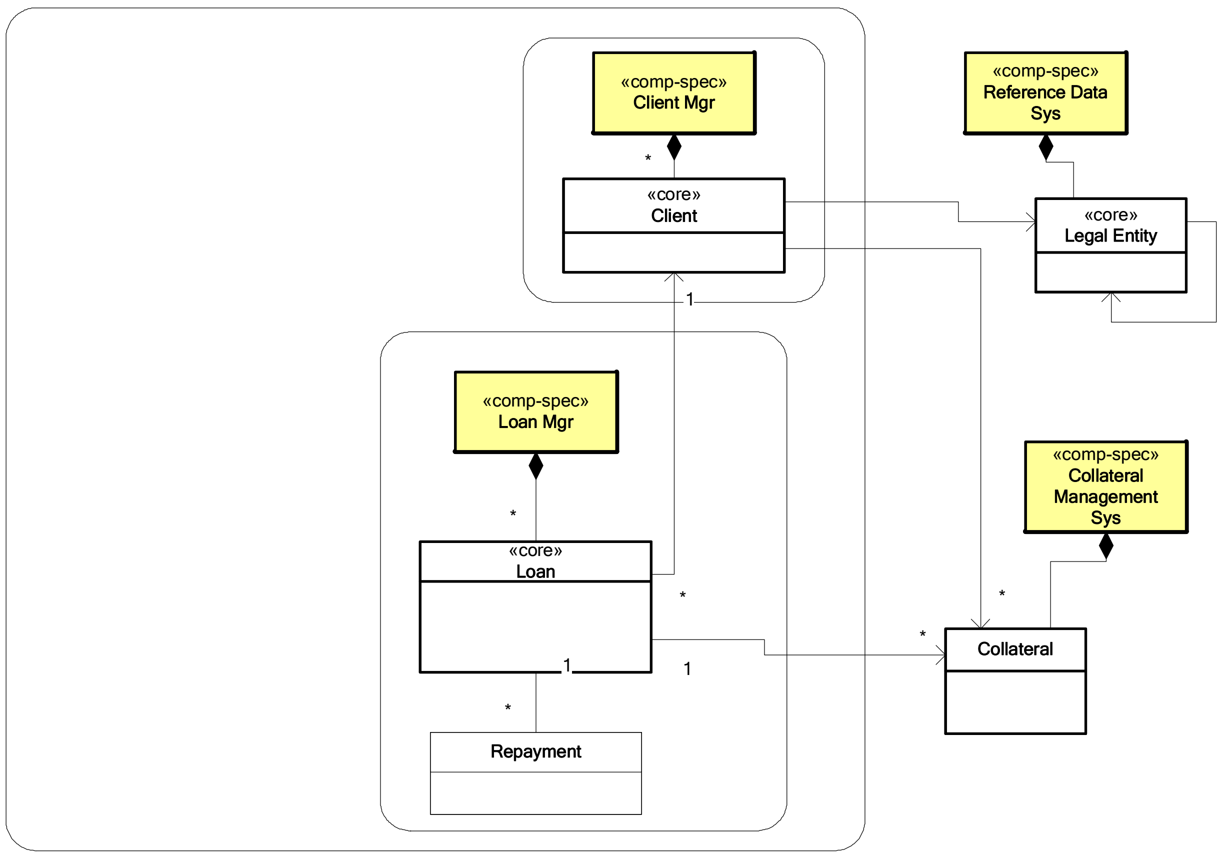
   <<event>> collateral_posted (loan_id)
8. ….

6.6 Black Box Ports and Assembly

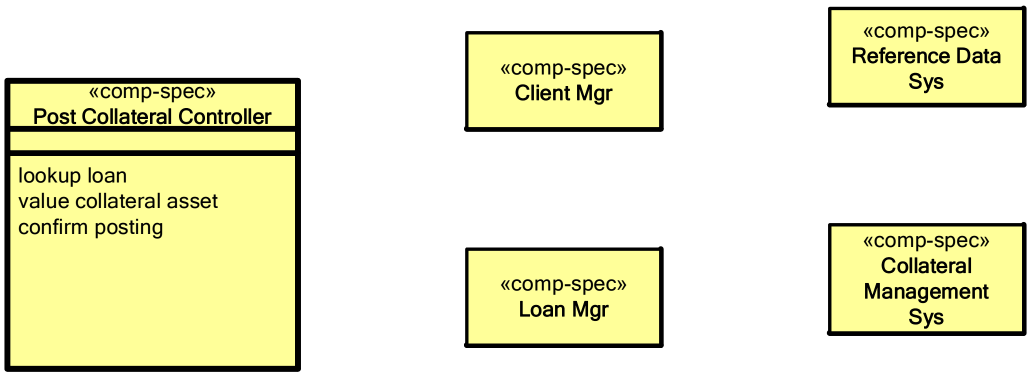


7.1 Draft Core Type & Use-Case Based Components

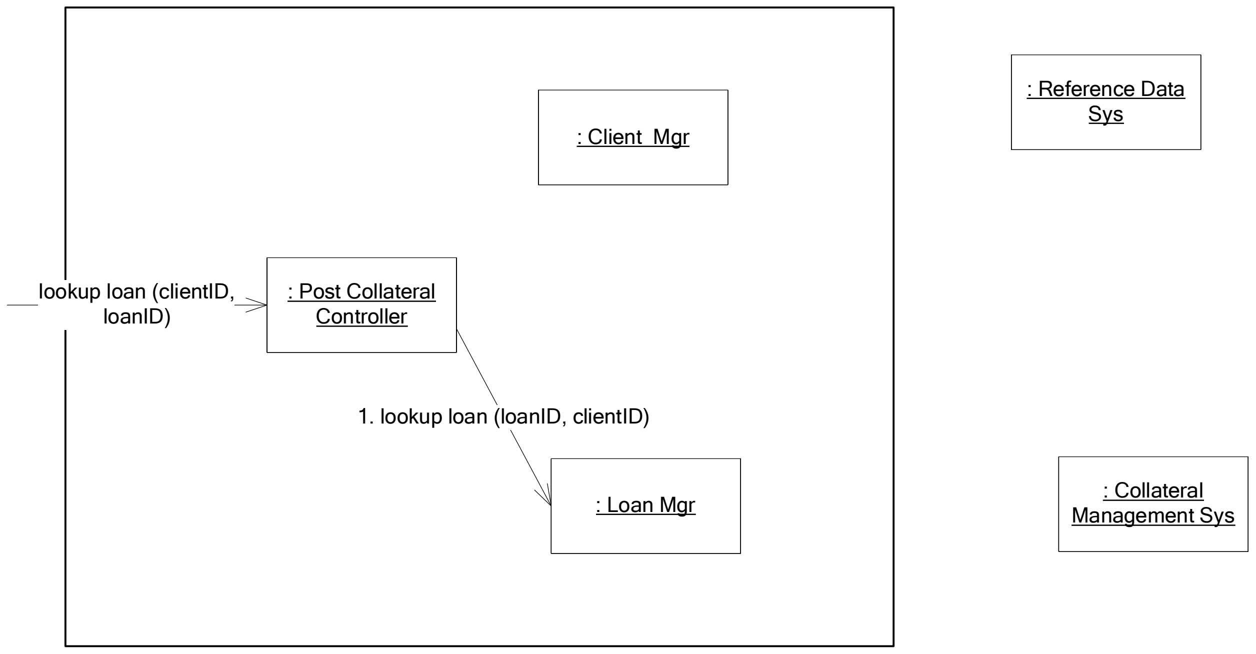
a) find core type components

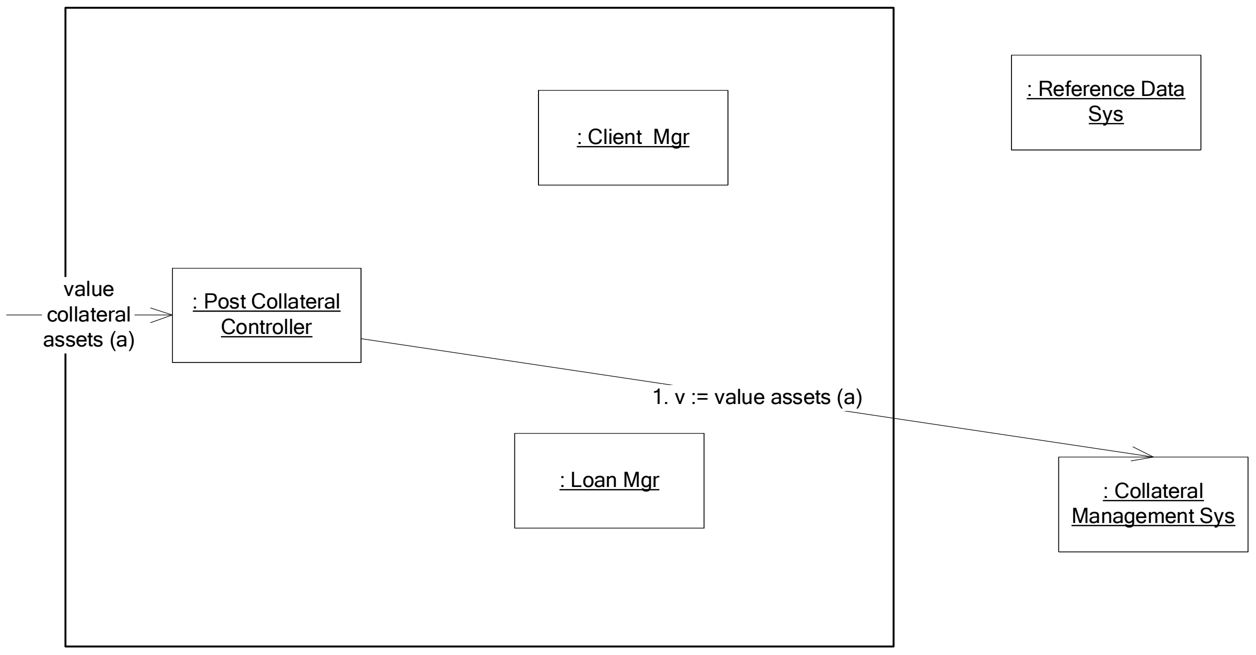


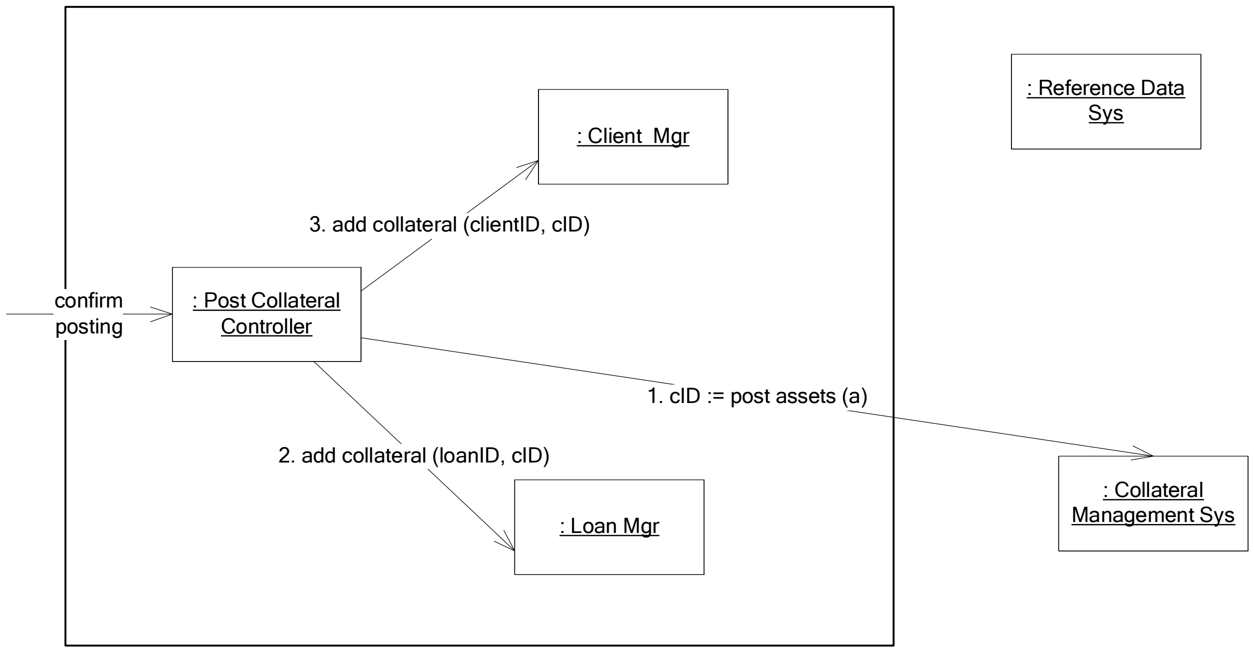
b) add use-case based components



7.2 Design Collaborations across Sub-Components


: Client Mgr
: Reference Data Sys
lookup loan (clientID, loanID)
: Post Collateral Controller
1. lookup loan (loanID, clientID)
: Loan Mgr
: Collateral Management Sys


: Client Mgr
: Reference Data Sys
value collateral assets (a)
: Post Collateral Controller
1. v := value assets (a)
: Loan Mgr
: Collateral Management Sys


: Reference Data Sys
: Client Mgr
3. add collateral (clientID, cID)
confirm posting
: Post Collateral Controller
1. cID := post assets (a)
2. add collateral (loanID, cID)
: Loan Mgr
: Collateral Management Sys

8.1 Apply a Pattern

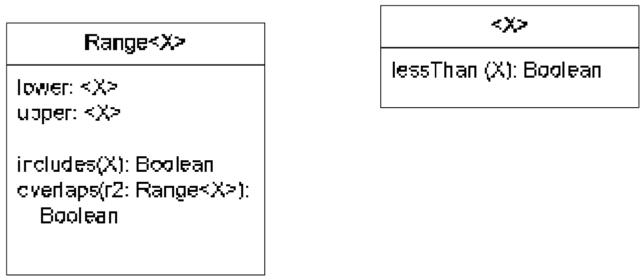
```
package Loans_Manager;

interface  payment_due_listener        {
  void notify_payment_due (T);
}

interface collateral_posted_listener               {
  void notify_collateral_posted (T2);
}

public class Loan_Events       {
   add_payment_due_listener ( payment_due_listener );
   remove_payment_due_listener (payment_due_listener);
    notify…
}
```

8.2 Extract a Pattern


Range<X>
lower: <X>
uper: <X>
includes(X): Boolean
overlaps(r2: Range<X>): Boolean
<X>
lessThan (X): Boolean