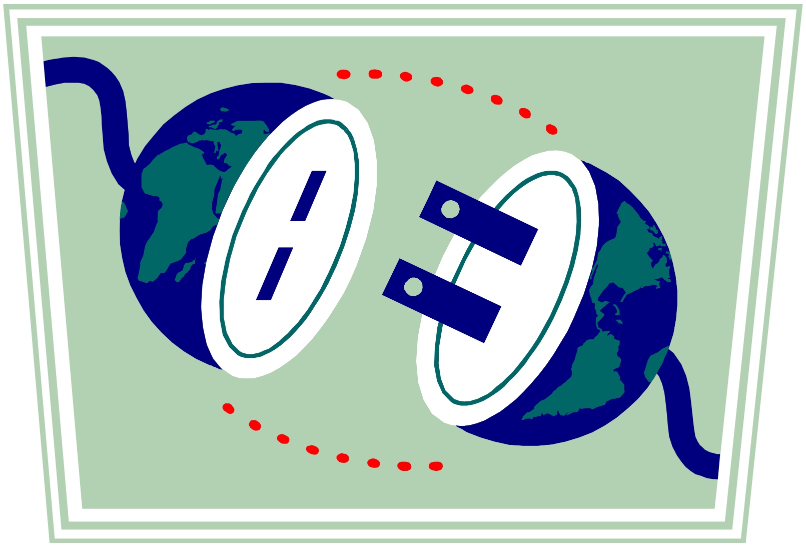
Modeling Goals, Processes, and Systems



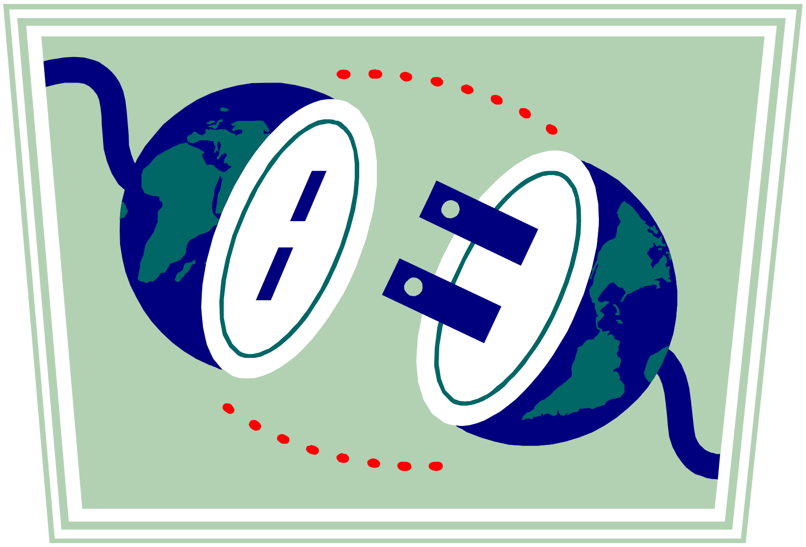
Kinetium Inc.

Version 1.0 Beta

(Back of cover sheet)

(1^{st} page after cover sheet)

Modeling Goals, Processes, and Systems



Kinetium Inc.

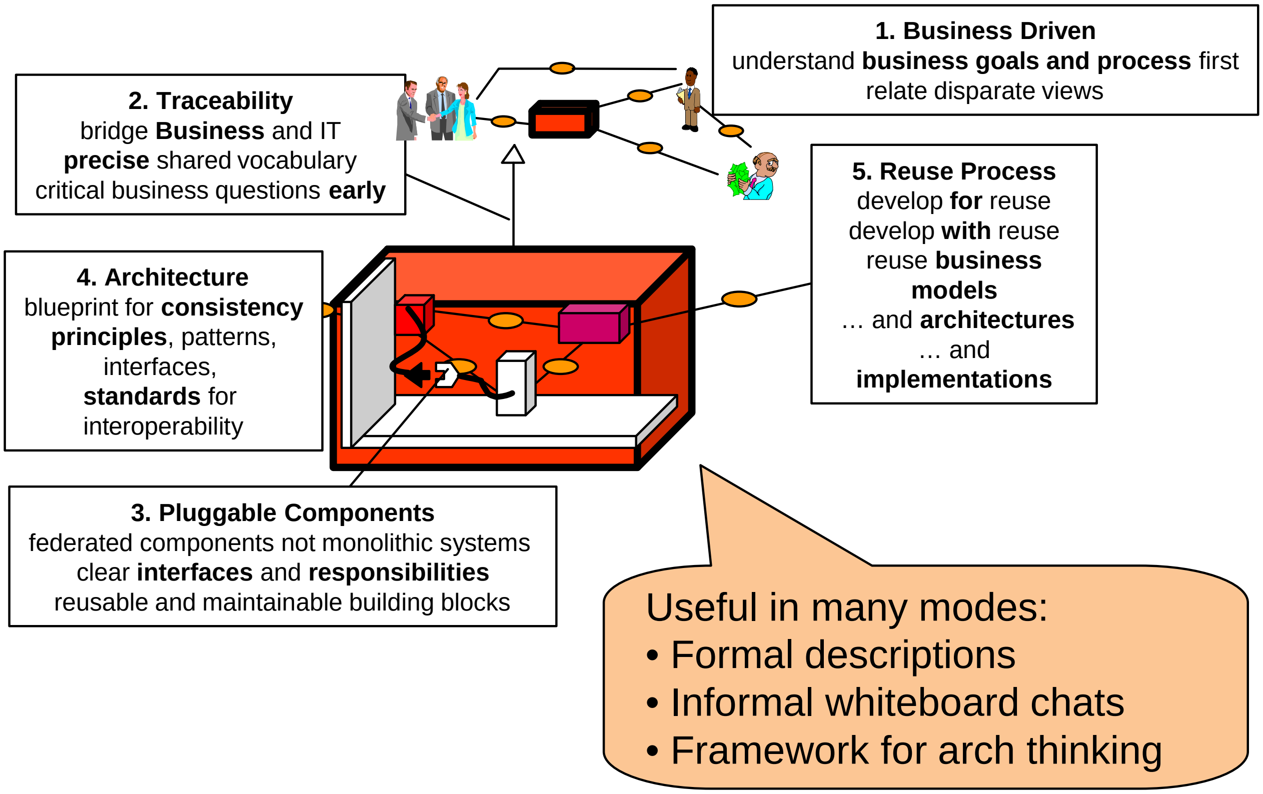
Course Mechanics

- 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?
- This is **“beta”** material … rushed, compressed
- A ? on a slide means 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
  - The instructor will work an example as part of most labs, either on line in the exercise document or offline
  - The students then work out the corresponding student lab
  - If solutions are distributed, use the previous solution as starting point for subsequent lab where appropriate.
- Please fill the course evaluations before leaving

The Approach

- A systematic approach to architect, plan, develop, and evolve software systems
  - Driven by business needs to meet goals and support processes
  - Based on the design and assembly of software components
  - Built consistently to architectural standards
  - Using a model-driven approach to describe those architectures

- We will sometimes call this “CAM” for “Component Architecture Modeling”
  - Sometimes to mean “Component Architecture Method”



1. Business Driven
understand **business goals and process** first
relate disparate views

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

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

4. Architecture
blueprint for **consistency**
principles, patterns, interfaces,
standards for interoperability

3. Pluggable Components
federated components not monolithic systems
clear **interfaces** and **responsibilities**
reusable and maintainable building blocks

Useful in many modes:
- Formal descriptions
- Informal whiteboard chats
- Framework for arch thinking

- Background on the approach
  - Developed, integrated, evolving to meet project requirements since ‘01
  - Builds upon best practice and industry standards for architecture

Course Outline

- Overview of Approach
- Basic Modeling Techniques
  - Modeling information: concrete and abstract
  - Modeling Behavior: concrete and abstract
  - Relating information to behavior
- Business goals, policies, obstacles
  - How to find them
  - How to describe them
  - How to analyze them
- Business processes
  - How to find and describe them
  - How to analyze them
- Current state systems
  - How to describe them

Chapter 1
Overview of Approach

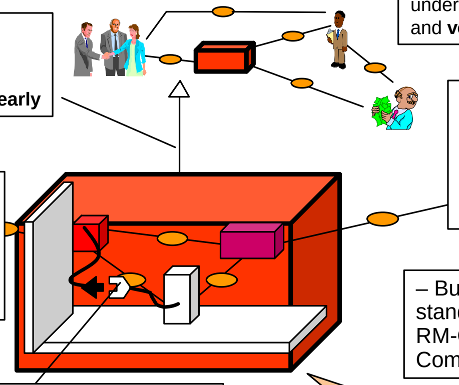
This chapter provides an overview of the approach to Component Architecture Modeling, and how it describes systems in terms of interacting objects at multiple levels of abstraction.

It covers:

- Multiple Viewpoints and Concerns
- Key Concepts of CAM

What is CAM?

- CAM is a systematic approach to **architect**, plan, develop, and evolve software systems
  – Allows clearly separated viewpoints and models to describe and analyze an architecture from business through components to technology



1. Business Driven
understand **business goals, process** and **vocabulary** first, relate disparate views

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

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

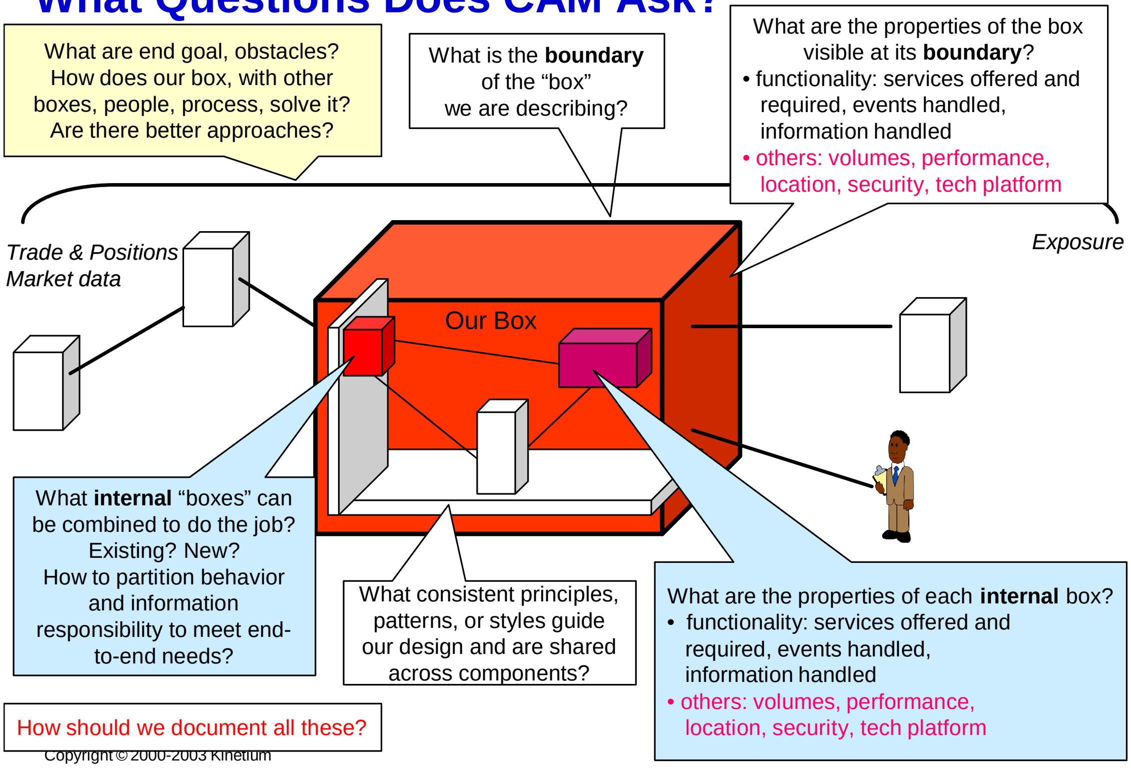
4. Consistent Architecture
blueprint for **consistency**
principles, patterns, interfaces, platforms, …
standards for interoperability

– Builds upon best practice and industry standards for architecture: UML, EDOC, RM-ODP, Catalysis, Simple Components, …

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

CAM is useful in many modes:
• Formal descriptions
• Informal whiteboard chats
• Framework for arch thinking

What Questions Does CAM Ask?



What are end goal, obstacles? How does our box, with other boxes, people, process, solve it? Are there better approaches?

What is the **boundary** of the “box” we are describing?

What are the properties of the box visible at its **boundary**?

- functionality: services offered and required, events handled, information handled
- others: volumes, performance, location, security, tech platform

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?

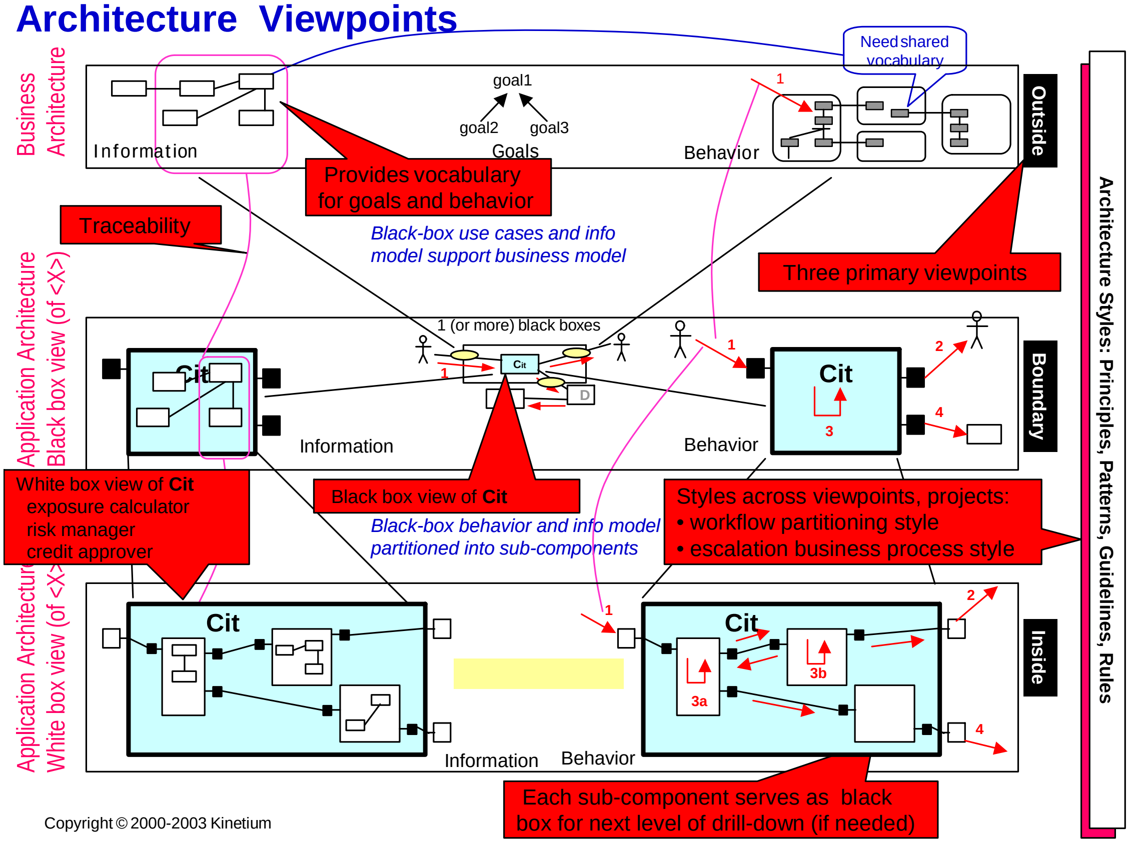
What consistent principles, patterns, or styles guide our design and are shared across components?

What are the properties of each **internal** box?

- functionality: services offered and required, events handled, information handled
- others: volumes, performance, location, security, tech platform

How should we document all these?

Architecture Viewpoints



Provides vocabulary for goals and behavior

Traceability

Black-box use cases and info model support business model

Three primary viewpoints

White box view of **Cit**
exposure calculator
risk manager
credit approver

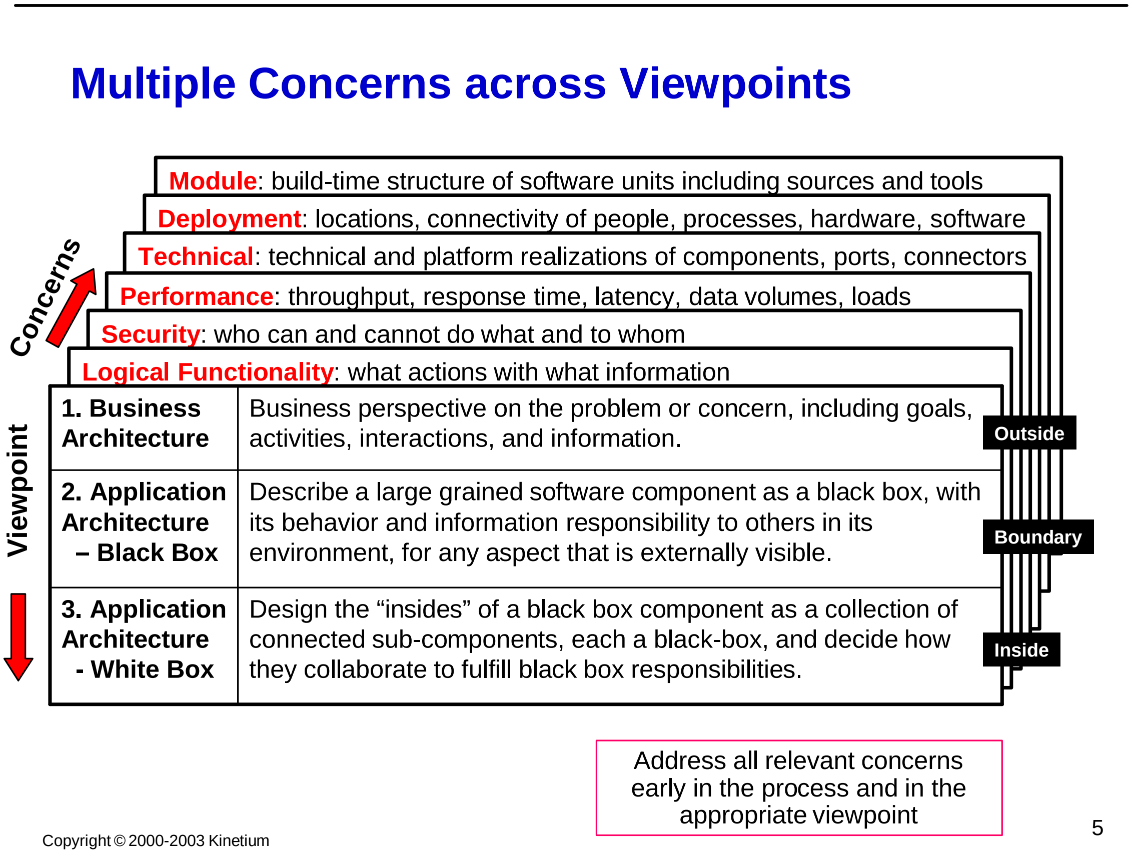
Black box view of **Cit**

Black-box behavior and info model partitioned into sub-components

Styles across viewpoints, projects:

- workflow partitioning style
- escalation business process style

Each sub-component serves as black box for next level of drill-down (if needed)



Multiple Concerns across Viewpoints

Concerns

Module: build-time structure of software units including sources and tools

Deployment: locations, connectivity of people, processes, 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

Viewpoint

1. Business Architecture	Business perspective on the problem or concern, including goals, activities, interactions, and information.
2. Application 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. Application 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.

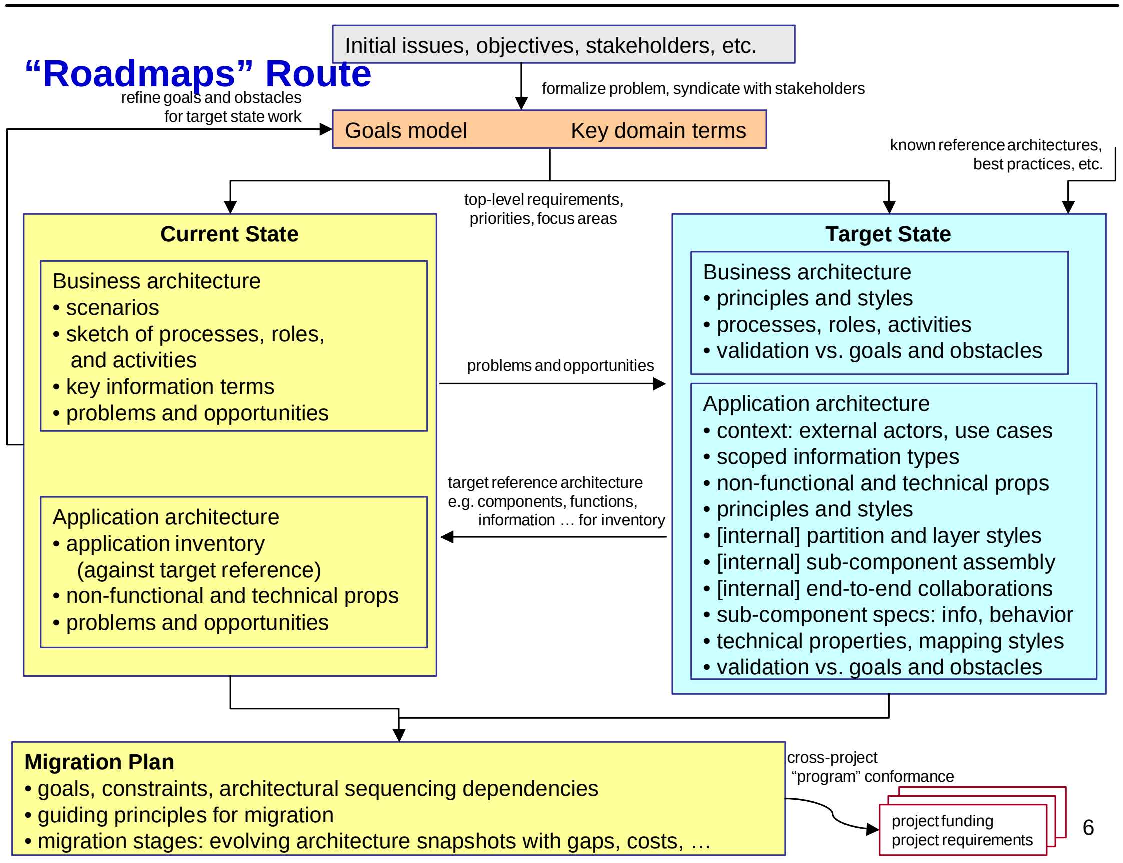
Outside

Boundary

Inside

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

5



“Roadmaps” Route

Initial issues, objectives, stakeholders, etc.

formalize problem, syndicate with stakeholders

refine goals and obstacles for target state work

Goals model Key domain terms

known reference architectures, best practices, etc.

top-level requirements, priorities, focus areas

Current State

Business architecture
- scenarios
- sketch of processes, roles, and activities
- key information terms
- problems and opportunities

Application architecture
- application inventory (against target reference)
- non-functional and technical props
- problems and opportunities

problems and opportunities

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

Target State

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

Application architecture
- context: external actors, use cases
- scoped information types
- non-functional and technical props
- principles and styles
- [internal] partition and layer styles
- [internal] sub-component assembly
- [internal] end-to-end collaborations
- sub-component specs: info, behavior
- technical properties, mapping styles
- validation vs. goals and obstacles

Migration Plan
- goals, constraints, architectural sequencing dependencies
- guiding principles for migration
- migration stages: evolving architecture snapshots with gaps, costs, …

cross-project “program” conformance

project funding
project requirements

6

Chapter 2
Basic Modeling Techniques

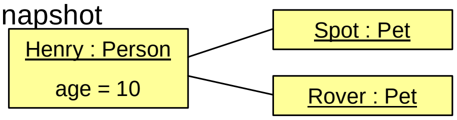
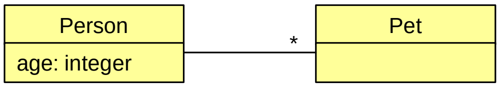
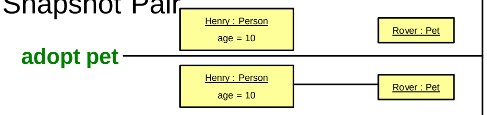
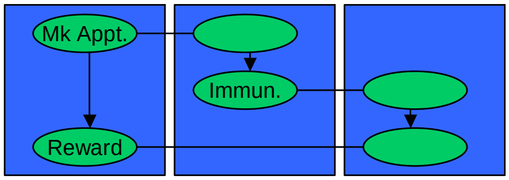
This chapter introduces basic techniques for modeling information and behavior.

Note that these are just the basic tools for our toolkit. We will apply these later in Business, Application black box, and Application 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: Concrete Examples and Abstract Models

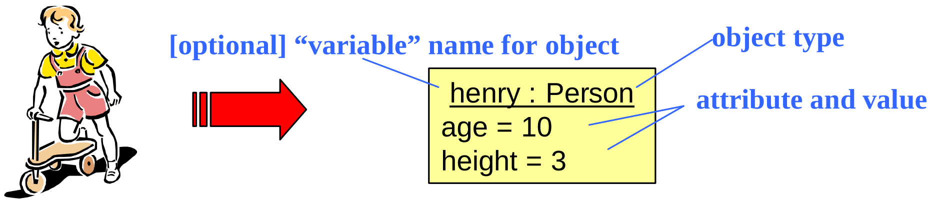
	Concrete Example	Abstract Model
Information	Object <u>Henry : Person</u> age = 10	Type Person age: integer
	Snapshot 	Information Model 
Behavior	Snapshot Pair  adopt pet Henry : Person age = 10 Rover : Pet Henry : Person age = 10 Rover : Pet	Action action **adopt pet** (person, pet) Precondition: … 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, .. 

Objects and Types

This section covers:

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

Modeling Objects



[optional] "variable" name for object

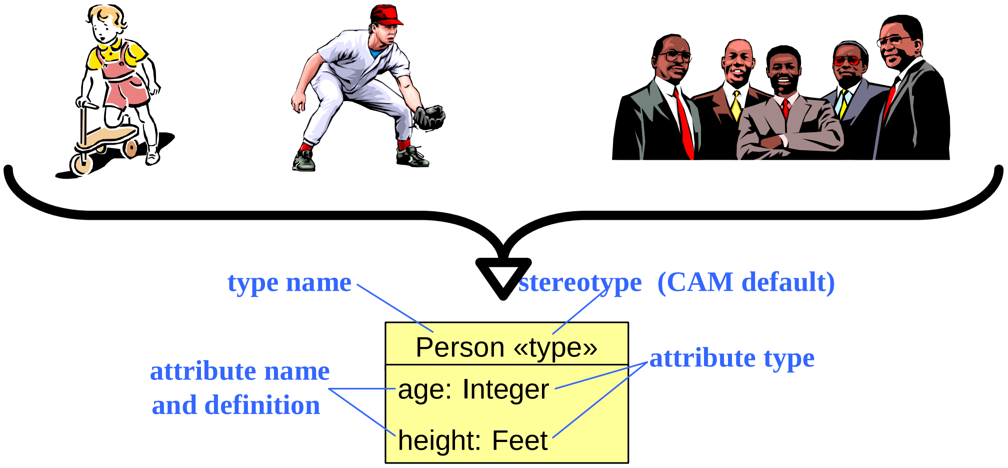
object type

henry : Person
age = 10
height = 3

attribute and value

- Object = a specific individual, identifiable thing that you need to refer to for its
  - **State: attributes with values, or links to other objects**
  - **Behavior: things it does, or that are done to it**
- Single box with underlined title, object ("variable") name, its type, attribute values
- Can refer to object by the variable name
- Can omit variable name; refer to it as **:Person** if unambiguous

Types: Generalized Objects



type name

stereotype (CAM default)

Person «type»

age: Integer

height: Feet

attribute name and definition

attribute type

- **Type** = external **specification** of visible properties of some object
  - independent of implementation – **how** data is stored (or **if**), step-wise procedures
- 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 logical structure and/or state

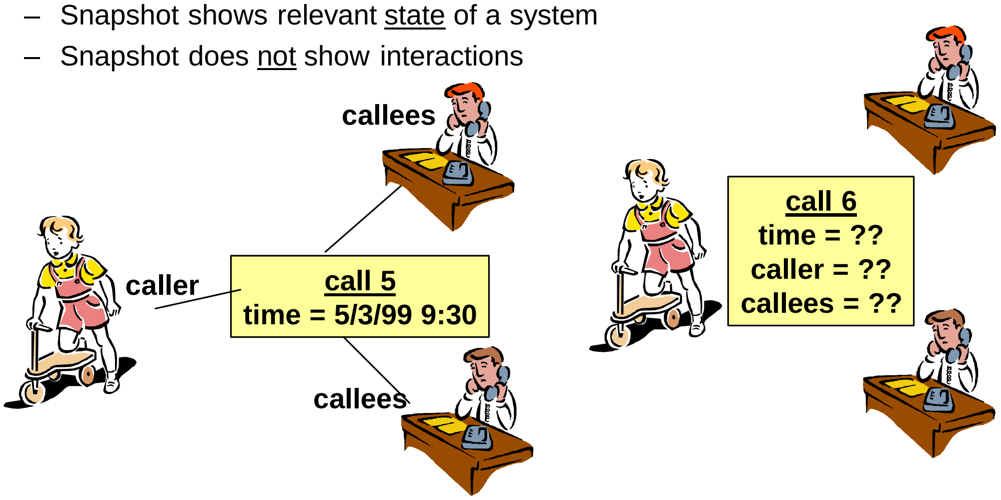
Snapshots and Information Models

This section covers:

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

Snapshots: Linked Objects

- A snapshot is a configuration of objects that are linked to each other at some 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



callees

caller

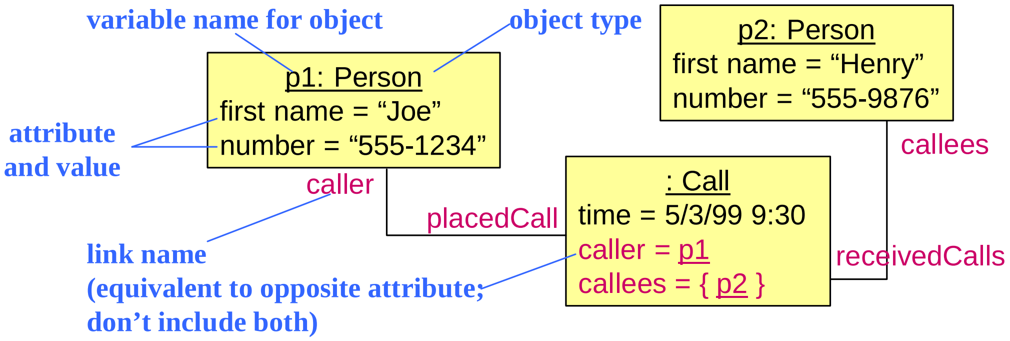
<u>call 5</u>
time = 5/3/99 9:30

callees

<u>call 6</u>
time = ??
caller = ??
callees = ??

How to Read a Snapshots in UML

- Uses UML "collaboration diagram" but without showing interactions



variable name for object

object type

<u>p1: Person</u>
first name = "Joe"
number = "555-1234"

attribute and value

caller

placedCall

link name (equivalent to opposite attribute; don't include both)

<u>: Call</u>
time = 5/3/99 9:30
caller = <u>p1</u>
callees = { <u>p2</u> }

<u>p2: Person</u>
first name = "Henry"
number = "555-9876"

callees

receivedCalls

<u>p1 is a Person</u>

- p1's name is "Joe"
- p1's placedCall is **:Call**

<u>**:Call** is an Call</u>

- its time is 5/3/99 9:30
- its caller is p1
- its callees is p2

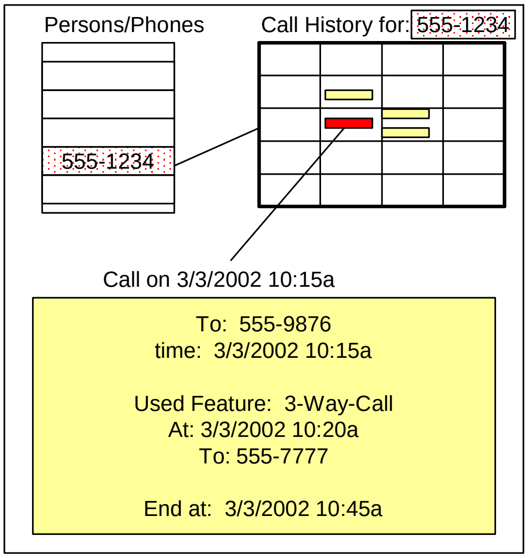
<u>p2 is a Person</u>

- p2's name is "Henry"
- p2's receivedCall is **:Call**

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

Populated User Interface shows Snapshots

- 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.
  - floor plans
  - factory flows
- Use as a means to uncover Information Model, not as an end



Persons/Phones

Call History for: 555-1234

555-1234

Call on 3/3/2002 10:15a

To: 555-9876
time: 3/3/2002 10:15a

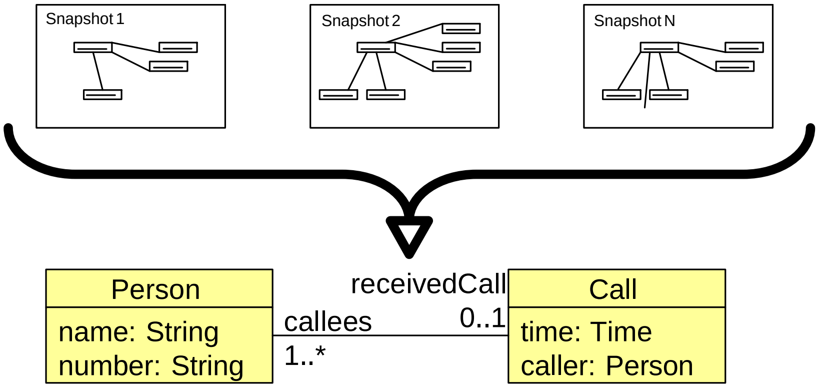
Used Feature: 3-Way-Call
At: 3/3/2002 10:20a
To: 555-7777

End at: 3/3/2002 10:45a

Exercise 3.1

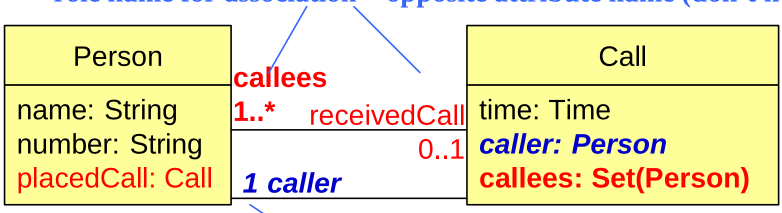
Information Model Generalizes Snapshots

- 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)

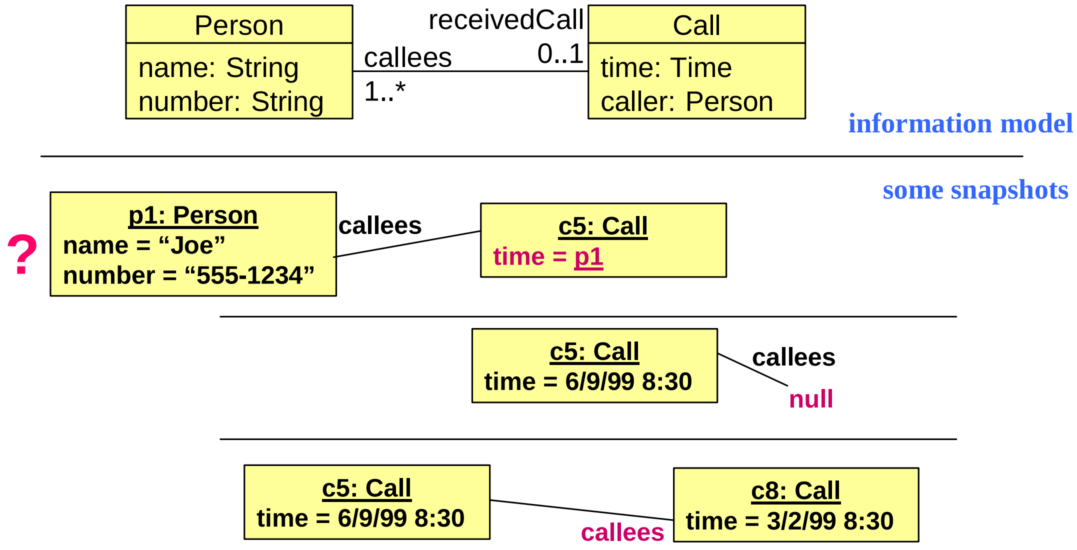


multiplicity: 1..* = 1 or more; 0..1 = optional
(multiplicity annotation permitted on attributes as well)

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?

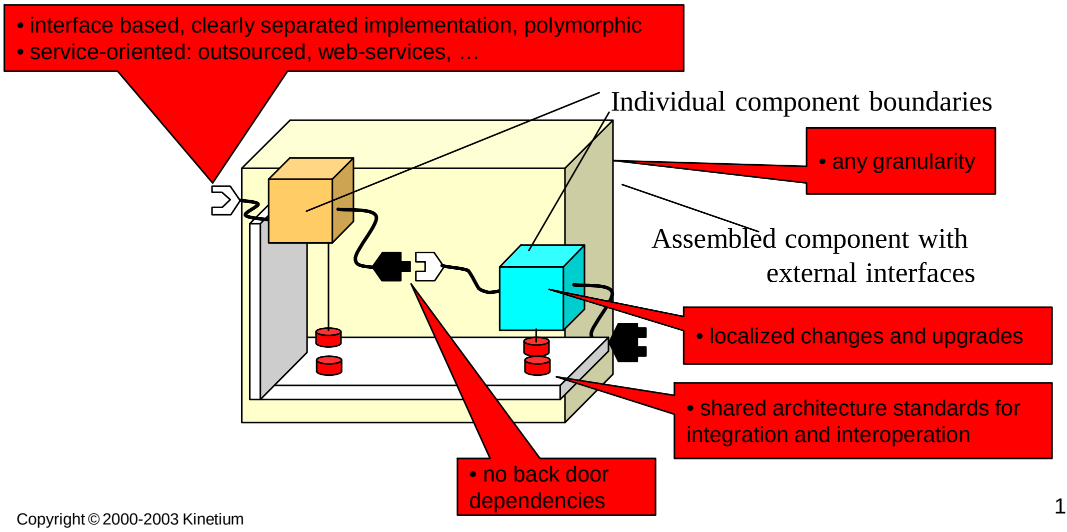


- Snapshots can be incomplete
  - Information, attributes, links omitted are not *null* unless explicitly marked
- Which of these snapshots is invalid and why?

Exercise 3.2

What is a Component?

t Somewhat like a large grained object.
t A package of software that can be independently replaced.
t It both provides and requires services with specified interfaces.
t It conforms to architectural standards to interoperate with other components.
t It has an external specification and an internal implementation.



• interface based, clearly separated implementation, polymorphic
• service-oriented: outsourced, web-services, …

Individual component boundaries

• any granularity

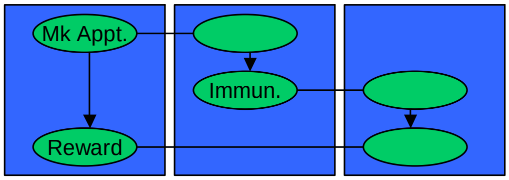
Assembled component with external interfaces

• localized changes and upgrades

• shared architecture standards for integration and interoperation

• no back door dependencies

Contd. Concrete Examples and Abstract Models

Concrete Example	Abstract Model
Object Henry : Person age = 10	Type Person age: integer
Snapshot Henry : Person age = 10 Spot : Pet Rover : Pet	Information Model Person age: integer * Pet
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, .. 

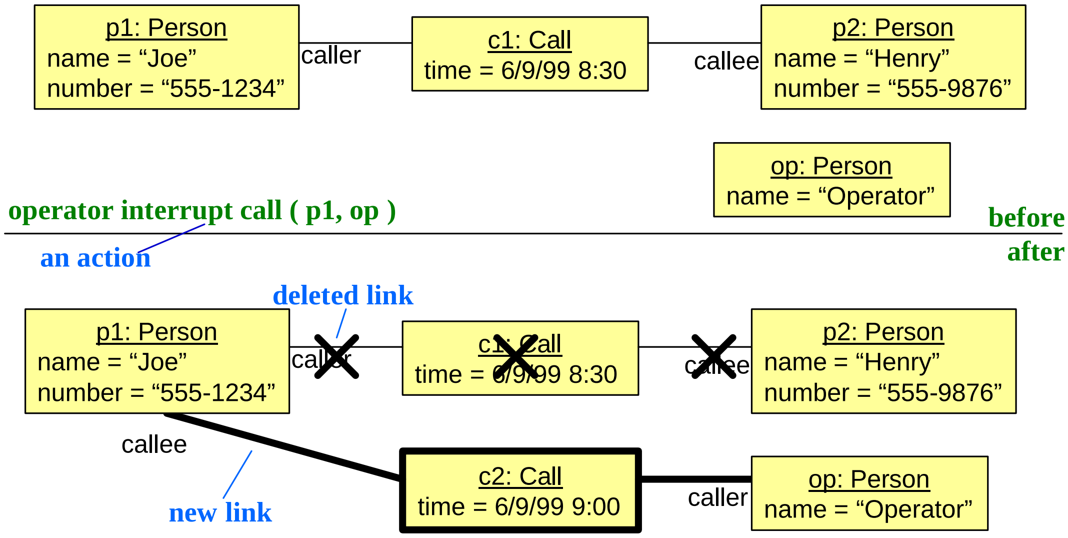
Snapshot Pairs and Actions

This section covers:

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

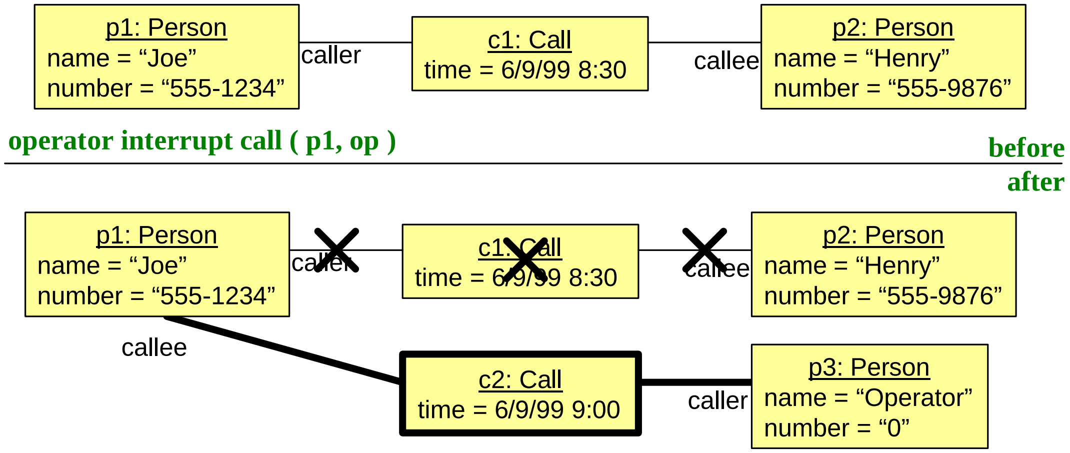
Actions Correspond to Snapshot Changes

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



- Can simply sketch: overlay "frames" with color change for before à after

How to Read Snapshot Pairs



p1: Person
name = "Joe"
number = "555-1234"

caller

c1: Call
time = 6/9/99 8:30

callee

p2: Person
name = "Henry"
number = "555-9876"

operator interrupt call (p1, op)

before

after

p1: Person
name = "Joe"
number = "555-1234"

caller

c1: Call
time = 6/9/99 8:30

callee

p2: Person
name = "Henry"
number = "555-9876"

callee

c2: Call
time = 6/9/99 9:00

caller

p3: Person
name = "Operator"
number = "0"

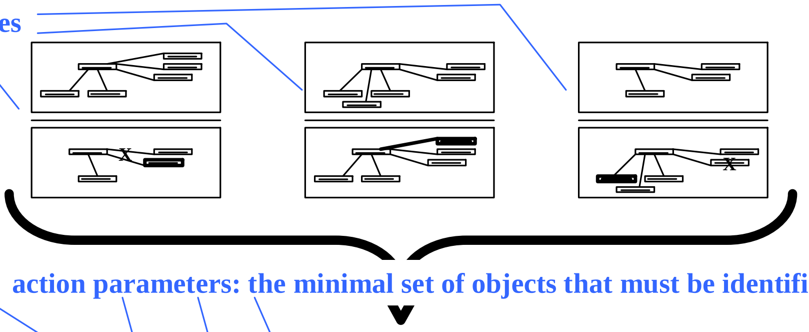
- Before: person p1 had a call c1 to person p2
- A **operator interrupt call** happens with op
- After: c1 is no more; new call c2 exists with caller op (the operator) and callee p1
- Underlined terms are names of objects (or name of attribute that refers to object)
- "Parameters" are those objects that 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, ...



action name

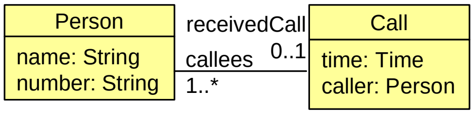
action parameters: the minimal set of objects that must be identified

action specification

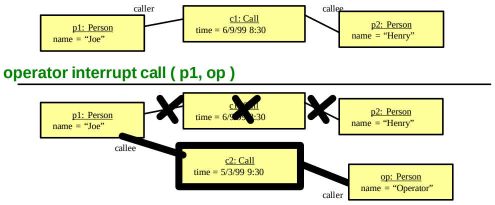
action a1 (t: T, r: R, s: S)
description narrative description of the action
precondition condition that, if true in a “before” snapshot …
postcondition … will result in the “after” snapshot in this condition

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
-- 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 Activity Diagrams

This section covers:

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

Scenarios

- A scenario is a story of a concrete 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.

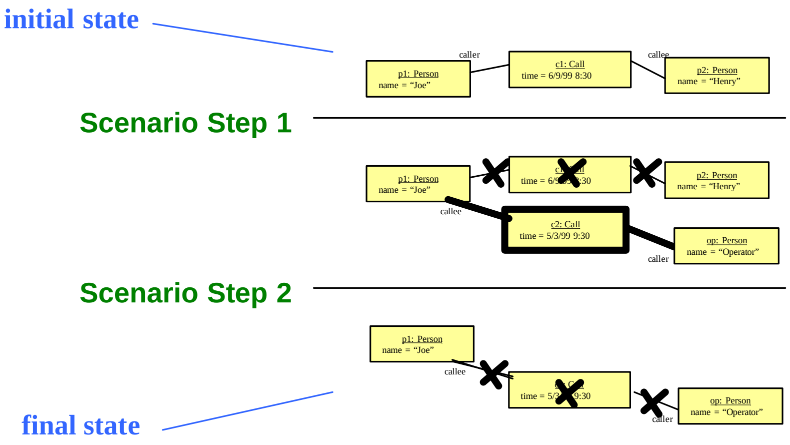
4: The operator interrupts the call and gives Joe an emergency message.

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

Note: scenario can also be shown with UML diagrams (e.g. sequence or collaboration diagram), provided they show specific objects, values, and states

Scenarios with Snapshots

- Any scenario can be explored and understood better with snapshots

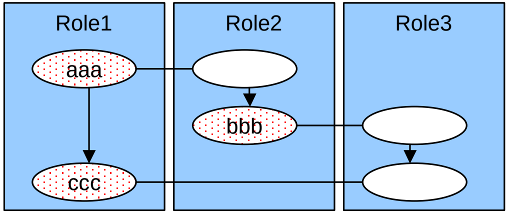


- Gives a "filmstrip" view of the changes that happen through the scenario

Exercise 3.5

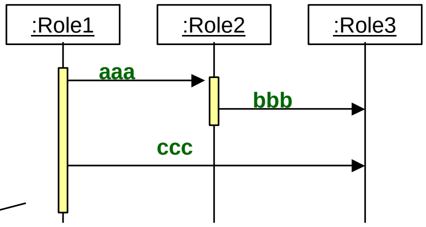
Alternatives to Specify Action Sequences

- Action specifications implicitly define action sequences by pre/post

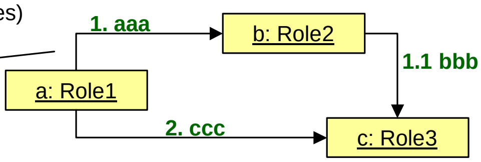


- Activity Diagrams
  - Useful for business process modeling
- Use cases
  - Textual and compact

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 …



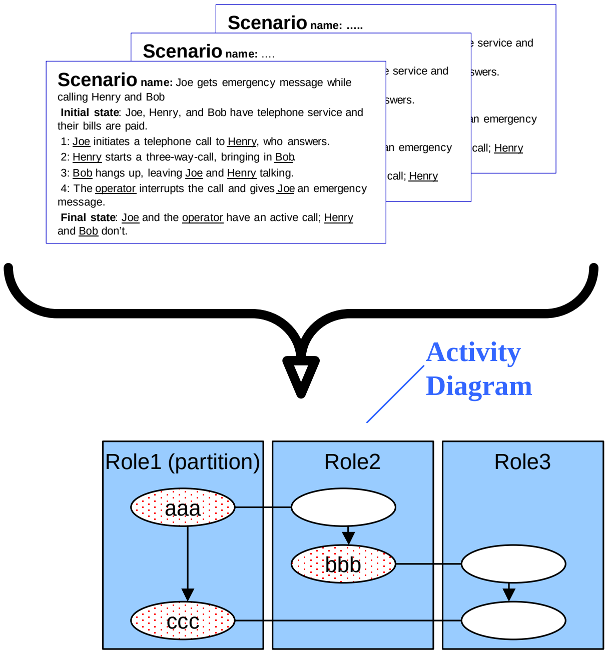
- UML Sequence Diagrams
  - Good for showing time sequence (often examples)



- UML Collaboration Diagrams
  - Good at showing 2D layout ("The print server is always on the bottom left…")
  - Often used for examples

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: 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 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.

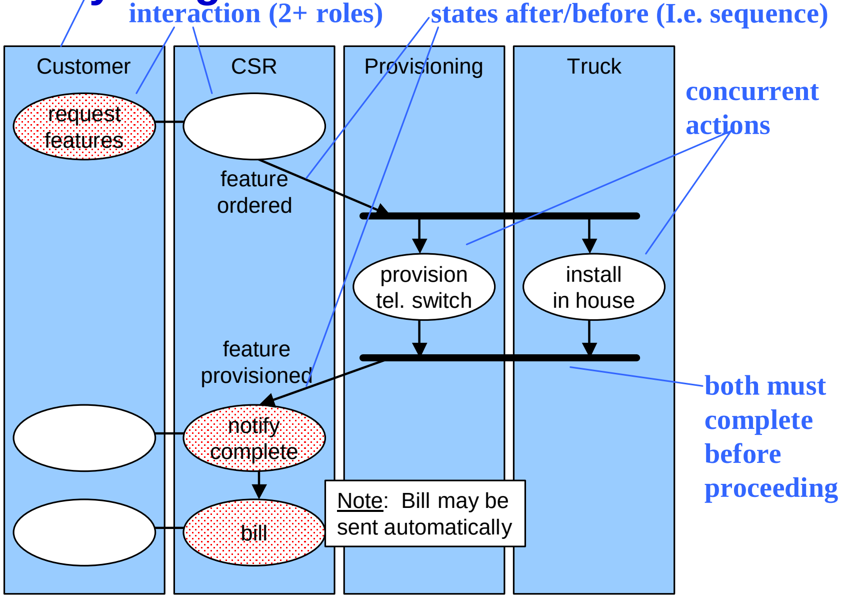
2: The switch provisioning tech activates 3WC

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.

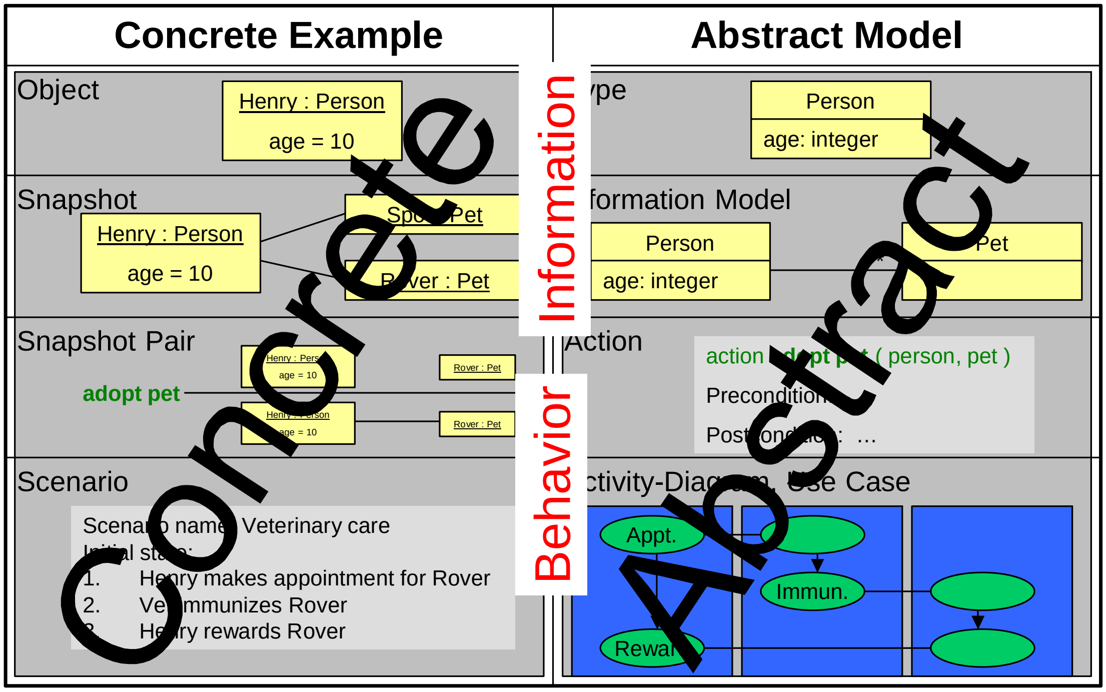
- ***Final state**: Joe has 3WC and DSL active on his phone line and MegaTel remembers to bill him each month.*



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

Exercise 3.6

Concrete vs. Abstract : Information and Behavior



Scenario name: Veterinary care
Initial state:
1. Henry makes appointment for Rover
2. Vet immunizes Rover
3. Henry rewards Rover

Summary

This chapter showed information and behavior in the concrete and abstract

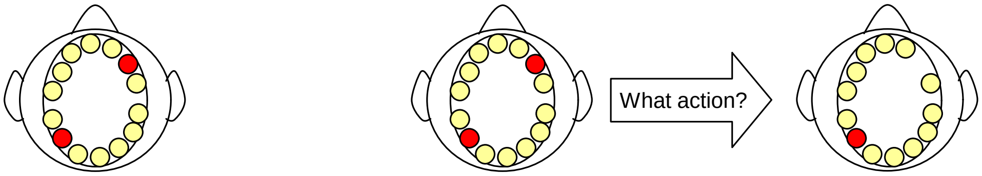
- Information
  - Objects are and snapshots are concrete examples of information state
  - Types and Information Models are general specifications of information
- Behavior
  - Snapshot pairs and scenarios are concrete examples of behaviors
  - Action specs, activity diagrams, use cases... are general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

Information Supports Behaviors / Constraints

This section covers:

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

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
  - The interactions that are possible

Information Model supports Behaviors, Constraints

Behaviors or constraints to be captured

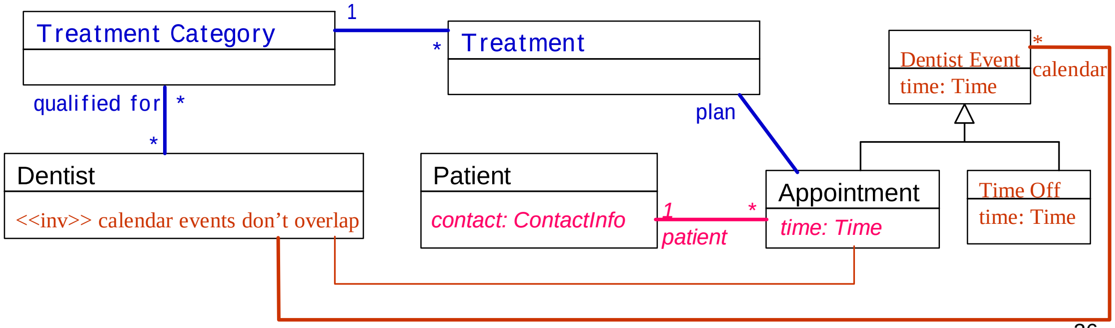
One day before the appointment, the system should contact the patient with a reminder.

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

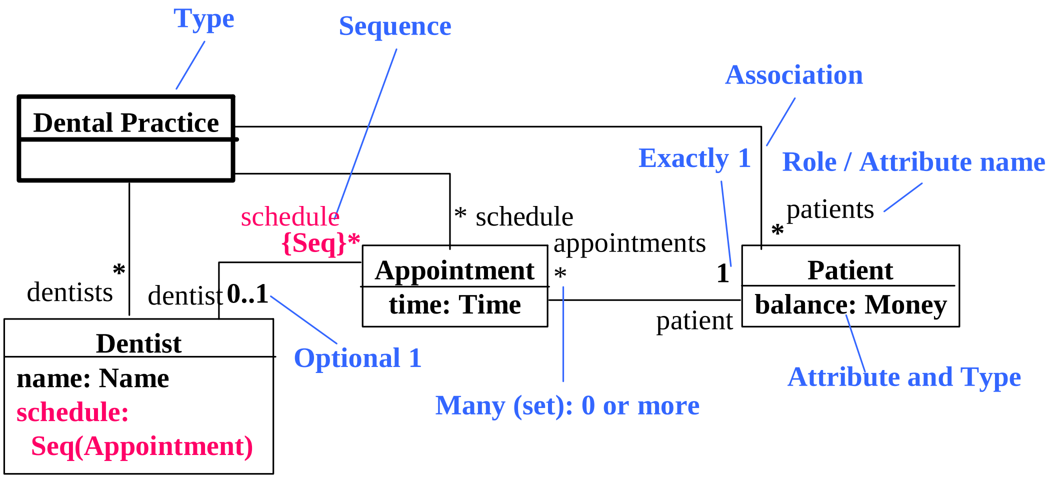
A dentist should not be assigned an appointment at a time when he/she has personal time off scheduled

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**

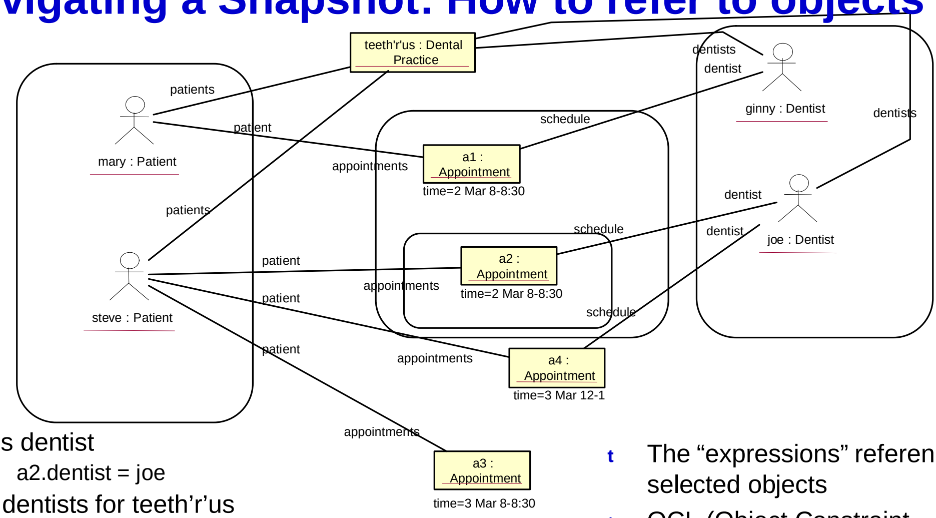


Information Model (adding “top-level” Object)



- 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 à select(time=2Mar) = { a1, a2 }

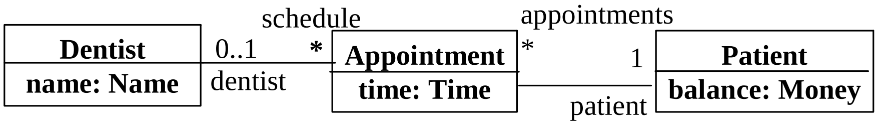
  Full form: **set_expression à select (iterator | condition_on_iterator)**
- All appointments for joe on 2Mar: joe.schedule à select(a | a.time=2Mar) = { a2 }

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

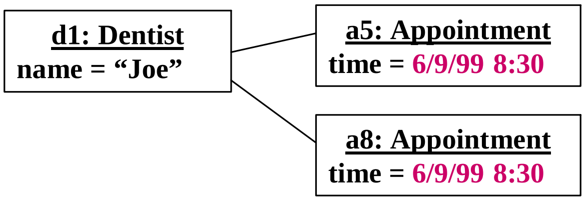
In this course we could use OCL to learn clarity in models, although many projects will not use it much in practice.

Exercise 4.1

Cannot Capture All Constraints Graphically



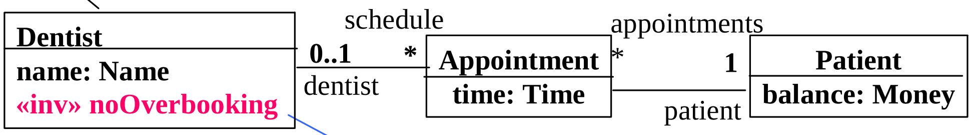
- Can a dentist have overlapping appointments?



Invariants are Explicit Constraints on State

(for any **dentist**)
No two **appointments** in its <u>schedule</u> with overlapping <u>times</u>

Invariant entered either (1) a UML "Note" (referring to <u>attributes</u> or <u>vars</u> and **types**)



Or (2) separate description in text document (referring to <u>attributes</u> or <u>vars</u> and **types**) [optionally with named «inv»]

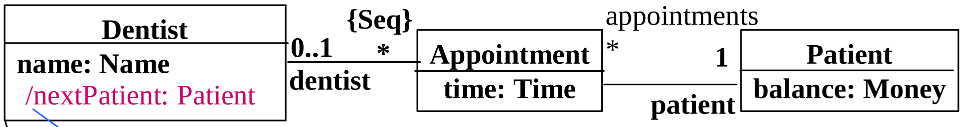
Dentist (invariants)

–noOverbooking — For any **dentist**, there are never two or more **appointments** in its <u>schedule</u> with overlapping <u>times</u>

- 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
- 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

"Convenience" attributes simplify terms



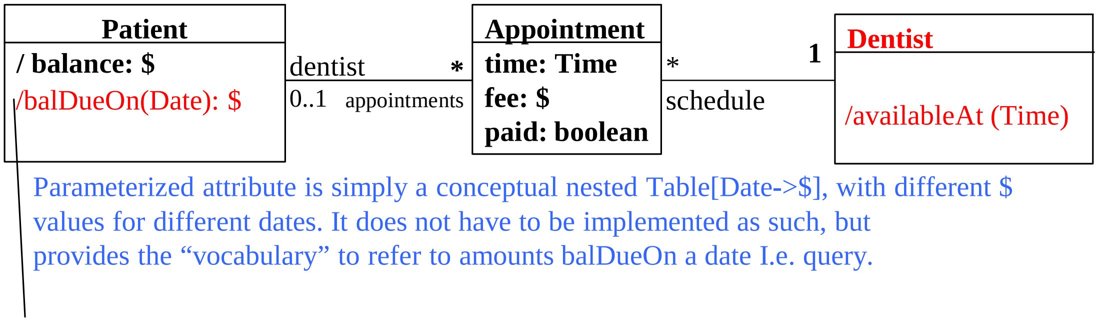
"/" means attribute is for convenience; it can be "derived" from others.
It should be accompanied by a derivation expression (formal or informal)

<u>nextPatient</u> means the <u>patient</u> in the first <u>scheduled</u> **appointment** in the future

[OCL: nextPatient = schedule->select (time > now)->first.patient]

- "The next patient for <u>d1</u>" 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

Attribute as Query with Parameters

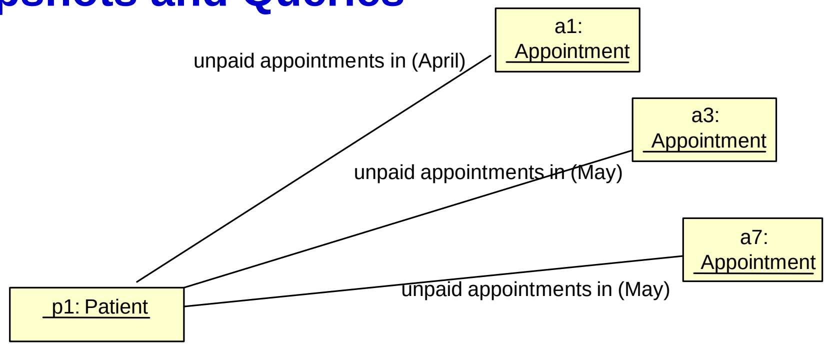


Parameterized attribute is simply a conceptual nested Table[Date->$], with different $ values for different dates. It does not have to be implemented as such, but provides the “vocabulary” to refer to amounts balDueOn a date I.e. query.

balDueOn (d: Date) is the total fees of unpaid appointments 45+ days older than d

- 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** (Time)?
- Why isn’t this in the Behavior Model?

Snapshots and Queries

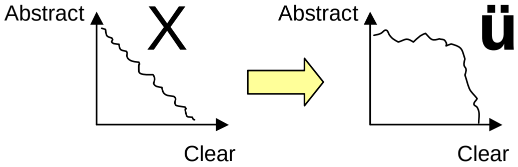

a1: Appointment
unpaid appointments in (April)
a3: Appointment
unpaid appointments in (May)
a7: Appointment
unpaid appointments in (May)
p1: Patient

balanceDueOn (6/1/99) = $500

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

Primary Goal: Clarity

1. Abstract ≠ Fuzzy   Clear ≠ Detailed
2. **Clear terms** give clear specifications



“Trust nothing except code”

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

Chapter 3
Goal Modeling

Business goals motivate the design of processes, roles, responsibilities. Goals ultimately motivate choice of software architecture.

This chapter introduces techniques for modeling business goals, obstacles, and stakeholders.

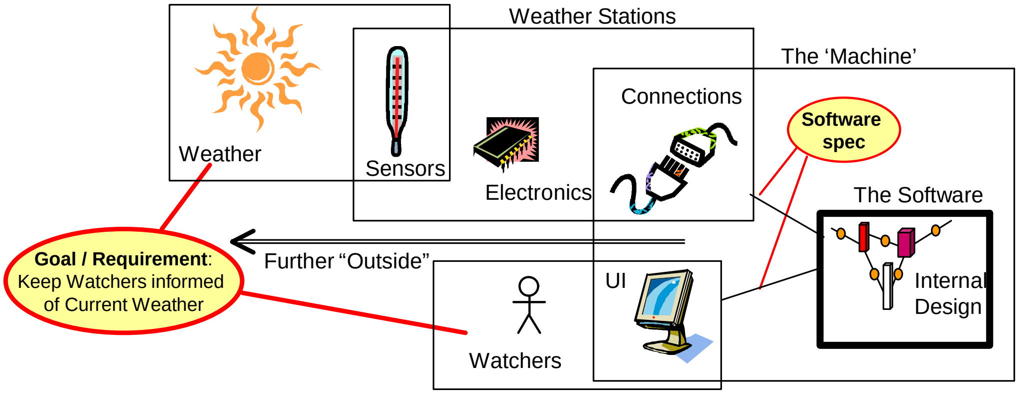
It covers:

- Goals
- Obstacles
- Responsibilities
- Processes

Outline

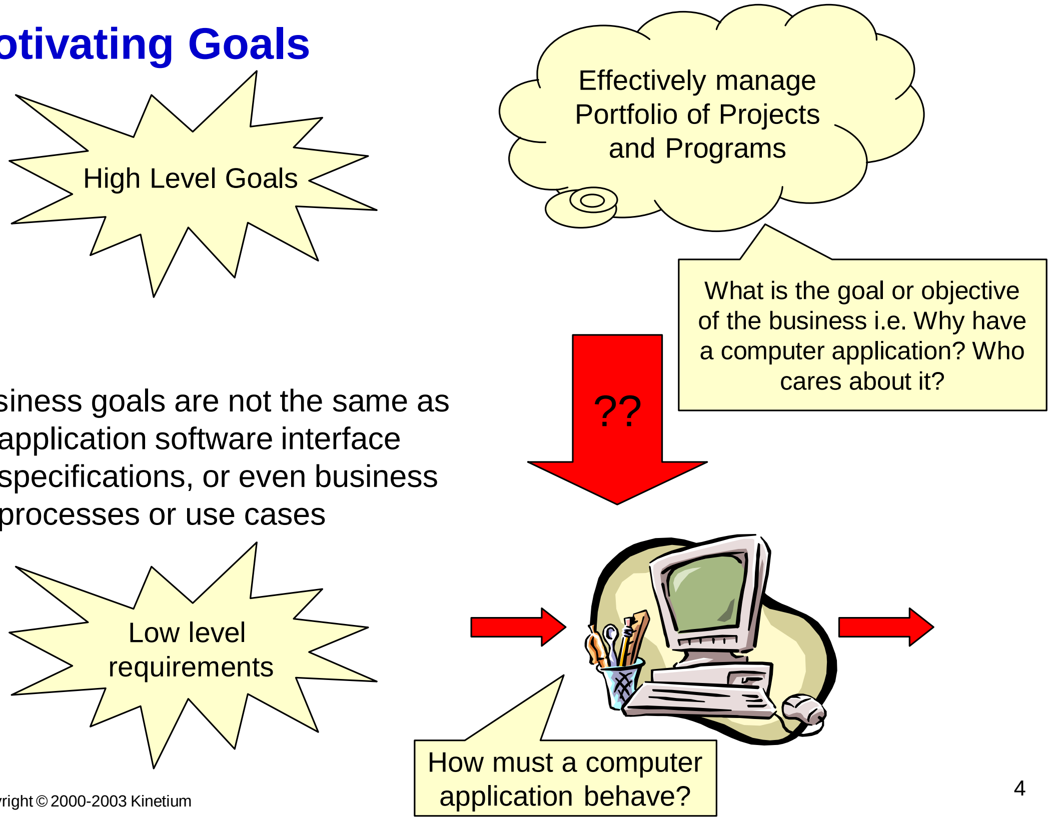
- **What is the Motivation for “Goals”?**
- Goals Modeling and Refinement
- From Goals to Responsibilities, Processes and Actions
- Obstacles and Conflicts
- Soft Goals
- “List” and “Draft” versions of goals models

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



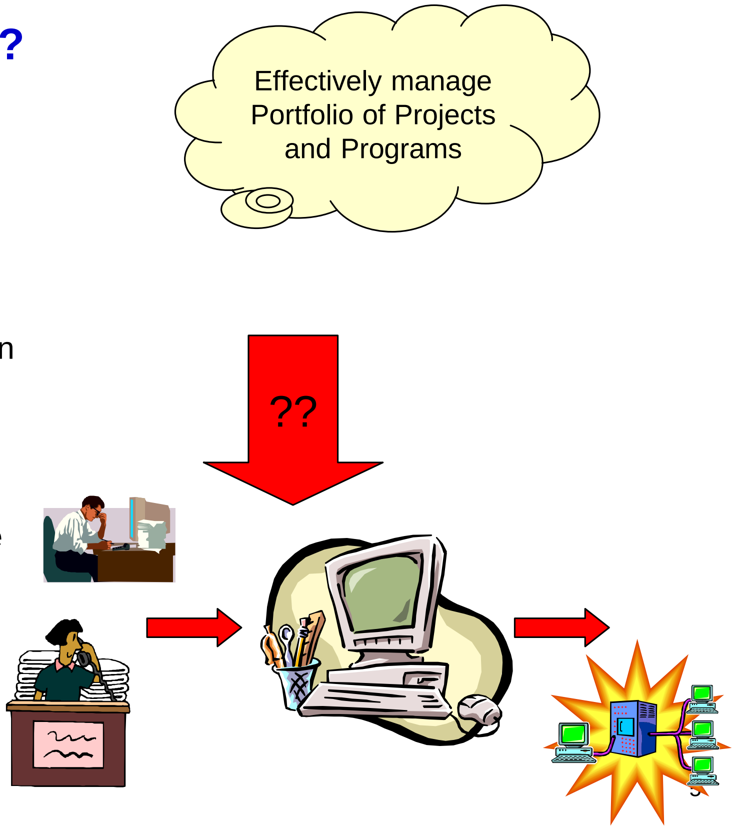
- A domain is a part of the world with some shared state or interactions with other domains
  - Weather, weather stations, watchers, the machine
- Problem domain is different from Machine domain and intermediate "connection" domains
  - The problem to be solved may be **outside** the machine; the machine + connections help solve it
- **Goals** (objectives, outcomes) 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 a part of the solution, when combined with other domains and human operating procedures

Motivating Goals


High Level Goals
Effectively manage Portfolio of Projects and Programs
What is the goal or objective of the business i.e. Why have a computer application? Who cares about it?
??
Low level requirements
How must a computer application behave?

Business goals are not the same as application software interface specifications, or even business processes or use cases

Who Meets Goals?

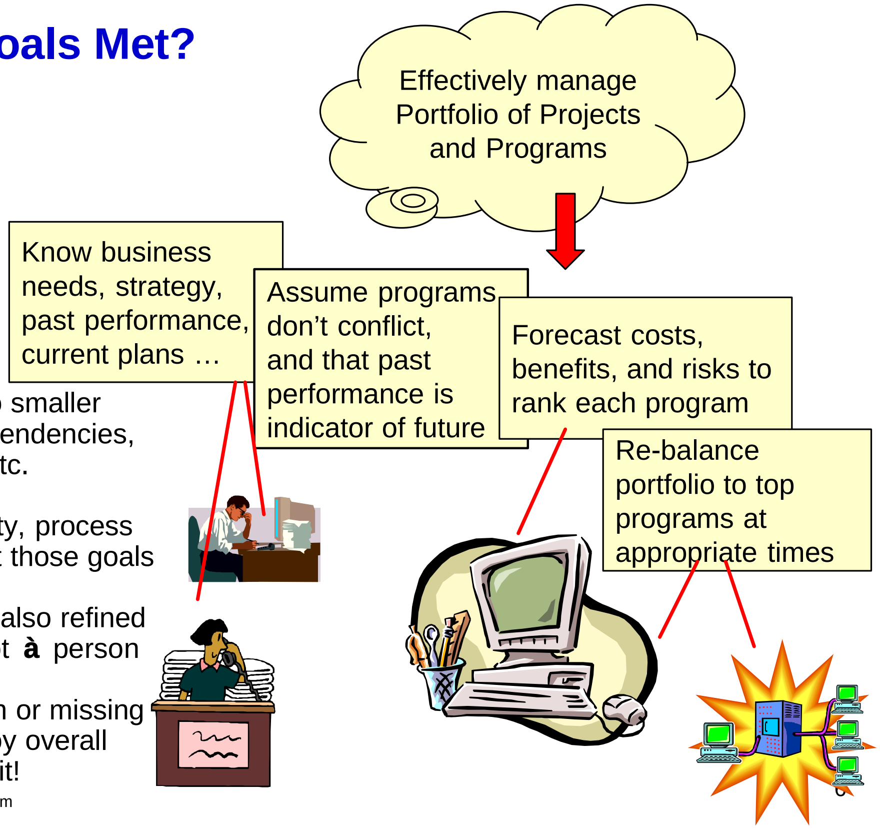


Effectively manage Portfolio of Projects and Programs

Achieving complex high-level goals will always require the co-operation of many players

Identifying these players, their granularity, roles, and responsibilities are part of ensuring that goals are adequately met

How are Goals Met?



Effectively manage Portfolio of Projects and Programs

Know business needs, strategy, past performance, current plans …

Assume programs don't conflict, and that past performance is indicator of future

Forecast costs, benefits, and risks to rank each program

Re-balance portfolio to top programs at appropriate times

Goal is refined into smaller goals, with dependencies, assumptions, etc.

Roles, responsibility, process chosen to meet those goals

Role granularity is also refined e.g. firm à dept à person

Broken assumption or missing goal can destroy overall business benefit!

The Ultimate Success Criterion

Effectively manage Portfolio of Projects and Programs ???

Know business needs, strategy, past performance, current plans …

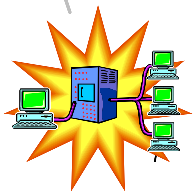
Assume programs don't conflict, and that past performance is indicator of future

Forecast costs, benefits, and risks to rank each program

Re-balance portfolio at appropriate times

The final success criterion:

Do the applications, humans, and processes together meet the goals?



Hence Goals Need Clear Definition

Effectively manage Portfolio of Projects and Programs

At all times maintain the firm's top ranked fundable programs within the project portfolio.

Program rank is based on

§ forecast program value, from plan, actual, and forecast of all projects in that program
§ it's overall alignment with firm strategy
§ it's minimal dependence on other programs

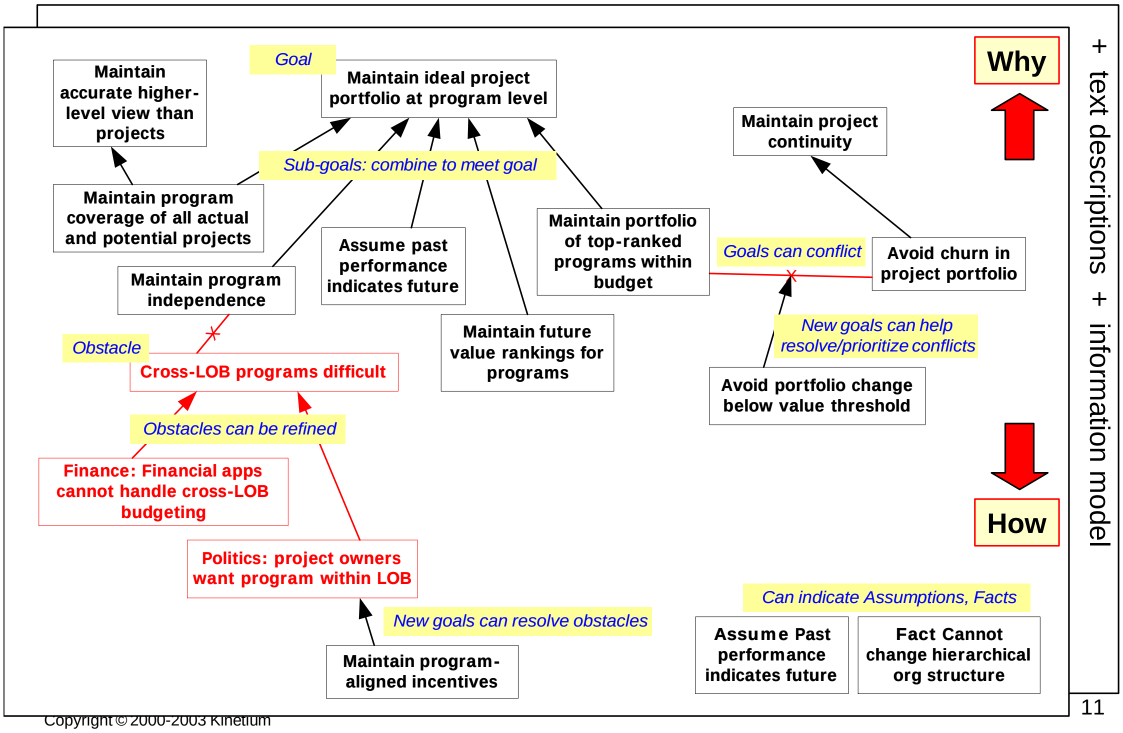
Outline

- What is the Motivation for “Goals”?
- **Goals Modeling and Refinement**
- From Goals to Responsibilities, Processes and Actions
- Obstacles and Conflicts
- Soft Goals
- “List” and “Draft” versions of goals models

What is a Goal?

- A goal defines something **desired** by some **stakeholders**, to be fulfilled by some system that is usually composed of multiple cooperating human, hardware and software agents

Goals Model: Goals, Refinement, Obstacles



Why

How

+ text descriptions + information model

Goal

Maintain ideal project portfolio at program level

Maintain accurate higher-level view than projects

Sub-goals: combine to meet goal

Maintain program coverage of all actual and potential projects

Maintain program independence

Assume past performance indicates future

Maintain portfolio of top-ranked programs within budget

Maintain future value rankings for programs

Maintain project continuity

Goals can conflict

Avoid churn in project portfolio

New goals can help resolve/prioritize conflicts

Avoid portfolio change below value threshold

Obstacle

Cross-LOB programs difficult

Obstacles can be refined

Finance: Financial apps cannot handle cross-LOB budgeting

Politics: project owners want program within LOB

New goals can resolve obstacles

Maintain program-aligned incentives

Can indicate Assumptions, Facts

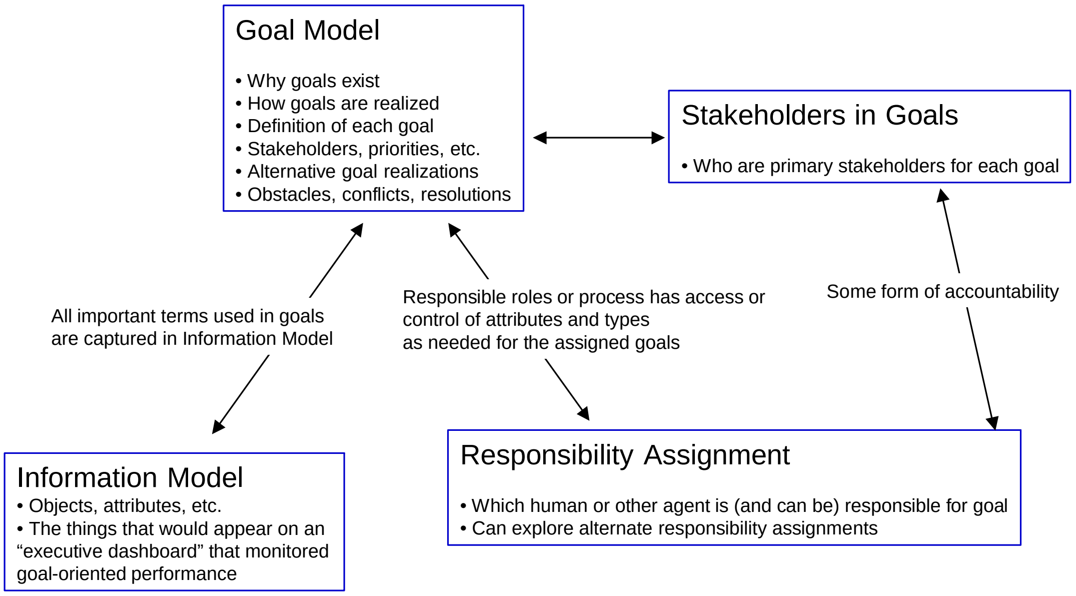
Assume Past performance indicates future

Fact Cannot change hierarchical org structure

Goals Modeling Process – Fully Dressed

- Find all stakeholders: external roles, users, customers, regulators, marketing, etc.
- Identify initial goals: from stakeholders, VOC, pain-points, documentation, etc.
- Define each goal clearly; if formalized, use one of the following forms where possible
  - Maintain < condition to be maintained >
  - Avoid < condition to be maintained >
  - Achieve < condition to be achieved >
  - Cease < condition to be ceased >
  - Min / Max / Optimize < condition to be minimized, maximized, optimized >
- Uncover key underlying terms of goals in a glossary or information model
- Explore hidden and alternate goals and assumptions, by asking for each goal, G
  - **Why** is G a goal: this will uncover rationale and find a higher-level goal, G1
  - **How** is goal G1 fully realized: this will find sub-goals, assumptions, alternatives to realize G1
- Find obstacles to goals, by asking what might keep each goal G from being met
  - Address obstacles with changed goals e.g. obstacle prevention, obstacle minimization, goal weakening, etc.
- Find conflicting goals and the conditions under which they conflict
  - Resolve conflicts by changed goals e.g. prioritize one goal over the other in condition of conflict
- Refine goals until “realizable” by process or roles (at chosen level of granularity)
  - Goal requires to control something (e.g. portfolio balance) and observe others (e.g. market), to meet certain conditions (e.g. maintain risk/return ratio within targets)
  - Goal is “realizable” by process or roles if it can control and observe as required by the goal
  - Lower level: use-case goals, pre/post conditions, etc. do not need separate goals model

Main Elements of Goal Modeling – “Dressed”



Goal Model

- Why goals exist
- How goals are realized
- Definition of each goal
- Stakeholders, priorities, etc.
- Alternative goal realizations
- Obstacles, conflicts, resolutions

Stakeholders in Goals

- Who are primary stakeholders for each goal

All important terms used in goals are captured in Information Model

Responsible roles or process has access or control of attributes and types as needed for the assigned goals

Some form of accountability

Information Model

- Objects, attributes, etc.
- The things that would appear on an “executive dashboard” that monitored goal-oriented performance

Responsibility Assignment

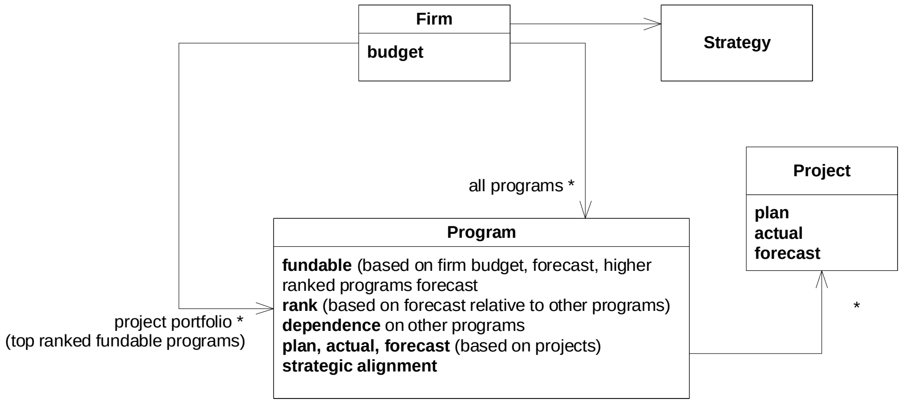
- Which human or other agent is (and can be) responsible for goal
- Can explore alternate responsibility assignments

Light weight “List” and “Draft” versions are equally useful

Goals and Information Model

Goal Maintain ideal project portfolio at program level

At all times maintain the **firm’s** top **ranked** **fundable** **programs** within the **project** **portfolio**. Rank is based on
§ **forecast** program value, from **plan**, **actual**, and **forecast** of all **projects** in that program
§ it’s overall **alignment** with firm strategy
§ it’s minimal **dependence** on other programs



Firm
budget

Strategy

Project
plan
actual
forecast

all programs *

*

Program
fundable (based on firm budget, forecast, higher ranked programs forecast
rank (based on forecast relative to other programs)
dependence on other programs
plan, actual, forecast (based on projects)
strategic alignment

project portfolio *
(top ranked fundable programs)

t Goals become clearer and fewer, goal structure better motivated

Information Model – Pragmatic Formalization

- Ambiguities are inevitable and natural in initial goals definition
  - Inherent ambiguity in language, or to deliberately avoid committing too early
  - Definition of objects/attributes and the definition of goals themselves
- For example, *“forecast value of program”* could mean
  - the value the local sponsor assigns to the program
  - a risk-adjusted and present-valued benefit to the entire firm from the program
- Missing key definitions makes it very hard to know what is meant
- **However**
  - Not all ambiguities need to be solved as soon as they arise
  - Solve after further elaboration when you have better understanding of overall issues
- **Hence**
  - Start with idealized goals: they are often easier to make clear
  - e.g. “always maintain firm-wise optimal risk-adjusted and time-adjusted project portfolio”
  - Premature compromises on the ideal makes alternatives and trade-offs impossible

Kinds of Goals

- Goal classifications: Achieve, Cease, Maintain, Avoid
  - Achieve goals: goals requiring that some property become true
    - If Condition is true then State should become true [within some time frame]
  - Cease goals: goals requiring that some property will stop being true
    - If Condition is true then State should become false [within some time frame]
  - Maintain goals: goals requiring that some property remain true
    - If Condition is true then State should be maintained always true [until Stop_Condition, within time frame]
  - Avoid goals: goals requiring that some property remain false
    - If Condition is true then State should never become true [until Stop_Condition, within time frame]
- Soft goals
  - Somewhat “fuzzy” goals that do not have clear-cut satisfaction criterion
  - Optimization goals e.g. Min, Max goals: goals to minimize or maximize some property
  - Soft goals are useful to guide and choose between alternative approaches
- As we get towards software requirements consider different kinds of requirements
  - Functional: use cases, feature sets, capabilities
  - Security: access control, non-repudiation
  - Usability: human factors, aesthetics, help, documentation and training
  - Reliability: frequency and severity of failure, recoverability, predictability, accuracy
  - Performance: response time, scalability, throughput
  - Supportability: test, extend or adapt, maintain, configure, operate and manage

Scenarios and Goals

- Scenarios and goals have complementary characteristics
- Scenarios
  - concrete examples
  - often narrative and procedural in nature
  - **leave intended properties implicit**
- Goals
  - abstract
  - declarative
  - **make intended properties explicit**
- Scenarios and goals complement each other
  - Scenarios validate and help uncover and clarify goals
  - Standard CAM techniques of concrete snapshots and scenarios

Assumptions and Facts

- An **Assumption** is a goal assumed to be satisfied in the domain
- A **Fact** is something known to be true in the problem domain
- Assumptions and facts help understand goals and refinement
- They should be described only to the extent they are helpful to understand goals, goal refinement, and obstacles

Goal Refinement

- A goal may be met by some combination of (smaller) goals
- **Goal:** Maintain Confidentiality of Data
  - **Goal:** Avoid Classified Data Flowing Inappropriately
  - **Goal:** ….
- This is called “goal refinement”
- Goal refinement requires corresponding information model refinement
  - What is classified data? Data: classification
  - What is Inappropriate Flow? Component: clearance
  - No flow of data to component with lower clearance than data classification

What is a “Good” Goal Refinement?

- Formally, a set of goals {G1, ..., Gn} refines a goal G given some domain facts if the refinement is:

  1. Complete: The sub-goals (with domain facts) do, in fact, meet the goal **G**
  2. Minimal: All the sub-goals are necessary to meet the goal **G**
  3. Consistent: The sub-goals are consistent with each other and with the domain

[Advanced] Some Goal Refinement Patterns

- If a role or process lacks adequate information access to meet a goal
  - Add information access
  - Split lack of direct access by introducing intermediate information
    - Introduce intermediate tracking information + some accuracy goal
    - Achieve goals: split into sequence with intermediate state, sub-goals access intermediate state
    - Maintain goals: split into sub-goals that maintain the intermediate information
    - Split by cases and resolve each case separately
    - Replace inaccessible state S by accessible events S_true, S_false
- If a role or process lacks adequate information control to meet a goal
  - Add information control
  - Split lack of direct control by introducing intermediate information
    - Introduce new info to control goal + actuation goal
    - Similar variations for Achieve, Maintain, etc. goals
- If an (ideal) goal involves knowing about the future
  - Prediction pattern: sub-goals to predict future, act based on prediction, learn from past
- Goal-refinement patterns rationalize the structure of the goal model

Outline

- What is the Motivation for “Goals”?
- Goals Modeling and Refinement
- **From Goals to Responsibilities, Processes and Actions**
- Obstacles and Conflicts
- Soft Goals
- “List” and “Draft” versions of goals models

Responsibilities

- Some object(s) can be made **responsible** for a goal
- Those objects must collaborate (some process) to meet that goal
- Refine goals until “realizable” by process or roles (at chosen level of granularity)
- **Goal:** *balance the portfolio to maintain expected risk/return ratio while the market changes*
- A goal requires
  - to observe some things (e.g. market)
  - to control some thing (e.g. balance of portfolio)
  - to meet certain conditions (e.g. maintain risk/return ratio within targets)
- A goal is “realizable” by process or roles **if** it can control and observe as required by the goal
  - There will be lower-level collaboration even to “observe” what needs to be observed
  - Just utilize use-case goals, pre/post conditions, as appropriate at lower levels

From Goals to Business Processes

- Goals uncover the basic information model of the domain and add desired constraints to it
- CAM case/case-coordination pattern provides a basis for top level process partitioning
- Goals help further define the processes
  - Identify what activities are needed in the processes
  - Properly specify those activities e.g. this activity
    - **trigger:** must be performed whenever: …
    - **prohibit:** must not be performed unless: …
    - **post:** must result in: …

Deriving Specifications of Actions from Goals

- e.g. for a goal: **Maintain** [while condition then state]
  - Op 1 to establish state **must** be performed whenever condition becomes true
  - Any Op 2 that exits state **must not** happen unless condition was false
- e.g. for a goal: **Achieve** [when condition then state change]
  - Op 1 to establish state change **must** be performed whenever condition becomes true
- Hence processes must observe the rules required to meet such goals
- Finer analysis possible by making finer-grained distinction of goal types
  - state invariants vs. transition invariants
  - invariant that hold over all system states, or only after or between some conditions
- A goal influences many actions; an action is influenced by many goals
  - Formally one might distinguish domain facts from goal-derived pre/post
- One can make reasoned argument about meeting goals

Outline

- What is the Motivation for “Goals”?
- Goals Modeling and Refinement
- From Goals to Responsibilities, Processes, and Actions
- **Obstacles and Conflicts**
- Soft Goals
- “List” and “Draft” versions of goals models

Obstacle Definition

- An obstacle is a high level “Exception”
  - It defines a set of undesirable behaviors of a system
  - Obstacles are so because they can prevent goals from being met
- **O** is an **obstacle** to a goal **G** if
  - **O** is sufficient (combined with domain facts) to prevent **G** from being met
- Obstacle modeling addresses “early-phase” goals
  - over-idealization and initial simplistic assumptions
  - lack of anticipation of exceptions or possible modes of failure
  - incorrect assumptions about humans or machines, …
- Set of obstacles **O1, O2, … On** is **complete** with respect to goal **G** if
  - … whenever all the obstacles in the set are avoided
  - … based on the domain properties
  - … you are **guaranteed** that goal **G** is met

How to find Obstacles

- Find the “Pain Points” e.g. from VOC interviews
- Ask “Why is this a problem” to find obstacle and goal description

- Negate goal or some part (usually only the lower-level goals)
- Refine the negation (for obstacles considered likely or high risk)
- Obstacle O to goal G
  - frequently refined into sub-obstacle O1 to sub-goal G1

- Focus on obstacles to leaf goals assignable to agents
- Focus on obstacles to high-priority, security, and safety goals; in general, some domain-specific cost-benefit analysis is needed
- Quit obstacle analysis when leaving unresolved is acceptable

Obstacles and Resolution

t Obstacles prevent goals from being met

t Obstacles have to be addressed in some way

- Eliminate obstacle: e.g.
  - devise alternative refinement of higher-level goal
  - add new goal to specifically avoid the obstacle
- Weaken or de-idealize the goal
  - Simply change from too ideal to more realistic goal definition
- Obstacle reduction e.g.
  - human incentives
  - self-correcting designs
- Obstacle mitigation e.g.
  - allow obstacle to occur and introduce a new goal to mitigate its effect

Conflicts between Goals

t Start with ideal goals

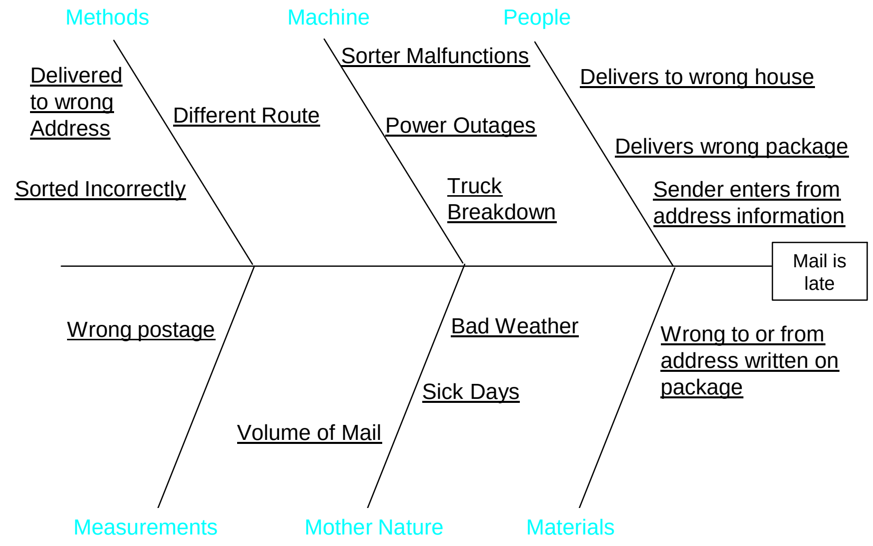
t Some goals may conflict under some “boundary conditions”

t Goals that share parts of the information model can conflict

t Mark important conflicts on the model, annotated with conditions

t Resolve conflicts e.g. by prioritizing one goal over the other, or introducing new goals to resolve the conflict

Cause and Effects Diagram à Goals Model Equivalent



Methods

Machine

People

Delivered to wrong Address

Different Route

Sorted Incorrectly

Sorter Malfunctions

Power Outages

Truck Breakdown

Delivers to wrong house

Delivers wrong package

Sender enters from address information

Mail is late

Wrong postage

Volume of Mail

Bad Weather

Sick Days

Wrong to or from address written on package

Measurements

Mother Nature

Materials

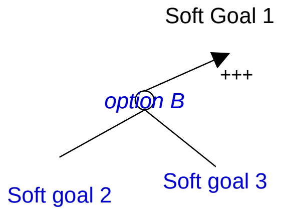
- Goal = on time mail
  - goal refinement = right address, delivery, power, weather, …
- Obstacle = late mail
  - obstacle refinement = goals and obstacles, sub-obstacles obstructing sub-goals

Outline

- What is the Motivation for “Goals”?
- Goals Modeling and Refinement
- From Goals to Responsibilities, Processes and Actions
- Obstacles and Conflicts
- **Soft Goals**
- “List” and “Draft” versions of goals models

Soft Goals and Alternative Designs

- Soft goals are …
  - Somewhat “fuzzy” goals that do not have clear-cut satisfaction criterion
  - e.g. Optimization goals e.g. Min, Max goals minimize or maximize some property
- **Soft goals can help guide choice between alternative solution approaches and architectural choices**
- Min/Max Goals naming
  - Min <Property to minimize>
  - Max <Property to maximize>
- Can add qualitative annotation to goal structure graph
  - +++, ++, + (degree of support for goal)
  - ---, --, - (degree of conflict with goal)
  - etc.



Outline

- What is the Motivation for “Goals”?
- Goals Modeling and Refinement
- From Goals to Responsibilities, Processes and Actions
- Obstacles and Conflicts
- Soft Goals
- **“List” and “Draft” versions of goals models**

Goals – summary tabular version

The element named	Described as	[Optional] Is the responsibility of [some may be marked Out of Scope]	[Optional] Conflicts, obstructs, is obstructed by, or resolves	[Optional] Variability	[Optional] Stakeholders [optionally with stakeholder view of goal]	[Optional] Priority
Goal <name>	<description>			<how variable or negotiable is this item>		
Goal <name>	<description>					
Obstacle <name>			<which goals does it obstruct>			
Assumption <name>						
Fact <name>						
Gap <name>						

Goals – detailed tabular version

Goal ID	**Goal**		**Date**
B1.0	Maintain optimal project portfolio at program level		1/2/03
Description Always maintain an optimal mix of top-ranked fundable programs in the project portfolio, where ranking is determined by (planned, actual, and forecast) business value, costs, risks, and inter-program dependencies, all derived from project-level intra-program information.			
Sub-Goals <Goal 1> <Goal 2> If needed, an explanation of how sub-goals combine.			
Obstacles <Obstacle 1> <Obstacle 2>			
Responsibility of Role / Process	**Stakeholder**	**Owner**	
Variability	**Parent Goal(s)**		**Priority**

Information Model – detailed tabular version

Order	Data associated with an instruction from an investor, to either buy or sell an instrument according to certain parameters conveyed with the instruction.	
	Aliases: None.	
	Supertypes: None.	
buyOrSell	{buy, sell}	Indicator to show whether the order is to buy or sell the instrument. Rule: <any rule or constraint applicable to this attribute, or combinations of attributes>
quantity	Integer	Quantity of the instrument to buy or sell.
price	Currency	Overall price for the order, as figured, and eventually approved, by the Trader.
instrument	Instrument	The instrument to be bought or sold.
trades	Set of Trades	All trades that fulfill the Order

Chapter 4
Process Modeling

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

It covers:

t Business Behavior Model

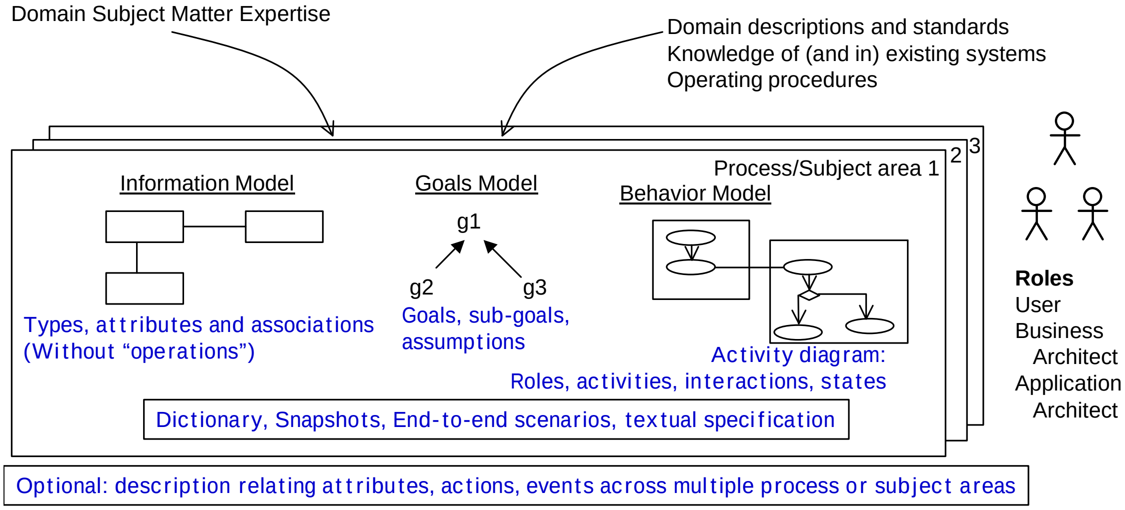
– Activity diagrams

– State diagrams

t Supporting Business Information Model

t [Optional] Defining a reference business process architecture

Preview of Business Architecture



Domain Subject Matter Expertise

Domain descriptions and standards
Knowledge of (and in) existing systems
Operating procedures

Process/Subject area 1 2 3

Information Model

Types, attributes and associations (Without “operations”)

Goals Model

g1 g2 g3

Goals, sub-goals, assumptions

Behavior Model

Activity diagram: Roles, activities, interactions, states

Dictionary, Snapshots, End-to-end scenarios, textual specification

Optional: description relating attributes, actions, events across multiple process or subject areas

Roles
User
Business Architect
Application Architect

Why do a Business Architecture?

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

Steps to Business Architecture

Reference: CAM PDF, page 9; links to steps and techniques. More being added e.g. goal models.

- **Goal model:** Define stakeholders, goals, explore **why** and **how** of goals for completeness
- **Scenarios:** Find outer boundary of the problem to define end-to-end scenarios
  - As far out in “space” and in “time” to include the real goal that must be met
  - Use these to validate business goals and process models
- [Optional] **As-Is model:** If appropriate, start with **As-Is Process**
  - Define as-is processes based on current practice
  - Identify stakeholder roles and their goals and CTQs (Critical-to-Quality attributes) for the process
    - Zoom out of detailed activities to larger grained ones and to more black-box views
    - What are post-conditions? What matters to Stakeholders? What are CTQ (Critical to Quality) attributes (goals)? How are the goals structured?
  - **Analyze** the as-is model, particularly with respect to CTQs
    - Identify failures, overlaps, redundancies, non value-adds, etc. and their causes
    - Inadequate resource co-ordination across multiple “cases”, business rule failures, process defects
  - Define **Essential** (“reference”) **Process** architecture (explained later)
- **To-Be model:** Brainstorm the **To-Be Process** using the above goals and as-is models
  - If you design multiple processes, also include an overview of how they interrelate
  - Define Information Model as needed to support activities in process and goals model
- **Why?**
  - Use goals and understanding of current-state to design future-state business process

Example of Boundary, Stakeholders, CTQs

- 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* **à** *complete visit* **à** *make appointment*
- Who are all the stakeholders, how to group them, and what are their goals and CTQs?
  - *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* - Want 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

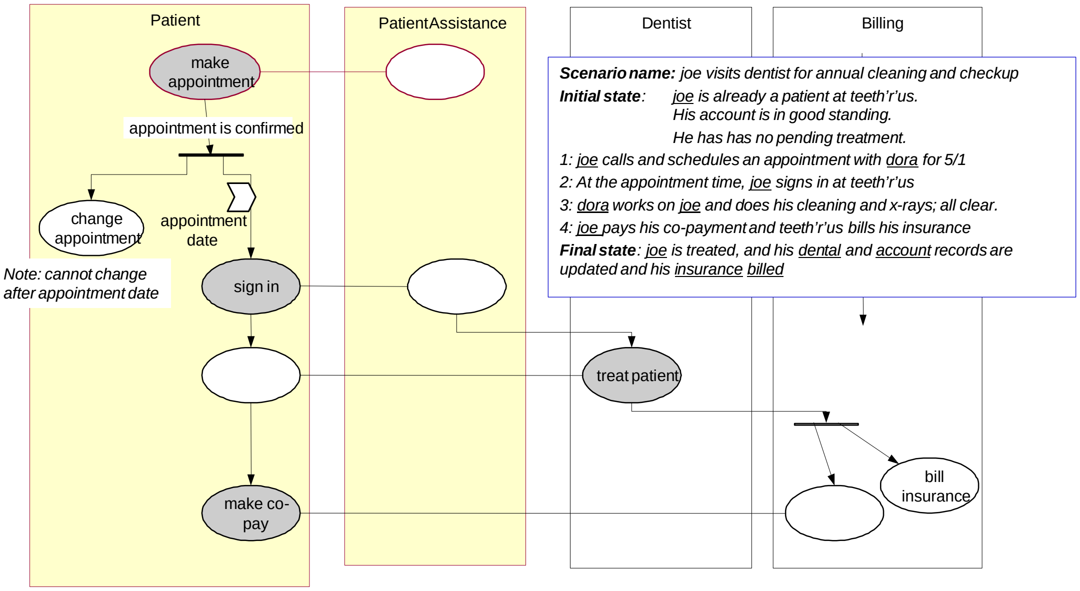
Business Behavior Model – Activity Diagrams

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

Activity diagram Generalizes End-to-End Scenarios

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



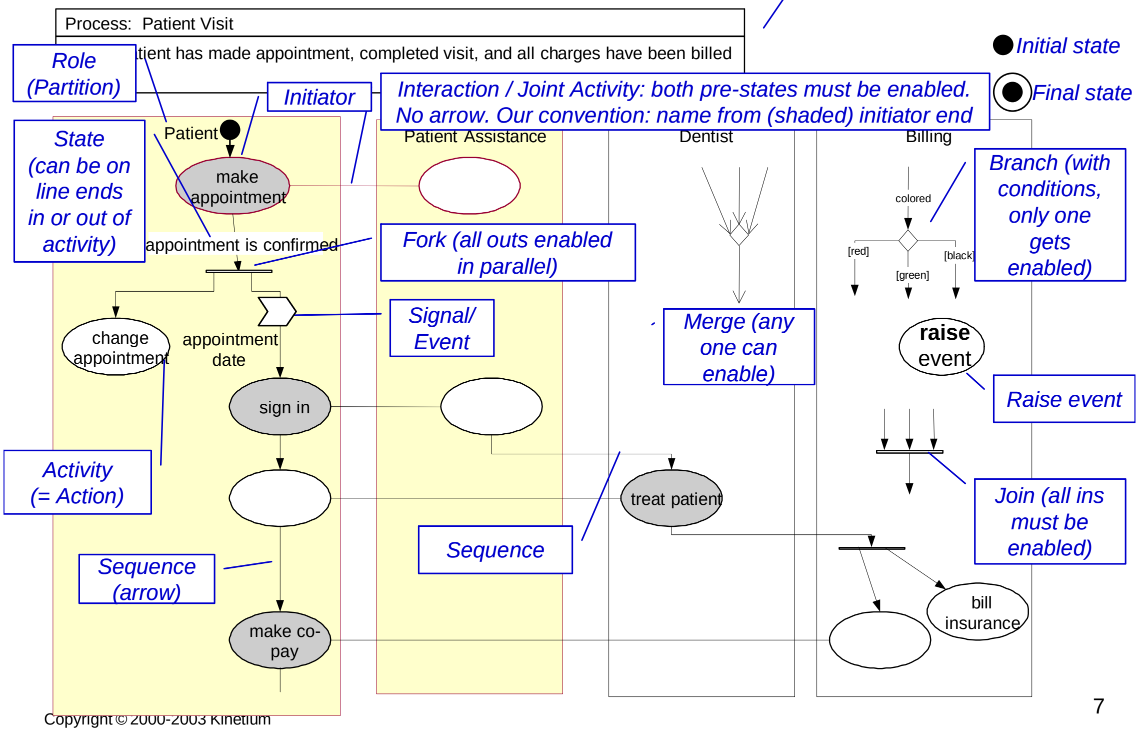
Scenario name: *joe visits dentist for annual cleaning and checkup*

***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



Process name and goal

Initial state

Final state

Role (Partition)

Initiator

Interaction / Joint Activity: both pre-states must be enabled. No arrow. Our convention: name from (shaded) initiator end

State (can be on line ends in or out of activity)

Branch (with conditions, only one gets enabled)

Fork (all outs enabled in parallel)

Signal/ Event

Merge (any one can enable)

Raise event

Activity (= Action)

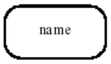
Join (all ins must be enabled)

Sequence

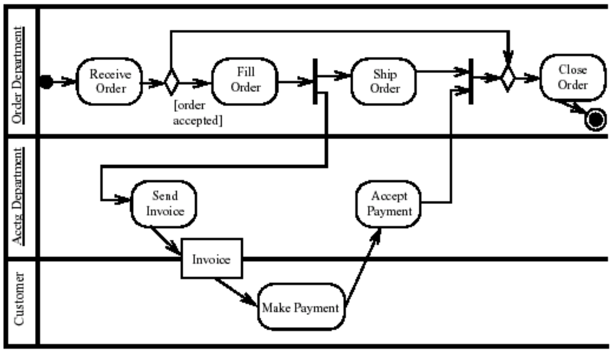
Sequence (arrow)

Preview of some UML 2.0 Differences

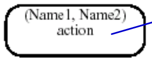
- UML 2.0 uses “Rounded Rectangle” for action and activity



- UML 2.0 “Partition” can indicate role, or other grouping of activities

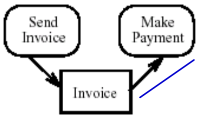


- UML 2.0 does **not** allow Interactions / Joint Activity on graphical “Partitions”, but text-annotation version allows “partition” list per activity.



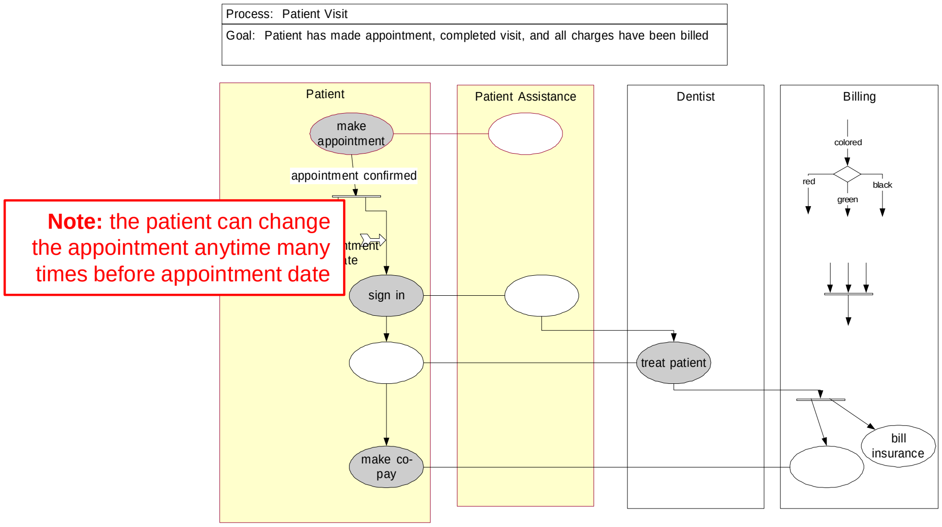
This action is contained within 2 partitions e.g. jointly performed by 2 roles

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



Object flow

Activity Diagram Pragmatics

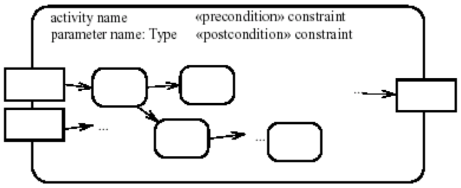


Note: the patient can change the appointment anytime many times before appointment date

- Formalizing concurrency, iteration, conditionals… can make diagram detailed
- Can simply add **annotations** to the activity diagram explaining these cases
  - 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

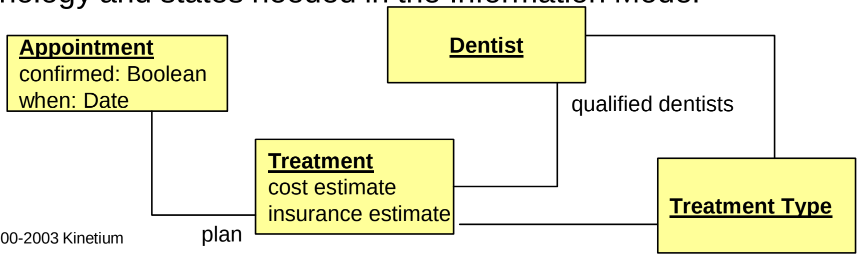
UML 2.0 composite activity with pre/post + explicit in/out parameters using object flows



- Actions specified textually

activity: make appointment
roles: Patient, Patient Assistance
inputs: symptoms: Set of Symptom [from Patient]
outputs: appointment details: Appointment [to Patient]
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



Appointment
confirmed: Boolean
when: Date

Dentist

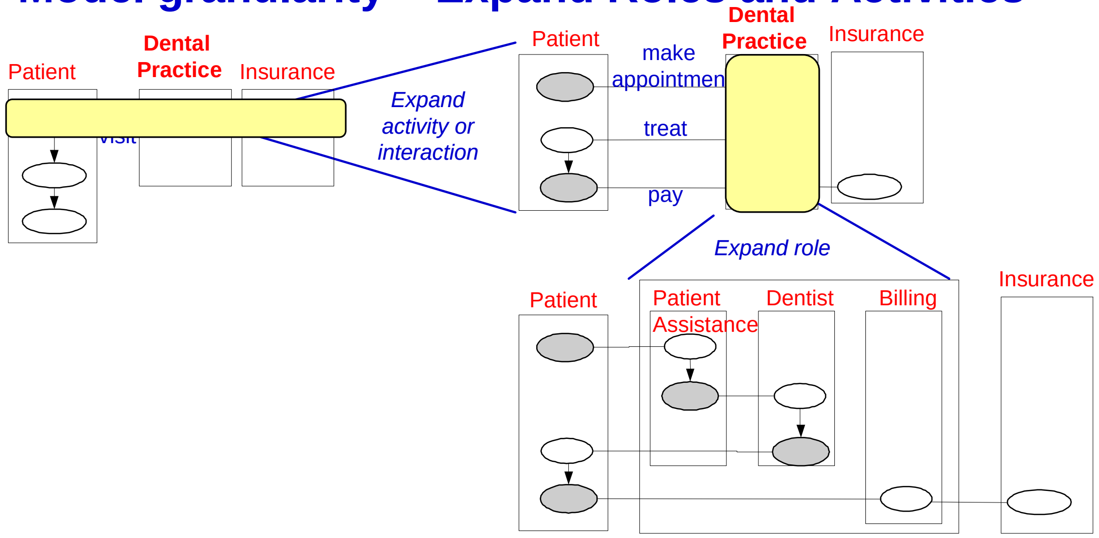
qualified dentists

Treatment
cost estimate
insurance estimate

Treatment Type

plan

Model granularity – Expand Roles and Activities



- 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
- "**Black box**" and "**white box**" is equally applicable in business modeling
- **Start with largest grained activities and roles, be willing to zoom in and out**

Exercise 5.1

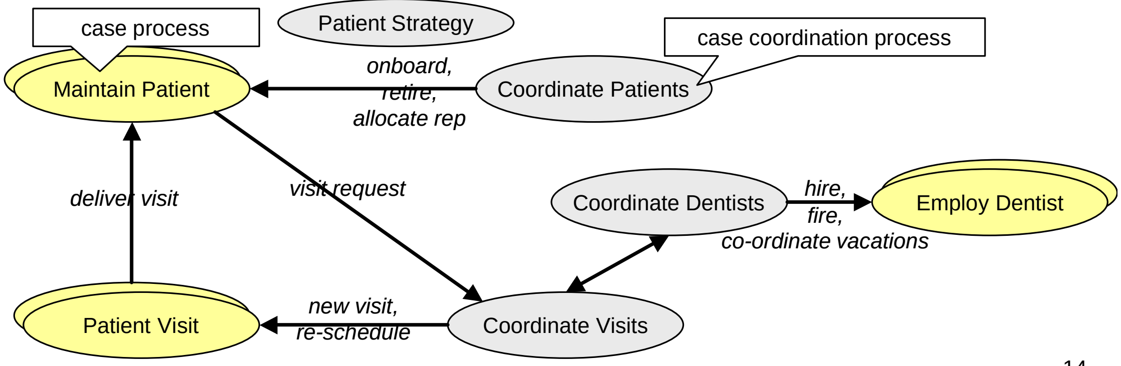
Essential Business Process Architecture

This section covers:

- Large domains have multiple interrelated business processes
- Case and Case Coordination gives one clear structure across processes

Essential Process Architecture with Example

- Find "essential" business entities: interesting lifecycles, you cannot imagine not having
  - "Essential" entities: Patient, Visit, Dentist, Payment
- Find "generates" relations to other entities spawned off, with their own lifecycles
- Define a "case" process: takes each instance through entire lifecycle e.g.make loan, order
- Define a "case coordination" process to start, stop, coordinate across multiple "cases"
- Define a "case strategy" process to evolve both case and case-coordination processes
  - Maintain Patient, Coordinate Patients
  - Patient Visit, Coordinate Visits
  - Employ Dentist, Coordinate Dentists
- Process architecture relates these together with summarized interactions



Business Behavior Model – Lifecycles

This section covers:

- Describing Object Lifecycle using a State Diagram and State Matrices

State Model Describes Lifecycle of Object

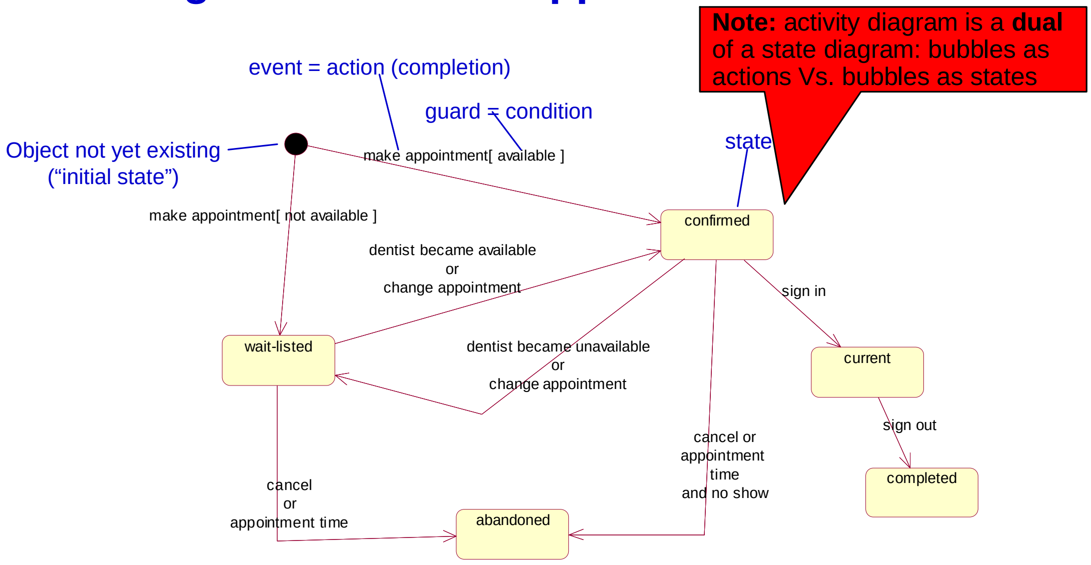
- Frequently, some objects in the domain have complex lifecycles
  - Insurance Company: a Claim
  - Pharmaceutical Company: a candidate Drug, a Clinical Trial
  - Financial Services: a Syndicated Loan
  - General Business: an Opportunity

- Question: Which of our activity diagrams represents such a “case” process?

- Such objects progress through an interesting state progression
  - Insurance Claim: filed, appraised, contested, in court, settled
  - Pharmaceutical Drug: in lab trials, in clinical trials, in sample production, FDA approved
  - Trade: modeled, approved, executed, booked

- **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 a **dual** of a state diagram: bubbles as actions Vs. bubbles as states

t States: confirmed, wait-listed, current, abandoned, completed

t Events: make appointment, change appointment, sign in, …

State Definition Table and State-Event Table

t Each state is equivalent to a Boolean attribute (or some “status” attribute)

t Each state should be derived from other attributes and associations

t A simple table (matrix) of states vs. other attributes and associations can be helpful

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

t Each transition event should correspond, directly or indirectly, to some action

t All events should be considered in all states

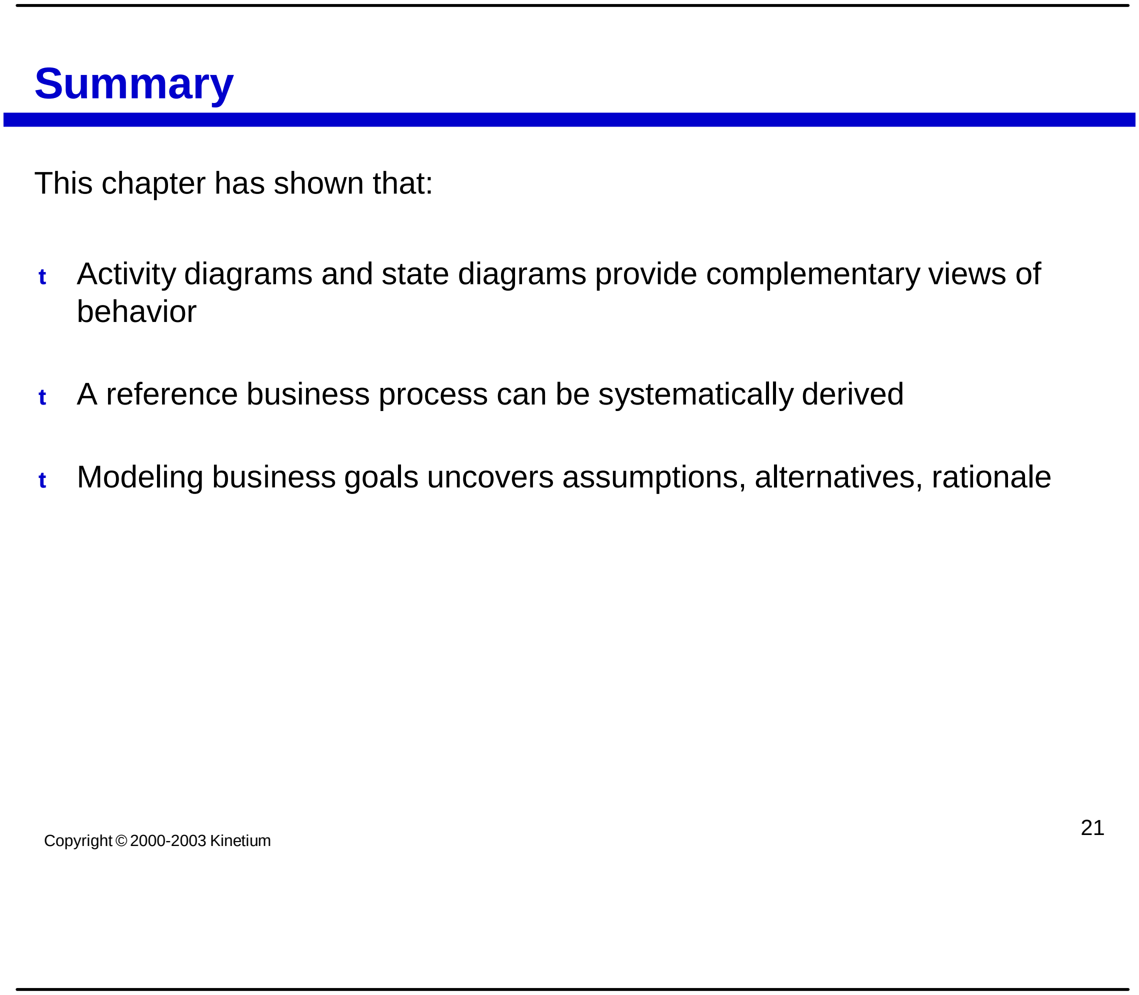
t A simple table of states vs. possible events helps find missing cases

State à **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 5.2

Business Architecture – Some Guidelines

- Focus on end goal and state change for every process or activity
- Distinguish black-box from white-box views of processes
- Support process behaviors and goals with information model
- Zoom in and out of processes, activities, and goals
- [Optional] Define reference process architecture from object lifecycles



Summary

This chapter has shown that:

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

Chapter 5
Current State Systems

This chapter discusses how to describe “current-state” systems in the context of a road-map or migration-related project.

It covers:

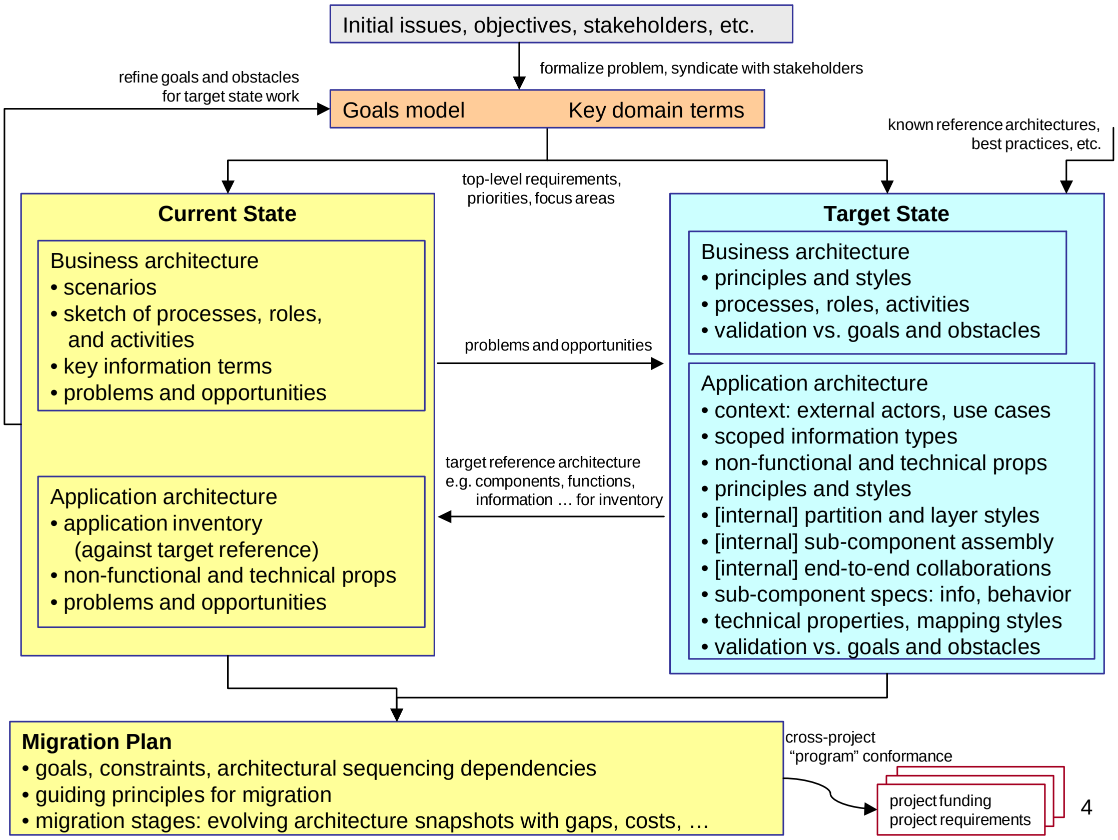
- Elements of a Road-Map
- Describing Current State Systems

Outline

- **What is a Roadmap Project?**
- Describing Current State Systems

Architecture Roadmap Projects

- Roadmap projects have specific characteristics
  - Involves serious long-term re-engineering to both business and technology
  - Produces rationalized target architecture and migration plan
  - Has significant business and technical drivers
  - Acts as basis of investment decisions for further build / buy / integrate / consolidate projects
  - Stakeholders and business processes are often cross-business
- Examples
  - Revamp risk management in response to recent disasters
    - major credit risk information and exposure capture and reporting challenges
    - existing infrastructure “unacceptable” and “needs improvement”
    - failure to capture adequate basis for accurate risk information
    - mixed inconsistent definitions of fundamental credit terms
    - multiple disjointed sources of data
    - reliance on manual processes
    - disparate heterogeneous environments make reconciliation painful
- Method demands of Roadmap projects led to “CAM For Roadmaps” route



Initial issues, objectives, stakeholders, etc.

formalize problem, syndicate with stakeholders

refine goals and obstacles for target state work

Goals model Key domain terms

known reference architectures, best practices, etc.

top-level requirements, priorities, focus areas

Current State

Business architecture
• scenarios
• sketch of processes, roles, and activities
• key information terms
• problems and opportunities

Application architecture
• application inventory (against target reference)
• non-functional and technical props
• problems and opportunities

problems and opportunities

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

Target State

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

Application architecture
• context: external actors, use cases
• scoped information types
• non-functional and technical props
• principles and styles
• [internal] partition and layer styles
• [internal] sub-component assembly
• [internal] end-to-end collaborations
• sub-component specs: info, behavior
• technical properties, mapping styles
• validation vs. goals and obstacles

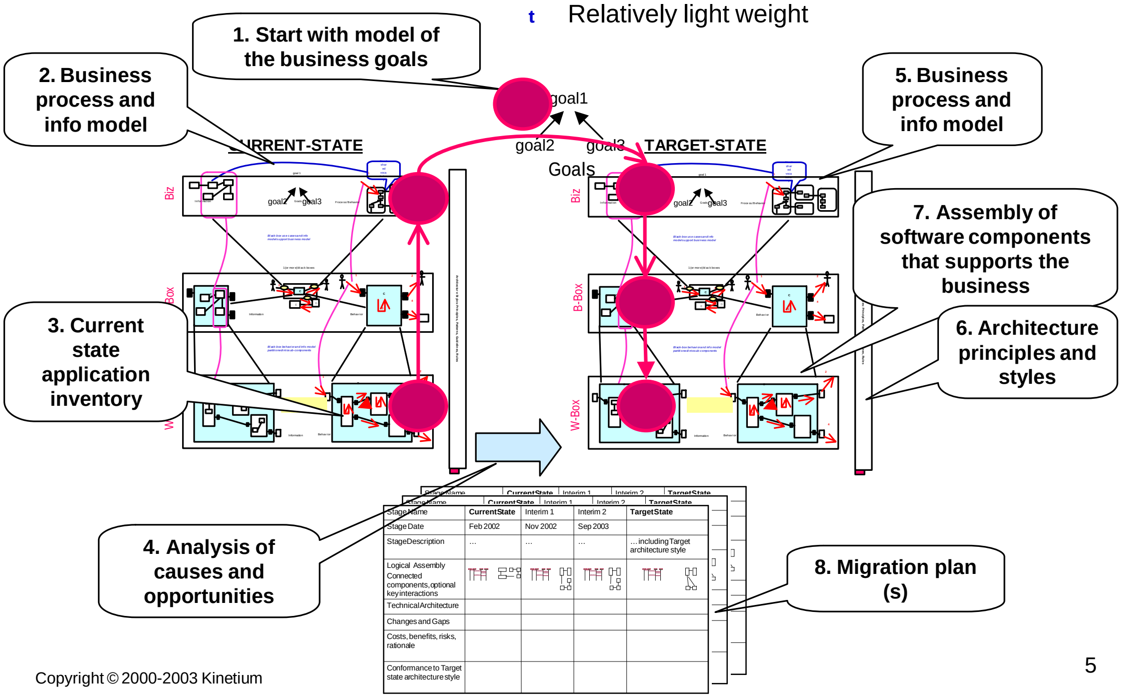
Migration Plan
• goals, constraints, architectural sequencing dependencies
• guiding principles for migration
• migration stages: evolving architecture snapshots with gaps, costs, …

cross-project “program” conformance

project funding
project requirements

Roadmap in CAM

- Not too strict about usage of UML
- Makes minimal assumptions about tools
- Relatively light weight



1. Start with model of the business goals

2. Business process and info model

3. Current state application inventory

4. Analysis of causes and opportunities

5. Business process and info model

6. Architecture principles and styles

7. Assembly of software components that supports the business

8. Migration plan (s)

Stage Name	**CurrentState**	Interim 1	Interim 2	**TargetState**
Stage Date	Feb 2002	Nov 2002	Sep 2003	
StageDescription	…	…	…	… including Target architecture style
Logical Assembly Connected components, optional key interactions				
Technical Architecture				
Changes and Gaps				
Costs, benefits, risks, rationale				
Conformance to Target state architecture style				

Roadmap Table of Contents

1. **Overview**
2. **Business Goals and Terminology**
   - 2.1 Business Goals
   - 2.2 Business Information Model
3. **Current State**
   - 3.1 Business Architecture
     - *3.1.1 [Optional] Architecture Principles and Styles*
     - *3.1.6 Problem and Opportunity Analysis*
   - 3.2 Application Architecture – White Box View of <System>
     - *3.2.1 Application Inventory*
     - *3.2.4 Problem and Opportunity Analysis*
4. **Target State**
   - 4.1 Business Architecture
     - *4.1.1 Architecture Principles and Styles*
     - *4.1.2 High Level Structure of Business Processes*
     - *4.1.4 Business Information Model*
   - 4.2 Application Architecture – Black Box View of <System>
     - *4.2.1 Black Box Context*
     - *4.2.2 Architecture Principles and Styles*
   - 4.3 Application Architecture – White Box View of <System>
     - *4.3.1 Architecture Principles and Styles*
     - *4.3.2 Sub-Component Assembly*
     - *4.3.3 Sub-Component Collaborations*
     - *4.3.4 Sub-Component Information Model*
     - *4.3.5 Sub-Component Behavior Spec for <Sub Component Name>*
     - *4.3.6 [Optional] Sub-Component Technical Mappings*
5. **Migration Plan**
   - 5.1 Migration Goals, Constraints, and Guiding Principles
   - 5.3 Migration Overview
   - 5.2 Stage: Interim 1
   - 5.3 Stage: Interim 2

Outline

- What is a Roadmap Project?
- **Describing Current State Systems**

Why Bother with Current State?

- More “real” understanding of problems and opportunities
- Sketchy current-state idea can be helpful (or not!) for target-state
- Required input for migration planning

Current State System Description

- Each application
  - Owns (performs) these types of actions [or groups]
  - Owns these information [types, attributes, or groups]
  - Responds to (uses) these types of actions or events [or groups]
  - Needs (uses) this information [types, attributes, or groups]
  - Is available from [elsewhere in firm, vendor, etc.]
  - Depends on these technical components or platforms

- A sketch of a CAM “white-box” assembly model is helpful
  - Logical connectivity of applications
  - Technical characteristics of connectors

- Problems and opportunities with current-state