

Software Engineering

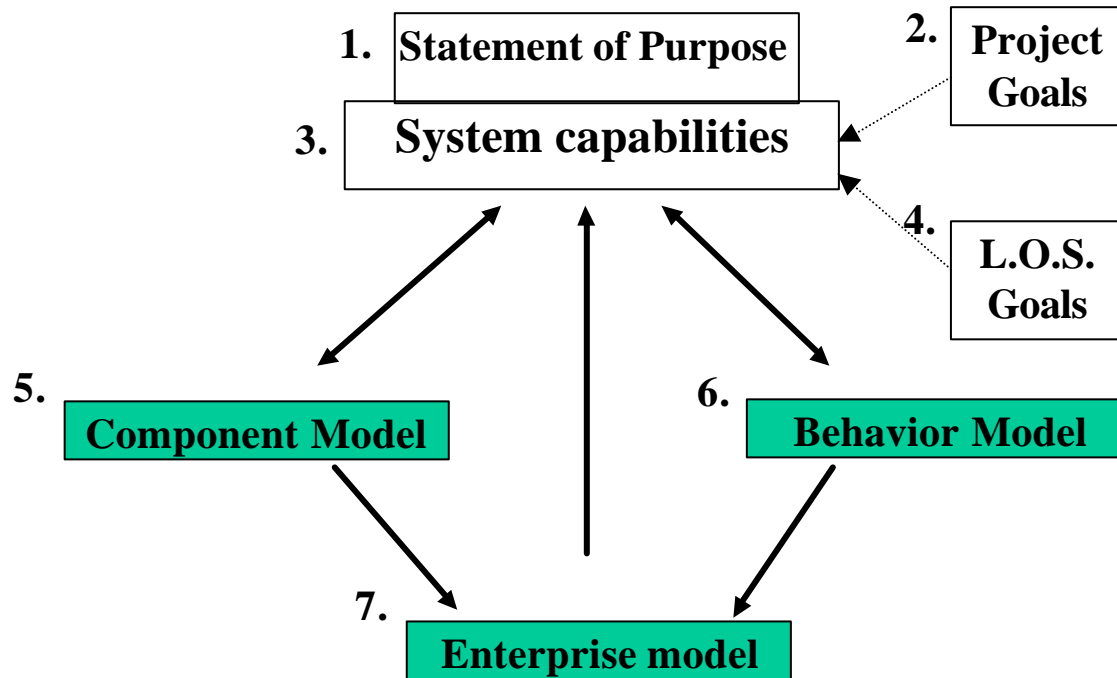
SSAD III: Behavior, Enterprise Models

CS577a

Fall 2000

Brief Review

Analysis Deliverables



OCD 4.0

SSAD 2.0

Components vs. Objects

- **Objects** are the smallest (most refined) entity we consider in our models prior to implementation
- **Components** are compositions (membership relationships) of objects with a high degree of *cohesion* within the domain
- Components are what you need to describe the system to domain experts at a higher level of abstraction (less detail)

Objects

- The Design phase may decompose components into objects.
- Objects are used to represent the system in software.
- An object is a specialization of a component.
- An object is an atomic unit for systems analysis purposes.

Design Preview:

Java Composites and Components

- Design Composites:
 - collection: Vectors
 - aggregation: HashTable
 - grouping: Array
- Java Components:
 - “Component”
 - Canvas
 - Window

End of Brief Review

Behavior Model

Examples:

(Archive Administrator Subsystem, Archive User Subsystem)

2.2 Behavior Model

- Start with the list of System capabilities (OCD 4.3)
 - Refine => sub-capabilities => behaviors
 - avoid system operations (data ops, eg. Event Notification)
 - Label system policies (<policy>), algorithms (<algorithm>), and events (<event>)
- Use Behavior Specification Template
- Must reference: System capabilities, Project Goals, or L.O.S. Goals

Behavior Modeling

- What is it?
 - Details what the specific desired system activities are within the domain; What we would do if we didn't know components.
 - Behaviors are as important as components. Behaviors and components together form a complete Enterprise model (our goal).
 - Task analysis, independent of component model, keep it separate. Ideal: do both simultaneously.
- How is it done?
 - refinements of system capabilities through iteration to create outline view

Deriving Behaviors From capabilities

System Responsibility 1

System Sub-Responsibility 1

System Behavior 1 <event>

System Sub-Responsibility 2 <policy>

System Sub-Sub-Responsibility 1 <policy>

System Behavior 1 <algorithm>

System Behavior 2 <event>

...

System Responsibility 2

System Sub-Responsibility 1

System Behavior 1

System Sub-Responsibility 2

System Sub-Sub-Responsibility 1

System Behavior 1 <event>

System Behavior 2

Example #1 Behavior Summary

2.3.1. Archive administrator subsystem

The capabilities of the administrator subsystem are [OCD 3.3] [SSRD 2.3]:

SR - 1 Maintain items

BH-1 Operator adds an item <event>

BH-2 Manager deletes archive item

BH-3 Operator modifies archive item

BH-4 Manager verifies archive item updates <event >

BH-5 Manager prints item summary report

SR - 2 Maintain types

BH-6 Manager adds item type

BH-7 Manager deletes item type <event>

BH-8 Manager modifies item type

SR - 3 Maintain item field options

BH-9 Manager adds a field option

BH-10 Manager deletes a field option <event>

BH-11 Manager modifies a field option

SR - 4 Maintain related links

...

Global Outlining (cont.)

- The productive questions (last two) are useful
 - “What do you need to do this?”
 - “What’s involved in this?”
- Avoid system operations (i.e., behaviors that operate directly on a piece of data or supply data, such as an Event Notification)

Outlining Syntax

- Label with
 - <policy> for system policies
 - choices that reflect an organizations workflow, mandates, or constraints
 - <algorithm> for respective algorithms that carry out a policy - detailed in design w/ ops
 - <event> for system events - a notable occurrence in the system - detailed w/ ops
 - important actions
 - notifications

Boundaries of Control

- Important to keep in mind the scope of the system when modeling events and operations
- "Handle new entries" is made up of many independent operations such as "accept name", "accept address", etc. Each of these operations requires action from outside the boundary of control since the user must intervene in each step
- Any responsibility that crosses a boundary of control should not be modeled as a single Behavior (especially for user-driven systems where the order of behaviors is not determined by the software)
- Balances and creates natural system behavior partitions

Example: Boundaries of Control

1. Sales transaction

Payment

Cash

Credit card

Validate charge

----- (boundary of control) -----

Check

Validate check

----- (boundary of control) -----

2. Validate charge

Acquire credit card number

Dial authorization service

...

Behavior Specification

Behavior Specification Template:

Trigger -

Preconditions -

Postconditions -

Inputs (with constraints and dependencies) -

Outputs -

Exceptions -

Use Case Diagram and/or Scenario-

Relates to - Reference corresponding System Responsibility (OCD 3.3)

Type: {<event>, <policy>, <algorithm>}

Example #1 Behavior Scenario 2

Scenario SC-02

<i>Name</i>	Maintain item type
<i>Trigger</i>	Administrator issues command – maintain item type
<i>Preconditions</i>	Administrator logged in as manager
<i>Postconditions</i>	Changes in item types are reflected when archive is updated.
<i>Inputs</i>	Choice of operation: ADD, MODIFY, DELETE, QUIT For ADD – Type name, Type description For MODIFY – Type name, type modification information For DELETE –Type name
<i>Outputs</i>	Type information in the archive
<i>Exceptions</i>	Type Exists, Type Issued
<i>Relates to</i>	RQ-02, RQ-03, RQ-04 of SSRD 2.3.1.1

Example #1 Behavior Scenario 3

Scenario SC-03

<i>Name</i>	Modify item field options
<i>Trigger</i>	Administrator issues command – maintain item field options
<i>Preconditions</i>	Administrator logged in as manager.
<i>Postconditions</i>	Changes in field options are reflected when archive is updated.
<i>Inputs</i>	Choice of operation: ADD, MODIFY, DELETE, QUIT For ADD – Field name, Option For MODIFY – Field name, Option modification information For DELETE –Field name, Option
<i>Exceptions</i>	No Such Option
<i>Relates to</i>	RQ-08 of SSRD 2.3.1.1

Example #1 Behavior Model

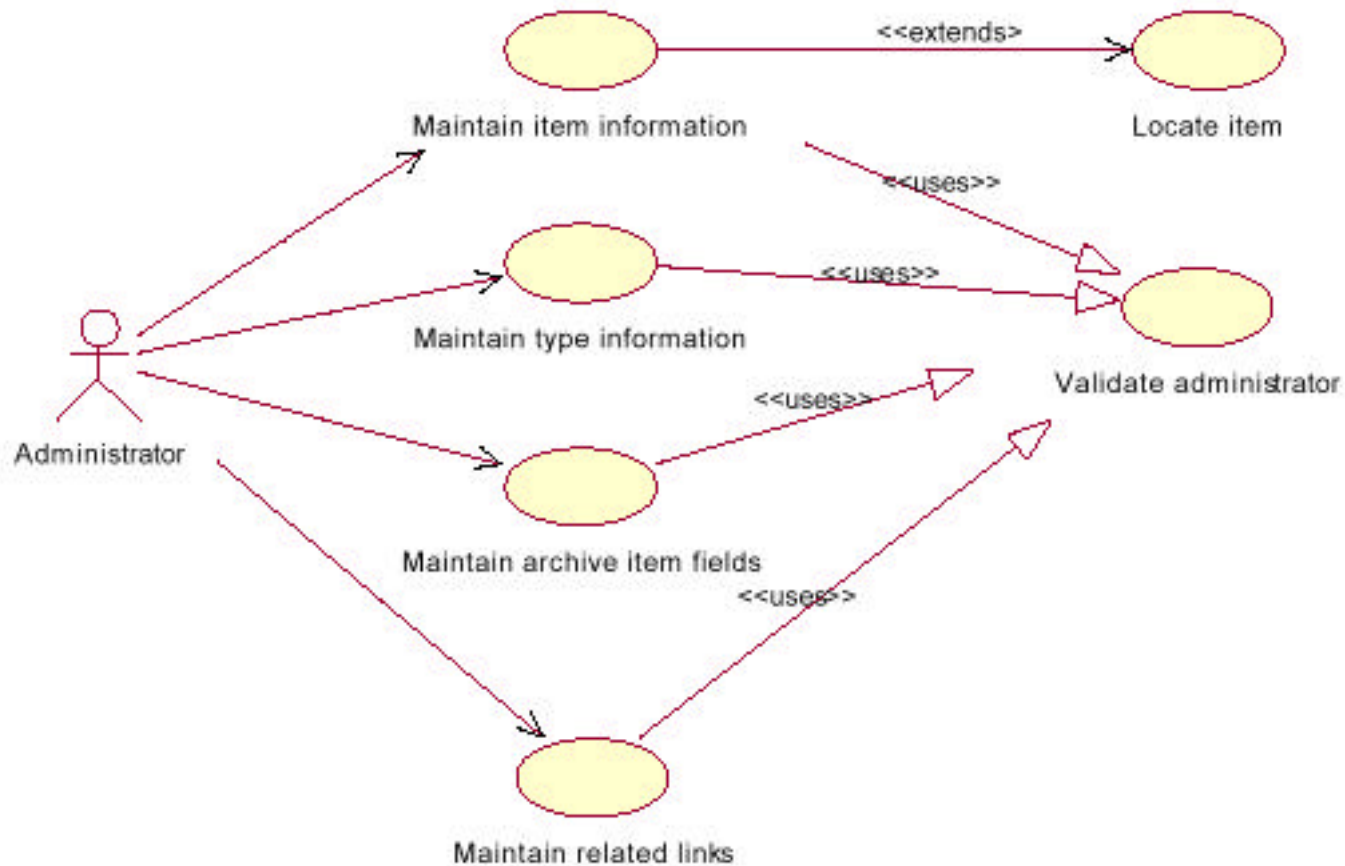


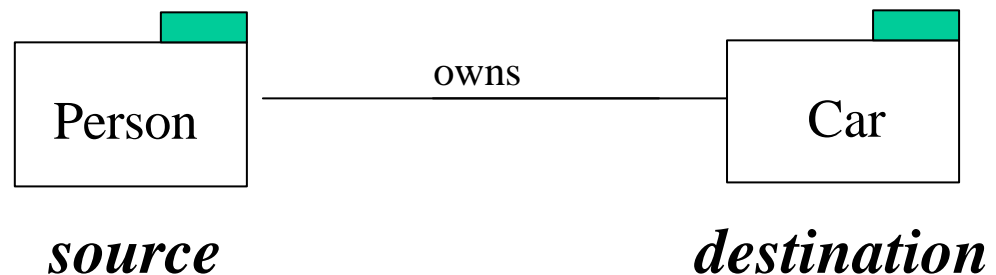
Figure 12 Behavior model – Administrator sub-system

System Analysis

Component Relationships

Simple Component Relationships

- What are they?
- Attributes vs. Relationships
- UML notation (may vary for other abstractions)



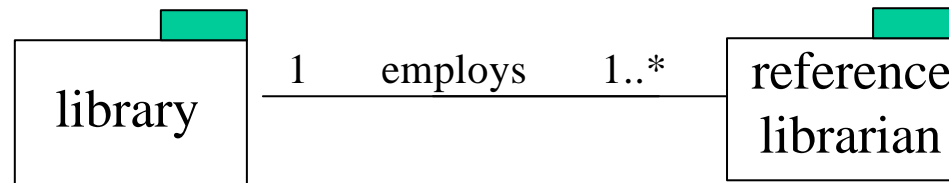
- The source is always the owner of a unidirectional relationship

Relationship Multiplicity (cardinality)

- to-one
 - a single instance at the destination
 - an implicit constraint, always challenge to-one's!
- to-many
 - relationship is to a group of components
 - role name of relationship is for the group
 - e.g. “Company employs many employees”
- to-n
 - relationship role names
- many-many
 - can't have many-many in unidirectional relationships
 - bi-directional relationships can accommodate

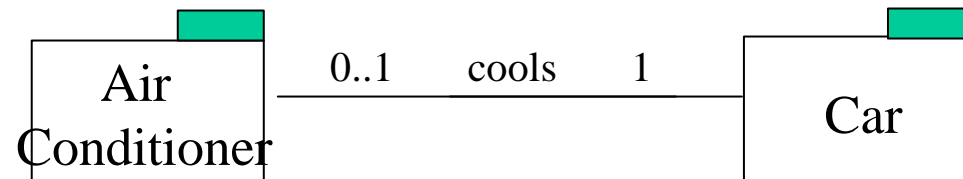
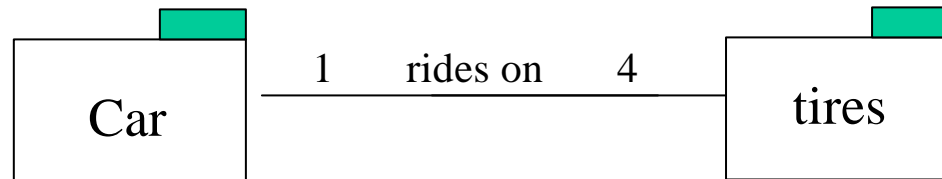
UML Notation

- to-1: ...1



- to-many: ...*

- to-n: ...n



Identifying Relationships

- static conditions
- avoid actions
- avoid comparative statements

Relationship Qualities

- Similar to attributes except destination resolves

Relationship Inspector

Defining L.O.S. -

Name -

Accessibility -

*{readable, settable, modifiable,
fixed}*

Scope - *{shared, unique}*

Constraints -

Required:

Initial Value:

Cardinality:

Dependencies:

Derived From:

Relational Attributes:

Role Names -

RoleGroupName 1: {..., ..., ...}

RoleGroupName 2: {..., ..., ...}

...

Dependencies

- Are constraints
 - once a constraint appears, it carries through
 - effect faithfulness directly
- Are not formally relationships, but similar
- Any model element can have dependencies
 - common: attribute, relationship, component

Dependencies (continued)

- how one element of the system uses another
 - Ex. "area" attribute of "circle"
- Dependencies can effect any element of a system, including constraints.
 - Ex. in a date the upper limit on "day" depends on the value of "month"
- Some dependencies occur at the object level.
 - Ex. a window is dependent on its delegate for custom closing behavior
- Most components level dependencies are described by a relationship between two components in which case it is important to document how the dependent components uses another components

Semantic Dependencies

- Semantic Dependencies: describe how a L.O.S. of an abstraction is resolved under a given set of conditions
 - Ex. "vacation days" has a semantic dependency on "start date"
- When a component uses an element in your model to make a decision as to how it should behave, it is a semantic dependency
- If the state of a component depends upon an element either within the component itself or within a component that it is related to, it is a semantic dependency
 - Ex. account change of state from “Solvent” to “Overdrawn” has a semantic dependency on the account balance

Functional Dependencies

- Functional Dependencies describe how components use other components to provide or assist in providing behavior
 - Ex. In handling options in securities trading, can have a security component that knows its own price. The option depends on the security components to handle price-setting
- Functional dependencies provide behavior the components could not exhibit on their own (the difference between functional, semantic and attribute dependencies)
- Functional dependencies are often implemented using delegation or forwarding; (delegation by itself does not necessarily imply functional dependency, it often just indicates semantic dependency)
- Other examples of functional dependencies are Client-server, Alarm response, CGI

Conceptual Dependencies

- Effects the defining qualities of the components
 - Ex. cars: may have a "generic" car components that depends on a separate "model" component
- Altering the dependency in some cases may change the component from one type to another

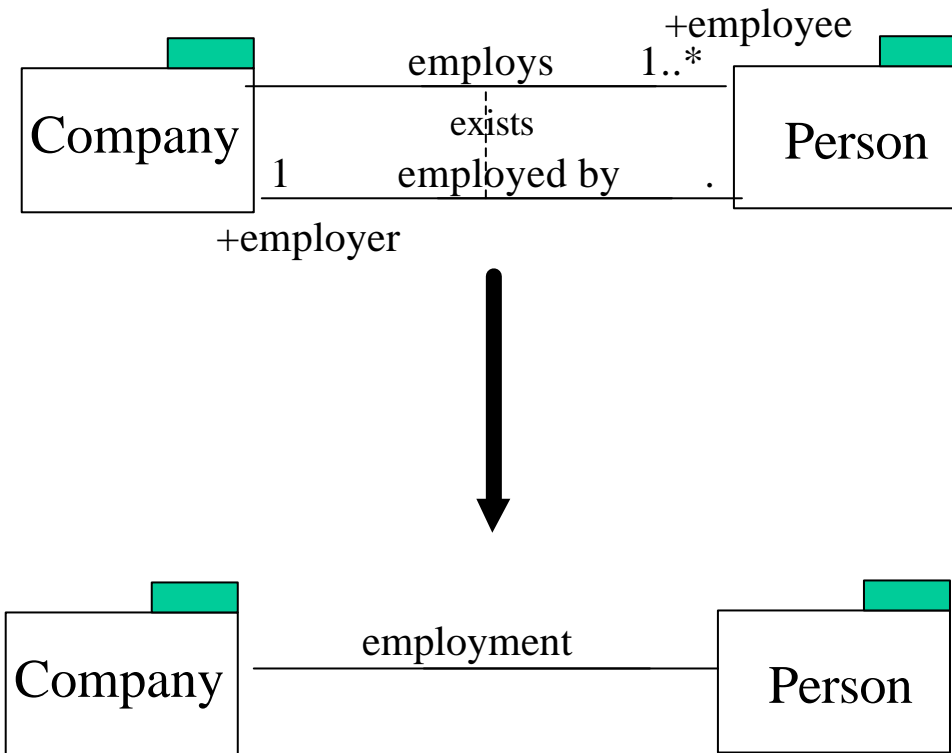
Documenting Constraints and Dependencies

- Not truly abstractions, but have strong effect on the resolutions of other elements
- Often left out of documentation, forgotten, or incorrectly specified or assumed to be so well known they don't require comment
- **Warning:** omission of constraints and dependencies in analysis often has dire effects later (reduced faithfulness)

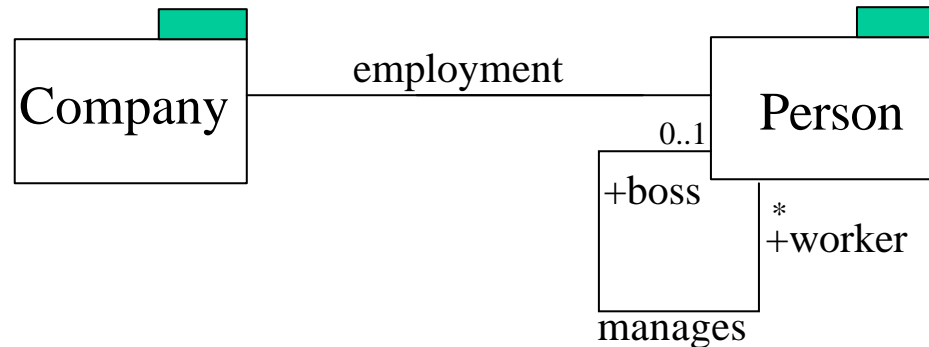
Complex component relationships

- Bi-directional Relationships
- Symmetrical Relationships
- Relational Attributes
- Relational Components

Bi-directional Relationships

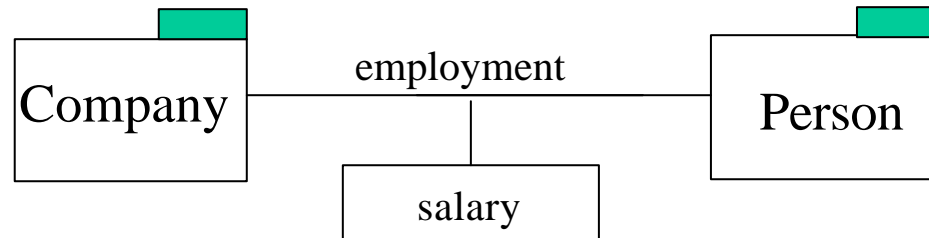


Reflexive Relationships

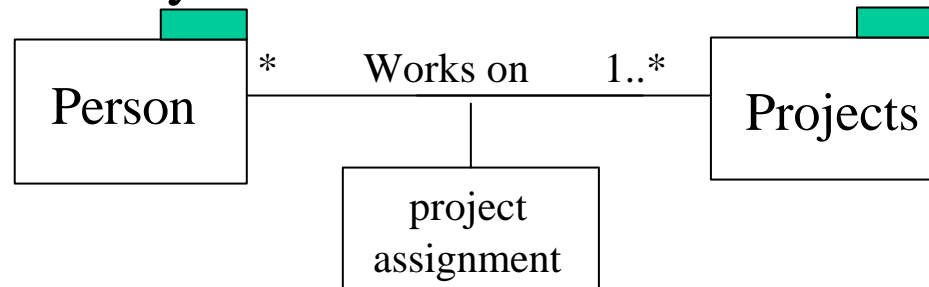


- A *symmetric* relationship: source and destination have the same role
 - “next to”, “friend of”
- Reflexivity is a constraint
 - source and destination can not be the same instance
 - most relationships are not reflexive
- Be careful of reflexive, transitive relationships!
 - A related to B related to C implies A related C

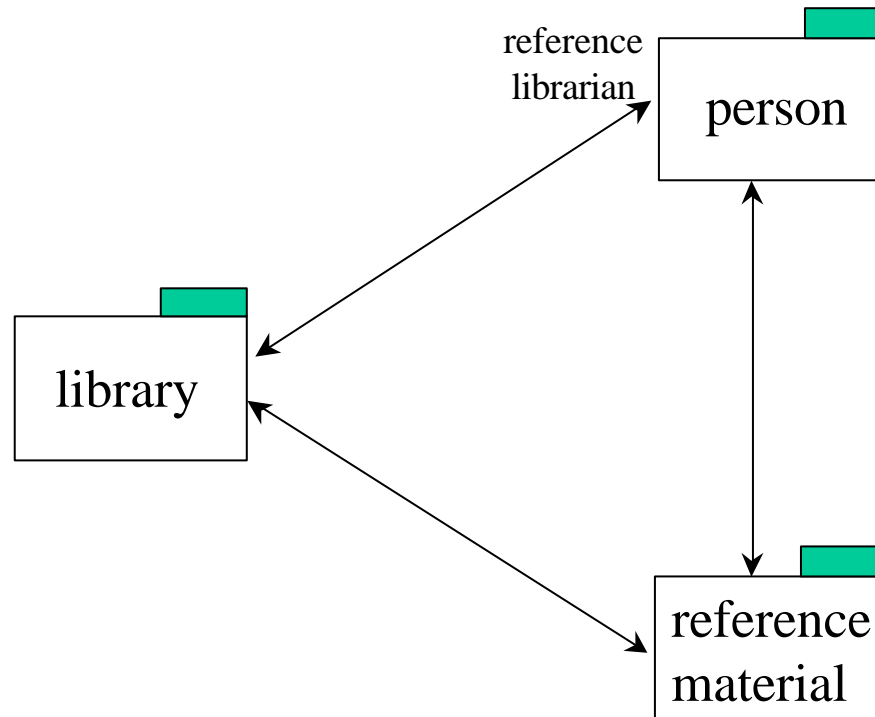
Relational Attributes



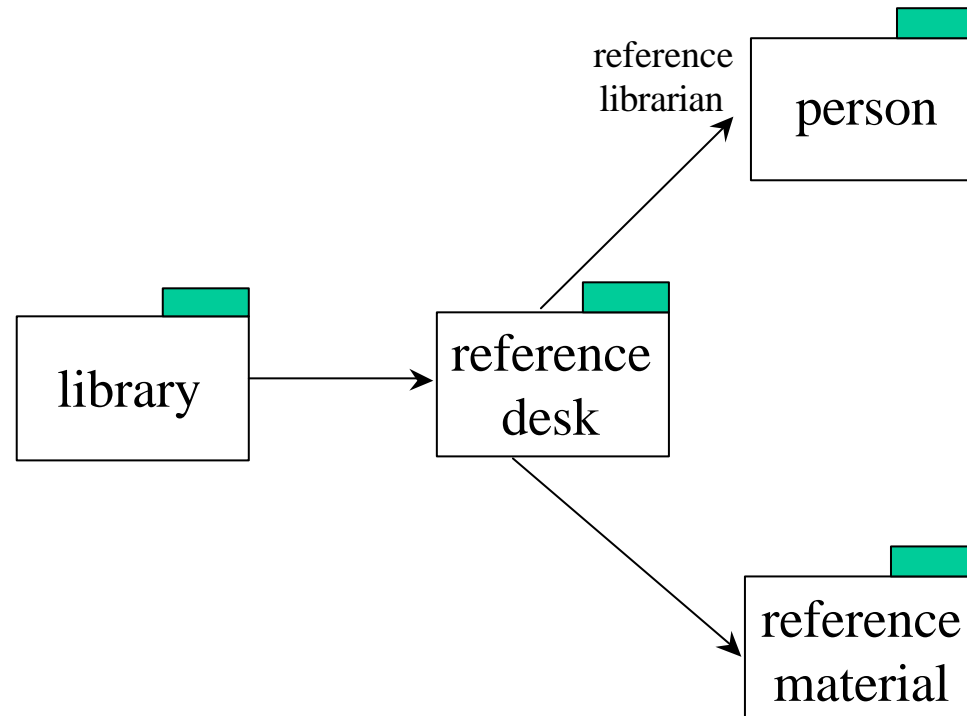
- Some relationships imply attributes that are not qualities of either the source or destination
- Will need to handle with a special component or object later
- *Selectors* are a special type of relational attribute that uniquely identify destination instances in a to-many



Multi-Way Relationships



Relational Components



About Components: Membership Relationships

- **Part-of Relationship**
 - relationship the objects within a component have to its container
 - A transitive relationship
 - Deletion of the components objects (the parts) triggers the deletion of the container
 - Ex. Car - remove engine, transmission, body, etc. you have no car
 - Visibility of component objects usually limited to the container
- **Contains Relationship**
 - the reverse of the part of relationship - from the container to the component objects
 - A transitive relationship
 - Deletion of the container (the component) causes the deletion of the components objects (the parts)

Composites

- Collections

- a “wrapper” for a number of contains relationships with identical objects
 - Ex. Address Book component contains a set of Address objects. Address Book is the composite component produced by applying the “collection” relationship

- Aggregation

- A contains relationship where the contained objects have a part-of relationship to the container
 - Ex. Automobile engine is an aggregate of pistons, crankshaft, spark plugs, etc. that are “part of” an engine. The engine “contains” the engine parts
 - What happens if you remove pistons? What about engine block?

- Groupings

- like aggregation, but if you delete all (not some) of the components objects the container is deleted as well.
 - Ex. Trades in a portfolio

Enterprise Modeling

Enterprise Modeling

- The goal is to get an overview of the overall structure of the system components and system behaviors
- How?
 - Through classification of components and behaviors
- Why?
 - So that choices on the "elegance" and faithfulness of the entities can be made

Abstraction Engineering

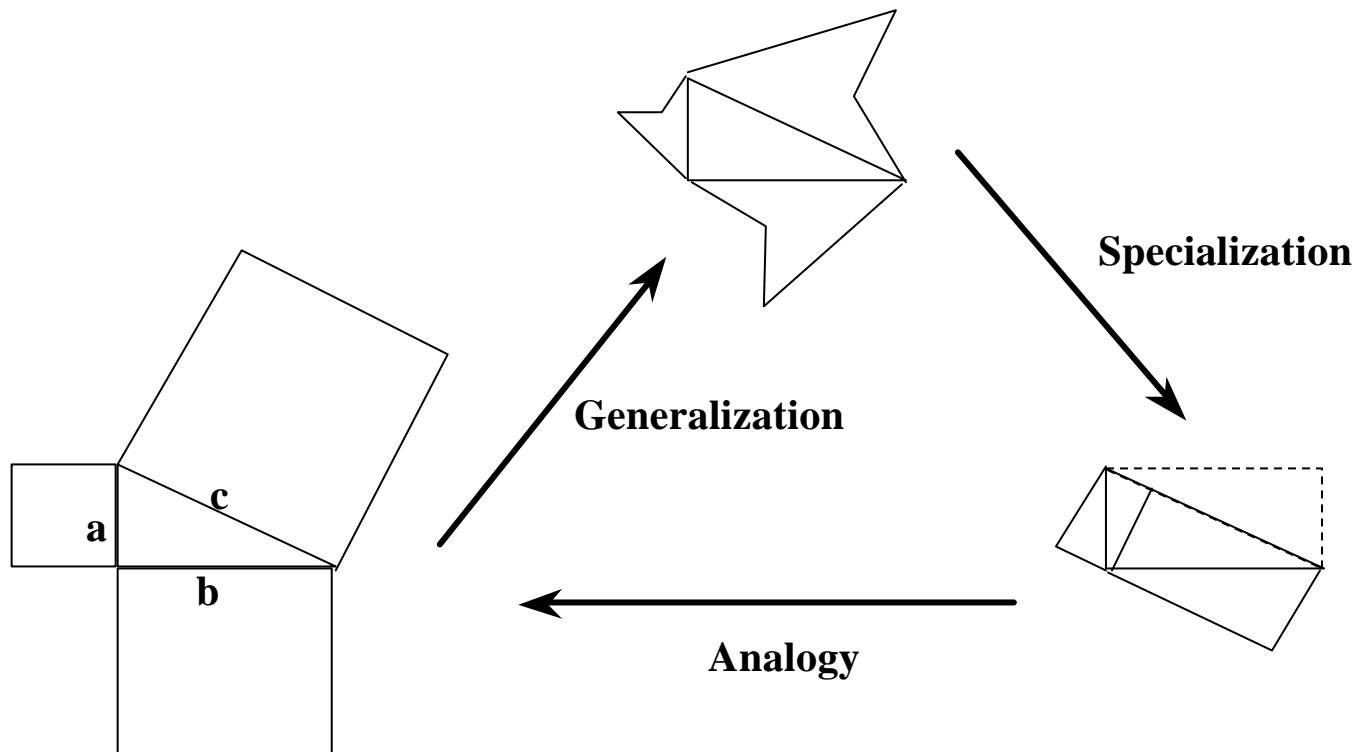
- Why necessary/useful
- Why powerful
- Can Be Powerful (But Subtle)

Abstraction Engineering

- Operations
 - Decomposition
 - Composition
 - Classification
 - Specialization
 - Generalization
 - Factoring

Abstraction Engineering

- Engineering operations can solve problems



Enterprise Modeling

- Classes and types
- Enterprise Classes and Analysis
- Why classify?

Classification

- Typing
- Specialization or Sub-typing
- Generalization or Supertyping

Component Classification

- Start building the Enterprise model by creating a class for each component in the Component model.
- Group the classes by:
 - *type-of* relationships
 - *kind-of* relationships
 - *part-of* relationships (optional)

Behavior Classification

- Behaviors are mapped to the components.
- Classify behaviors in an inheritance diagram, as this will make the assignment task easier
- Some behaviors will not map naturally to components and software level objects will need to be introduced later in the Design model to handle these.

Finding Component Classifications

- Requirements for Classification
 - type-of
 - kind-of
 - part-of

Equivalence

- attribute equivalence
- relational equivalence
- semantic equivalence
- roles
- operational equivalence

Discriminating Between Component Types

- constraints and dependencies
- comparators and discriminators
- comparators and main comparator
- discriminators and main discriminator

Example Enterprise Behaviors

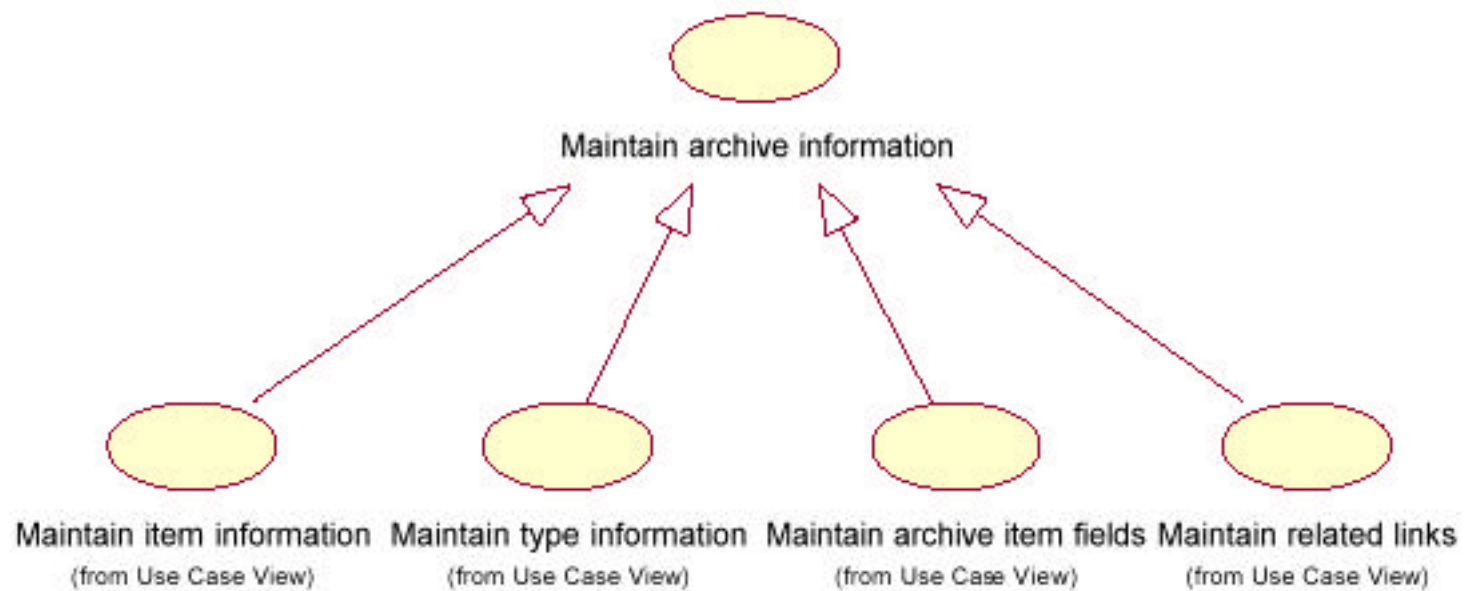


Figure 14 Enterprise model – Behaviors

Example Enterprise Components

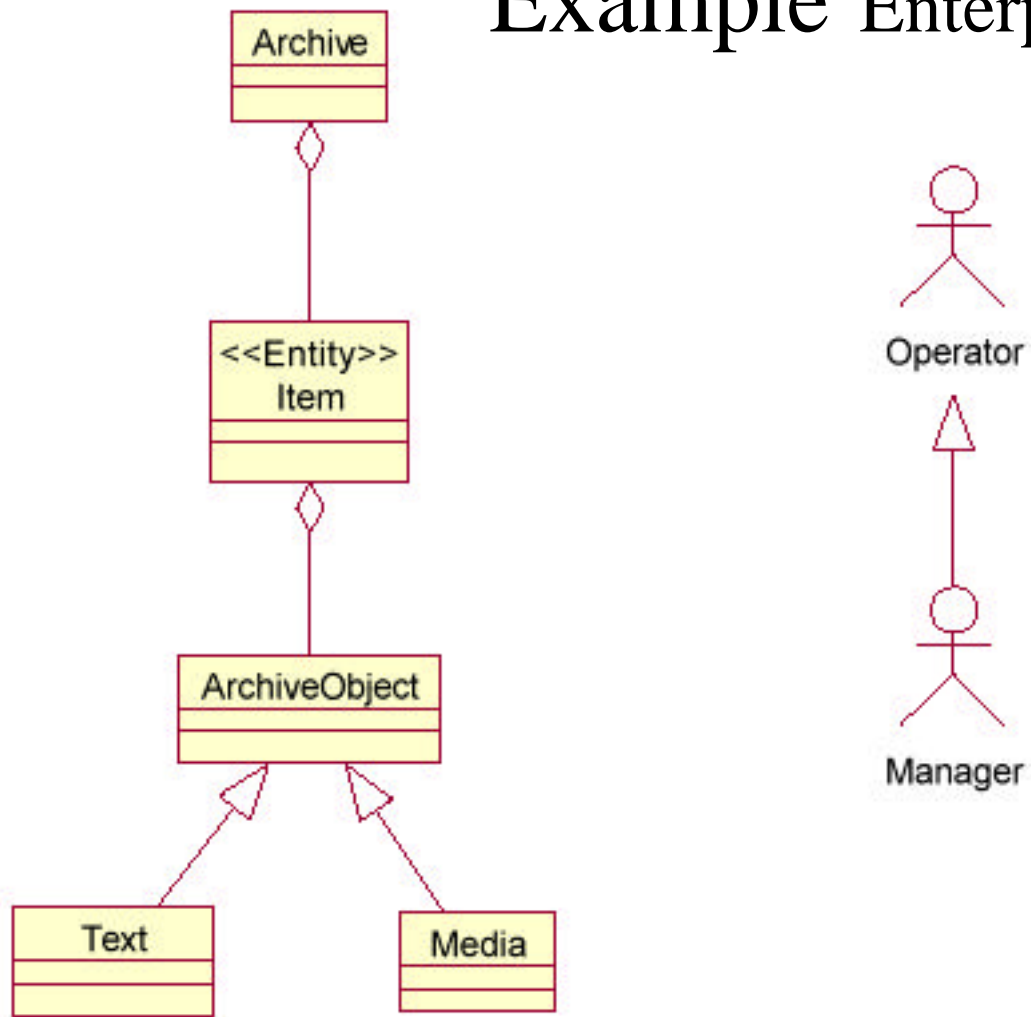


Figure 15 Enterprise model – components