System Analysis II: Component Modeling and the System and Software Architecture Description (SSAD)

CS577a

Fall 2000

SSAD Purpose in MBASE

Life Cycle Objective

- Top-level definition of at least one feasible architecture:
  - Physical and logical elements and relationships
  - precise descriptions of likely components, behaviors, objects, operations
  - Choices of COTS and reusable software elements
  - Detailed analysis, high-level Design.
- Identification of infeasible architecture options

Life Cycle Architecture

- Choice of architecture and elaboration by iteration
  - Physical and logical components, connectors, configurations, constraints
  - COTS and technology reuse choices
  - Architectural style choices, deployment considerations
  - Critical algorithms, Analysis issues resolved in Design
- Architecture evolution parameters

Overall Description

- Serves as “bridge” between concept and realization
- Goal is to describe a feasible Design that is faithful to the Analysis
- Intended audience
  - Domain expert for System Analysis
  - Implementers for System Design
- Participants
  - System Architect
  - Domain Experts
  - Implementers

Dependencies

- SSAD depends on OCD for:
  - Statement of Purpose
  - Project Goals
  - System capabilities
- SSAD depends on SSRD for:
  - System Requirements
  - Project Requirements

Dependencies (Continued)

- FRD depends on SSAD to make sure that:
  - Project Requirements
  - Capability Requirements
  - Interface Requirements
  - L.O.S. Requirements
  - Evolution Requirements

… are **A**chievable (for M.A.R.S.)

i.e. Why is the "how" is it going to be done feasable?

1.2 Standards and Conventions

- Describe:
  - Standards Used (DoD, IEEE)
  - Notation (UML) or adapted version
    - New symbols used
    - Stereotypes
  - Naming Conventions
    - Consistent use of names for elements
      - e.g. anObject, the_attribute, MyClass, theOperation()
      - e.g. nouns for Components, Objects, verbs for Behaviors, Operations

Brief Review

OCD 3.0 Proposed System (Analysis of)

- System Analysis involves several steps
  - Components - models, attributes, relationships, constraints, roles, and states
  - Behavior models
  - Engineering - Abstraction, enterprise class engineering
- This section is an overview of Analysis for OCD 3.0
- Details follow in later sections for SSAD 2.0

System Analysis

- The creation of precise, consistent description of a conceptual system in terms of its high-level components
- Description is within the organization domain, independent of implementation
- Analysis goes beyond simple checklists and pictures
- Analysis ties the domain description to the system design and implementation

Analysis Defined

- A separation of a whole into its component parts
- An examination of a complex system, its elements, and their relations
- A statement of such an analysis
- A method in philosophy of resolving complex expressions into simpler or more basic ones

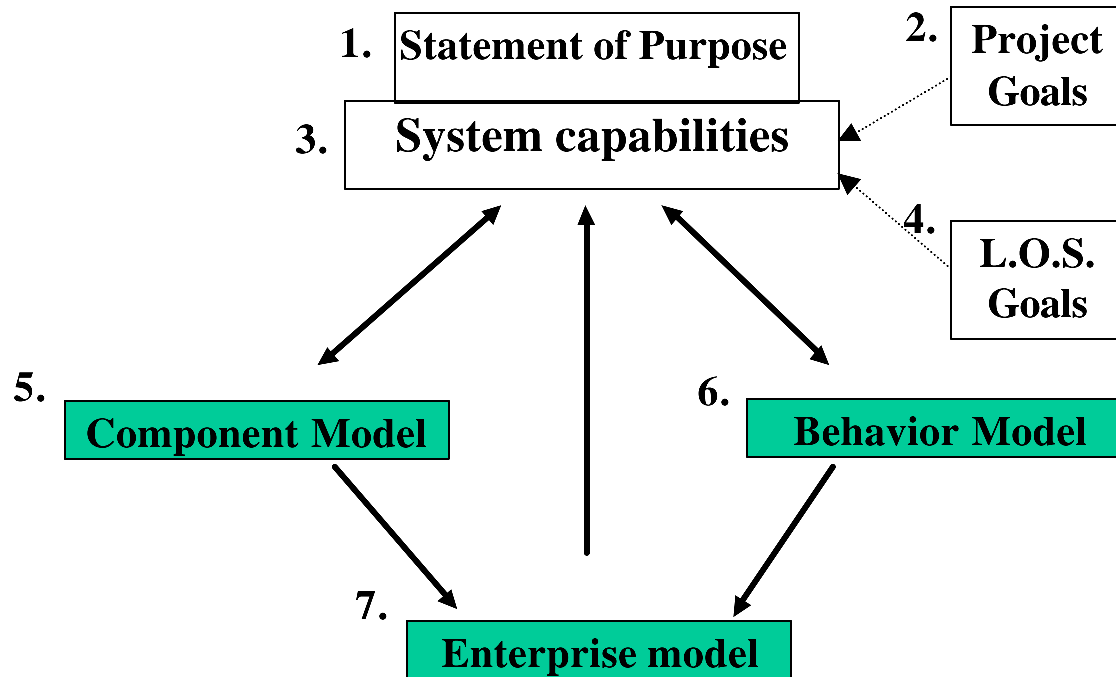
Analysis Goals

- Quantify *what* we want to represent, not *how* it is done
- Formalize and refine the specific parts of the organizations capabilities, entities, activities, and interactions described in the domain description that are to be automated
- Capture the high level architectural information that will represent (I.e. model) the conceptual system

Analysis Audience

- The Domain Description is for all constituents of the project
- Analysis is for Domain Experts - the high level leaders who understand the domain, know what they want, and have the authority to make decisions
- Not for implementers, who prefer design and implementation details (“hows”)

Analysis Deliverables



OCD 3.0

SSAD 2.0

Classifying Last

- Classification should be done after establishing Components and their behaviors
  - need things to classify before creating classes
  - intent is very important here
- This provides well named structures that will flow into the final implementation
- Accordingly, requirements should be delayed until later (design)

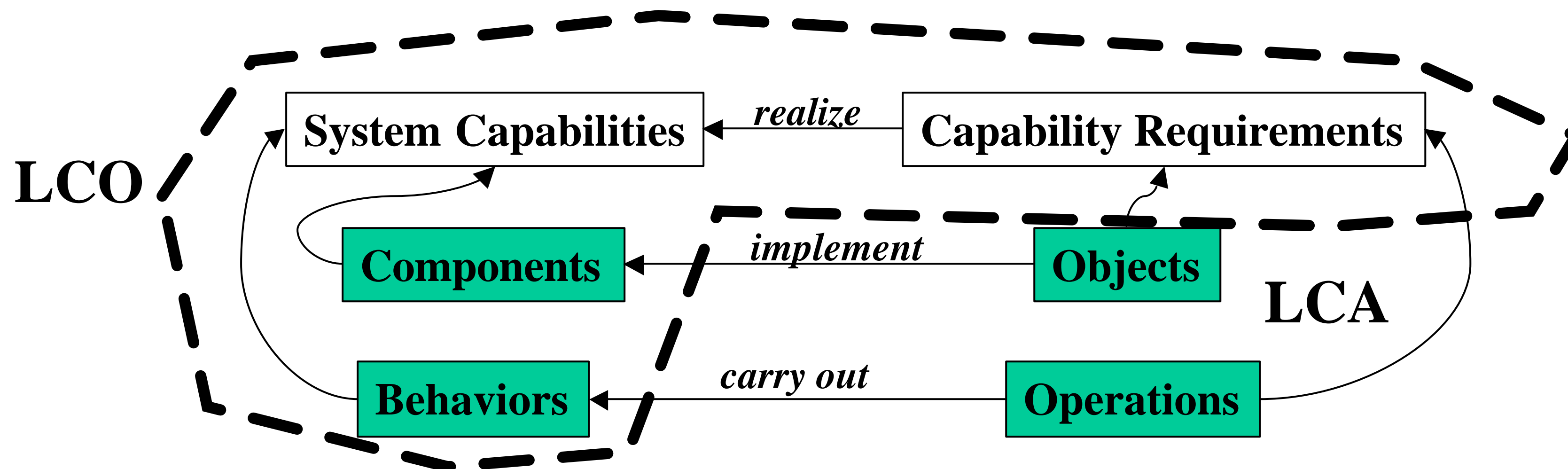
End of Brief Review

2. Architectural Analysis

- Main intent is to describe precisely "what" the domain experts vision of the system without implementation details
- As described via ISDM Component/Object Oriented Analysis (C/OOA)
  - Component, Behavior, Enterprise models
- Defines the “high-level” architecture of the system in a more precise way than the OCD

2.1 Component Model

- High-level architectural decomposition of the target system into functional partitions
- Should be consistent with System Capabilities(OCD 4.3)
  - Requirements in SSRD 3.0 will be based on what is mandated to be built in order to realize these capabilities
  - Object models in SSAD 3.0 will be derived from components in such as way as to realize the requirements in SSRD 3.0
- Use ISDM Component modeling techniques



Component Model

- A component model is
  - an intermediate step that reflects what is wanted and allows for trace-ability to what is implemented in design
  - at a higher level of abstraction, so is easier to work with (esp. with Domain Experts)
  - helps to manage complexity

Component Modeling

- Output is a collection of *diagrams* and *specifications*
- Provides details on important component qualities required for a *faithful* implementation
- Serves as the overall view of the system "parts" and their relationships (a system *partition*)

Component Modeling

- Why model components?
  - A language is needed to explain why a component exists or not, and to get a handle on the “things” which comprise the system
  - Systems are not often built entirely from scratch anymore, identification needed for integration
- How the components can or will be implemented is a design issue (will discuss in OOD)
- All components should be faithful as determined by Domain Experts (i.e. ask them!)

Component Modeling

- Component modeling
  - is not an exact science, but puts limits on the "emotional" approach, "it feels like a component"
  - Is not a set process, but a tool for descriptive purposes
  - does not involve design, just domain work: No user interface, no network protocols, etc.

2.1 Components and System Capabilities

- Components provide:
  - Views of the system's "things" whose interactions will carry out the system capabilities
  - An architectural breakdown of the system in terms of basic tangible entities that arise from the system capabilities

Building the SSAD 2.1 Component Model

- List possible components
  - Start with the Entity Model (OCD 4.5.2)
  - Look for "things" (e.g. nouns) in System capabilities (SC's) OCD 4.3
  - Search for less obvious possible components from SC's
  - Accept/reject possible components
  - Accepted components should have a Component Specification

Component Specification

Component 1:

Identity -

Defining Quality -

Name -

Attributes - Enumerate and use ***Attribute Specification*** template

Behaviors - Enumerate and use ***Behavior Specification*** template and/or UML Use Cases

Relationships - Use ***Relationship Specification*** template and/or UML Object/Relationship diagram

Roles - Describes how one component views another component through a relationship

a) role name, relationship or role diagram

b) role name, relationship or role diagram

Component Specification (continued)

State Groups - Use State transition diagram(s)

Constraints -

Dependencies -

Candidate Key - combination of attributes uniquely identifying a component or an object

Cardinality -

Others -

Relationship Specification

Relationship 1:

- Identity -
  - Defining Quality -
  - Name -
- Accessibility - *{readable, settable, modifiable, fixed}*
- Scope - *{shared, unique}*
- Constraints -
  - Required:
  - Initial Value:
  - Cardinality:
- Dependencies -
  - Derived from:
- Role names -
  - *group 1: {..., ..., ...}*
  - *group 2: {..., ..., ...}*
  - ...

Attribute Specification

Attribute 1:

- Identity -
  - Defining Quality -
  - Name -
- Accessibility - *{readable, settable, modifiable, fixed}*
- Scope - *{shared, unique}*
- Constraints -
  - Required:
  - Initial Value:
  - Cardinality:
- Dependencies -
  - Derived from:

Attribute 2:

…

What is a Component?

- An abstraction that represents both *memory* and *functionality* within
  - *memory*: resolution of a component's static qualities such as attributes and relationships.
  - *Functionality*: a set of methods (qualities) that embody operations
  - Represents a major highly cohesive "block" of the system
  - The components *partition* the system
    - Many ways to do this, you want an "elegant" one
- MBASE: A refinement of an Entity within the Domain Description

Component Qualities

Identity -

    Defining Quality -

    Name -

Attributes -

Behaviors -

Relationships -

Roles -

State Groups

Constraints -

Testing for Components

- Analysis components always have direct counterparts in the domain
- Important test: Components have *form* which allow them to transition from one *state* to another and take on *roles*

Advice for Finding Possible Components

- Start with a single fundamental component you know must be part of the system and fill out a specification template for it.
- Take a component it has a relationship to, and do the same.
- Repeat until no more components are found
- You will often need to draw upon the "possible components" list when detailing components relationships

Finding Possible Components

- From system capabilities and domain description look for “things” and “actors” that carry out some action.
- Start with Entities from Domain Description.
- Underline nouns, but not all nouns are guaranteed to be components

Components as Owners of Capabilities

- All capabilities must be eventually mapped to components
- some components may be identified by looking for entities to address specific capabilities
- Many participants will also become owners, e.g., owner of store is probably person working in the store

Caution: May translate into large blobs. e.g., Be careful of things like “Account Manager” or “Employee Tracker” as they may contain too many capabilities.

Components as Actors

- “notify users when appointments expire” must have a component doing the notification (be careful of “schedulers” - usually a design object).

Other Components

- Anything else that has component characteristics - (identity, memory, and operations)

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.

Component Pitfalls

- Every component has to have at least one state group: if you have difficulty specifying states or roles for a component, then it is probably not a component
- Do not include Components that do not reference Entities
- A component though may participate in more than one role at a given time: some roles may be assigned to multiple components, but the relationships must be specified

Design Preview: Java Composites and Components

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

Component Model

Simple Example

Example System Capabilities

Statement of purpose for the system:

The Columbia Library Reference-Librarian Management System (CLRMS) enables the Library system to locate and allocate appropriate library staff to serve Library Patrons in locating and borrowing Library reserve materials.

System capabilities:

1) Assign Reference Librarians to Library areas
2) Track consulting Reference Librarians
3) Manage Reference Librarian services to Patrons (including other Libraries) for particular reference collections
4) Assess need for consulting Reference Librarians (where, when, who)

Identify possible components

- Possible entities
  - Library
  - Library Staff
  - Library Patrons
  - Reserve Materials
  - Library Area
  - Consulting Librarian
  - Reference Collection

Example: Component Definition

Component 3

Identity:

Defining Quality.:

Scheduling and tracking information for permanent and temporary staff for the Columbia Libraries.

Name: Library_Staff

Attributes:

1) name
2) address
3) skills
4) schedule
5) availability
6) area_assignment
7) contract_info
8) service_requests

Example: Component Definition

Behaviors:

1) manage Reference_Material (organize, locate)

2) service Patrons

Relationships: <<see C3 view>>

Roles:

1) reference librarian:{services <service>, manages <reference material management>, service area assignment <librarian assignment>}

2) consulting librarian:{services <service>, organize and maintains <reference material management>, contracted <librarian assignment>}

Example: Component Definition

State groups:

1) employment_status:{contracted, permanent}

2) service_status:{serving_patron_request, managing_material, waiting_for_request, off_duty}

3) area_assignment:{assigned_to_area, available_for_area, unavailable}

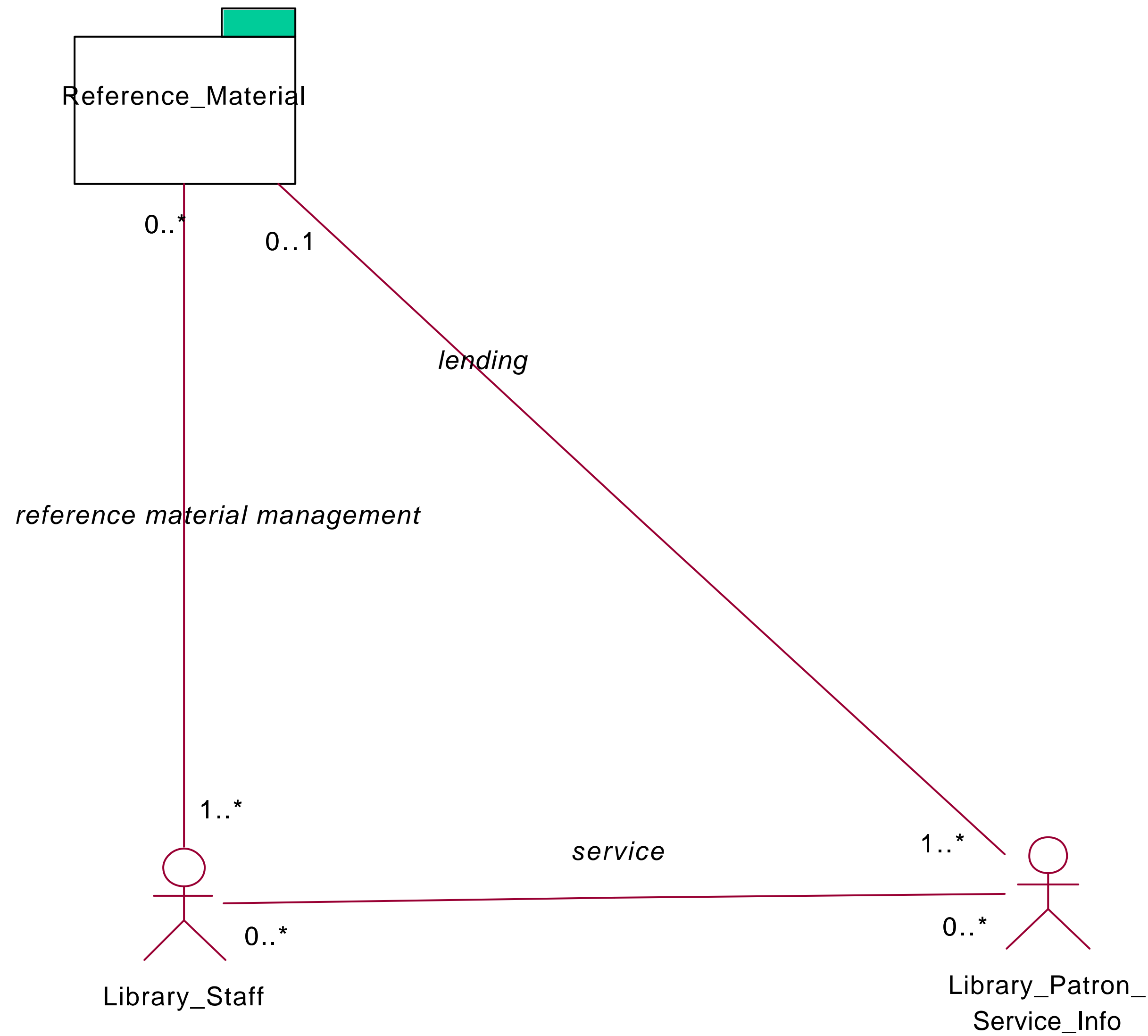
Constraints:

1) can not be consulting librarian and reference librarian for same Reference_Material

2) dependency: service_status is off_duty when area_assignment is unavailable

Derived from Library_People_Info

View C3: (SSAD 2.1)



SSAD Architectural Analysis

Course Evaluation System Example

(best guess)

> WCES_Student_Login

> WCES_Student_Login_Response_Parser

> WCES_Registrar_Class_List

> WCES_Student_Class_List

> WCES_Student_Class_Selector

> WCES_Evaluation_Questions

> WCES_Evaluation_Form

> WCES_Evaluation_Processor

> WCES_Evaluation_Response_Repository

> WCES_SEAS_Class_Structure

> WCES_Administrator_Login

> WCES_Administrator

> WCES_Student_Login

    * likely in interface for a Student entity, so might be a decent
    * component

> WCES_Student_Login_Response_Parser

    * Parsing is usually "how" an input is interpreted in the
    * software, hence an object

> WCES_Registrar_Class_List

    * Seems like a bone fide "what" needs to be represented from the
    * domain within the system they need to build - component

> WCES_Student_Class_List

    * This seems like the above in a different role or state

> WCES_Student_Class_Selector

    * Something that "selects" a group of items is actually a
    * relationship. Generally in design you will introduce a
    * "container" such as a hash table, list, tree structure to
    * hold the items and allow selection of them in the
    * appropriate way.

> WCES_Evaluation_Questions

    * Seems like a "what" with no behaviors (in the domain that is)
    * so it's likely an attribute of some other component - me thinks
    * WCES_Class or WCES_Course or the possible component below

```
> WCES_Evaluation_Form
          * Forms are general "how" you get data into your system, but in
          * this case he may be referring to a physical evaluation form
          * in which case this  is a component, but care should be taken to
          * see if it really needs to be separate from any type of
          * "class" or "course" component as from the systems perspective
          * the evaluation form for a class is the same as the class
          * itself as it never uses anything else about the class (such
          * as its location, room size, cost, etc.

> WCES_Evaluation_Processor
          * Anything that is an "action" or verb is generally a behavior or
          * an object. Why wouldn't the parser be in here?

> WCES_Evaluation_Response_Repository
          * Since there is likely an entity existing in the domain for this
          * (even if it's a simple folder or notebook with evaluation
          * forms), and the system conceptually needs a place to collect the
          * evaluations, this is a likely component. Note that no mention
          * of "how" the evaluations will be stored as software is
          * indicated (i.e. database, file, etc.).

> WCES_SEAS_Class_Structure
          * Seems like an attribute of another component, but it's vague.

> WCES_Administrator_Login
          * Like the WCES_Student_Login this may be a component that
          * represents the Admin for the system (i.e. lets the Admin
          * interface with the system). Care must be taken to ensure
          * that this truly has a different set of relationships to
          * the other components of the system, otherwise the Student
          * and Admin components should be combined into a more general
          * component "WCES_User_Login" that uses two roles "student"
          * and "admin"

> WCES_Administrator
          * This is an entity in the domain likely take care of by the above.
```

SSAD Architectural Analysis

HDA Example

Example #1 Component Specification

COM-01 HDA_Material

<table>
<tr><td>Defining Quality</td><td colspan="2">Digitized information on Hispanic material of the Boeckmann Center.</td></tr>
<tr><td>Attributes</td><td>a) Location
b) Box number
c) Folder number
d) Item number
e) Item type
f) Title
g) Source</td><td>h) Date
i) Descriptors
j) Country
k) Language
l) Notes
m) Collection</td></tr>
<tr><td>Behaviors</td><td colspan="2">Display information
Display images</td></tr>
<tr><td>Relationships</td><td colspan="2">a) Hispanic_Digital_Archive
b) HDA_Patron_Interface
c) HDA_Manager
d) HDA_Operator</td></tr>
<tr><td>Roles</td><td colspan="2">Viewed item {view < patron_brief_view> , view < patron_detail_view> , view < administrator_view> }</td></tr>
<tr><td>State Groups</td><td colspan="2">See Figure 2</td></tr>
<tr><td>Constraints</td><td>Candidate Key</td><td>Item Id (Not from domain)</td></tr>
<tr><td></td><td>Cardinality</td><td>® 1 Hispanic_Digital_Archive
® n HDA_Manager
® n HDA_Patron_Interface</td></tr>
<tr><td>Relates to</td><td colspan="2">Entity E-03 of OCD 2.4</td></tr>
</table>

From Hispanic Digital Archive LCA SSAD 2.2

Rejecting Components

- Why is this important?
- After listing potential components, need to filter those which are not components
- Even more important than finding components - we need to make sure components are not attributes, states, behaviors, or roles.
- When in doubt, leave as a component. Easier to go from component to attribute, etc. than vice versa

Attribute Specification

Attribute 1:
- Identity -
  - Defining L.O.S. -
  - Name -
- Accessibility - *{readable, settable, modifiable, fixed}*
- Scope - *{shared, unique}*
- Constraints -
  - Required:
  - Initial Value:
  - Cardinality:
- Dependencies -
  - Derived from:

Attribute 2:

…

Example #1 Component Attributes

Attribute ATR-01

Name	**Item Type**	
Defining Quality	**The item type determines the relevance of various fields describing an archive item**	
Accessibility	**Readable, Settable**	
Scope	**Unique, Shared**	
Constraints	*Initial Value*	**Book**
	Modality	**Required**
	Cardinality	**® n**

Attribute ATR-02

Name	**Collection**	
Defining Quality	**A group of items belonging to a logical grouping as determined by the Administrator**	
Accessibility	**Readable, Settable**	
Scope	**Unique, Shared**	
Constraints	*Initial Value*	**None**
	Modality	**Optional**
	Cardinality	**® n**

From Hispanic Digital Archive LCA SSAD 2.2

Rejecting Attributes

- Attributes are the “memory” part of a component
- Attributes “do nothing” i.e. have no behavior e.g., Address

Relationship Specification

Relationship 1:

- Identity -
  - Defining L.O.S. -
  - Name -
- Accessibility - *{readable, settable, modifiable, fixed}*
- Scope - *{shared, unique}*
- Constraints -
  - Required:
  - Initial Value:
  - Cardinality:
- Dependencies -
  - Derived from:
- Role names -
  - *group 1: {..., ..., ...}*
  - *group 2: {..., ..., ...}*
  - ...

Example #1 Component Relationships

Relationship REL-01

Name	**Inspects**	
Defining Quality	**A patron requests to inspect an item.**	
Accessibility	**Readable**	
Scope	**Shared**	
Constraints	*Modality*	**Optional**
	Cardinality	**® n**
Role names	**Inspected Item {Viewed, Retrieval Requested}**	

Relationship REL-02

Name	**Maintains**	
Defining Quality	**An administrator performs maintenance operations on items such as add, modify, delete.**	
Accessibility	**Settable**	
Scope	**Shared**	
Constraints	*Modality*	**Optional**
	Cardinality	**® n**
Role names	**Maintained Item: {Added, Deleted, Modified,Approved}**	

From Hispanic Digital Archive LCA SSAD 2.2

Example #1 relationship diagram

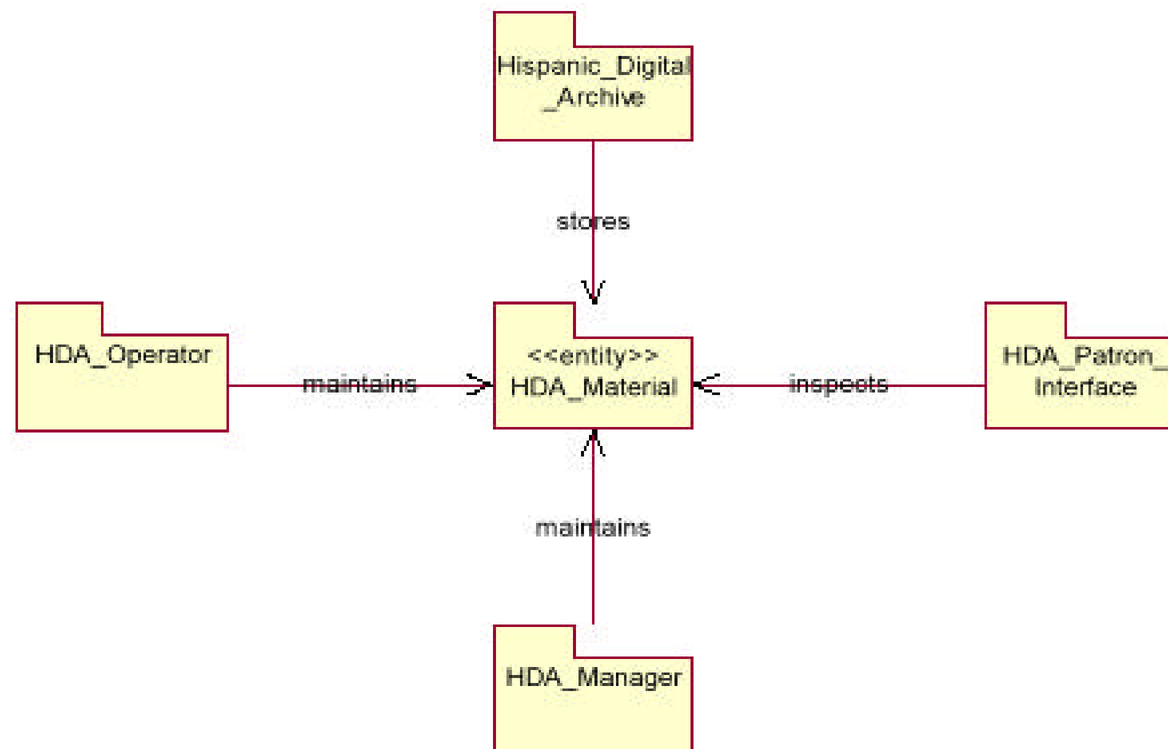


Figure 1 COM-01 HDA_Material

From Hispanic Digital Archive LCA SSAD 2.2

Rejecting Roles

- Roles: occur frequently and cause lots of confusion.
- Identified by how something is used, as opposed to what it actually is
- Example: In a company payroll program, “Manager” and “Subordinate” are “Person”

Example #1 State Diagram

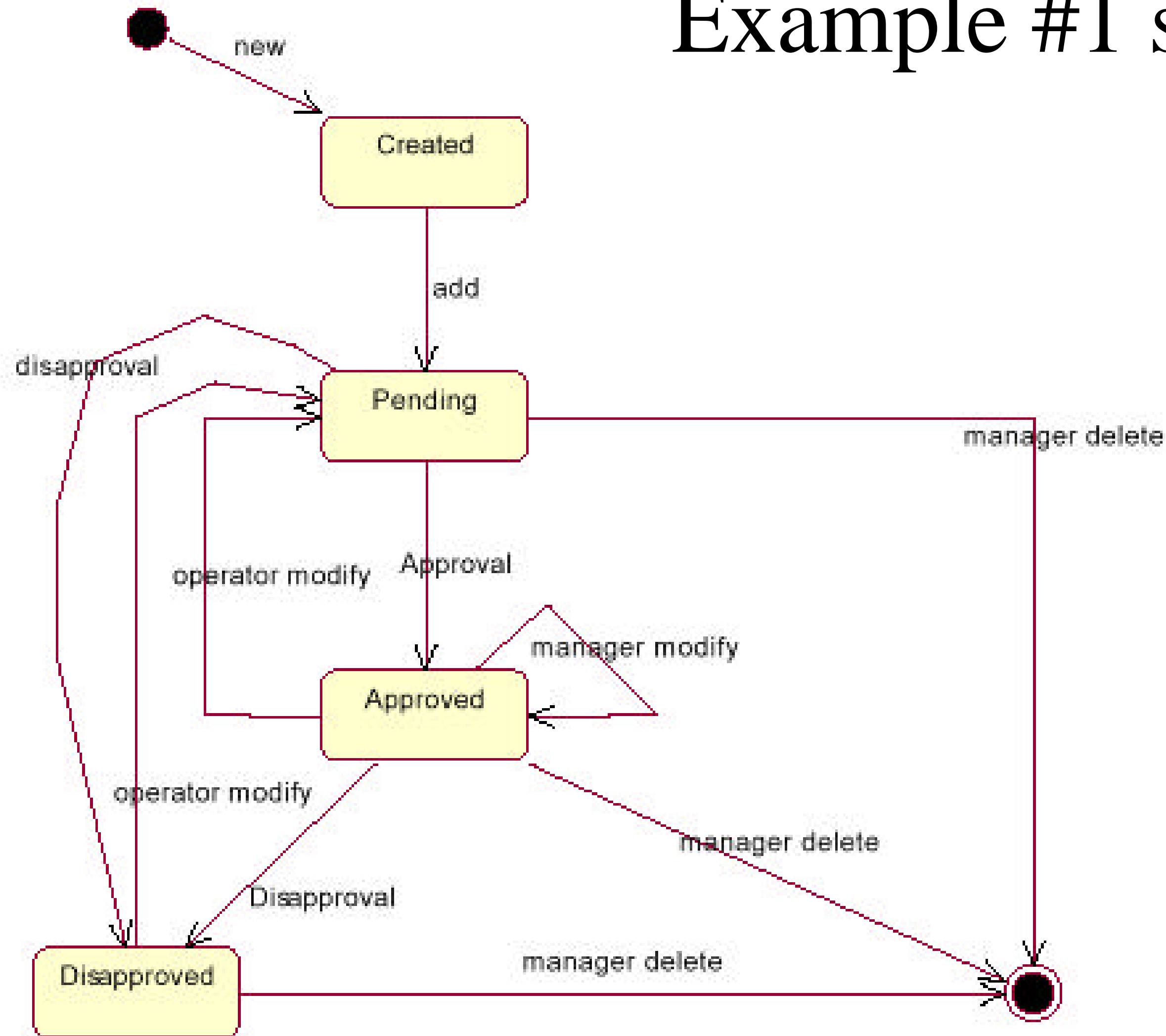


Figure 2 HDA_Material State Diagram

From Hispanic Digital Archive LCA SSAD 2.2

Rejecting States

- State: attributes and relationships that affect the behavior of an object.
- Has well-defined transitions to or from other states.
- Example:
  - "Solvent account" and "Closed account" should be combined into a single "Account" component with states {open, closed}, {solvent, insolvent}

Rejecting Behaviors

- Behavior: Maps to objects
- Operations or Behaviors: “Withdraw”, “Deposit”, “Access” may not be components

Design goals

- Do not clutter up analysis with information relevant to design only.
- Ask, “Is this something the domain expert will understand?”
- Filter out:
  - System Requirements
  - Runtime requirements (e.g. file access and memory allocation),
  - UI, Network, Implementation objects (e.g. strings, pointers, etc.)