

System Analysis II & III

CS577a

Fall 2002

Ed Colbert

USC Center for Software
Engineering

Who Am I?



- Research Associate, Center for Software Engineering
- 20 years industrial teaching & consulting on object-oriented methods, software engineering, & programming languages
- Consultant on definition of Architecture Design Language (ADL) for real-time, safety-critical systems
 - Based on Unified Modeling Language (“UML”) and Honeywell’s MetaH
 - To be proposed as standard of Society of Automotive Engineers (SAE)
- Created Colbert Object-Oriented Software Development method (“OOSD”)
 - Noted for strength in real-time software development
 - NASA Langley Research Center used for software engineering process manual
- MBASE developer
- Founded Absolute Software Co., Inc. in 1986

Course Etiquette

- Observe normal rules of classroom etiquette
 - Be on time
 - If you are late, don't slam the door
 - (tell your friends)
 - One conversation at a time
 - One topic at a time
 - Direct all comments to instructor
 - Encourage (rather than criticize) other students
 - E-mail & browsing at breaks
 - Turn off/silent cell phones & pagers

- Respect others & yourself

- Be ready to learn....

Goal of Presentation

- Understand how to perform System Analysis
 - Using
 - MBASE
 - Object-oriented techniques
 - RUP
 - Rational Rose
- Understand how to document analysis
- Presentation is part 2 & 3 of 3 on System Analysis



Outline

- **Key Concepts**
- Process Overview
- Example Project Description
- Process by Example

Purposes of Analysis & Design

- To transform requirements into design of system
- To evolve robust architecture for system
- To adapt design to match implementation environment
 - Designing it for performance
 - RUP 2001

Analysis & Design Approach

- Architecture–centric
 - Define stable architecture early
- Use–case Driven
 - Use–case describe capabilities/usage of system
 - Use–case help identify objects & operations needed
- Iterative & Incremental

Analysis Goals

- Quantify ***what we*** want to represent
 - ***Not how*** it is done
- Formalize & refine automated parts of organizations capabilities, entities, activities, & interactions described in domain description
- Model high-level architectural of system

Analysis Audience

- Domain Description is for all constituents of project
- Analysis is for Domain Experts
 - i.e. high level leaders who
 - Understand domain
 - Know what they want,
 - Have authority to make decisions
- Not for implementers
 - Who prefer design & implementation details (“how”)

What's A Software Architecture?

- *Software Architectures* [Shaw & Garlan 96] defines for a system
 - Computation components
 - Clients
 - Servers
 - Databases
 - Filters
 - Layers
 - Interactions among components
 - Subprogram calls
 - Shared data
 - Client–server
 - DB–accessing protocols
 - Asynchronous even multicast
 - Piped streams
 - etc.

What's A Software Architecture? (cont.)

- *IEEE Standard Glossary of Software Engineering Terminology* [IEEE Std 610.12-1990]
 - Organizational structure of a system or component.
- *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems* [IEEE Std 1471-2000]
 - Fundamental organization of a system embodied in
 - Its components,
 - Relationships among the components,
 - Relationships to the environment,
 - Principles guiding its design & evolution.

What's A Software Architecture? (cont.)

■ RUP 2002

- Highest level concept of a system in its environment
- Organization or structure of system's significant components
- Interacting through interfaces
- With components composed of successively smaller components & interfaces

What's a Component?

■ Many Definitions

- *Software Architectures* [Shaw & Garlan 96]
 - Loci of computation & state
 - Has an interface specification that defines its properties

- *Component Software: Beyond Object-Oriented* [Szyperski 97]
 - A unit of
 - independent deployment
 - third-party composition
 - Has no persistent state

What's a Component? (cont.)

□ UML v1.4

- Modular, deployable, & replaceable part of system that
 - Encapsulates implementation
 - Exposes set of interfaces
 - Specified by one or more classifiers that reside on it
- May be implemented by one or more artifacts
 - e.g., binary, executable, or script files

□ RUP 2002

- A **non-trivial, nearly independent**, & replaceable part of system that fulfills clear function in context of well-defined architecture
- Conforms to & provides physical realization of a set of interfaces

What's a Component? (cont.)

□ MBASE Guide

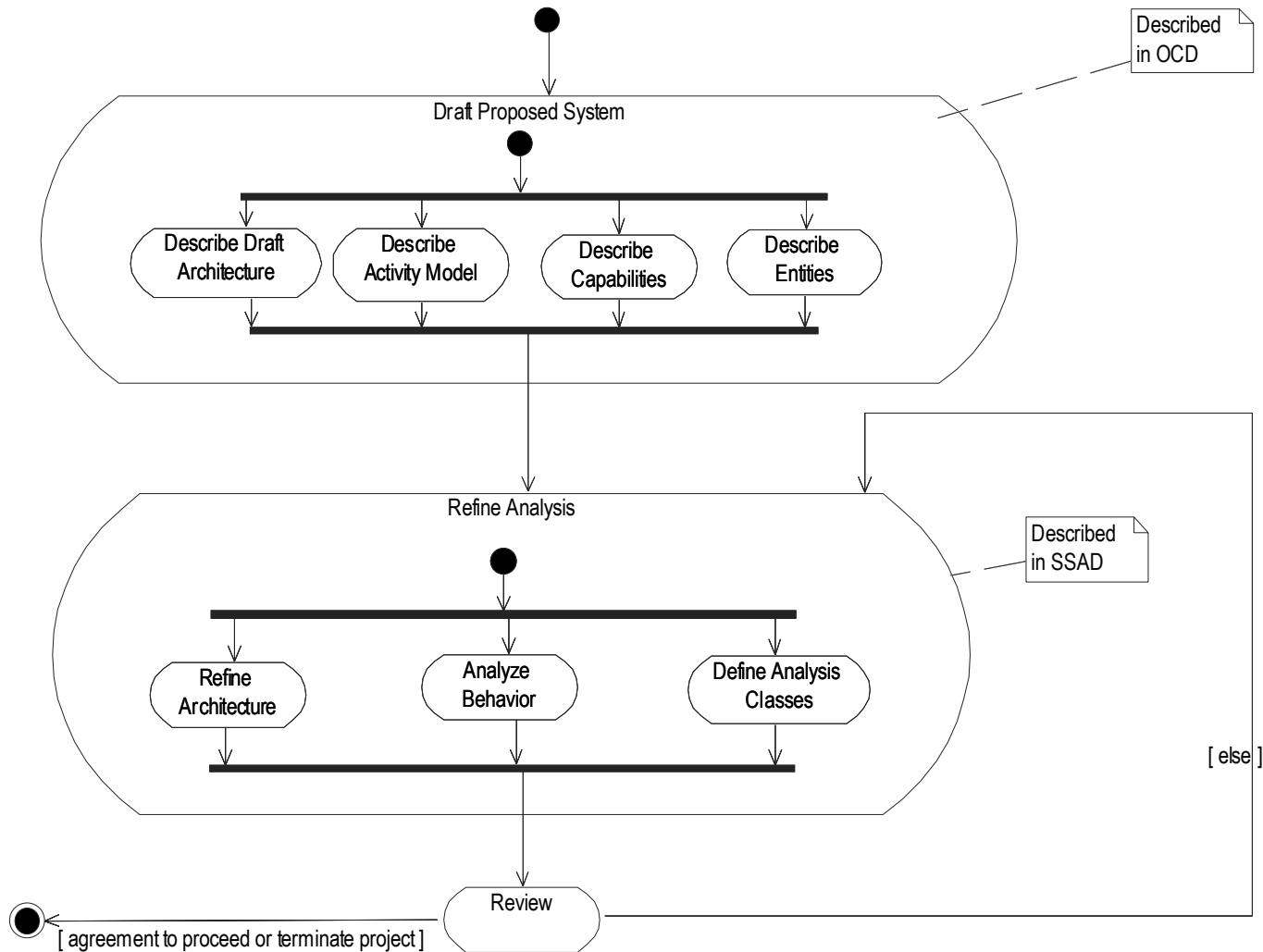
- An abstraction that represents both memory & functionality
- A partition of system
- A part of system
- Implemented by objects
- Detail guides for identifying components are typically used for objects
- We'll use
 - “component” to mean part
 - “*Component*” to refer to UML/RUP definition



Outline

- Key Concepts
- **Process Overview**
- Example Project Description
- Process by Example

System Analysis Process Overview





Outline

- Key Concepts
- Process Overview
- Example Project Description
- Process by Example

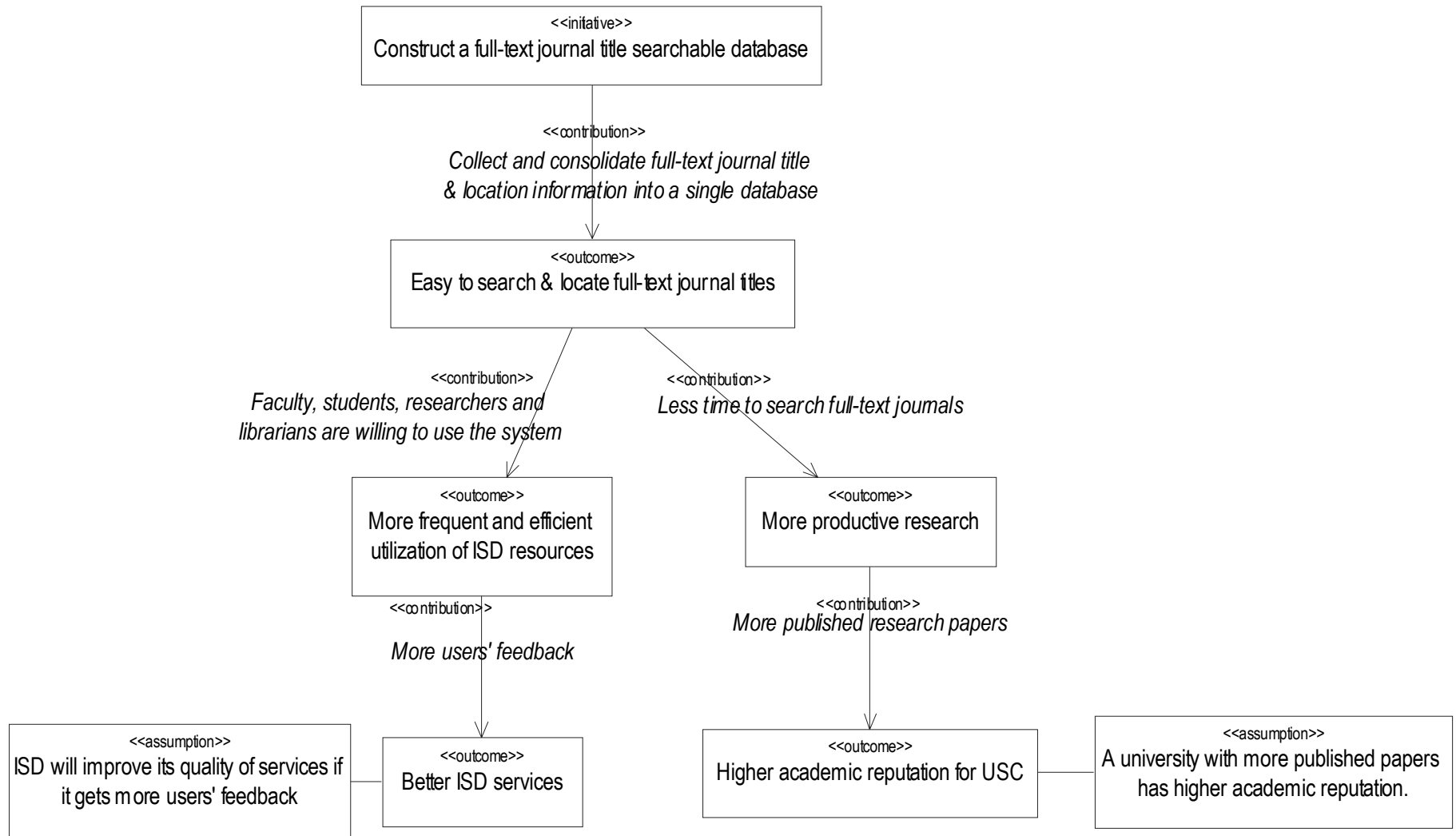
OCD 2.1 System Capability Description

- Target customer: Information Service Division (ISD) in University of Southern California
- The system is proposed and implemented for faculty, students, researchers and librarians at USC who need to identify where a particular periodical title is indexed, the date of coverage and whether it is available in full-text.
- The Full-text Title Database is a web-based full-text journal title searchable database.
- That system retrieves and consolidates journal title information from different vendors and provides it to general public users such as faculty, students, researchers and librarians.
- Unlike the Jake project at Yale University and full-text journal and newspaper search engine at Indiana University, our product is designed dedicated to USC which is accessible globally via World Wide Web and provides full-text journal title information to general public especially USC community. The Jake project of Yale University and the journal and newspaper search engine at Indiana University provide similar functionality. However, the former includes some information in those vendors' databases that USC does not has access to and the later only contains journal title information from databases that Indiana University has access to which is inadequate.

OCD 2.1.1 Benefits Realized

- The database we are going to build will let the general public users (faculty, students, researchers, librarians, etc.) easy to find the full-text journal title information they need. They can do a search in our database instead of searching each vendor's database USC has access to one after another. Thus it will much reduce the full-text journal searching time in the research. Since it's an easy to use and powerful system, more and more users are willing to use our system. It leads to more frequent and efficient utilization of resources of ISD, and more productive research work by faculty, students and researchers at USC. And it will lead to better ISD service and a higher academic reputation for USC.
- The database we are going to build will provide a web-base interface to system administrator, so ISD can easily find a person who has some basic computer knowledge to maintain the whole system, and doesn't have to provide much training to him/her or additional resources. Therefore ISD can save budget and time on a system which is easy to maintain like this proposed system.

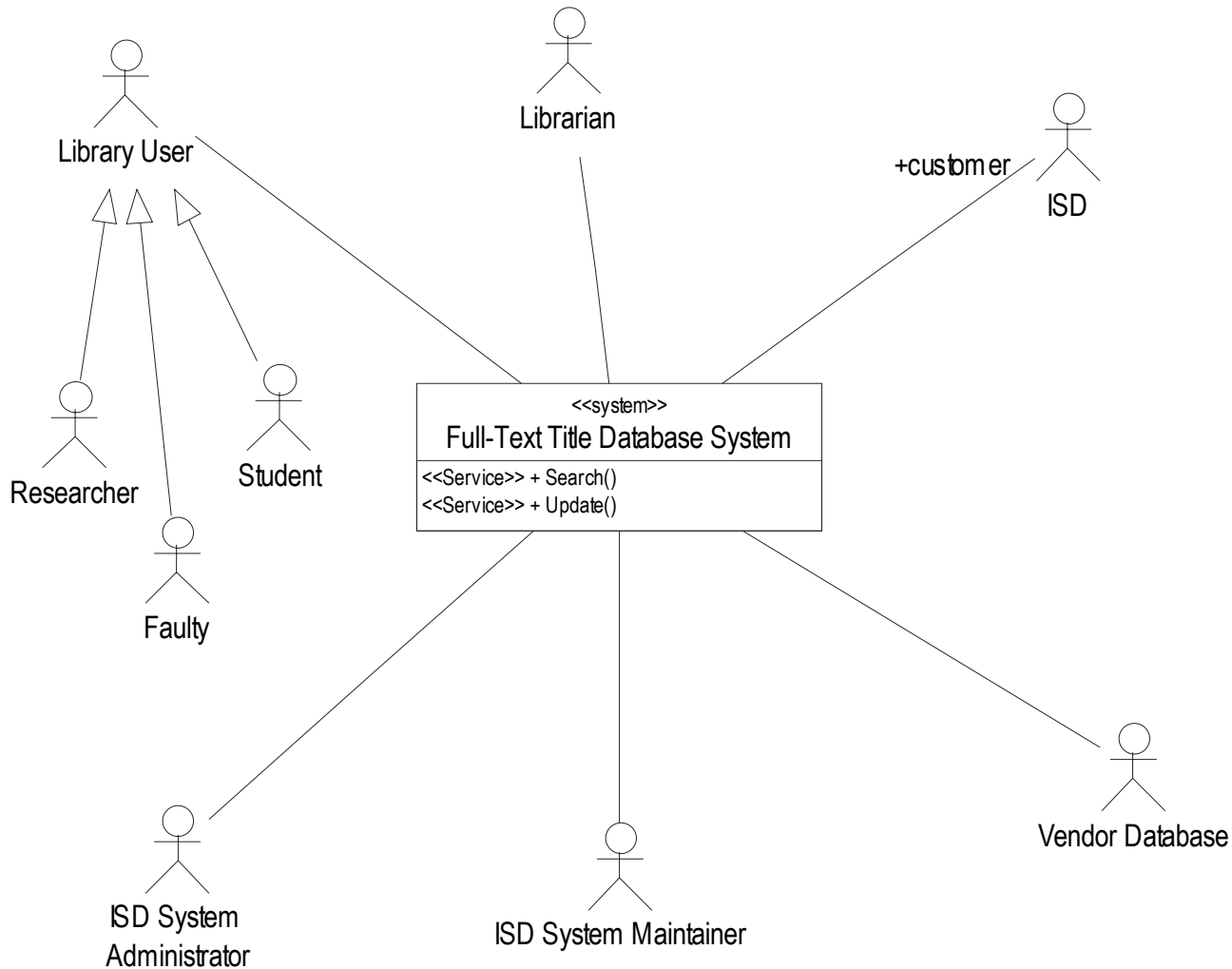
OCD 2.1.2 Results Chain



OCD 2.2 Key Stakeholders

- **Software/IT system's users: Faculty, Researchers, Students, and Librarians at USC.**
 - The relationship with result chain: The users will be the ones who are going to interact with our system through user interface. If they are willing to use our system, they will get the benefit which is to “fill a serious gap in the tools available and result in more efficient research for researchers and students”.
- **Customers: ISD**
 - The relationship with result chain: Whom we are going to develop the system for. This is also the one who is going to inspect the result chain.
- **Developers: CS577a Team8 & CS577b Team for Full-text Title Database**
 - Developers are responsible for design, construct and implement the software system. They are also responsible for helping the customer and provide some training to the system administrator and software maintainer during the transition phase of the system.
 - Relationship with result chain: We are the people who develop the result chain.
- **System Administrator: Greg Fleming**
 - Greg Fleming will be the one who is assigned by ISD and responsible for updating the system database. He needs to observe the vendors' databases. Once vendors update their databases, our unified database should also be updated.
 - Relationship with result chain: He is responsible for ensuring the system to provide the updated information to users so that they can actually get the expected benefits.
- **Software Maintainer: ISD employee**
 - ISD will assign a person to maintain the system. He/she is responsible for maintaining the software code and crash recovery of the system.
 - Relationship with result chain: He/she is responsible for ensuring the system stability and availability so that the expected benefits can be realized.
- **Data Source: Vendors**
 - Vendors are the commercial organizations which provide databases containing the full-text journals and subscription service to others.
 - Relationship with result chain: The initiative of the result chain. We assume the vendors' databases are searchable. Our unified Fulltext Title Database will be generated from their databases

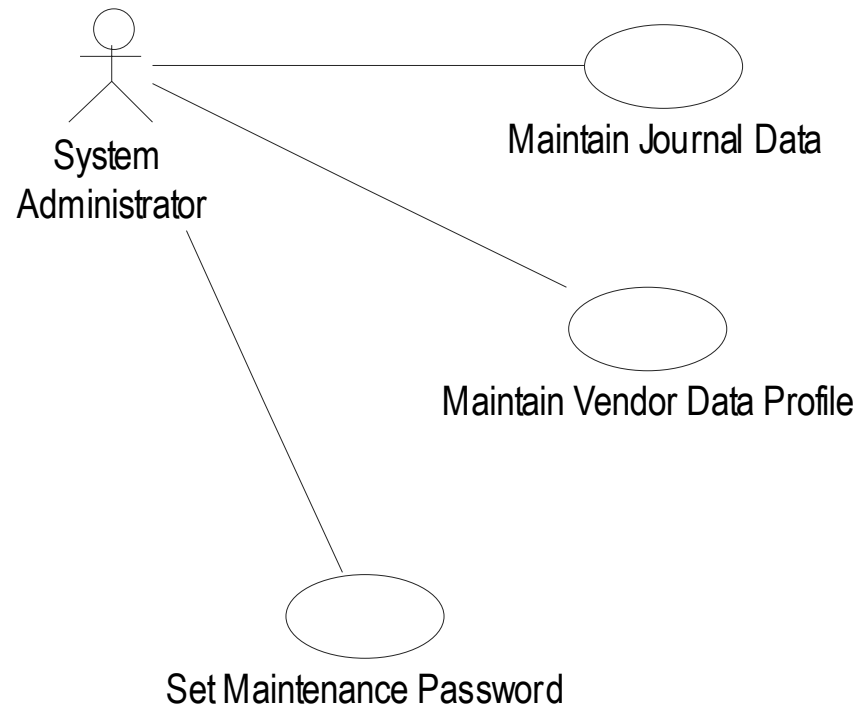
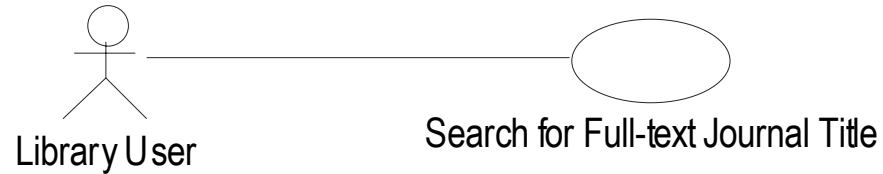
OCD 2.3 System Boundary & Environment



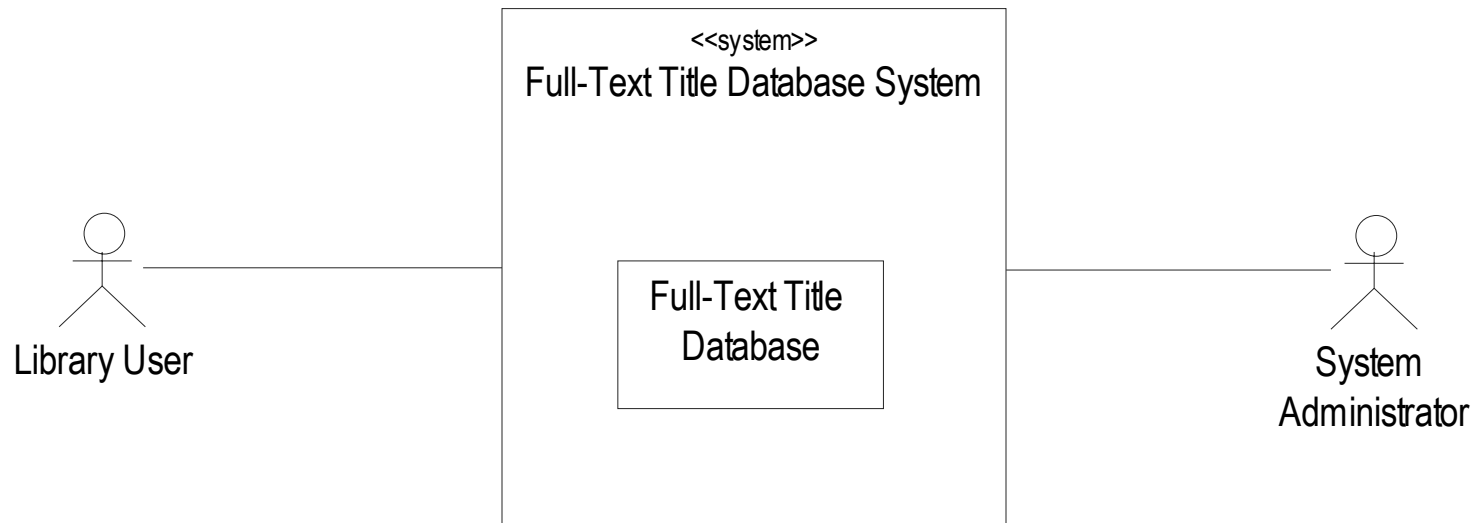
OCD 3.2 Organization Goals

- **OG-1: Fast Information Access**
 - Provide a faster way to access information
- **OG-2: Reduced workload**
 - Allowing the librarians to focus their time & energy on those patrons that really need help
- **OG-3: User friendly environment**
- **OG-4: Enhance the library collection**
 - Enhance the library collection by subscribing to vendors' full-text journal databases
- **OG-5: Ensure the Integrity of Information**
 - Provide a way to ensure the integrity of the information and protect the contents of the system
- **OG-6: Make the Resource Available to Distributed Users**
 - Users may be locate at any place and want to access the resources

OCD 4.3 Capabilities

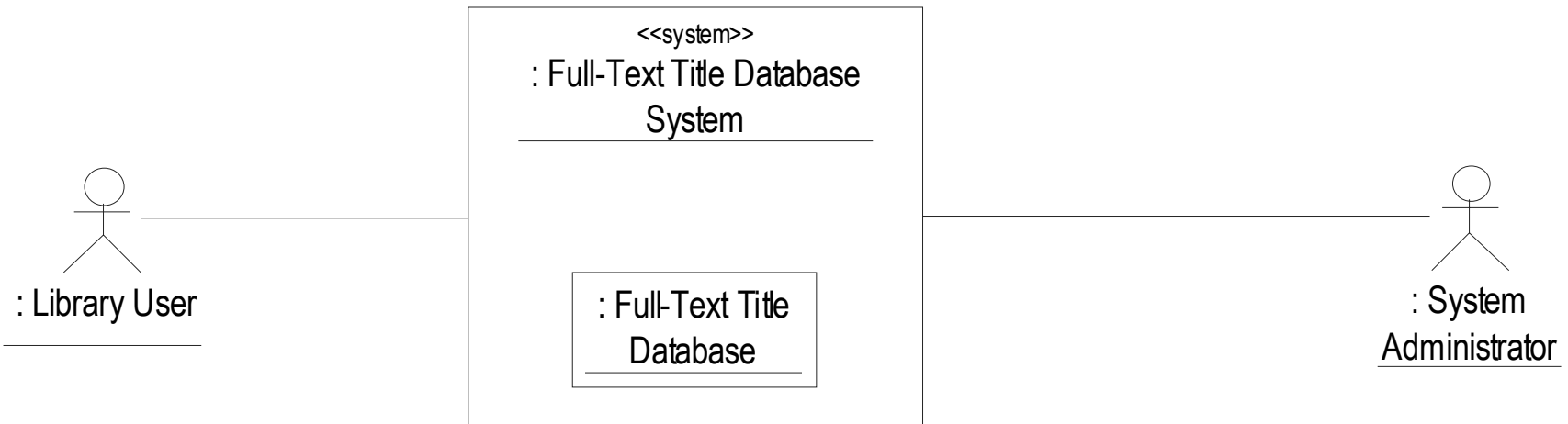


OCD 4.5 Proposed System



OCD 4.5 Proposed System (cont.)

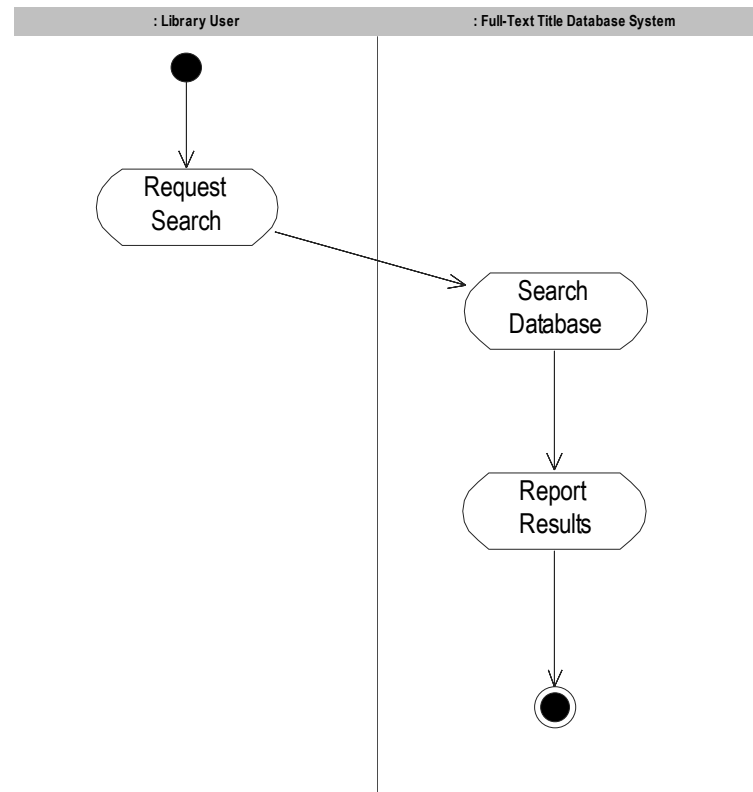
Typical Configuration



- For this system,
 - There's really only 1 configuration
 - Doesn't add much information

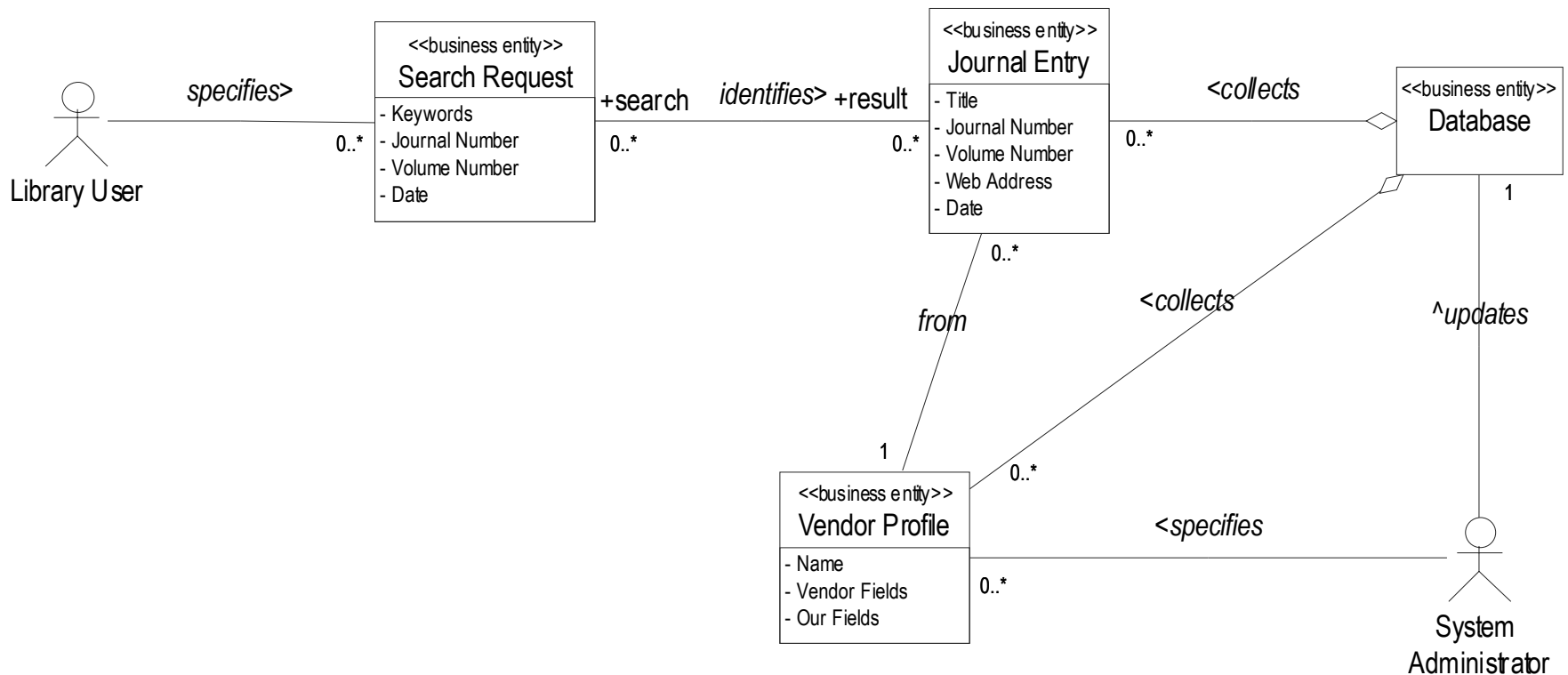
OCD 4.5.1 Proposed Activities

Search for Full-Text Journal Title



- Other activities not shown

OCD 4.5.2 Proposed Entities



Iteration Plan (Informal)

■ 2 Iterations

□ Capabilities in Iteration#1

- Search for Full-Text Journal Title
- Maintain (Enter) Journal Data

□ Additional Capabilities in Iteration#2

- Maintain Vendor Data Profiles
- Maintain (Change) Journal Data
- Set Maintenance Password



Outline

- Key Concepts
- Process Overview
- Example Project Description
- **Process by Example**

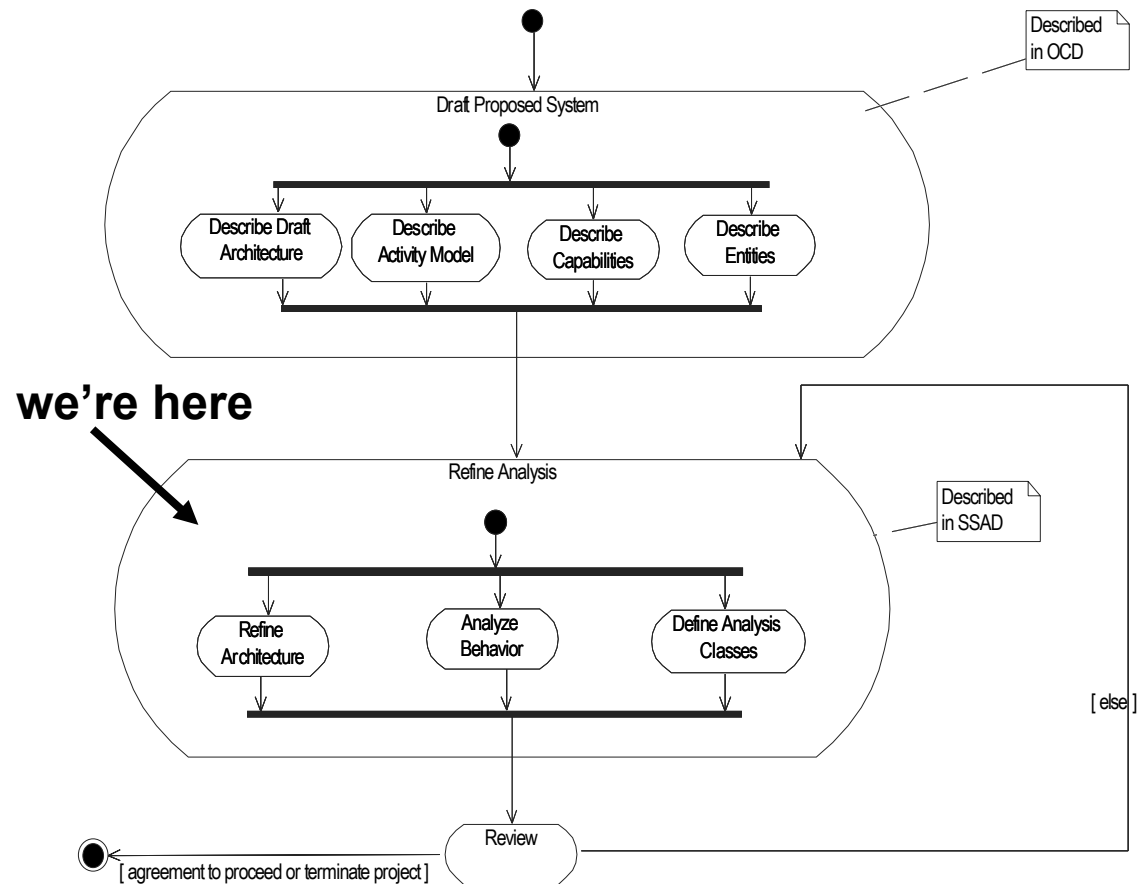
System Analysis Process Overview

- Sub-activities not ordered

- i.e. can start anywhere

- Why not?

- Iterative Process



Analyze Behavior – LCO

■ Purpose:

- Improve understanding of behavior requirements

■ Inputs:

- Proposed System Context
- Activity Model
- Capability Model

■ Artifacts:

- Use-cases Model
 - Use-case Diagram(s)
 - Descriptions of each use-case
 - See Form

Analyze Behavior – LCO

Guide for Creation of Use–Case Model

- For each capability in Capability Model
 - If abstraction of more specify behavior
 - Create use–case for each specific behavior
 - e.g. Manage Vendor Profile => Add, Edit, Delete Profile?
 - Show generalization relation in Behavior Classification Model
 - Otherwise, create use–case for capability
- Add actors that participate in each use–case
- For all actors,
 - Verify that have identified all capabilities to be included in current iteration

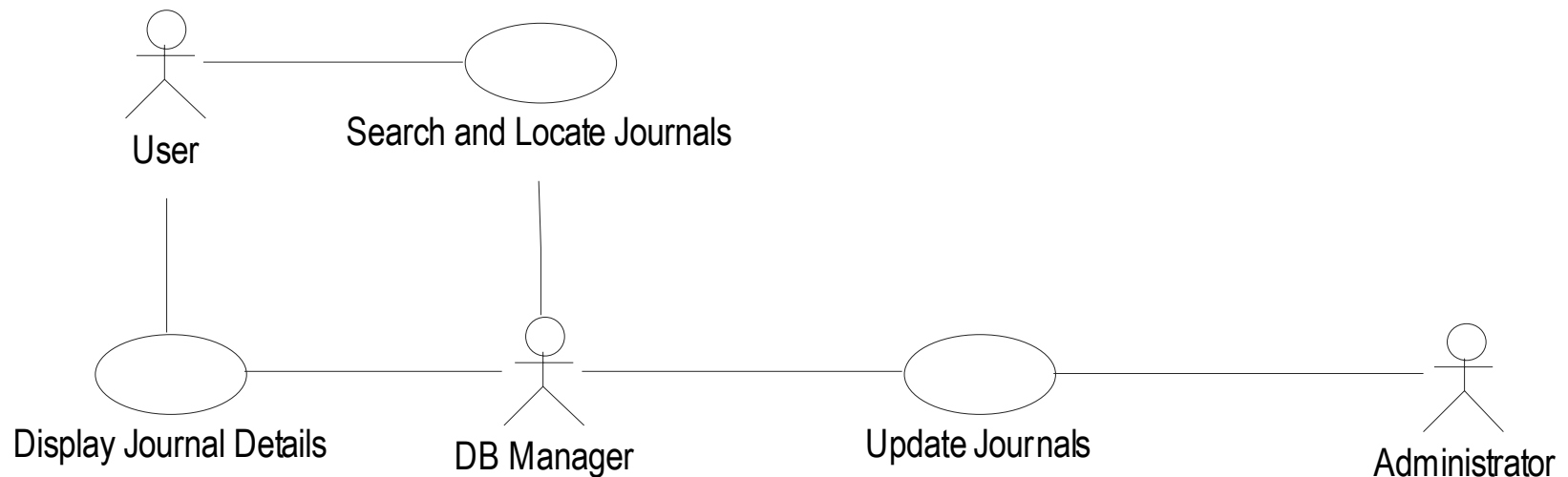
Analyze Behavior – LCO

Guide for Creation of Use–Case Model (cont.)

- Define relations
 - For each use–case that an actor participates in
 - Draw bi-directional association between actor & use–case
 - If actor specializes another actor
 - Draw generalization from specialized actor to general actor
 - If use-case extends other use-case
 - Draw extends relation
- Describe each actor
- Describe each use–case
 - Form
 - Sequence Diagrams (optional at LCO)
- Identify common behavior in different use–cases
 - Create use–case for common behavior
 - Draw include relation from all behaviors that share this common behavior to the use–case for common behavior

Analyze Behavior – LCO

Use-Case Diagram Example



Analyze Behavior – LCO

Use-Case Description Example 1

Use-Case Name	Full-text Journal Title Search
Abstract	
Purpose	To allow a user to search for journals to which USC Libraries subscribes by keyword
Actors	User, DB Manager
Importance	Primary
Requirements	Full-text Journal Title Search
Risks	
High-Risk?	No
Architecturally Significant?	Yes
Development Status	Draft LCO
Overview	User enters search criteria and system returns lists of journals matching criteria
User Interface	
Pre-conditions	Database has been initialized
Post-conditions	Displayed List of all the complete journal titles containing the user's search criteria
Includes	
Extension Points	

Analyze Behavior – LCO

Use-Case Description Example 1 (cont.)

■ Typical Course of Action

Seq. #	Actor Actions	System Response
1.	User requests search	
2.		Displays search page
2	User enters search criteria	
3		Queries the database asking for journal titles that match the user's search criteria
4		Displays the journal list in search result page

Analyze Behavior – LCO

Use-Case Description Example 1 (cont.)

- Alternate Course of Action: No results match search criterion

Seq. #	Actor Actions	System Response
4.		Displays error page that asks user to search again

- Exceptional Course of Action: None

Analyze Behavior

Exit Criteria for Use–Case Descriptions

■ LCO

- High–risk, architecturally significant, or particularly complex use–case
 - Include detailed courses of actions with exception & alternate courses of action identified
- Other use–cases need only include high–level overview

■ LCA

- High–risk, architecturally significant, or particularly complex use–case
 - Designed using one or more Sequence Diagrams
 - Interaction Model
- Other use–cases need only include high–level overview

■ IOC

- All use–cases should be designed using one or more Sequence Diagrams

Define Analyze Classes – LCO

■ Purpose:

- Developing understanding of key concepts in this build

■ Inputs:

- Proposed System
 - Revised?
- Proposed Entity Model

■ Artifacts:

- Component Model (part)
- Enterprise Classification Model
- Draft Logical Classification Model

Define Analyze Classes – LCO

Guide to Creating Class Model

- Create draft Enterprise Classification Model
 - Create diagram(s) that shows preliminary
 - Classes
 - Relations
 - Associations
 - Dependencies?
 - Generalizations
- Create Tracing Diagram (recommended)
 - Shows relation of
 - Analysis actors to Proposed actors
 - e.g. if use different names
 - Analysis classes to entities in proposed system
- Warning: don't get carried away defining class

Define Analyze Classes – LCO

Guide to Creating Class Model

■ First-cut class identification

- Define how entity in Proposed Entity Model will be represented
 - Often 1 class per entity
 - Stereotype <<entity>>
 - Define attributes for class based on attributes of entities
 - If state of some instances need to be maintained between execution
 - Set Persistence property to True
- For each actor in context
 - Create a class that either
 - Manages
 - Monitors
 - Acts for
 - Represents
 - Is Interface
 - We'll stereotype later

Define Analyze Classes – LCO

Guide to Creating Class Model (cont.)

- First-cut class identification (cont.)
 - For each window/screen/form in user interface
 - Create a class
 - Stereotype <<boundary>>
 - Review use-case description for nouns
 - Determine whether describes class or object
 - If object determine its class
 - If classes found that are not in model
 - Add it
 - Determine stereotype

Define Analyze Classes – LCO

Guide to Creating Class Model (cont.)

- First-cut class identification (cont.)
 - Create class to coordinate work of use-case
 - If logic is not particularly related to
 - user interface issues (boundary objects)
 - data engineering issues (entity objects)
 - Stereotype <<control>>

Define Analyze Classes – LCO

Guide to Creating Class Model (cont.)

- Identify preliminary relations
 - Associations
 - Relations between concepts that indicates some
 - Meaningful connection
 - Interesting connection
 - Associations worth noting & including on Conceptual Model
 - Imply knowledge of relationship that needs to be preserved
 - Derived from Common Associations list
 - Compelling or useful in context of requirements
 - High priority associations
 - Invariably useful in conceptual model
 - A is physically or logical part of B
 - A is physically or logically contained in/on B
 - A is recorded in B
 - Named based on
 - Matching the pattern “Class Name – Verb Phrase – Class Name”
 - Creating readable & meaningful sequence in model context

Define Analyze Classes – LCO

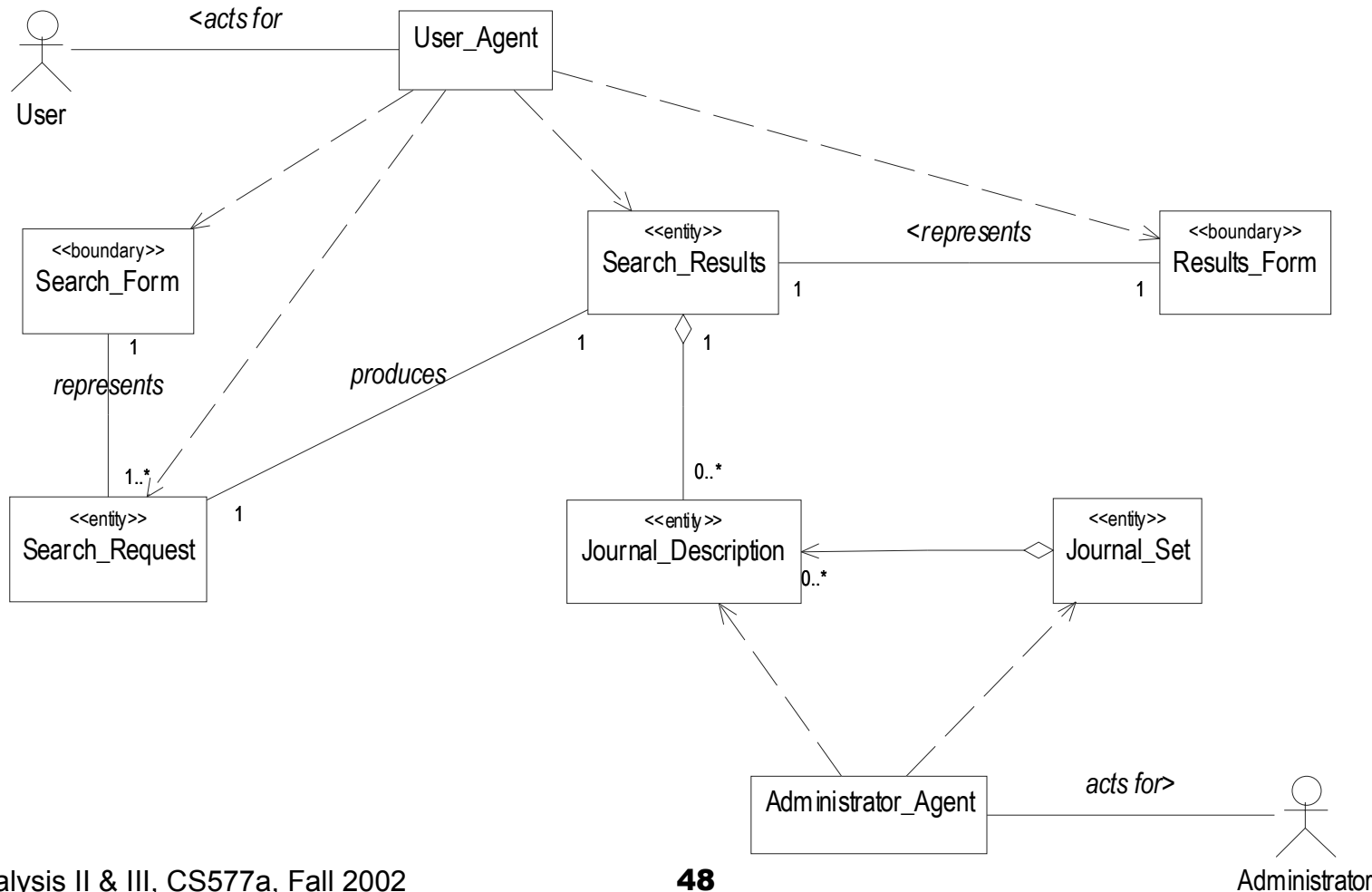
Guide to Creating Class Model (cont.)

- Identify preliminary relations (cont.)
 - Dependencies
 - Knowledge of other class required, but no permanent association
 - Hint: think of classes of parameters or temporary objects need in an operation
 - Generalizations
 - From special class to general class

- Repeat: don't get carried away

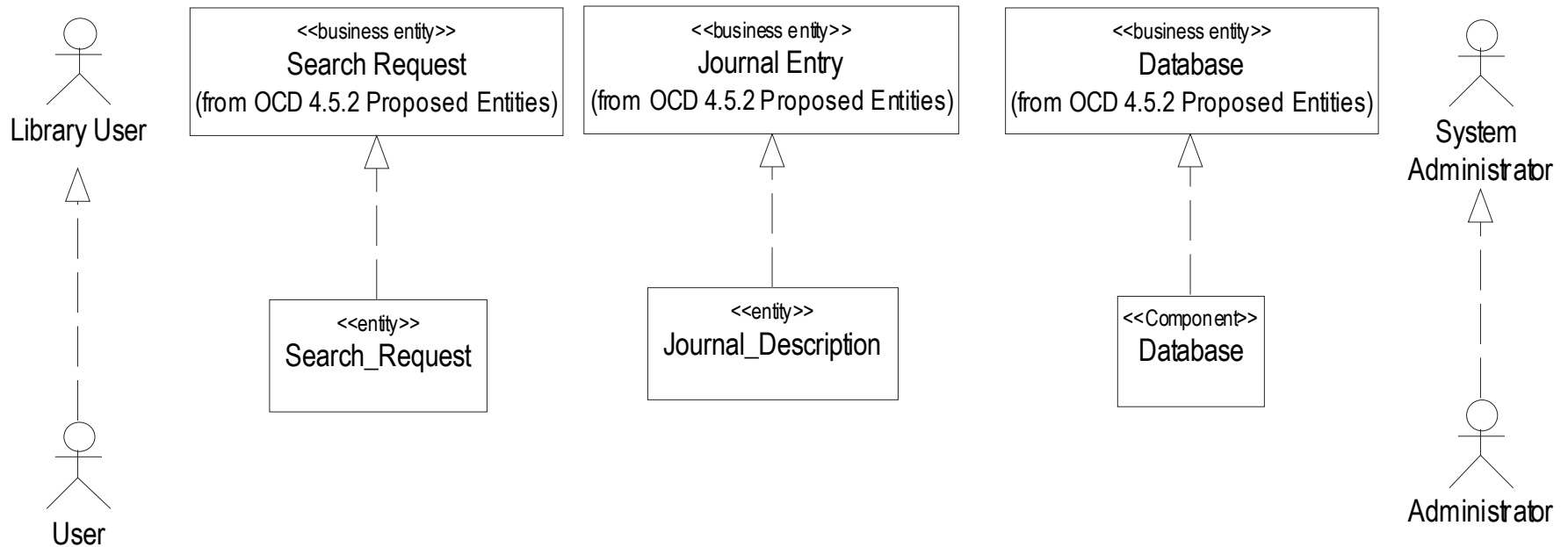
Define Analyze Classes – LCO

Enterprise Classification Model for Full-Text Title Database System



Define Analyze Classes – LCO

Business-Analysis Class Mapping for Full-Text Title Database System



Refine Architecture – LCO

- Purpose:
 - Define candidate architecture for system
 - Based on experience gained from similar systems or in similar problem domains
 - Identify Components
 - Understand
 - Hardware execution environment
 - Allocation of components to hardware
- Inputs:
 - Proposed System
 - Proposed Entity Model
 - Behavior Model
 - Architectural Patterns
 - L.O.S Requirements
 - Evolution Requirements

Refine Architecture – LCO Artifacts

- **Component Model**
 - Identify component classes
 - Assign classes to components
 - Identify dependencies between components
- **System Topology**
 - Identify layers
 - Assign components to layers
- **System Deployment Model**
 - Identify node classes & connections
 - Describe configuration of nodes
 - Allocate component instances to nodes

Refine Architecture – LCO

Considerations for E-Business Systems

- Existing network logical & physical design
- Existing databases & database design
- Existing Web environment
 - Servers, firewalls etc.
- Existing server environment
 - Configuration, software versions, planned upgrades
- Existing standards
 - Network, naming, protocols etc.

Refine Architecture – LCO Component Model

- Start with Proposed System Block Diagram
 - Did you identify parts of the system?
 - Likely components
- Common components
 - Clients
 - Servers
 - Databases
 - Filters
- Identify dependencies

Refine Architecture – LCO Component Model (cont.)

■ Look for Components

- Loci of computation & state
- Modular, deployable, & replaceable part of system
 - i.e. can be independently
 - Ordered, configured, or delivered
 - Developed, as long as the interfaces remain unchanged
 - Deployed across a set of distributed computational nodes
 - Changed without breaking other parts of the systems
- Units which can provide restricted security over key resources
- Existing products or external systems in design

Refine Architecture – LCO Component Model (cont.)

- Hints from Enterprise Classification Model
 - Look for optional groups
 - i.e. group of classes that represent optional behavior
 - Enclose in subsystem
 - Features which may be removed, upgraded, or replaced with alternatives should be considered independent

Refine Architecture – LCO Component Model (cont.)

- Hints from Enterprise Classification Model (cont.)
 - Look to user interface
 - If relatively independent of entity classes
 - i.e. interfaces & entities can & will change independently
 - Create components which are horizontally integrated
 - Component(s) for related user interface boundary classes
 - Component(s) for related entity classes
 - If tightly coupled with entity classes
 - i.e. change in one triggers a change in other
 - Create components which are vertically integrated
 - Component(s) for related boundary & entity classes

Refine Architecture – LCO Component Model (cont.)

- Hints from Enterprise Classification Model (cont.)
 - Look to Actors
 - Separate functionality used by two different actors
 - Each actor may independently change their requirements
 - Look for coupling & cohesion between classes
 - i.e. highly coupled or cohesive classes collaborate to provide some set of services
 - Organize highly coupled classes into component
 - Separating classes along lines of weak coupling
 - May be able to eliminate weak coupling by splitting classes into smaller classes with more cohesive responsibilities

Refine Architecture – LCO Component Model (cont.)

- Hints from Enterprise Classification Model (cont.)
 - Look at substitution
 - If there are several levels of service specified for a particular capability
 - i.e. high, medium and low availability
 - Represent each service level as separate subsystem
 - Each of which will realize the same set of interfaces
 - Subsystems are substitutable for one another

Refine Architecture – LCO Component Model (cont.)

- Hints from Enterprise Classification Model (cont.)
 - Look at distribution
 - Multiple instances of particular component can execute on different nodes
 - Single instance of component can't be split across nodes
 - If component behavior must be split across nodes
 - Decompose component into smaller components with more restricted functionality
 - Create component for functionality that must reside on each node

Refine Architecture – LCO Component Model (cont.)

■ Base on past experiences

□ General

- Buschman, F., R. Meunier, et al. (1996). A System of Patterns: Pattern-oriented System Architecture. John Wiley & Sons, Inc.
- Fowler, M. (1997). Analysis Patterns: Reusable Object Models. Addison Wesley Longman, Inc.
- Shaw, M. and D. Garlan (1996). Software Architecture: Perspectives on an Emerging Discipline. Prentice-Hall Inc.
- Szyperski, C. (1997). Component Software: Beyond Object-Oriented Programming. ACM Press Books.

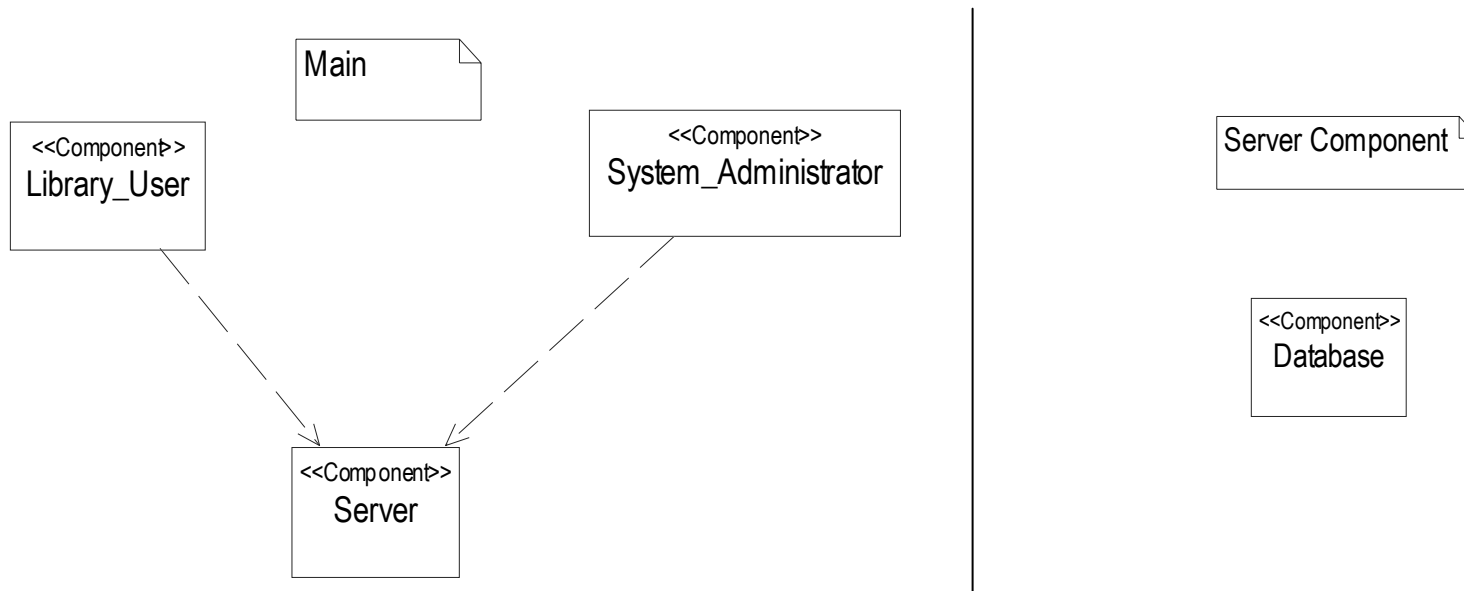
□ Domain-specific books & papers

□ Technology-specific

- During design

Refine Architecture – LCO

Component Model For Full-text Title Database System



- Database component from System Block Diagram
- Decision
 - Client-Server Model
- Note: Database is often deferred until later in development
 - Some classes just described as “persistent” early in process

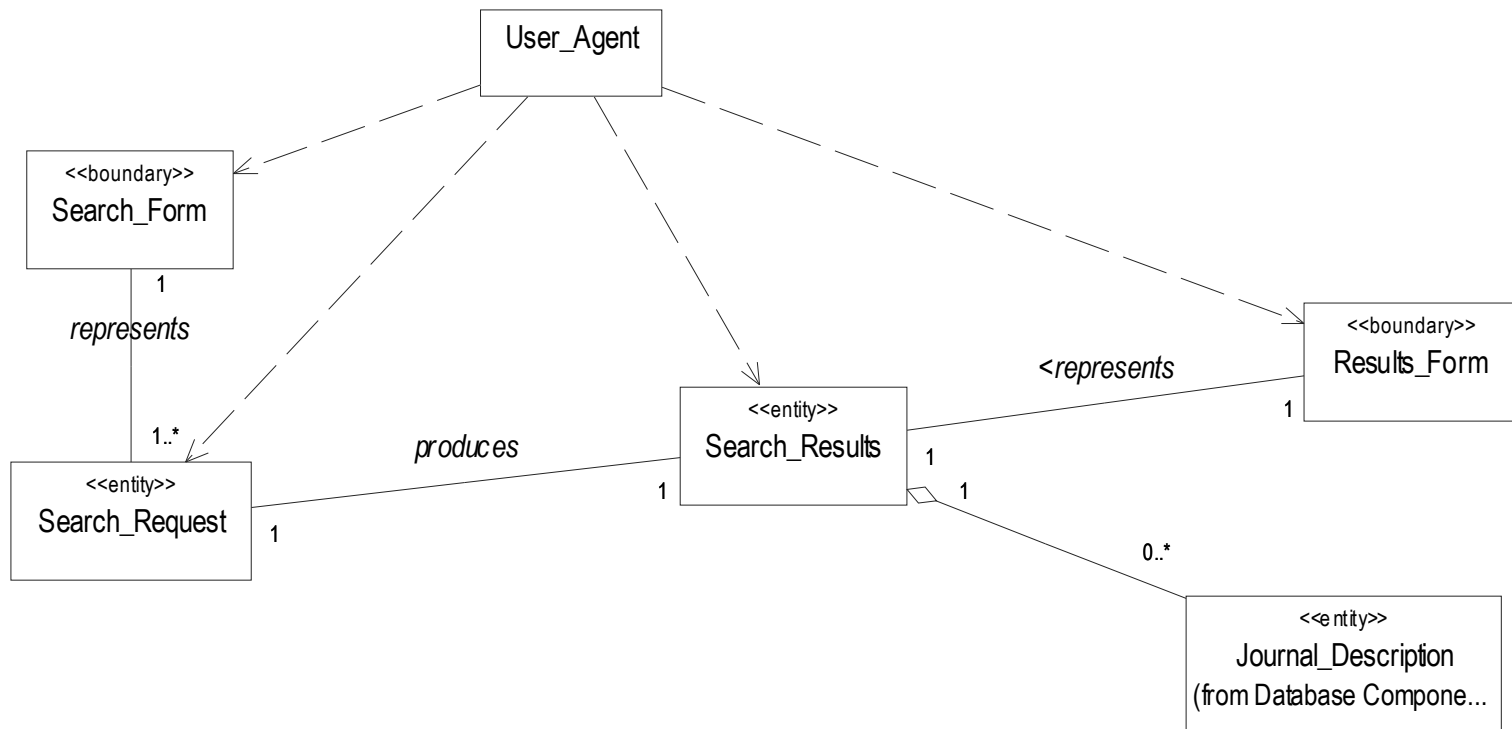
Refine Architecture – LCO

Assign Classes To Components

- Create preliminary Logical Class Model for each component
- Allocate each class in Enterprise Model to component
 - Consider hints for identifying components from classes

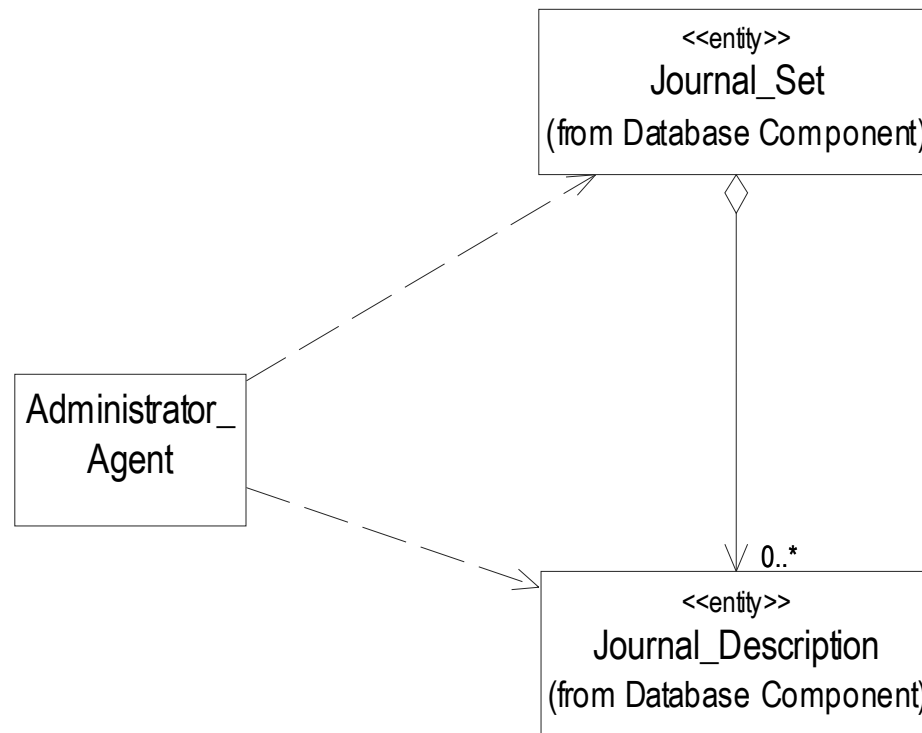
Refine Architecture – LCO

Logical Class Model For Library_User Component



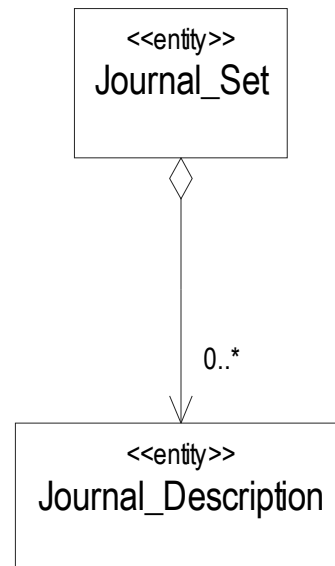
Refine Architecture – LCO

Logical Class Model For System_Administrator Component



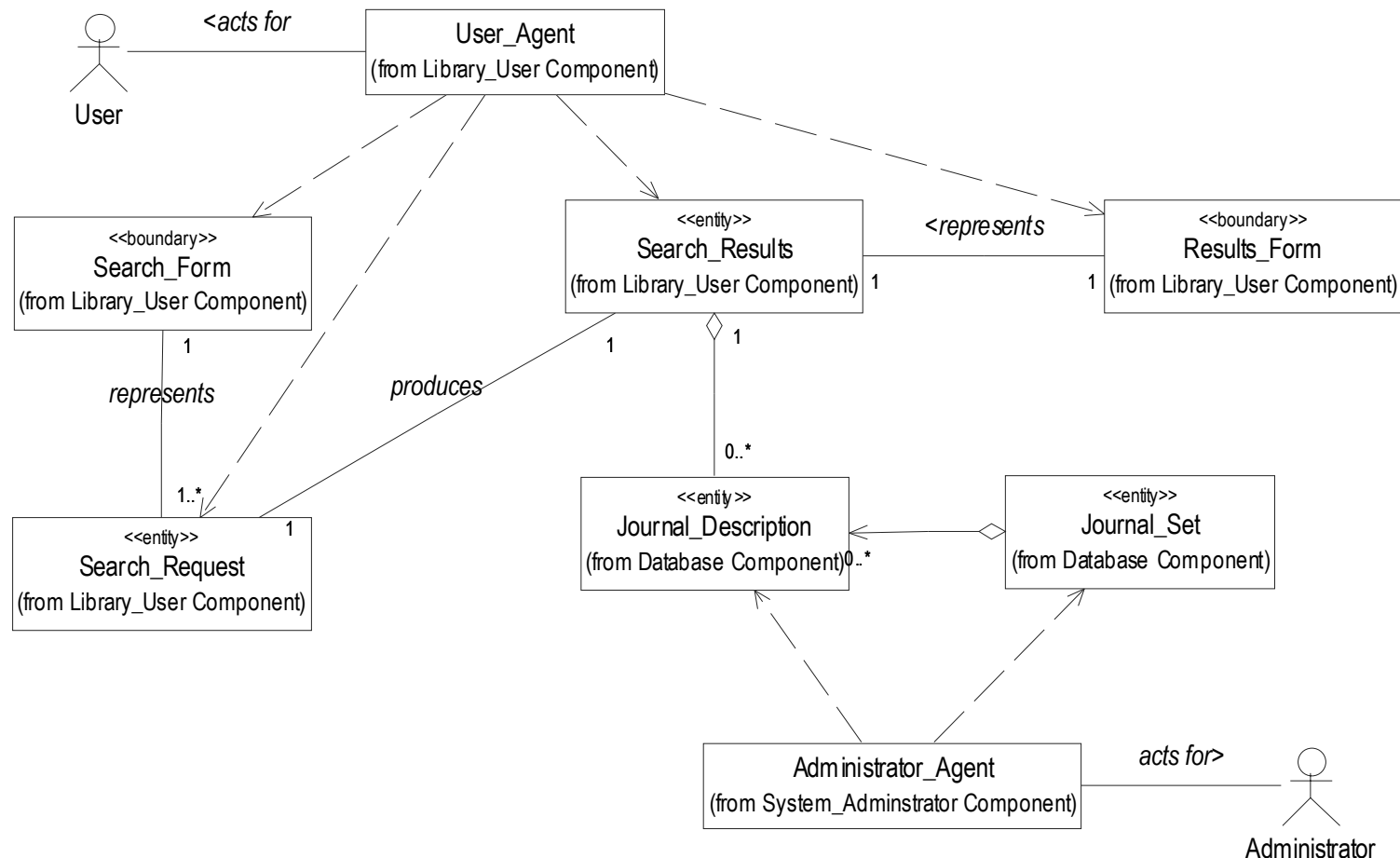
Refine Architecture – LCO

Logical Class Model For Server::Database Component



Refine Architecture – LCO

Enterprise Classification Model for Full-Text Title Database System — Updated



Refine Architecture – LCO

System Topology

- Goal
 - Easier maintenance
- How it works
 - Layering represents an ordered grouping of components
 - Number & composition of layers depends on complexity of
 - Problem domain
 - Solution space
 - Communication typically restricted
 - Component in layer can only request services in same or immediate lower layer
 - Some approaches also allow
 - Component to call services in any lower layer
 - Harder to maintain
 - Call-backs

Refine Architecture – LCO

System Topology – Layering Guidelines

- Number of layers
 - Small systems ~3
 - Large systems 5-7
- Visibility
 - Who can call whom
- Volatility
 - Components implementing
 - Most volatile requirements at top
 - Typically user/application-specific requirements
 - Least volatile requirements at bottom
 - Typically platform-specific
 - Hardware
 - Language
 - OS
 - Database
- Generality
 - By functionality
 - Higher-level functionality at top
 - By concept
 - More abstract model toward bottom
 - More implementation specific toward top
 - Interface-to-Entity
 - Interface-oriented at top
 - Representation-oriented at bottom
 - Application-to-Hardware
 - Application software at top
 - Hardware-specific software at bottom

Refine Architecture – LCO

System Topology – Common Topologies

■ 3-layer

- Application
- Business
- Software System

■ 4-layer

- Application
- Business
- Middleware
- Software System

■ 5-layer

- Application
- Business Services
- Business Objects
- Middleware
- Software System

■ Model-View-Controller

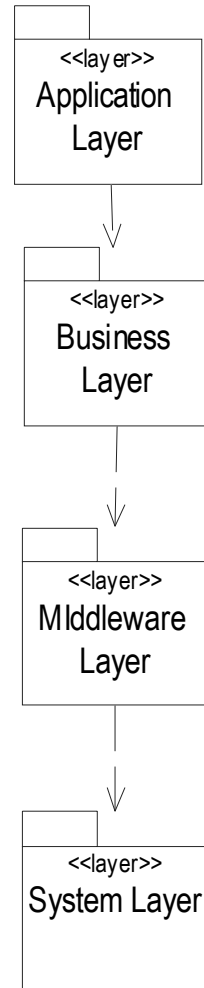
- Interface
- Controller
- Model
- Persistence (optional)

■ C2 Architectural style

- www.ics.uci.edu/pub/arch/c2.html

Refine Architecture – LCO

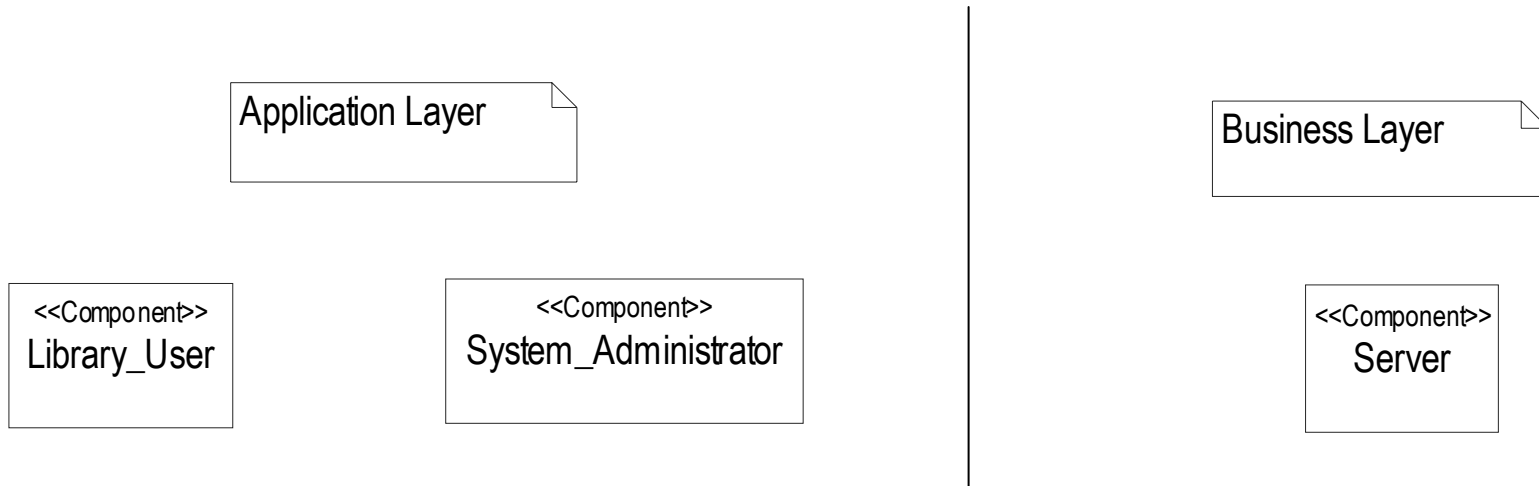
System Topology for Full-Text Title Database System



- Currently only have few classes
- But
 - We've just started
 - Planning for evolution

Refine Architecture – LCO

System Topology for Full-Text Title Database System



- Nothing in other layers
 - yet...

Refine Architecture – LCO System Deployment Model

■ Goal

□ Understand

- Execution environment
- Distribution of software components across environment

■ Model describes

□ Classification of hardware

- “node classes”

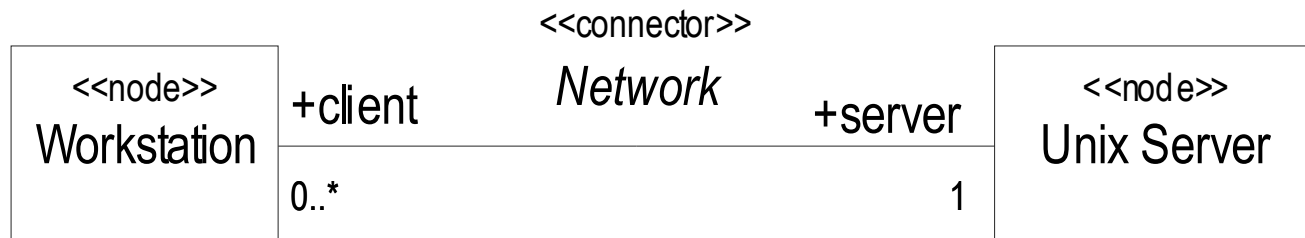
□ Classification of connections

□ Configuration(s) of node instances

□ Allocation of component instances to nodes

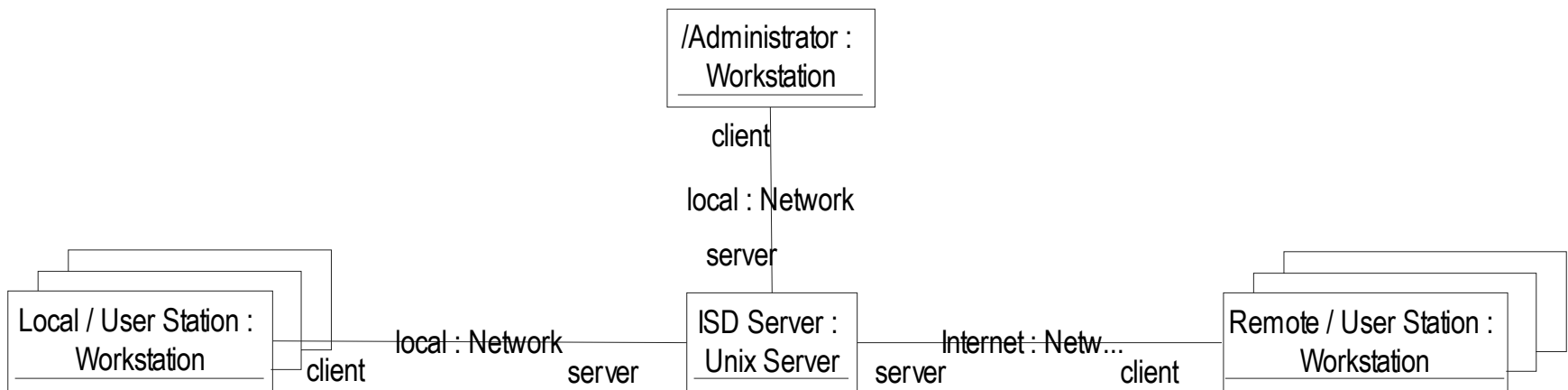
Refine Architecture – LCO

Node Classes for Full-Text Title Database System



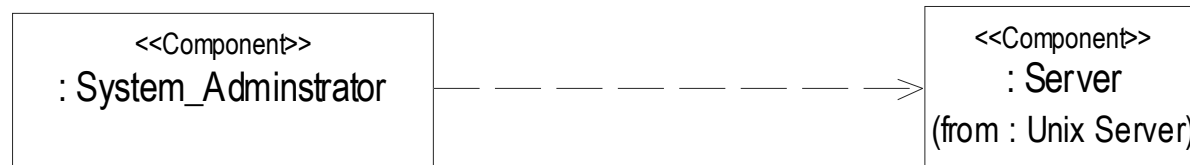
Refine Architecture – LCO

Node Configuration for Full-Text Title Database System



Refine Architecture – LCO

Component Configuration for /Administrator : Workstation



Refine Architecture – LCO

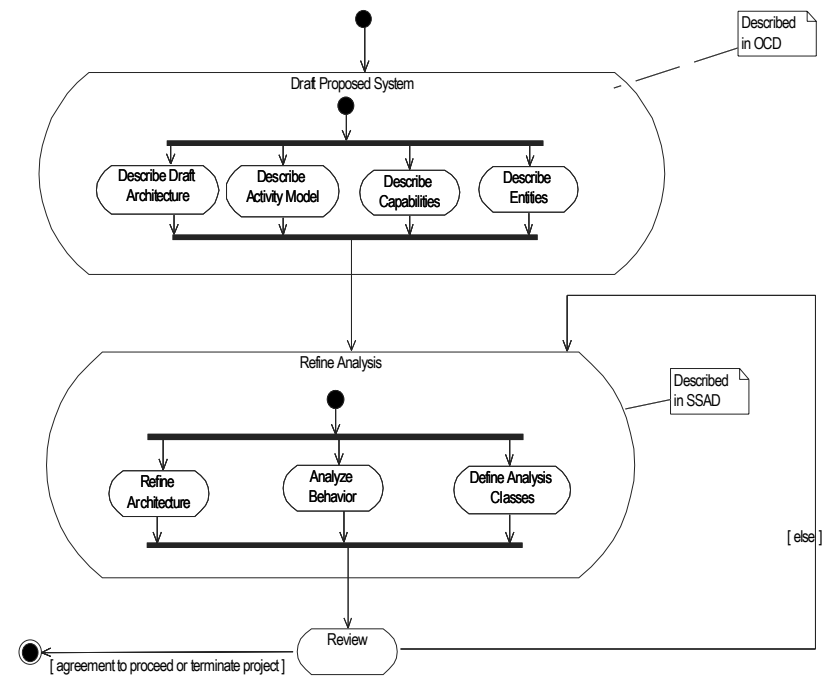
Component Configuration for / User Station : Workstation



System Analysis

What Have We Done?

- Developed a preliminary
 - Architecture model for our system
 - Behavior Model of system
 - Classes model



System Analysis

What Have Left To Do?

- Integrate models
 - Will set of classes & components implement behavior?
 - I'll cover the techniques when we cover system design
- Demonstrate the feasibility of our architecture
- Review

System Analysis Conclusions

- We now have seen
 - How to perform System Analysis
 - Using
 - MBASE
 - Object-oriented techniques
 - RUP
 - Rational Rose
 - How to document analysis
- Time for you to trying on your problem
 - Then you'll really understand!