School of Electrical Engineering & Computer Science @ Ohio University

OHIO UNIVERSITY

CS456/556 Software Design & Development

Analysis

Russ College of Engineering and Technology

RCENT

Analysis

- Analysis is a step between requirement capture and design.
- Analysis introduces notions that represent the *internals* of the system.
- Analysis prepares for the further refinement of the system (design).

- Input: the use case model
- Output: the analysis model
  - the functional model (use cases and scenarios)
  - the analysis object model (class and object diagrams)
  - The dynamic model (statechart and sequence diagrams)

The Analysis Workflow

- Architectural analysis
- Use case analysis
- Class analysis
- Package analysis

The Analysis Workflow (1)

- Architectural analysis
  - identify analysis packages
  - identify obvious entity classes
  - identify common special requirements
- Use case analysis
  - identify analysis classes
  - describe analysis object interactions (using collaboration diagrams)

The Analysis Workflow (2)

- Class Analysis
  - Identify Responsibilities
  - Identify Attributes
  - Identify Associations and Aggregations
  - Identify Generalizations
  - Capturing Special Requirements
- Package Analysis
  - Re-examine the division of analysis packages
  - Key: package dependency

Concepts in Analysis (1)

- **Analysis model**
  - A conceptual object model
  - Objects represent the internals of the system
- **Analysis Package**
  - **A means of organizing the artifacts of the analysis model in *manageable* pieces.**
  - **An analysis package can consist of analysis classes, use-case realizations, and other analysis packages recursively.** (There could be a hierarchy of analysis packages in large systems.)
  - **Key: tightly coupled internally. Loosely coupled externally.**

Concepts in Analysis (2)

- Service
  - A coherent and indivisible set of functionally related actions. (A use case specifies a sequence of such actions.)
- Service package
  - A coherent and indivisible set of functionally related classes.
  - A service package is usually of relevance to only one or a few actors.
  - Its functionality could be managed as a separate delivery unit.
  - Example: Mozilla thunderbird; spellchecker;
  - Or English spellchecker; French spellchecker;

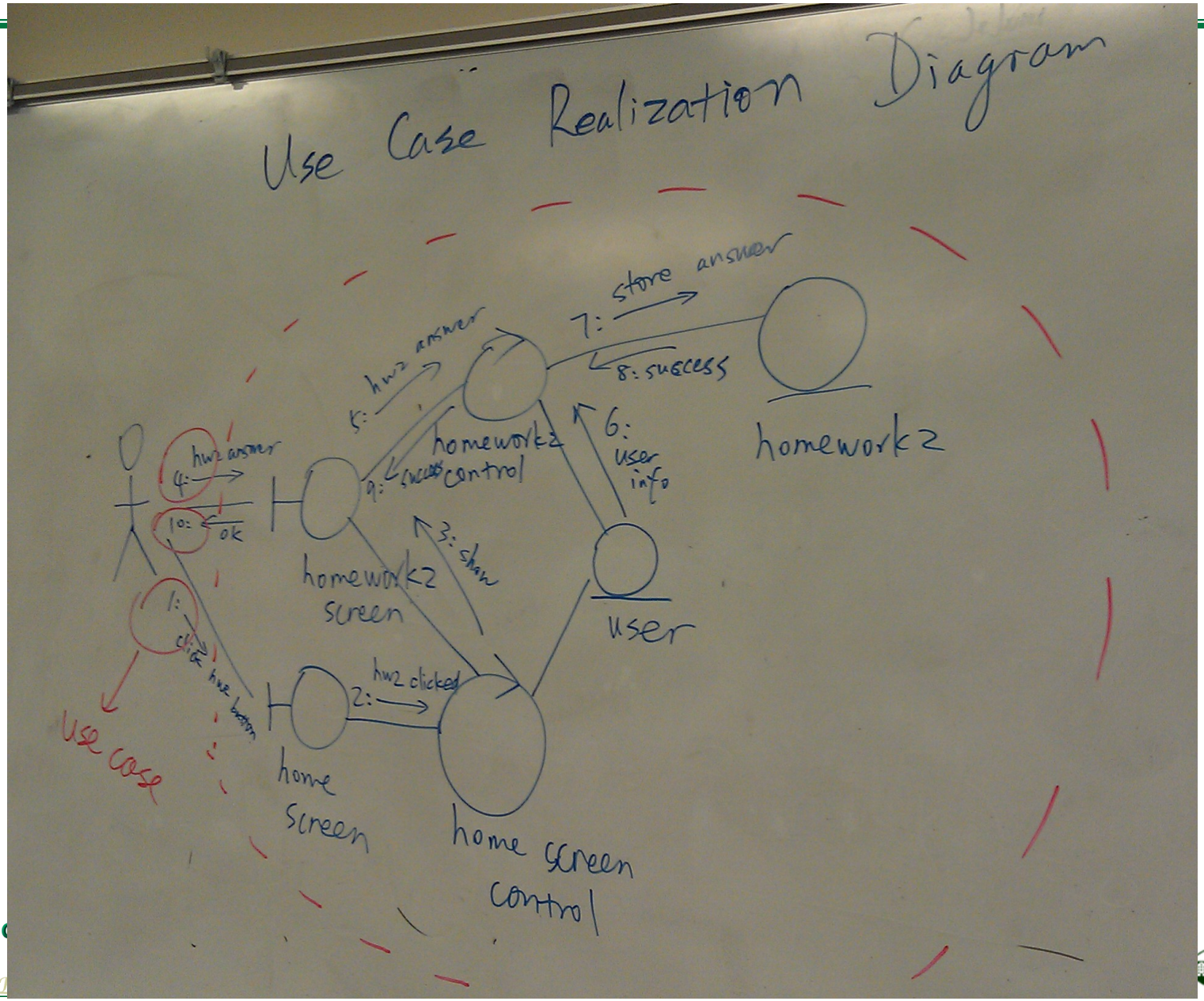
Concepts in Analysis (3)

- **Analysis Class**
  - **An analysis class represents an abstraction of one or several classes and/or subsystems in the system's design**
  - **Boundary Classes**
    - Interacts with actors. Example: Payment Request UI
  - **Entity Classes**
    - Persistent information. Example: Invoice
  - **Control Classes**
    - Coordination, sequencing, transactions, and control of other objects. Example: Payment scheduler

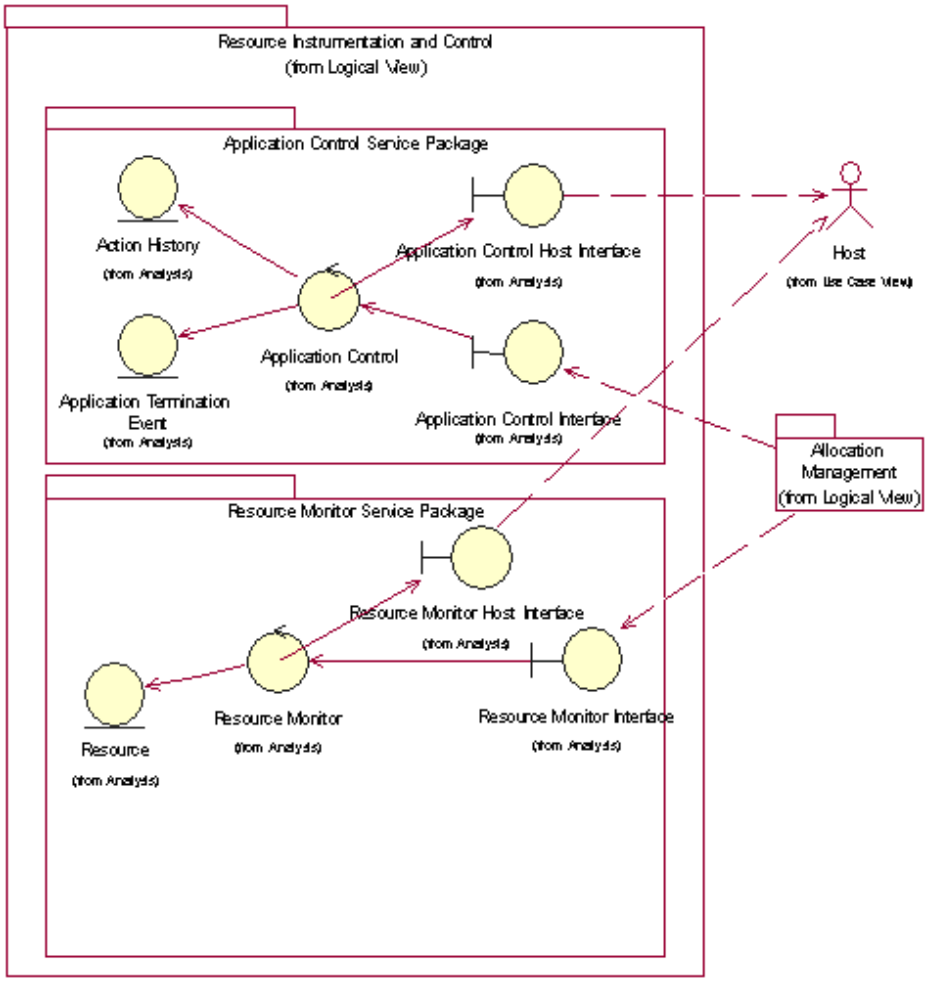
Concepts in Analysis (4)

- **Use-Case Realization**
  - **Class Diagrams**
    - Captures static relations
  - **Interaction Diagrams**
    - Include sequence diagrams and collaboration diagrams
      - Sequence diagrams are used more often in the design stage.
    - Captures dynamic relations
  - **Collaboration Diagrams**
    - To capture chronological sequences of interactions, sequence diagrams should be used

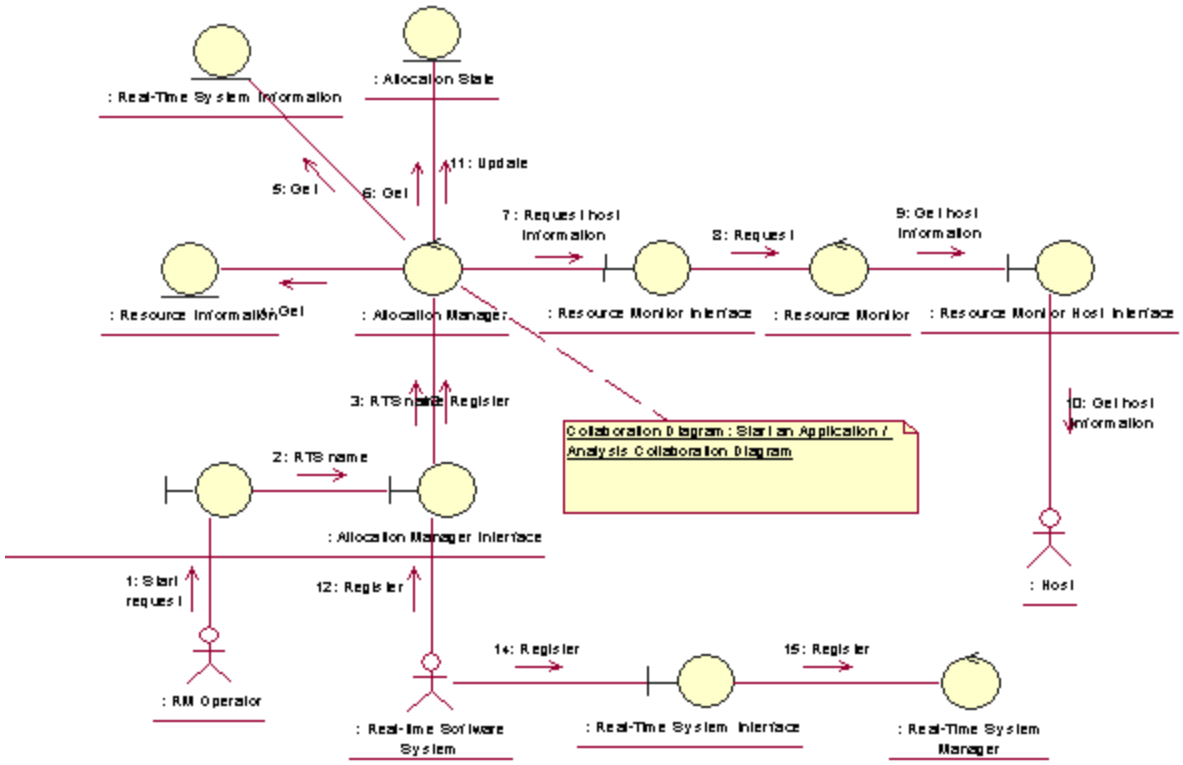
An Example


Use Case Realization Diagram
hw2 answer
4:
10: ok
1:
click hw2 button
use case
homework2 screen
5: hw2 answer
9: success
homework2 control
3: show
2: hw2 clicked
home screen
home screen control
7: store answer
8: success
6: user info
homework2
user

Analysis Class Diagram with Packages



Collaboration Diagram



Architectural Analysis

- To outline the analysis model and the architecture by identifying analysis packages, obvious analysis classes, and common special requirements

Architectural Analysis Workflow

- identify analysis packages
  - Grouping of use cases.
  - (This is just an initial grouping based on the nature of use cases. This grouping may be adjusted later on when analysis classes are identified and analyzed.)
- identify obvious entity classes
  - 10 to 20 in the textbook
  - Most likely much less in our projects.
- identify common special requirements
  - Restrictions and constraints on:
    - Persistence
    - Distribution and concurrency
    - Security features
    - Fault tolerance
    - Transaction management

Analyzing a Use Case

- Identify the analysis classes whose objects are needed to perform the use case's flow of events.
- Distribute the behavior of the use case to interacting analysis objects.
- Capture special requirements on the realization of the use case.

Analyzing a Use Case (Detail 1)

1. Identify entity classes needed to realize the use cases by studying use case and any domain model in detail.
2. Identify the information that is involved and that needs to be manipulated in use case realization.

Analyzing a Use Case (Detail 2)

3. Identify one central boundary class that interacts with the primary actor, if needed, and let this be represented in the primary window in user interface.
4. Identify a primary boundary class for each entity class found in step 1.

Analyzing a Use Case (Detail 3)

5. Identify a central boundary class for each external system actor and let this represent communication interface.
6. Start with one control class responsible for handling the control and coordination of the use case realization and then refine it.

Keep in mind classes identified in previous use-case realizations of other use-cases.

Class Analysis

- Identify Responsibilities
  - A combination of the roles it plays in different use-case realizations.
- Identify Attributes
  - Noun! (Will turn into data members eventually)
- Identify Associations and Aggregations
- Identify Generalizations
- Capturing Special Requirements

  - Note: Aggregation: individual versus collection
  - Generalization: concrete versus abstract

Package Analysis

- Grouping analysis classes according to the grouping of use cases as identified in architectural analysis.
- Goal: to encapsulate as many dependencies inside packages as possible.