Introduction to Unified Modeling Language

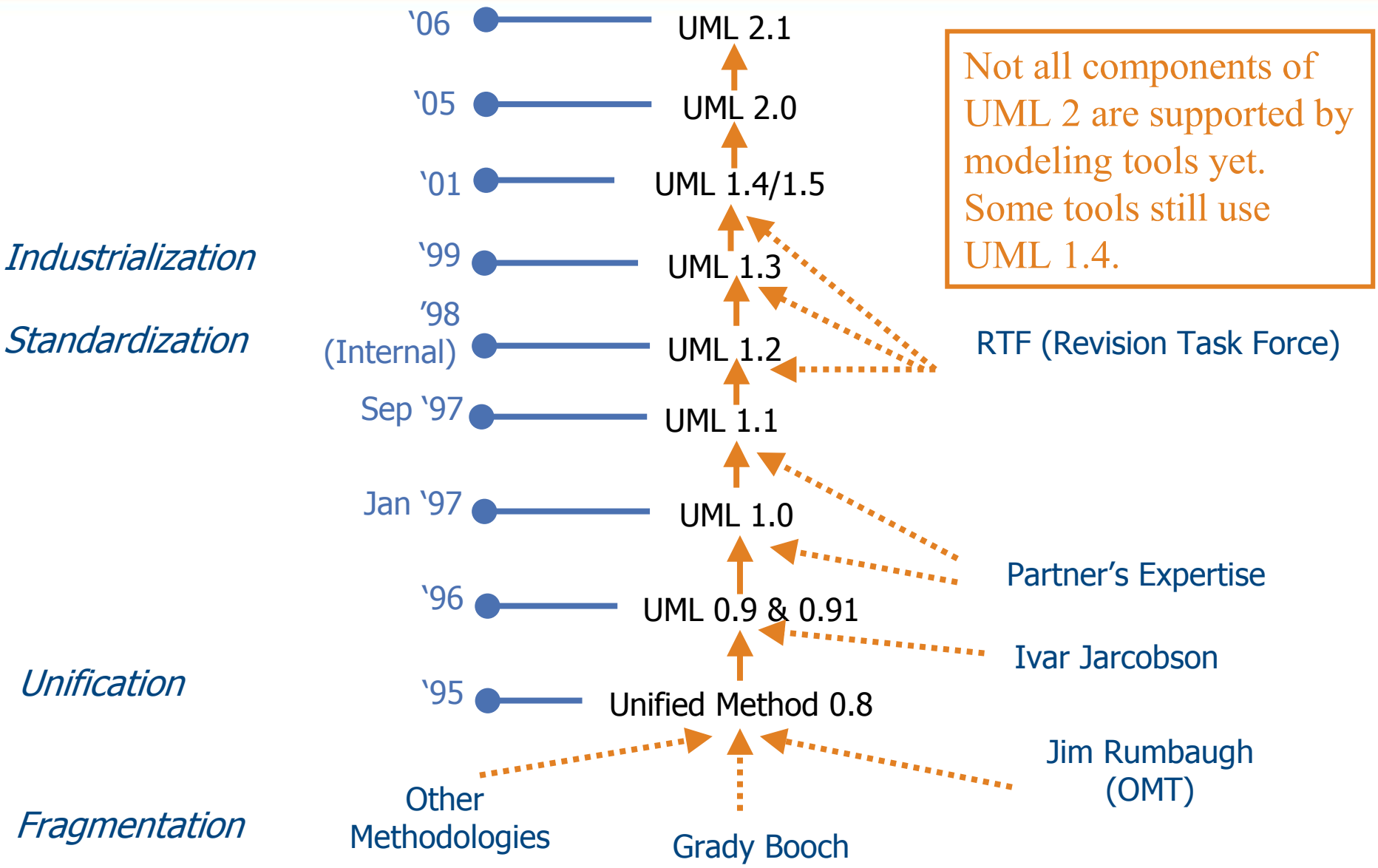
Overview of architectural views and UML 2 diagrams

What Is UML?

- A language (notation) for modeling object-oriented systems
- A standard maintained by the Object Management Group
- A modeling language including 13 diagrams
- A means for visualizing, specifying, constructing, and documenting software systems

→ http://www.uml.org

Long Story of UML



Why Do We Model?

- Furnish abstractions to manage complexity
- Provide structure for problem solving
- Experiment to explore multiple solutions

Modeling allows the following business benefits:

- Reduce time-to-market for business problem solutions
- Decrease development costs
- Manage the risk of mistakes

Why Do We Need UML?

- Graphical notation
  - A picture is worth a thousand words
- Standard communication language
- Provides multiple diagrams for capturing different architectural views
- Promotes component reusability

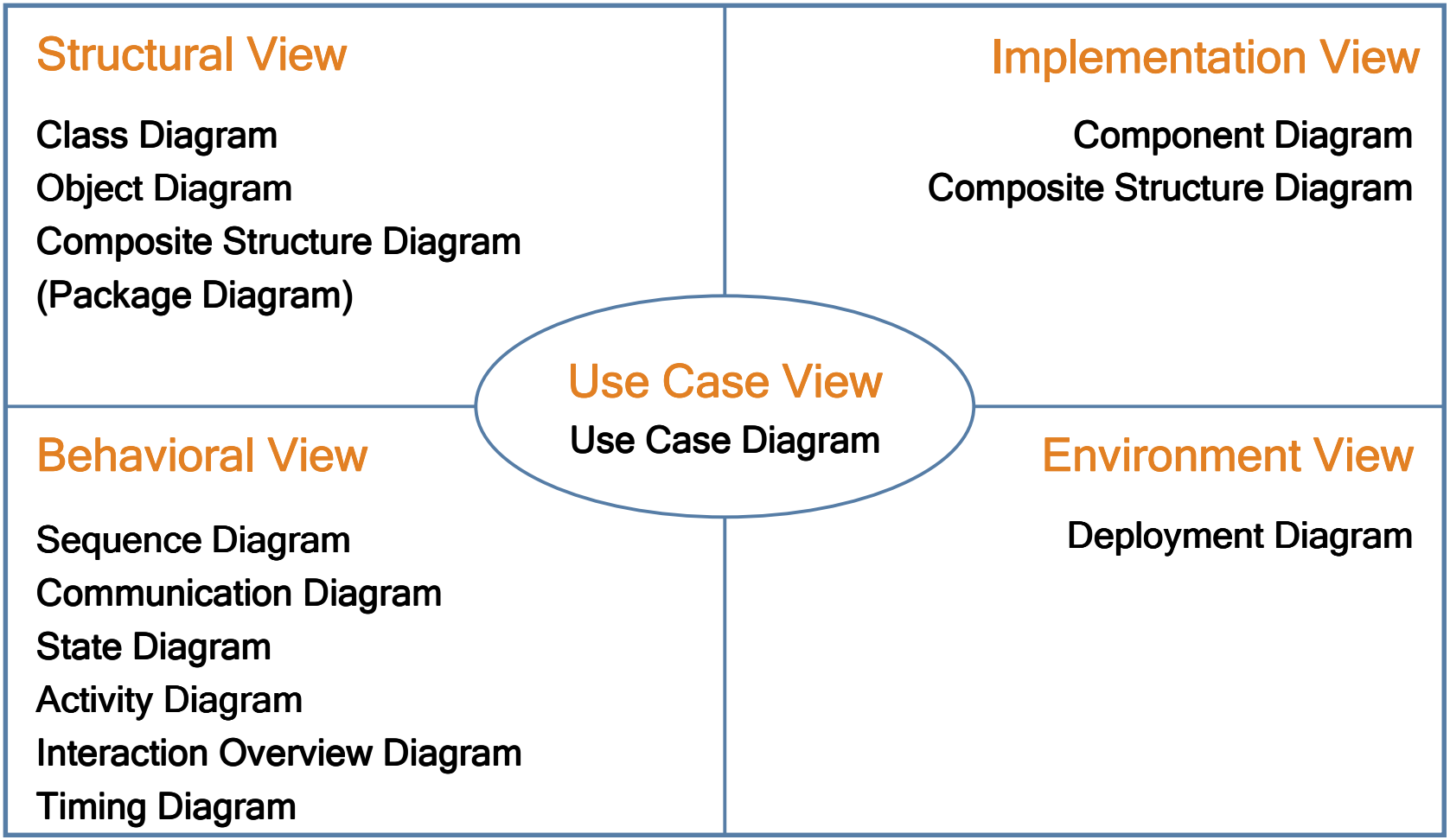
UML is a standard language for visualizing, specifying, constructing, and documenting software systems

How Can We Benefit from Using UML Modeling Tool?

- Repository of reusable model artifacts
- Visualize in multiple dimensions and levels of detail
- Use automated layout and visualization tools
- Harvest models from legacy systems
- Generate documentation from modeling environment
- Analyze traceability through relationships between elements
- Incremental development and refactoring
- Teamwork for parallel development of large systems
- Integration with other development tools

UML Architectural Views and Diagrams

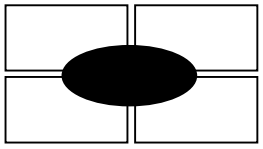
UML defines 13 diagrams that describe 4+1 architectural views

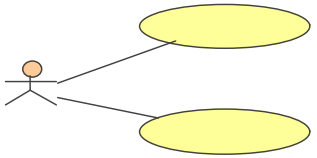


4+1 architectural views model was proposed by Philippe Kruchten, IBM

Use Case View

- The most important architectural view
- Describes use cases that provide value for the users
- Essential use cases are used as proof of concept for implementation architecture
- Use cases may be visualized in UML use case diagram
- Each use case may have multiple possible scenarios
- Use case scenarios could be described:
  - Using textual descriptions;
  - Graphically, using UML activity diagrams.

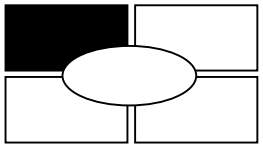


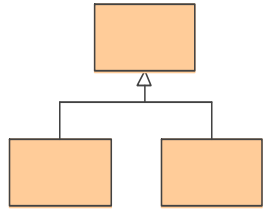


Structural View

- Represents structural elements for implementing solution for defined requirements
- Defines
  - Object-oriented analysis and design elements;
  - Domain and solution vocabulary;
  - System decomposition into layers and subsystems;
  - Interfaces of the system and its components.

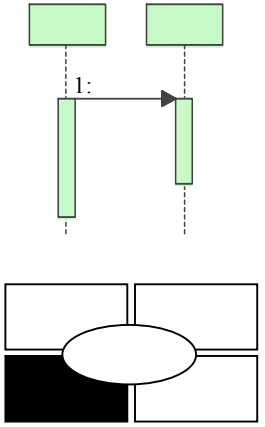
- Is represented by static UML diagrams:
  - Class diagrams in multiple abstraction levels;
  - Package diagrams;
  - Composite structure diagrams (new in UML 2).





Behavioral View

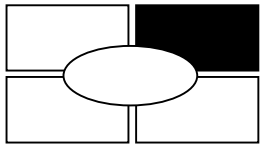
- Represents dynamic interaction between system components for implementing requirements
- Shows distribution of responsibilities
- Allows to identify interaction and coupling bottlenecks
- A means for discussing non-functional requirements
  - Performance, maintenance, …
- Is especially important for distributed systems
- Is represented by dynamic UML diagrams:
  - Sequence and/or communication diagrams;
  - Activity diagrams;
  - State diagrams;
  - Interaction overview diagram (new in UML 2);
  - Timing diagrams (new in UML 2).

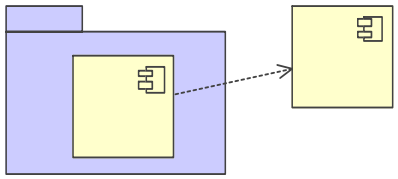


Implementation View

- Describes implementation artifacts of logical subsystems defined in structural view;
- May include intermediate artifacts used in system construction (code files, libraries, data files, …)
- Defines dependencies between implementation components and their connections by required and provided interfaces

- Is represented by these UML diagrams:
  - Component diagrams;
  - Composite structure diagrams (new in UML 2).

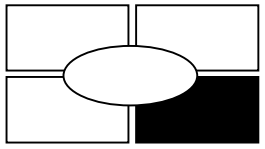


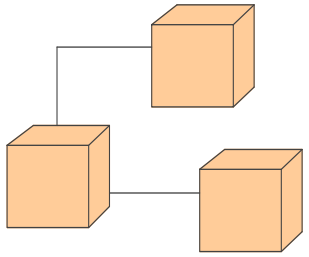


Environment View

- Represents system's hardware topology
- Defines how software components are deployed on hardware nodes
- Useful for analyzing non-functional requirements
  - Reliability, scalability, security, …
- Provides information for system installation and configuration

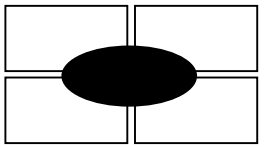
- Is represented by
  - UML deployment diagram

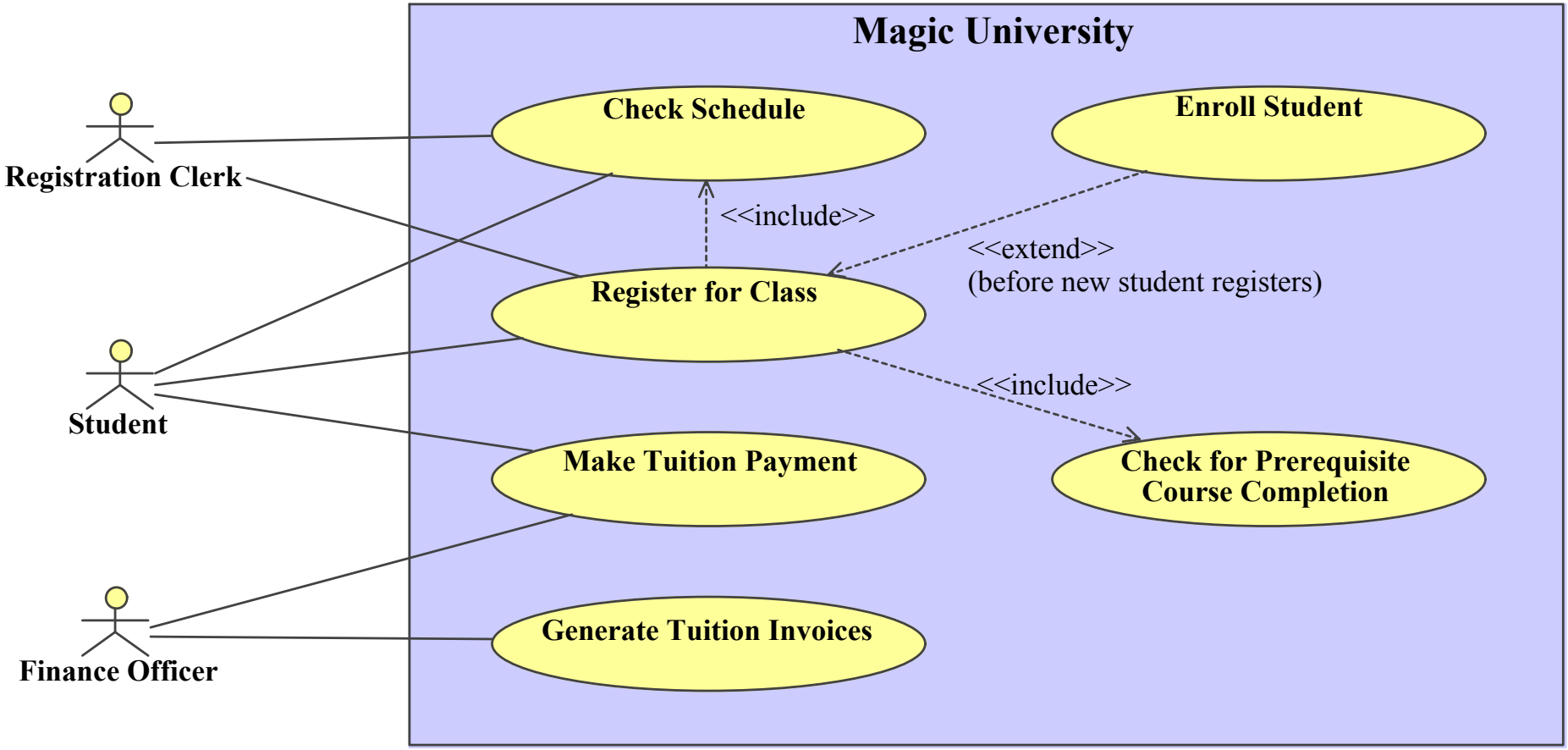




Use Case Diagram

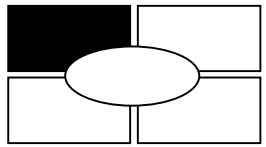
- Describes the functionality provided by system
- Contains actors, use cases, and relationships

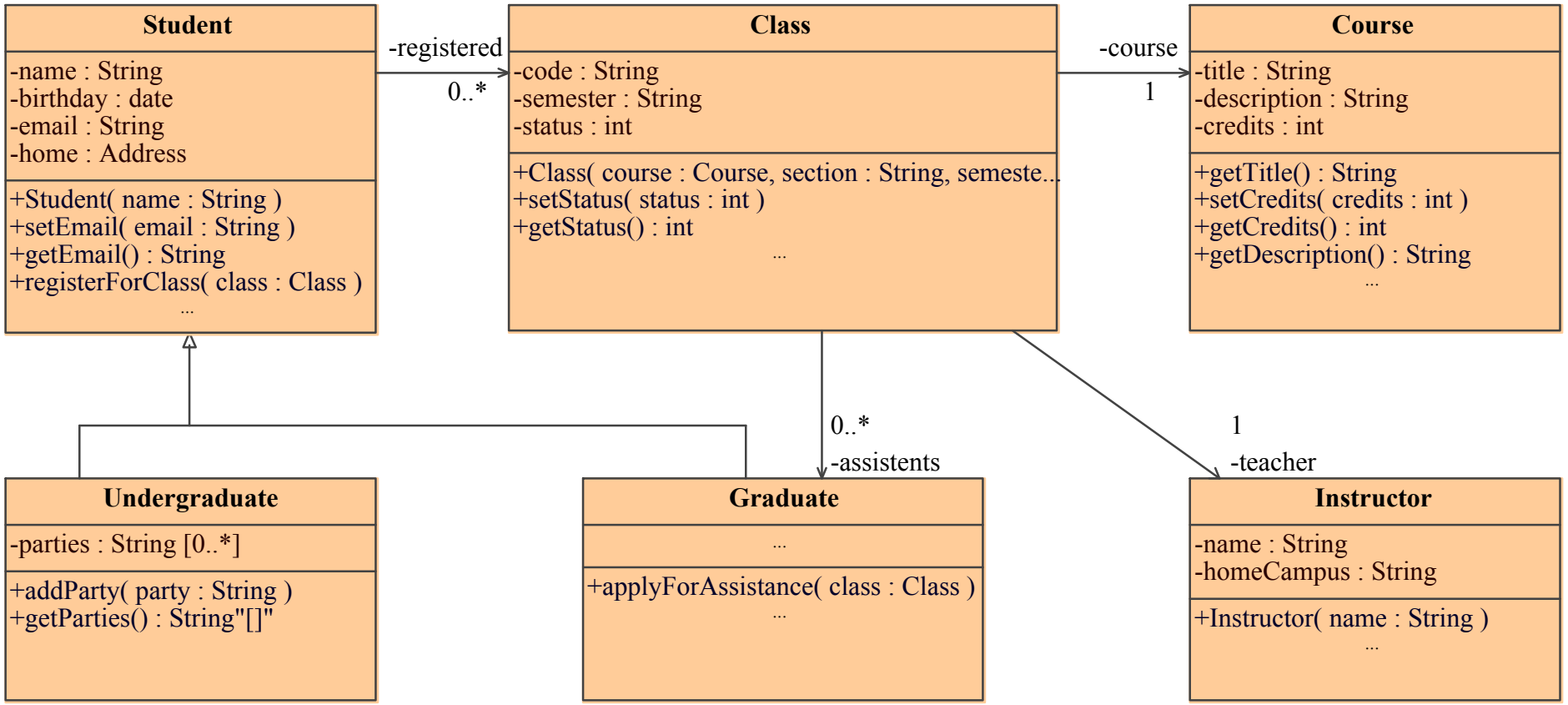




Class Diagram

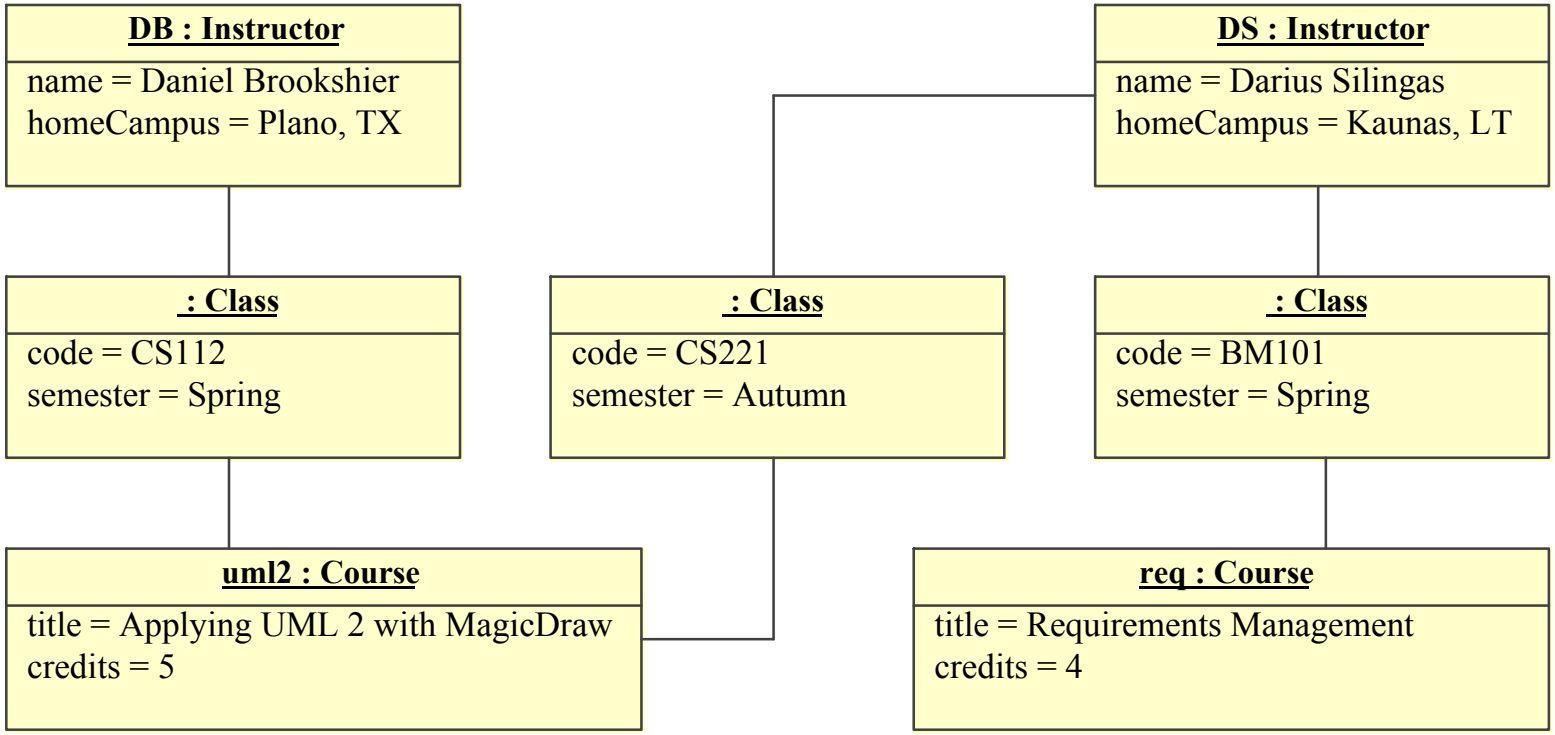
- Describes static structure of the system
- Contains classes and relationships

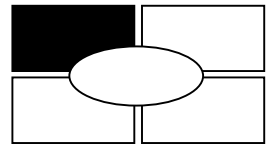




Object Diagram

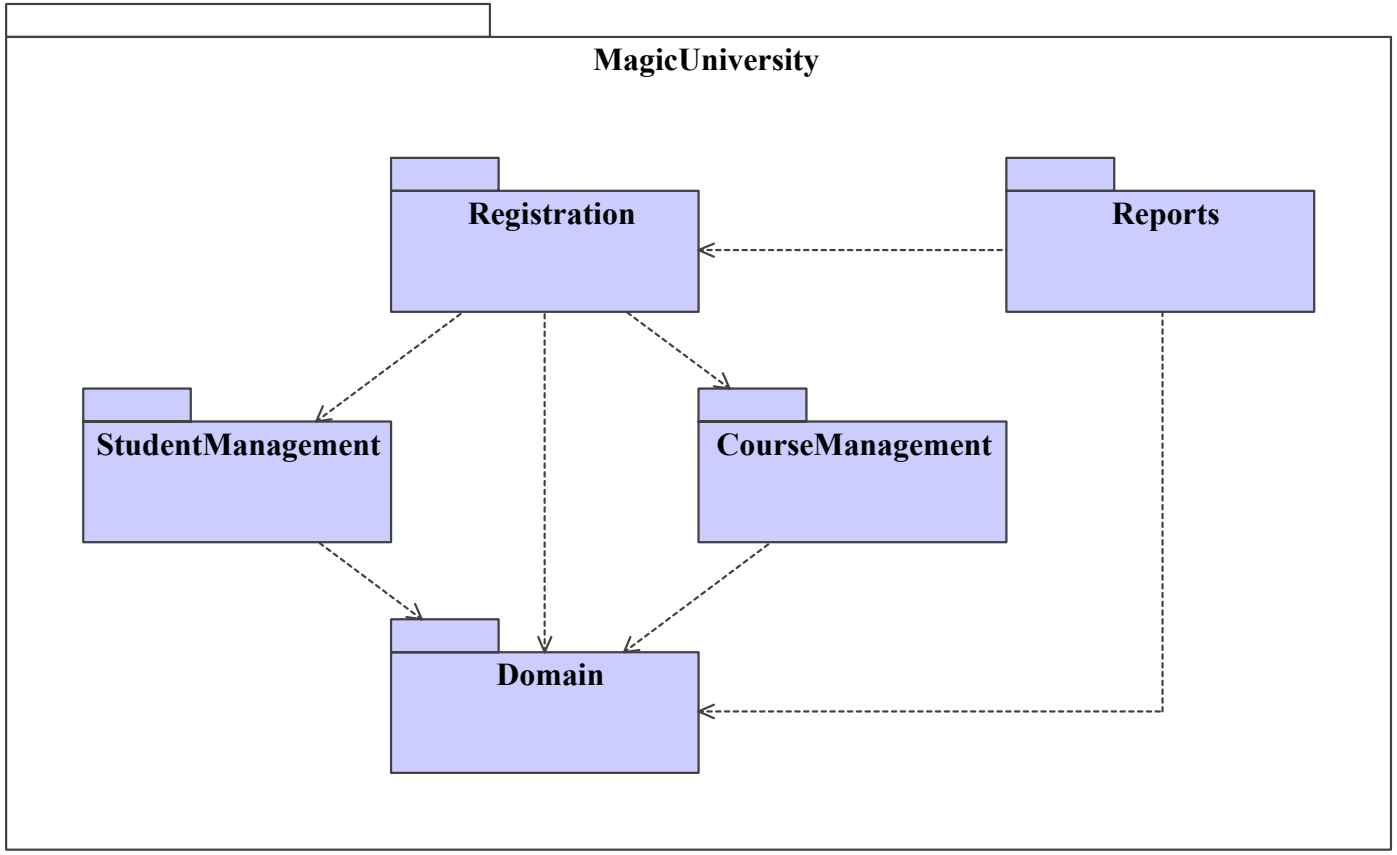
- Shows an example of objects with slots and links that could be instantiated from defined classes and relatioships
- Validates class diagrams

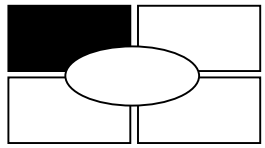




Package Diagram

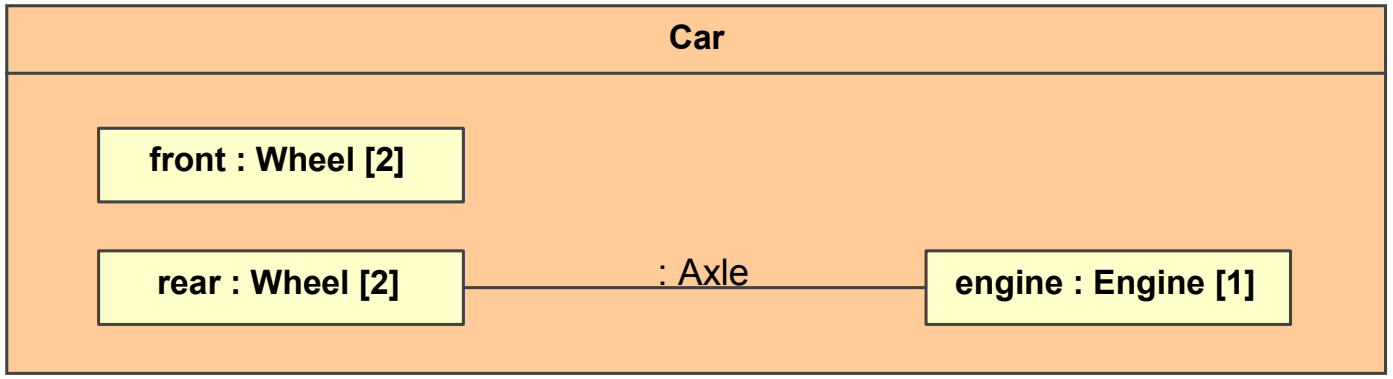
- Decomposes system into logical units of work
- Describe the dependencies between logical units of work
- Provide views of a system from multiple levels of abstraction

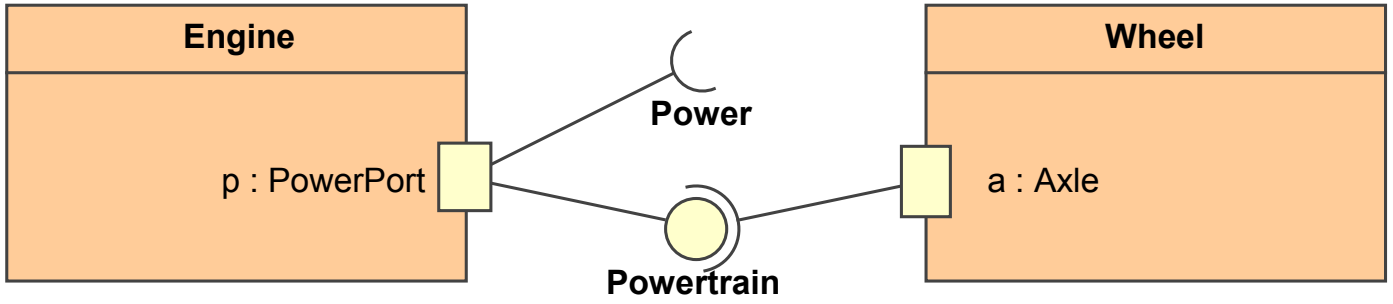


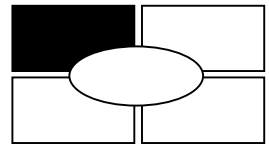


Composite Structure Diagram (1)

- Shows the internal structure of a classifier, including its interaction points to other parts of the system
- More useful for modeling hardware, real-time systems, integrated device modeling

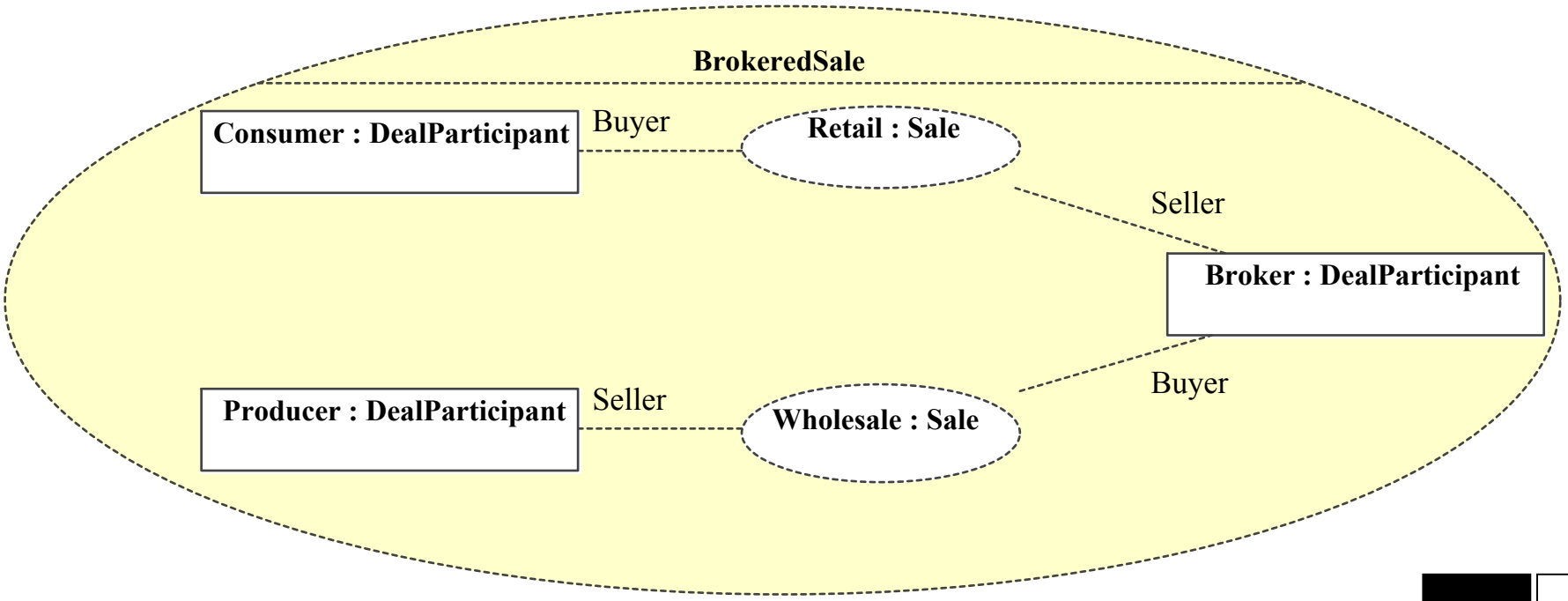


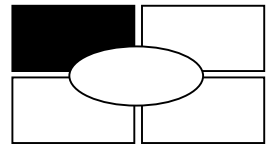




Composite Structure Diagram (2)

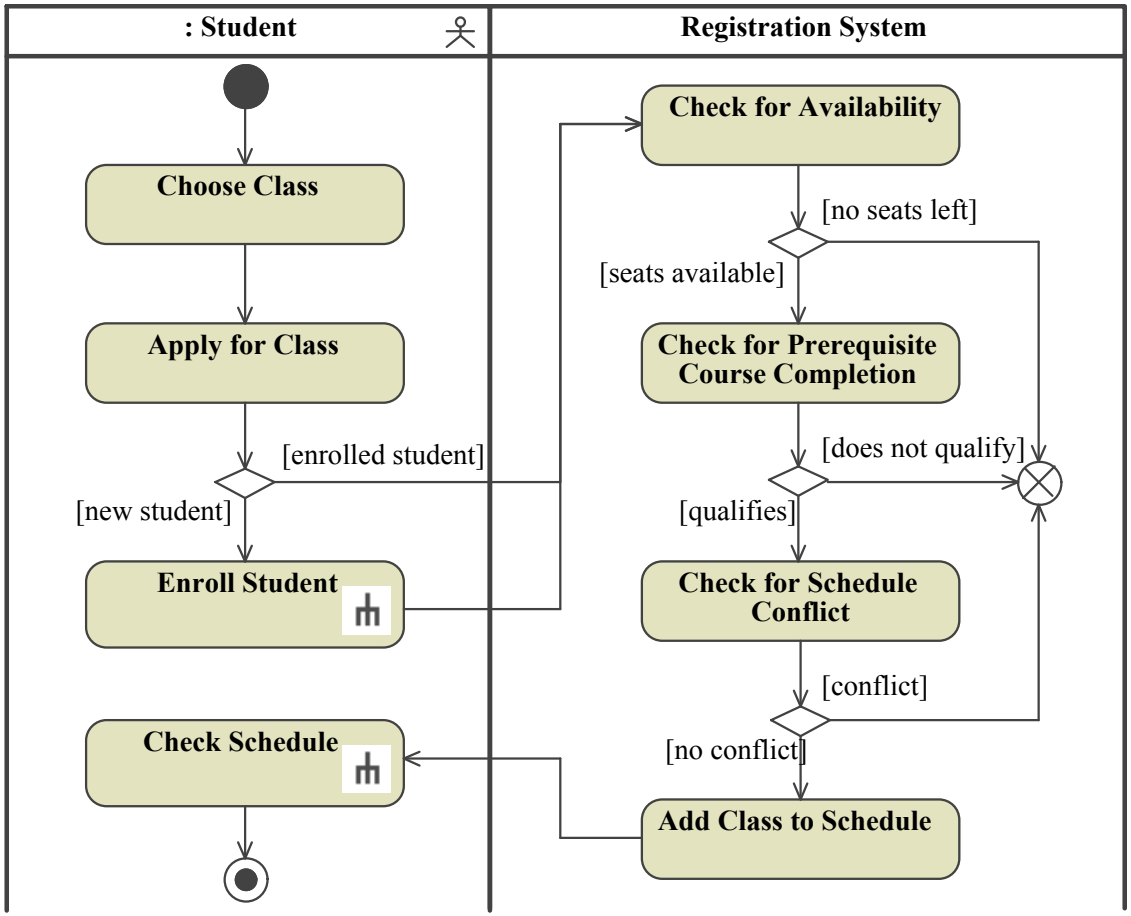
- Shows the configuration and relationship of parts that together perform the behavior of the containing classifier
- Useful for defining static structure of collaboration patterns

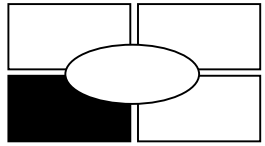




Activity Diagram

- Shows a procedural flow for a process
- Useful for workflow modeling
- Supports parallel behavior for multithreaded programming

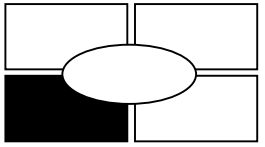




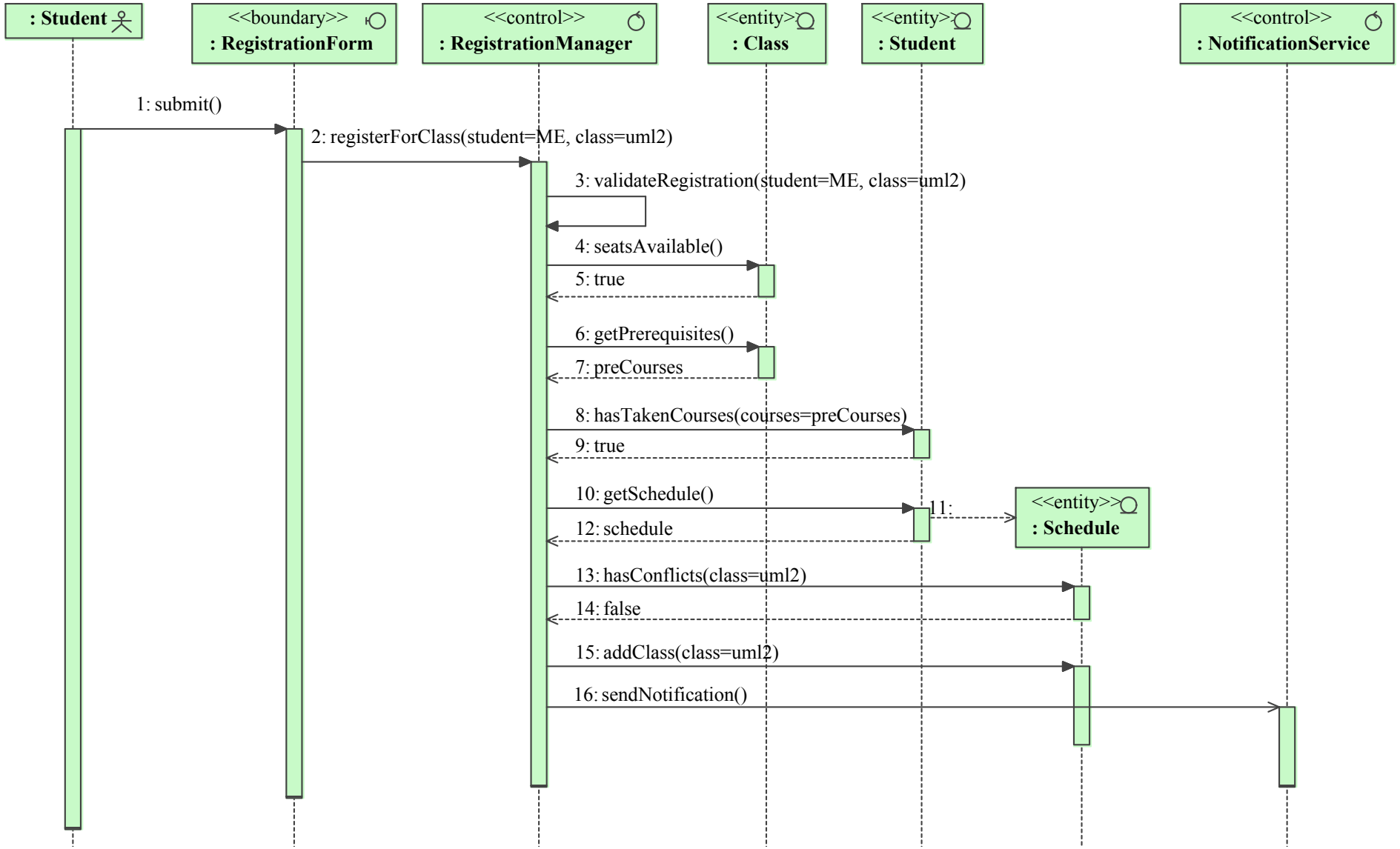
Sequence Diagram (1)

- Describes how a process is performed by a group of objects by a sequential set of interactions
- Provides an object-oriented view of a procedural views
- Facilitates assignment of responsibilities to classes
- Helps finding out new methods and new classes
- Shows timing very explicitly

- (Diagram on next slide)

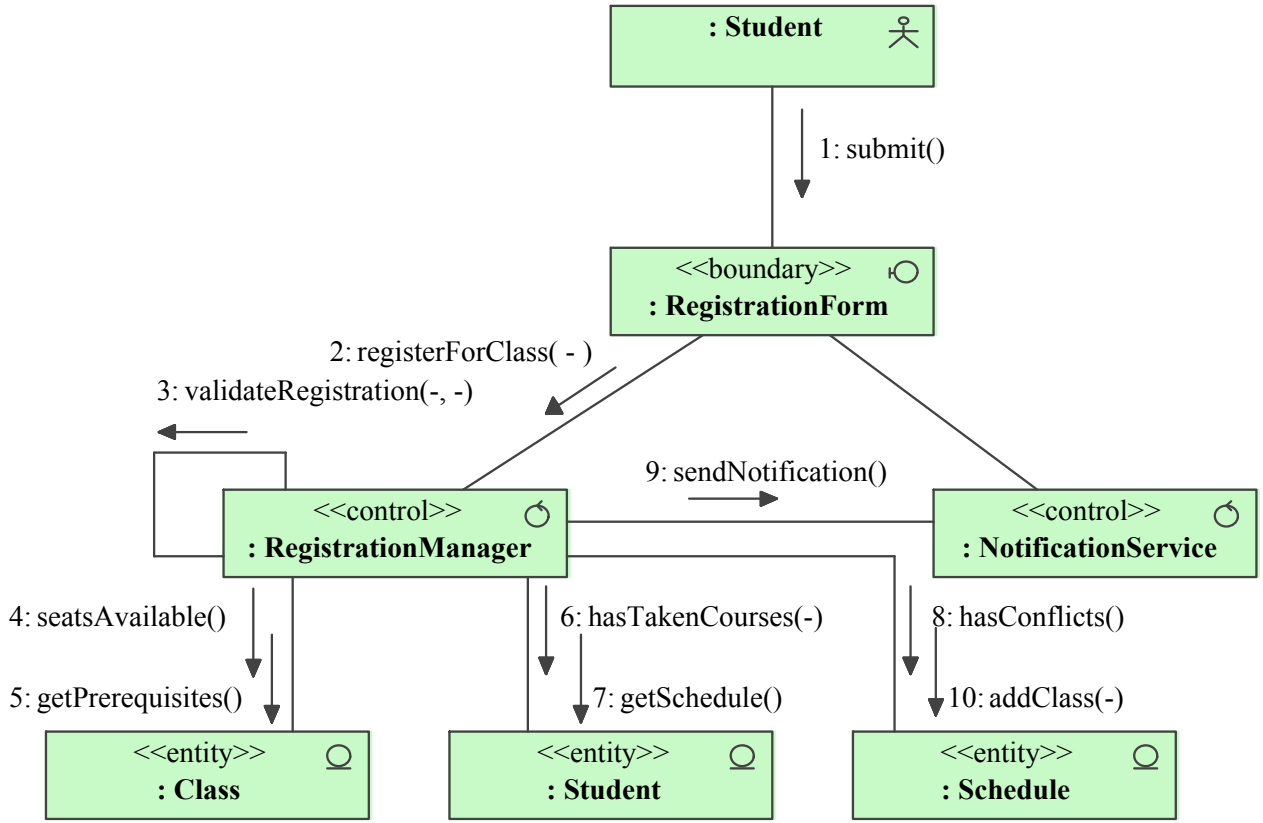


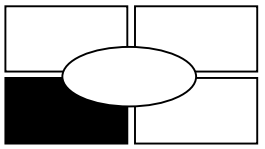
Sequence Diagram (2)



Communication Diagram

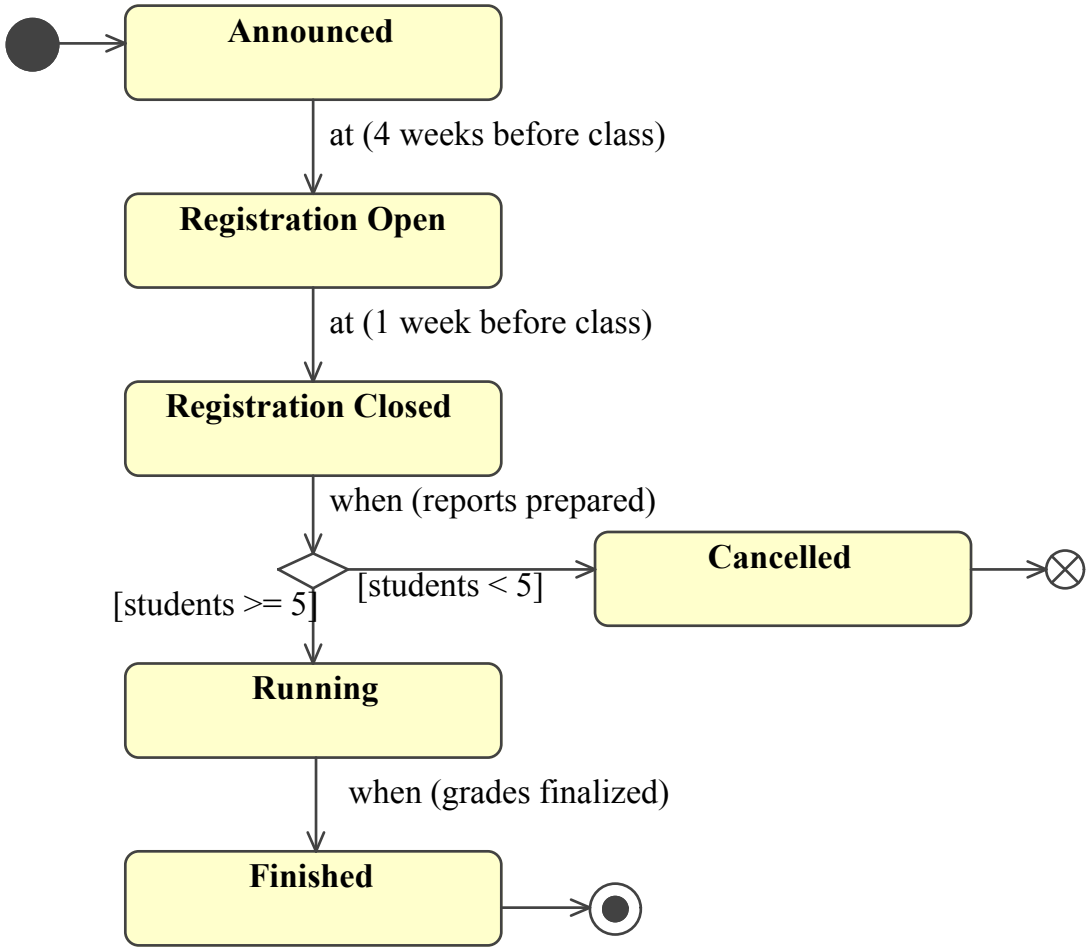
- Provides an alternative view to the sequence diagram in a format based on structure rather than time
- Emphasizes how objects interact with each other
- More efficient use of space

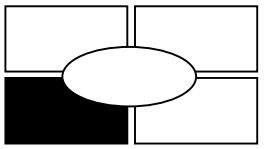




State Diagram

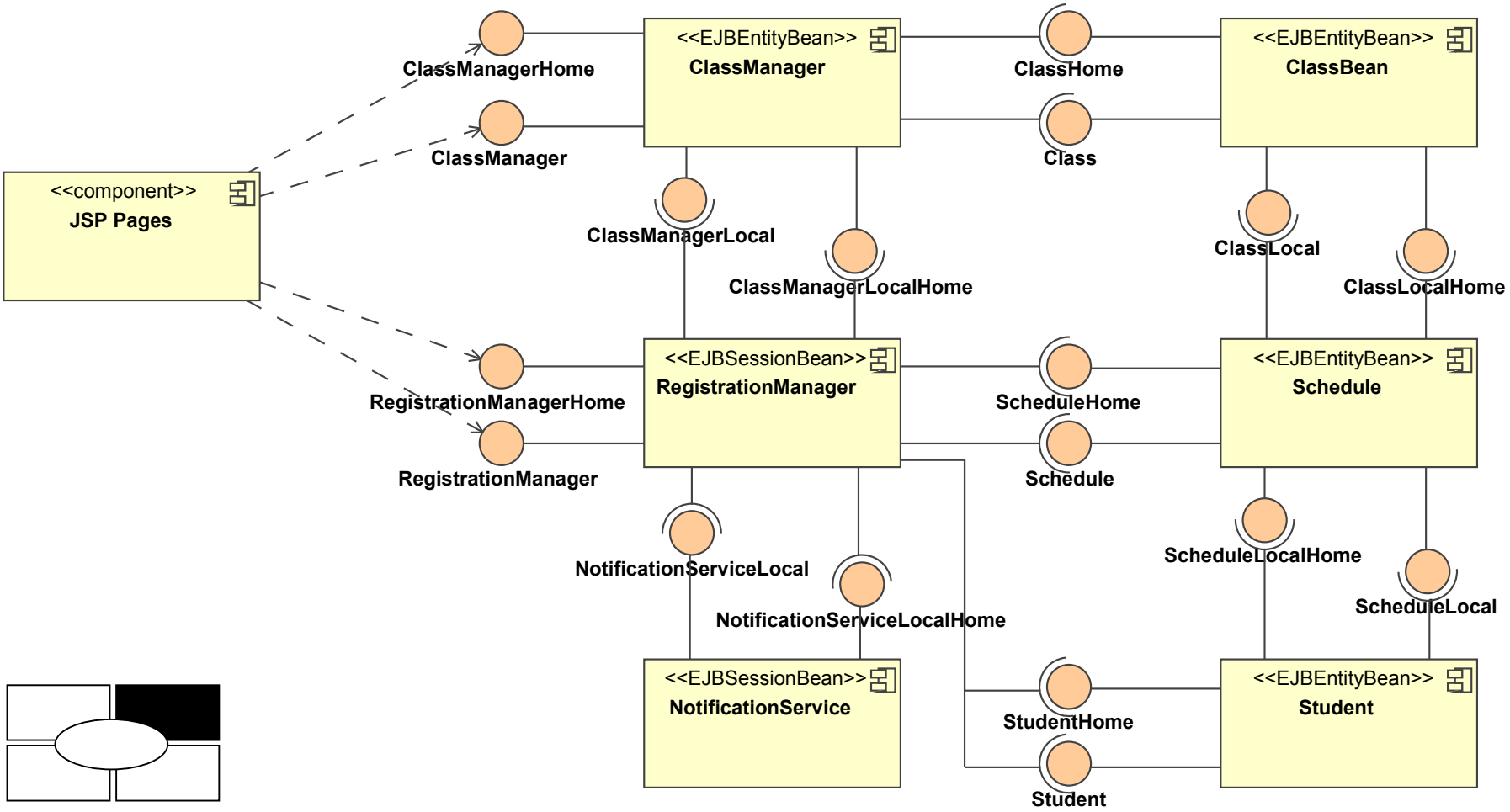
- Describes how an object changes its state that govern its behavior in response to stimuli from the environment





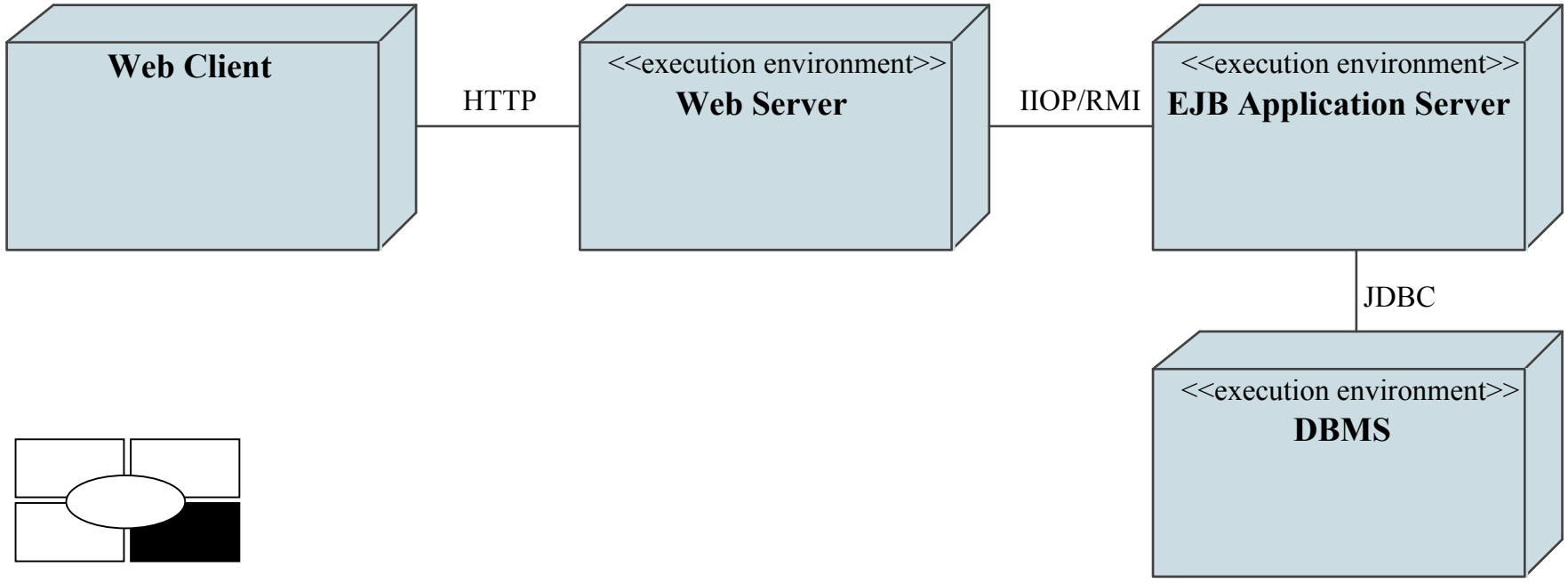
Component Diagram

- Describes software components that make up a system, their interfaces (optional) and relationships



Deployment Diagram

- Describes the configuration of hardware in a system in terms of nodes and connections
- Describes the physical relationships between software and hardware
- Displays how artifacts are installed and move around a distributed system



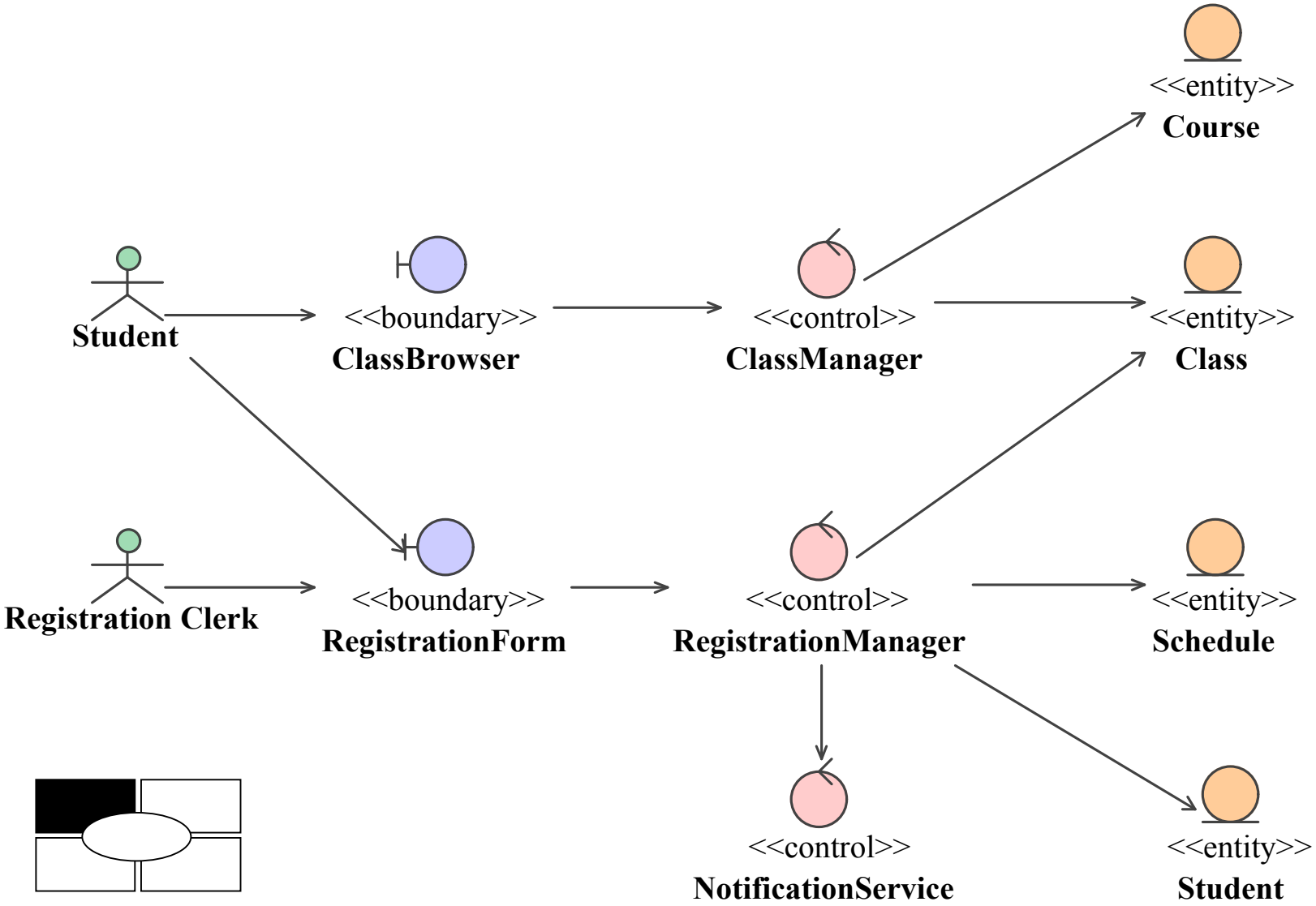
But That's Not All…

UML provides extensions to the language to create new types of diagrams

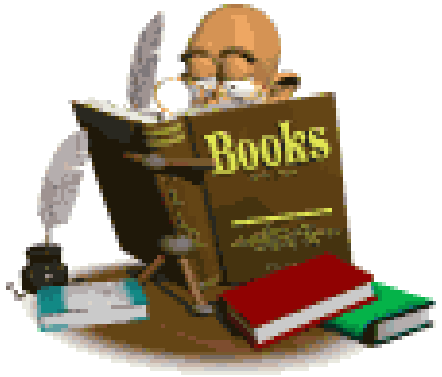
UML Profiles define a set of extensions for a specific usage, e.g. new domains, technologies, or methods

- Stereotypes «Process»
- Tagged Values approval_status="draft"
- Constraints {deliver within 48 hours}
- Customizable Icons

Extending UML – Robustness Diagram



Where To Go To Learn More



UML Web Resources:

- http://www.objectsbydesign.com
  - UML and OO links, forums, and resources
- http://www.devx.com/uml/
  - UML developer zone
- http://www.sdmagazine.com/
  - Magazine with many UML related articles
- http://www.omg.org
  - The UML Specification and other UML resources

UML Books

- UML Bible by Tom Pender
- UML Distilled by Martin Fowler & Kendal Scott
- Applying UML & Patterns by Craig Larman