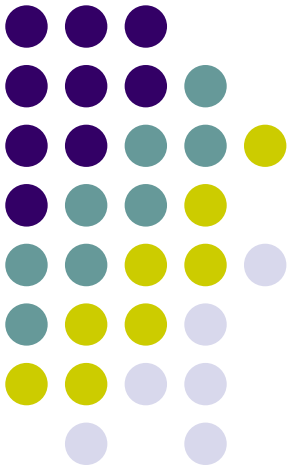
SCOUPE

Systems Engineering Tool designed to guide engineers through the process of front-end system development



Faculty: Dr. Mark Austin, Dr. John S. Baras,
Developers: Natasha Kositsyna, Vijay Krishnamurthy, Shah-An Yang

Why SCOUPE?

- Many engineers (students and industry personnel) have difficulties understanding/learning/practicing the elements of typical Systems Engineering Processes.
- They need a tool that would help them organize ideas and thoughts into a cohesive and consistent system.
- They need to learn basic techniques on visual modeling along the way (i.e., UML).
- They also need tools to facilitate the use of modern formal and visual tools, in a way that lets the engineers focus on the engineering task at hand.
- They need to understand how to model system behavior and system structure, and how to map fragments of behavior onto fragments of system structure (i.e. system components).

The Big Picture

- In ENSE 621/622/623 we promote systems engineering development procedures that employ **semi-formal and formal models** of the requirements and design, connected by **mappings** and **traceability**. This is facilitated by selected software SE tools.

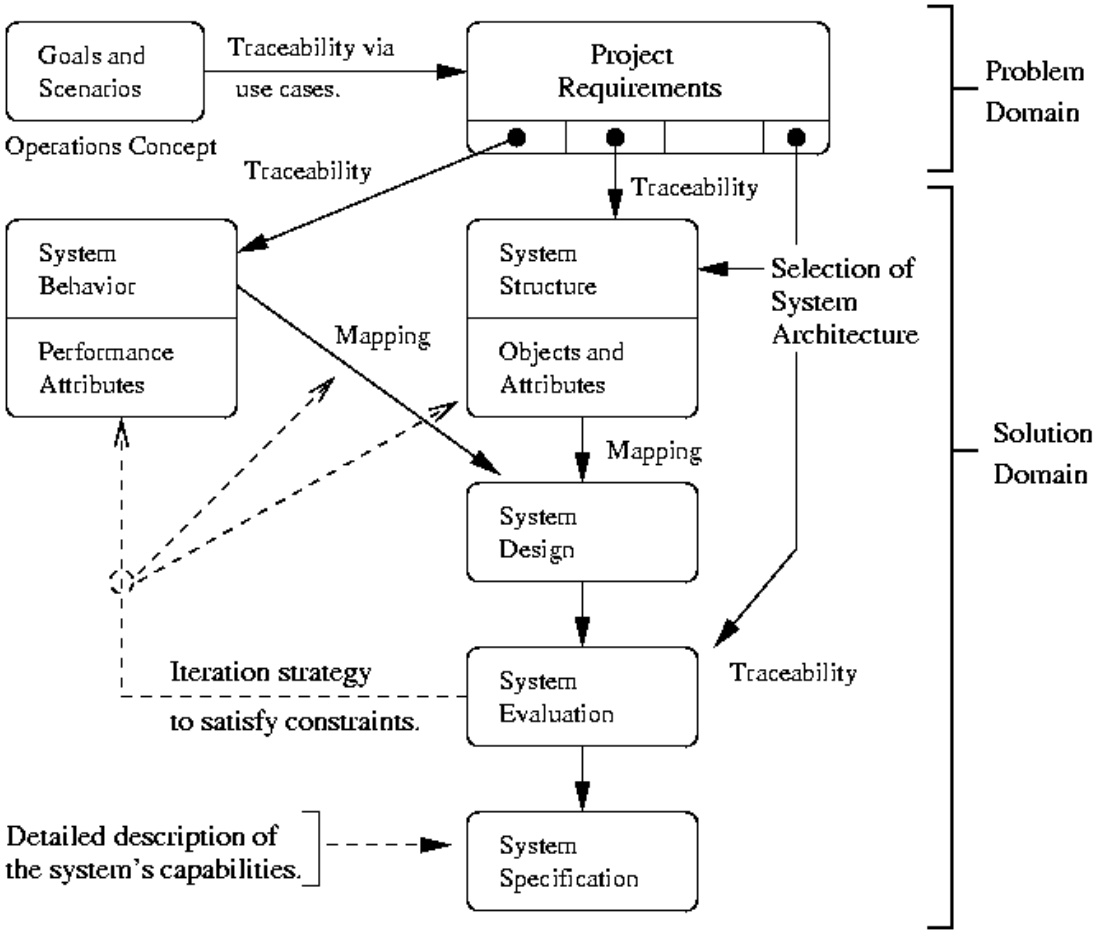
- Students begin their "case study" projects with the development of an operations concept. The operations concept should capture the goals and scenarios relevant to stakeholders needs. They use “use cases” to obtain a first visualization and organization of these operational concepts and ideas.

- Students learn that descriptions of **system behavior** can be independent of their implementation in the **system structure**. However, the selection of a suitable system structure/architecture will be guided by the required behavior.
- They verify that the system structure connectivity is consistent with the flows of data/information defined in the models of system behavior.
  (Mapping system behavior to system structure)

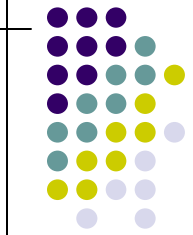
- They **organize requirements** according to their role in contributing to the behavior, structure, or testing of the system. **(Requirements allocation)**
- They transform requirements into detailed **specifications** -- that is, requirements that have an element that can be evaluated quantitatively or logically (true/false).
- They allocate fragments of system behavior to components in system structure.

- They perform **trade-off analysis**
- They develop a **system testing and validation** plan

The Big Picture



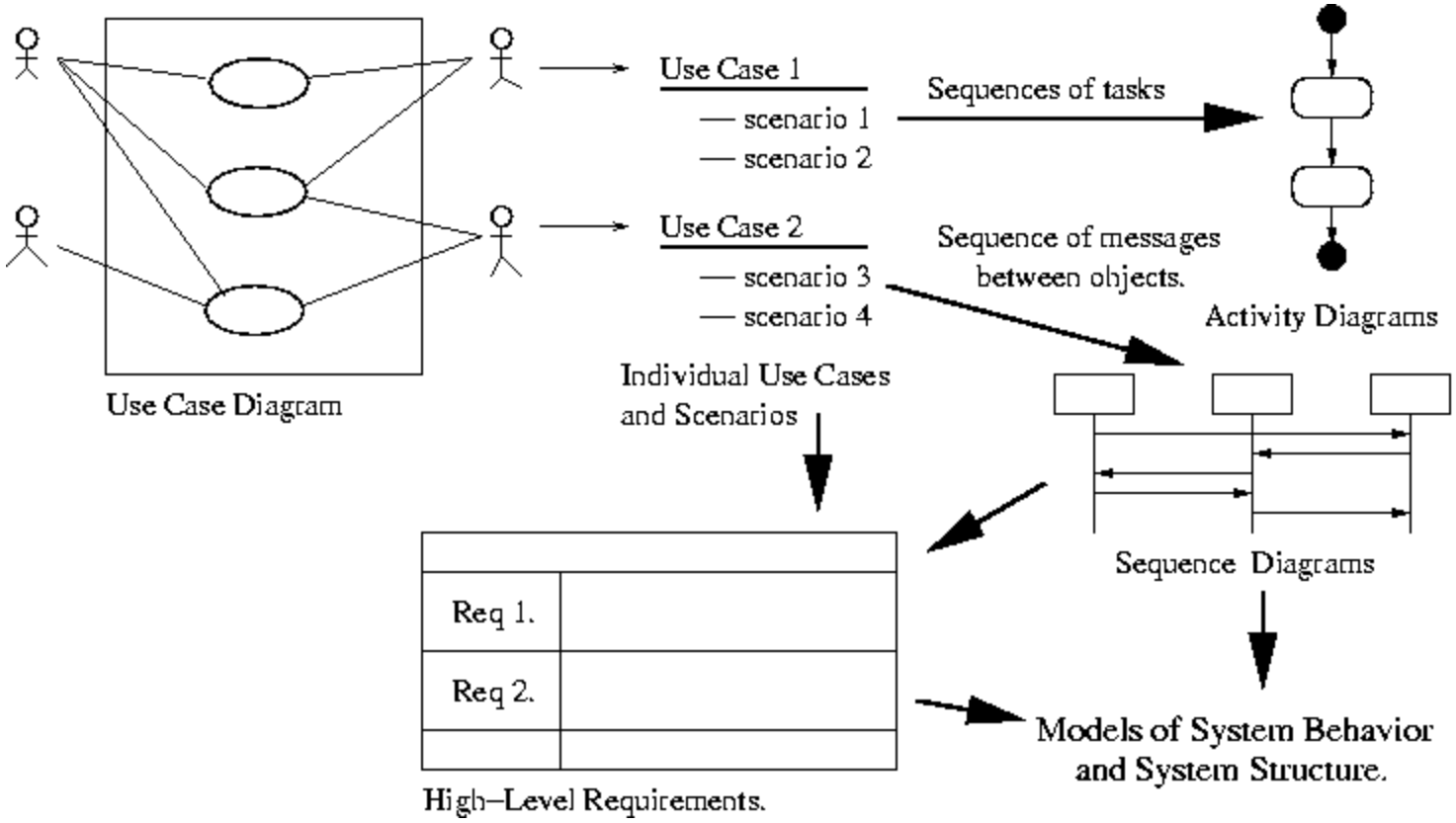
Near-Term Goals for SCOUPE



Development of a tool that would help engineers to:

- Specify UML Use Case Diagrams and Scenarios.
- Convert Scenarios into UML Activity and UML Sequence Diagrams.
- Identify Object Classes and create Class Diagrams.
- Facilitate linkage of chunks of system behavior to chunks of system structure.
- Describe, refine and allocate requirements along the way.
- Link requirements and the resulting specifications to specific components of system structure and system behavior.

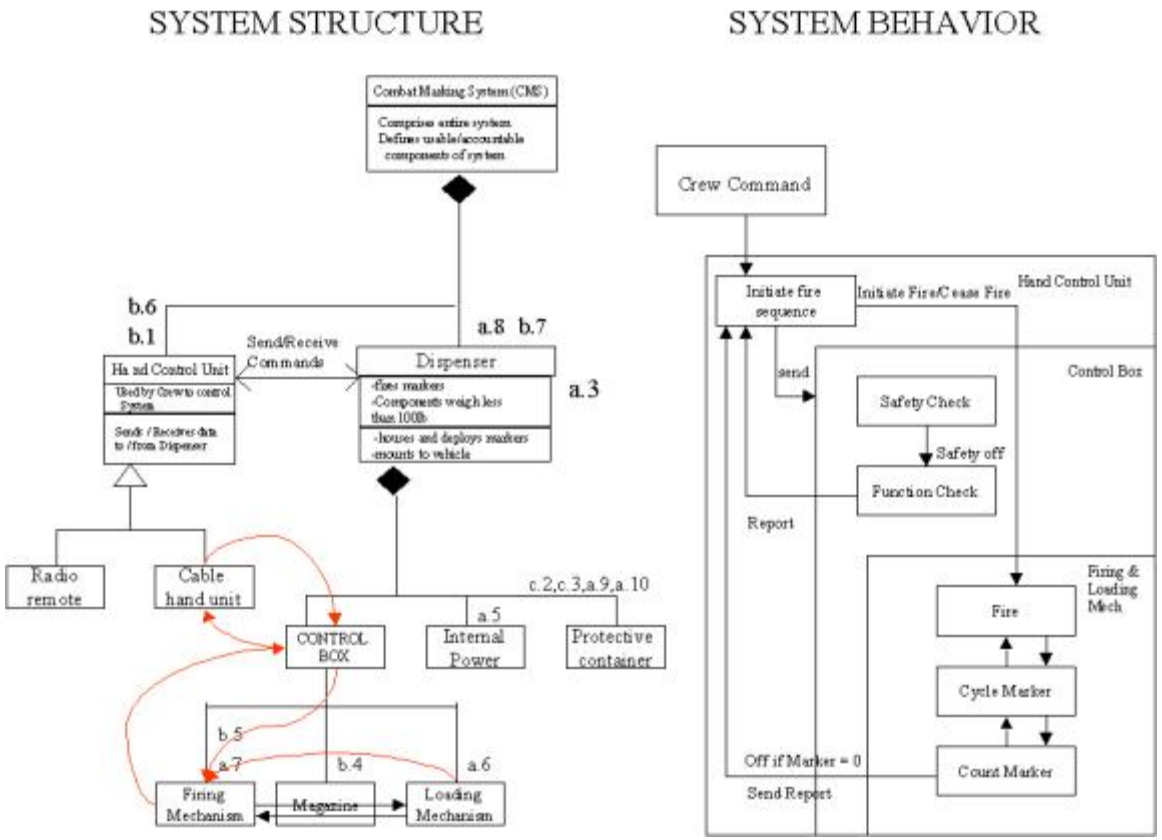
Pathway from Use Cases to Scenarios and High-Level Requirements



Visualizing Fragments of Behavior as Paths through the Class Diagram

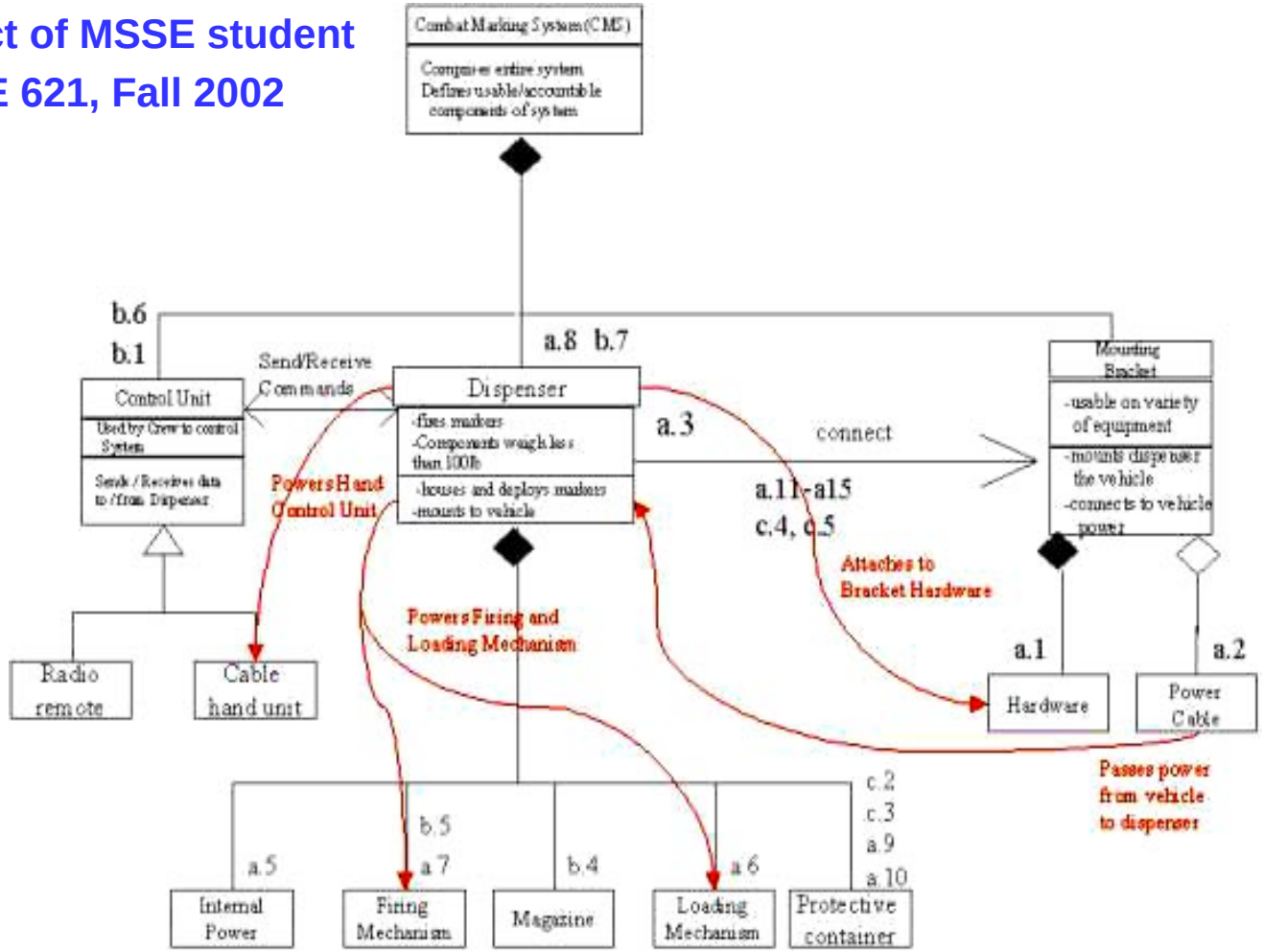
Based on the project of MSSE student

Adrian Marsh, ENSE 621, Fall 2002



Visualizing Fragments of Behavior as Paths through the Class Diagram

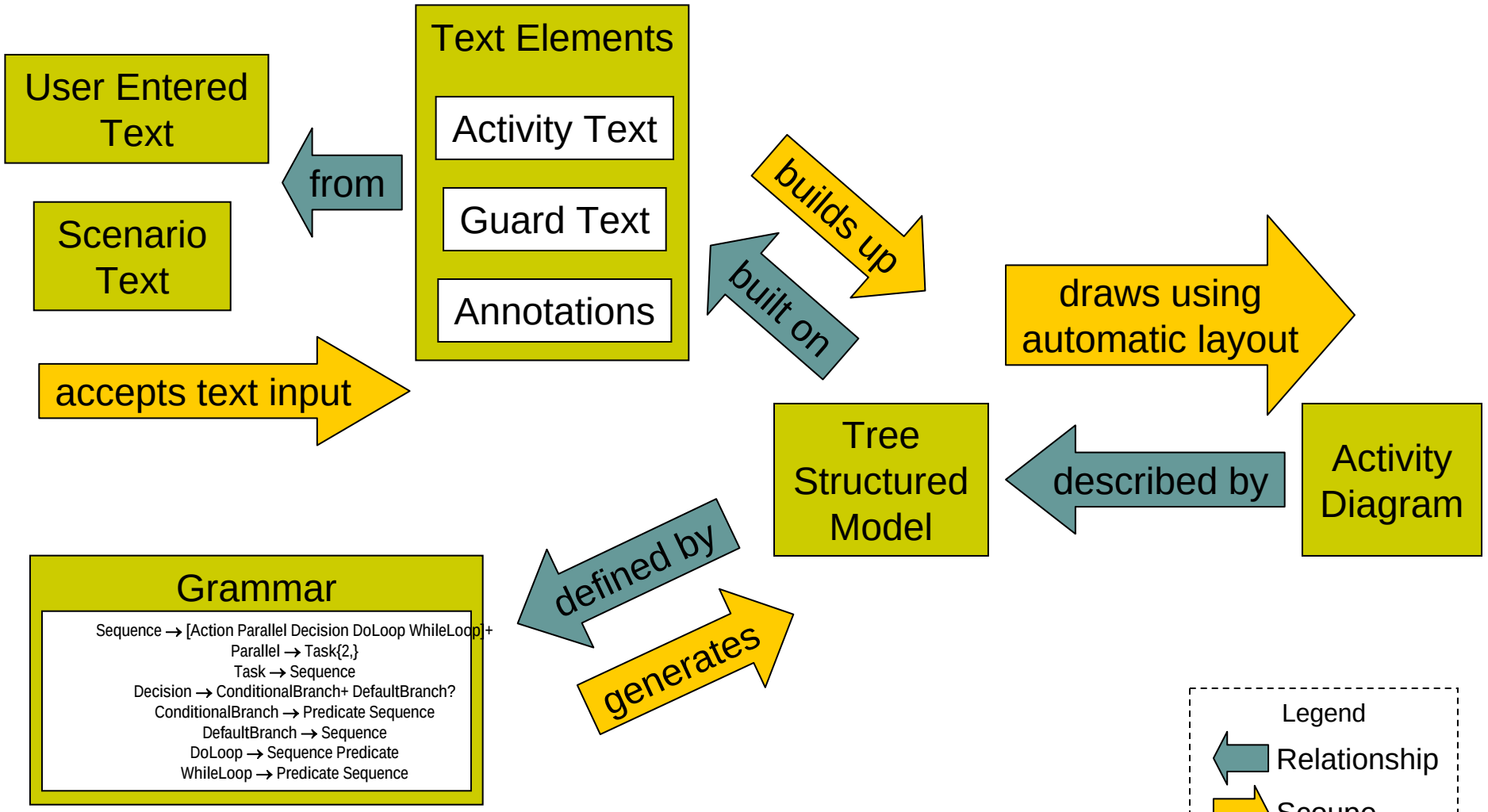
Based on the project of MSSE student
Adrian Marsh, ENSE 621, Fall 2002



Model-Driven Activity Diagram Generation

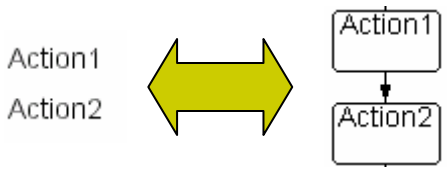
- Traditional UML tools allow inconsistent diagrams to be drawn
  - The tool does not tell a student if a diagram is syntactically consistent with the UML specification
  - Inconsistent diagrams are ambiguous
- Our approach is ***Model-centric***
  - A student focuses on developing a model
  - Consistency with the modeling language is enforced through a grammar
  - Diagrams are always syntactically consistent with UML
- Scoupe automatically generates diagrams from underlying models
  - It takes less work to change the model underlying the diagram than to line up the diagramming elements on screen
  - Feedback from changing the model is instantaneously provided through updates to the diagram.

Activity Diagram Software Overview

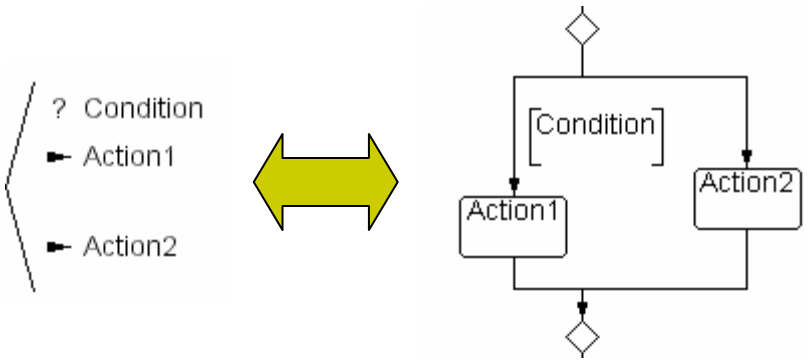


Modeling Language Elements

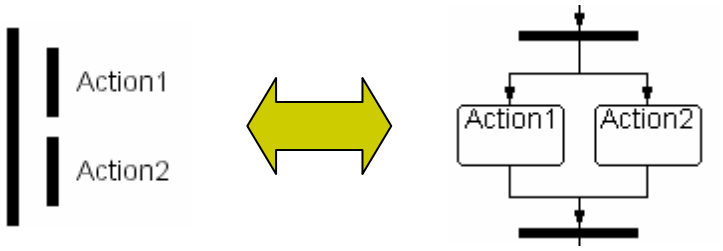
- Sequence



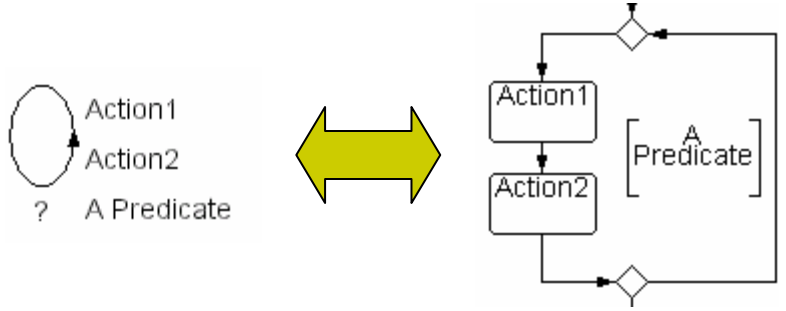
- Decision



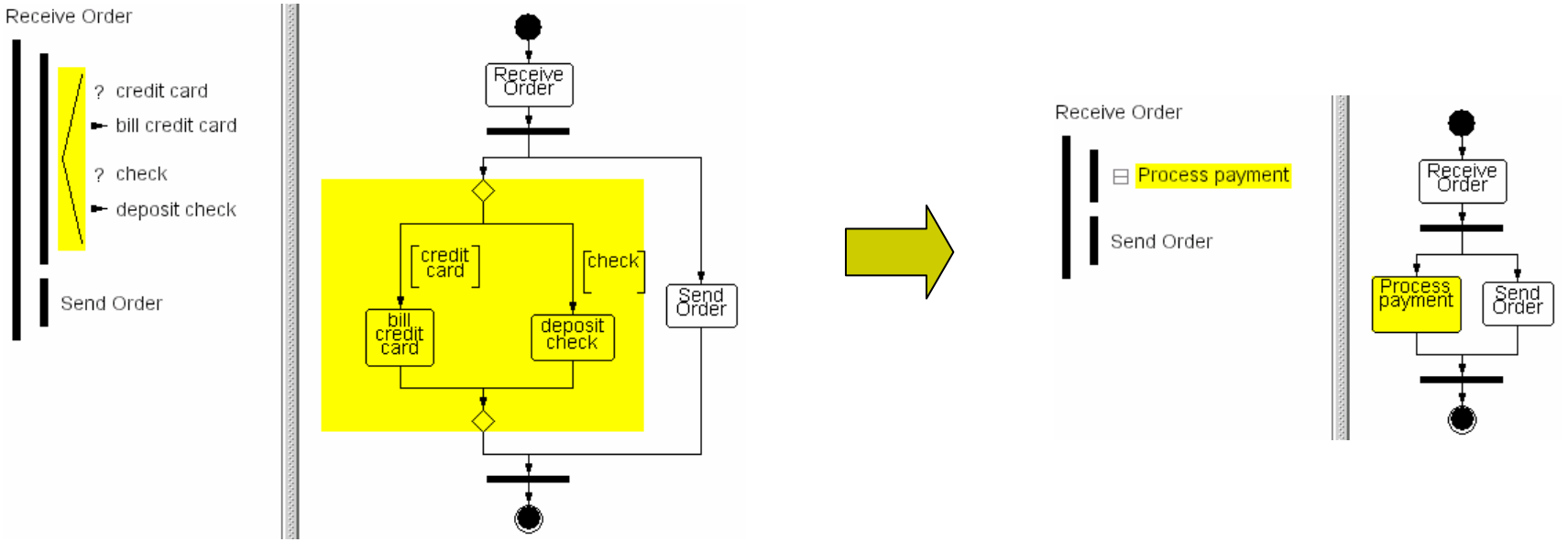
- Parallel



- Do Loop



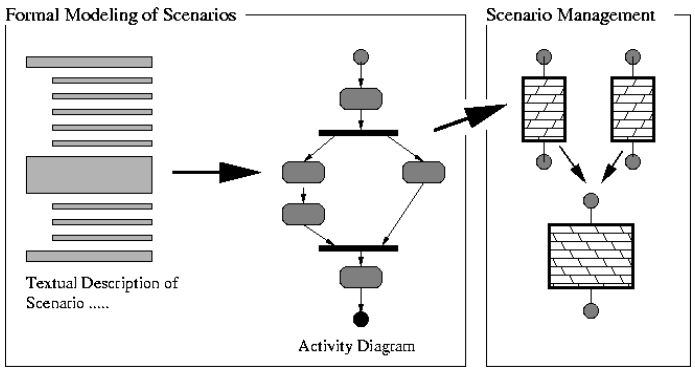
Management of Large Diagrams by Collapsing

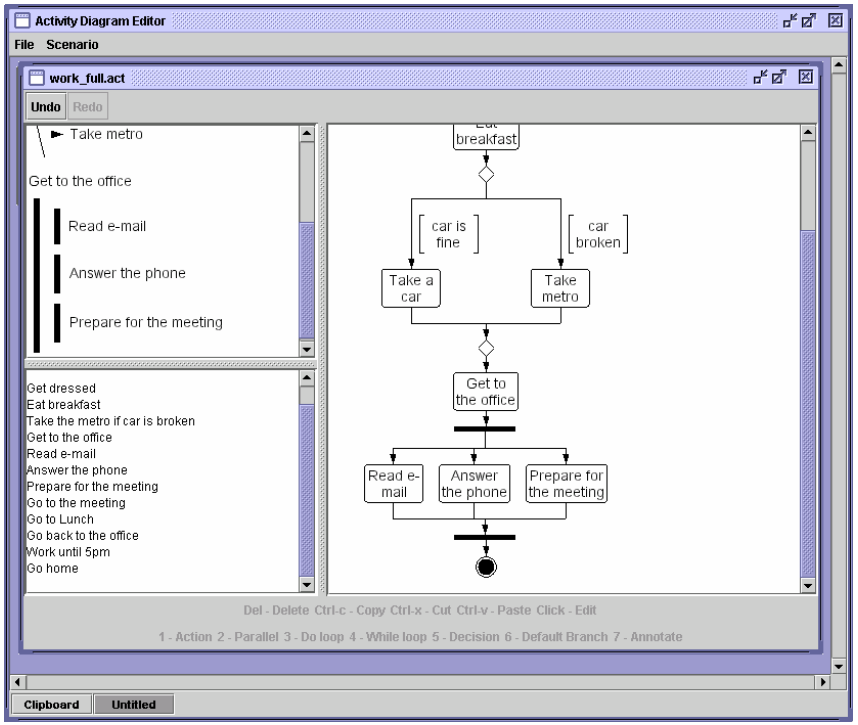


Advantages, Limitations

- Assists students in learning to produce *correct* UML diagrams
- Automatic diagram layout
- Blocks of activity diagrams are collapsible
- Copy and paste between diagrams is much easier
- Not all UML activity diagram expressible in language (only all behaviors)
- No swim-lanes

Where we are

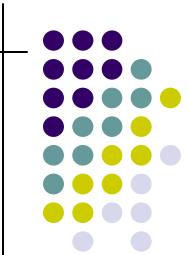




Students can already create UML Activity Diagrams from textual Scenarios and check the diagrams for correctness.

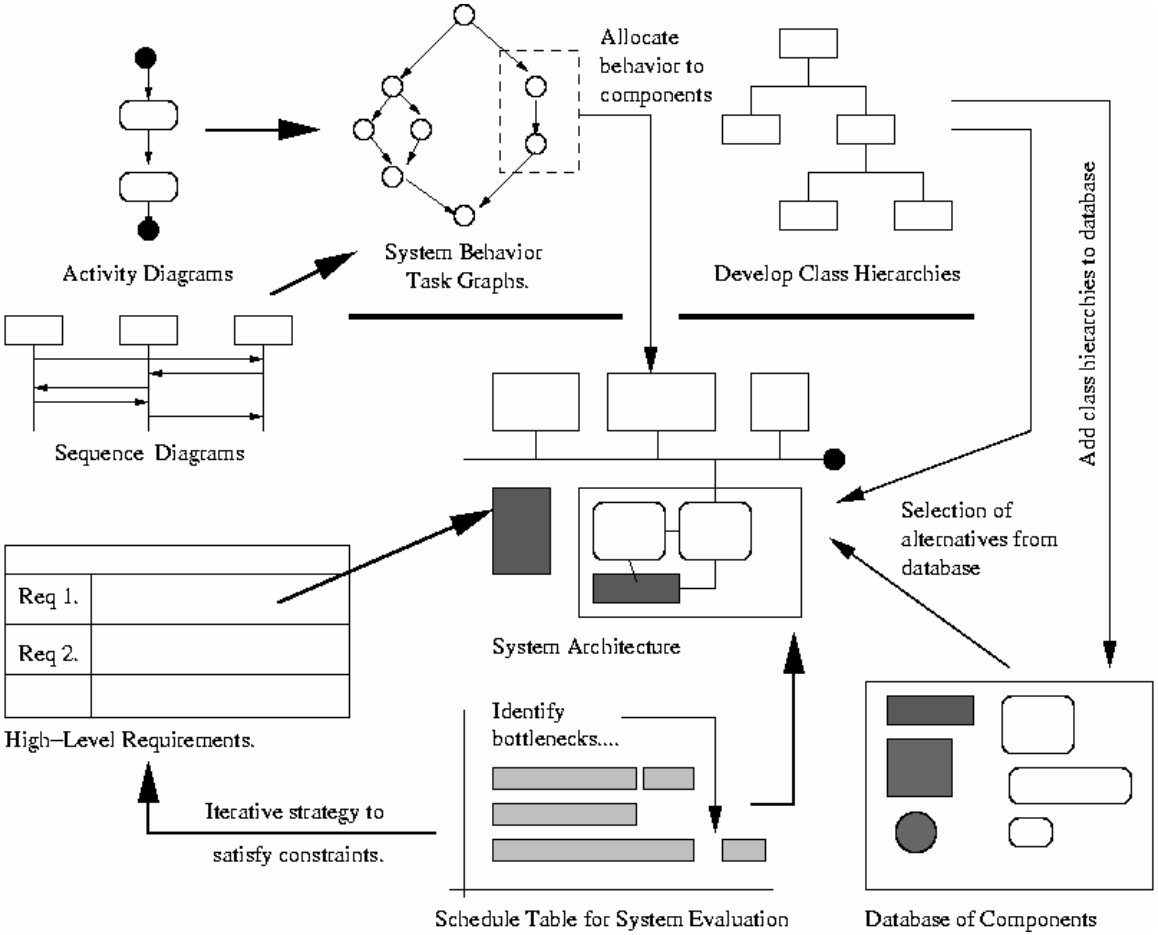
They can also create diagrams from scratch, save them into readable format, and manipulate the way they are represented on the screen (collapse/expand).

Looking ahead



- Engineers will be able to identify attributes of each component, and the fragments of functionality that each will perform. The identification process is linked back to the high-level requirements.
- They will formulate multi-objective trade-off analysis problems. Specifications are written as design constraints. Typical design objectives are cost, performance, throughput and reliability.
  Most of the constraints for trade-off analysis will come from the requirements-specifications.
- Deposition and recovery of the developed annotated (by requirements) model chunks of system behavior and system structure to and from an object-relational database is facilitated.

Looking ahead



Benefits of SCOUPE

- Scoupe ensures that all of the elements in the system description are linked together in a consistent way.
- Built-in traceability. By design, each object can be traced to its behavior , to corresponding requirements, and to other necessary elements in the the system.
- Students are constrained to complete all of the steps in the system development process.

DEMO of Basic Capabiliies of SCOUPE

SCOUPE Activity Diagram Editor

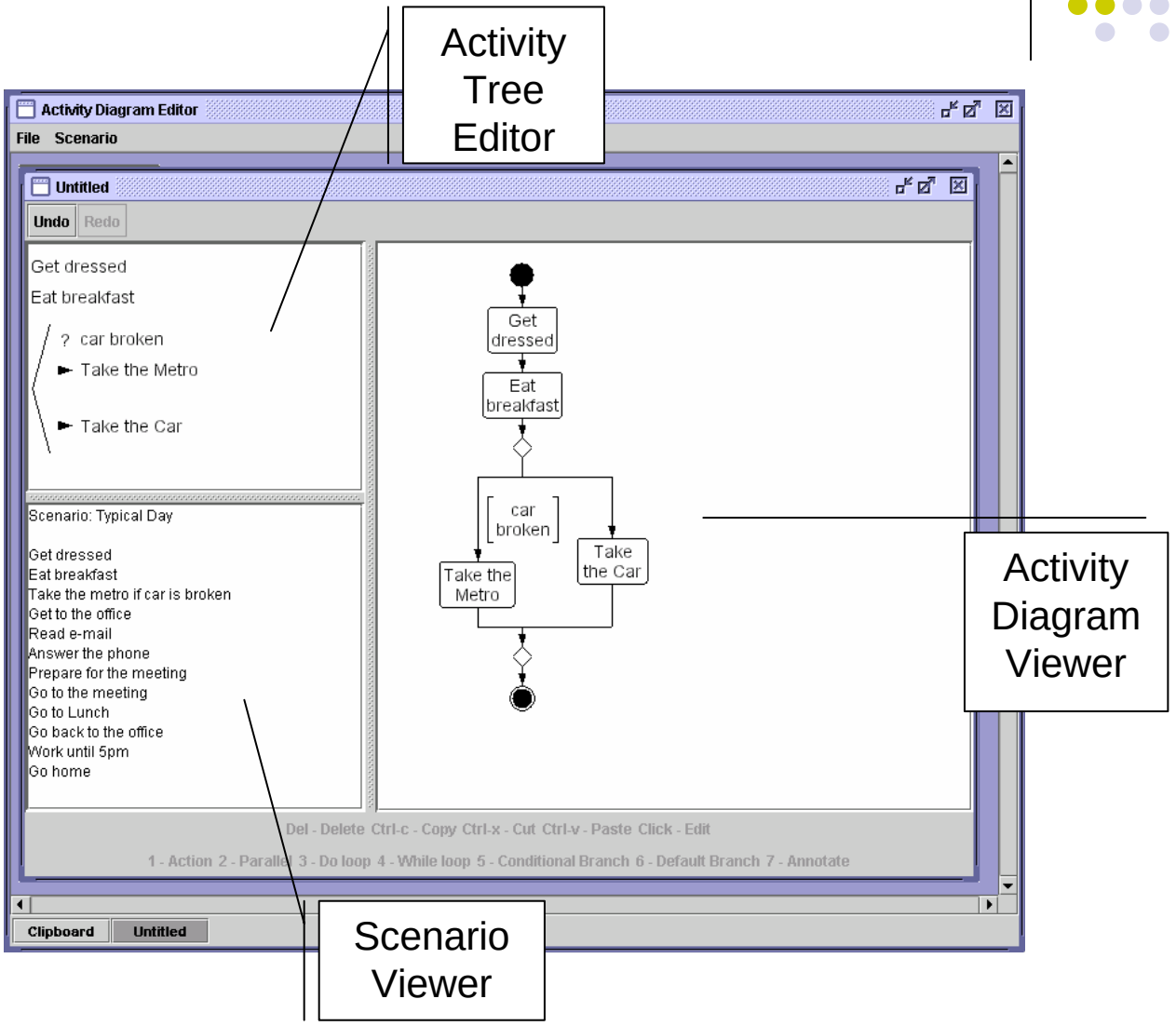
- Enables generation of model-driven UML Activity Diagrams.
- The users can generate diagrams from scratch using Activity Tree Editor.
- The tool also supports construction of the diagrams via scenarios.

Following earlier slides 10-12

DEMO of Basic Capabilities

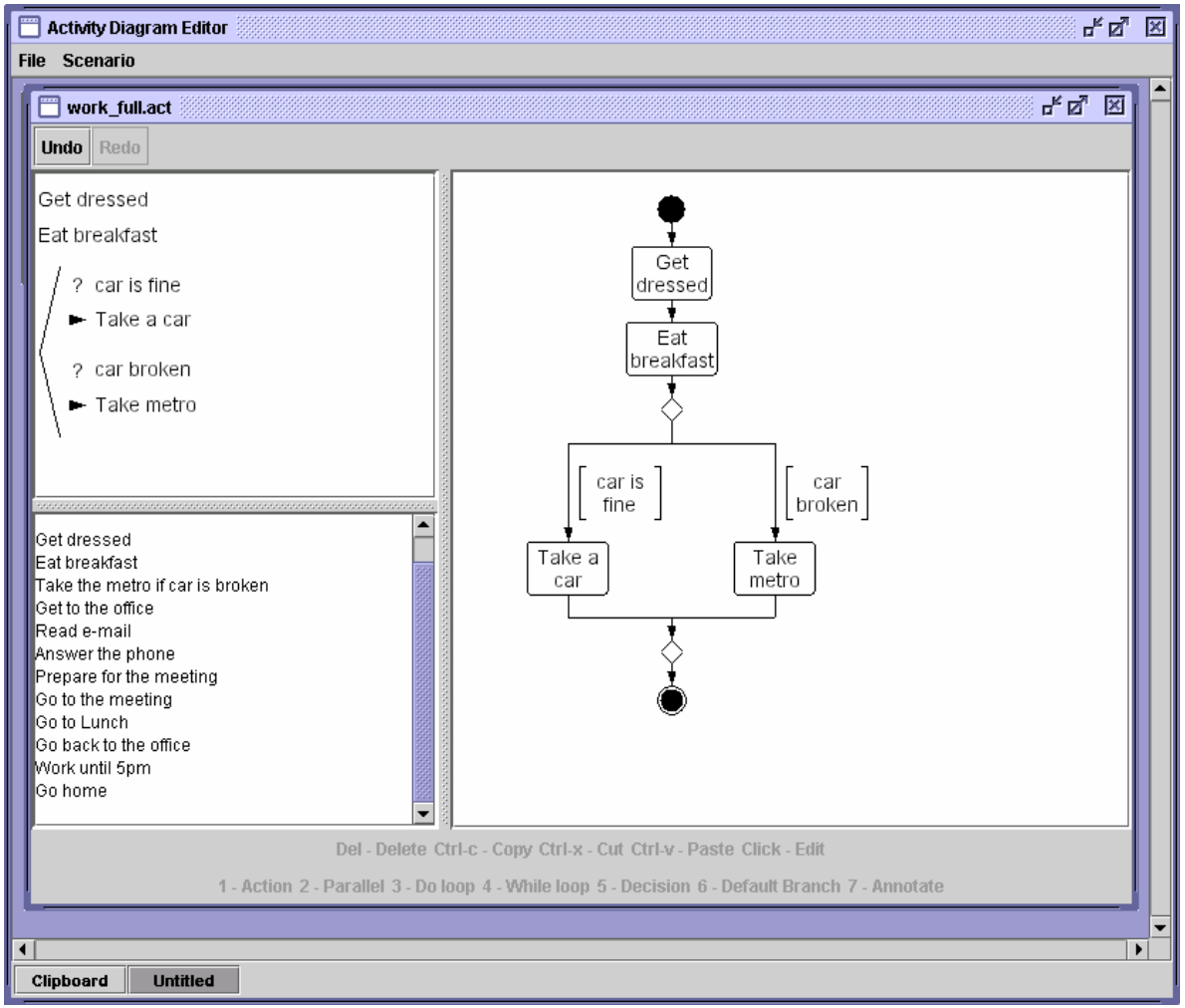
Activity Diagrams are used for describing a process, a use case, or a complicated method. The can depict data and information flows. They can also aid in capturing requirements.

Activity Tree Editor greatly simplifies the process of diagram creation. It effectively captures the structure of the Activity Diagram and enables its modification.



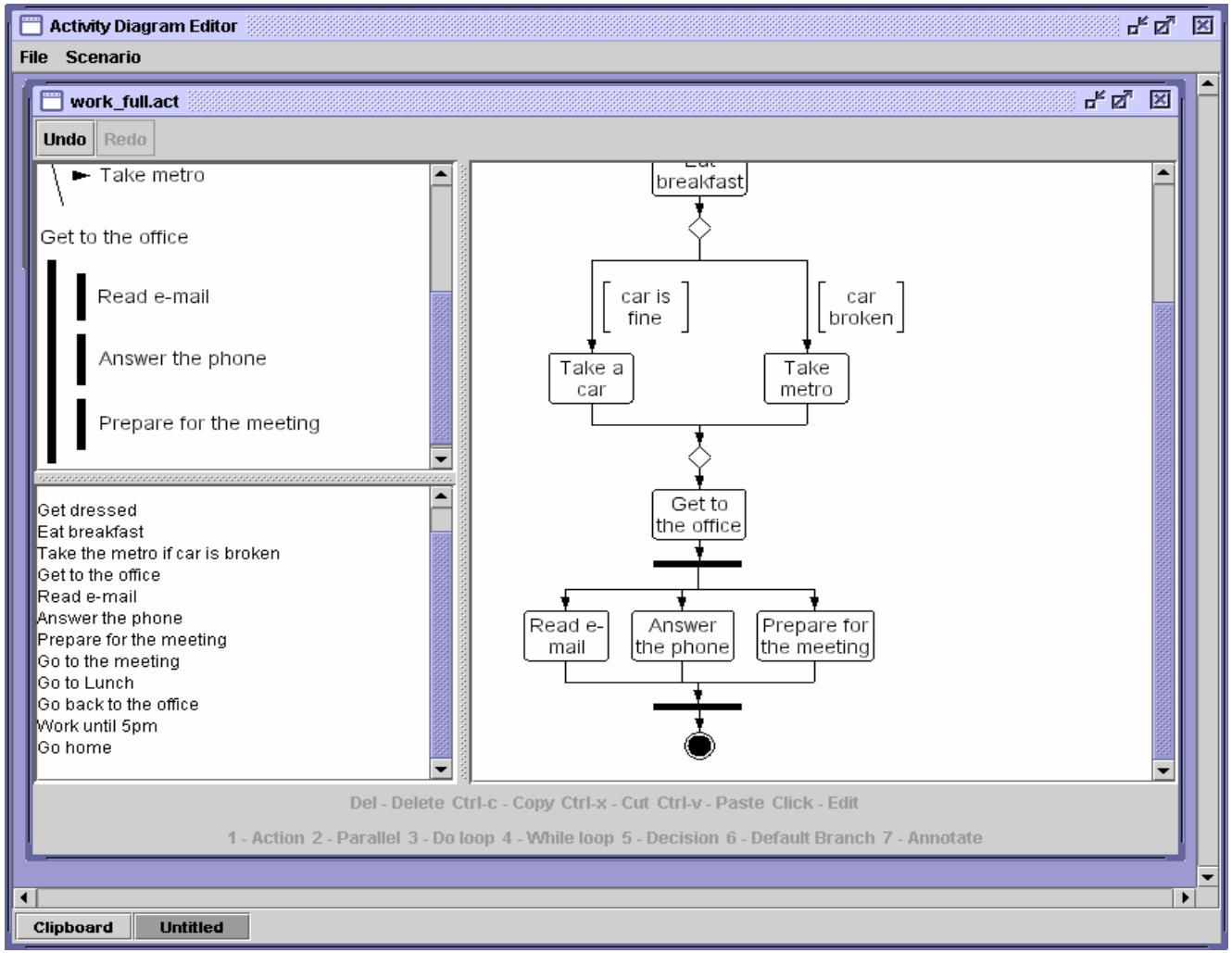
DEMO of SCOUPE Functionality

1. Open Textual Scenario: Typical Day
2. Create Actions
3. Create Decision Block
4. Create another action “Get to the office”



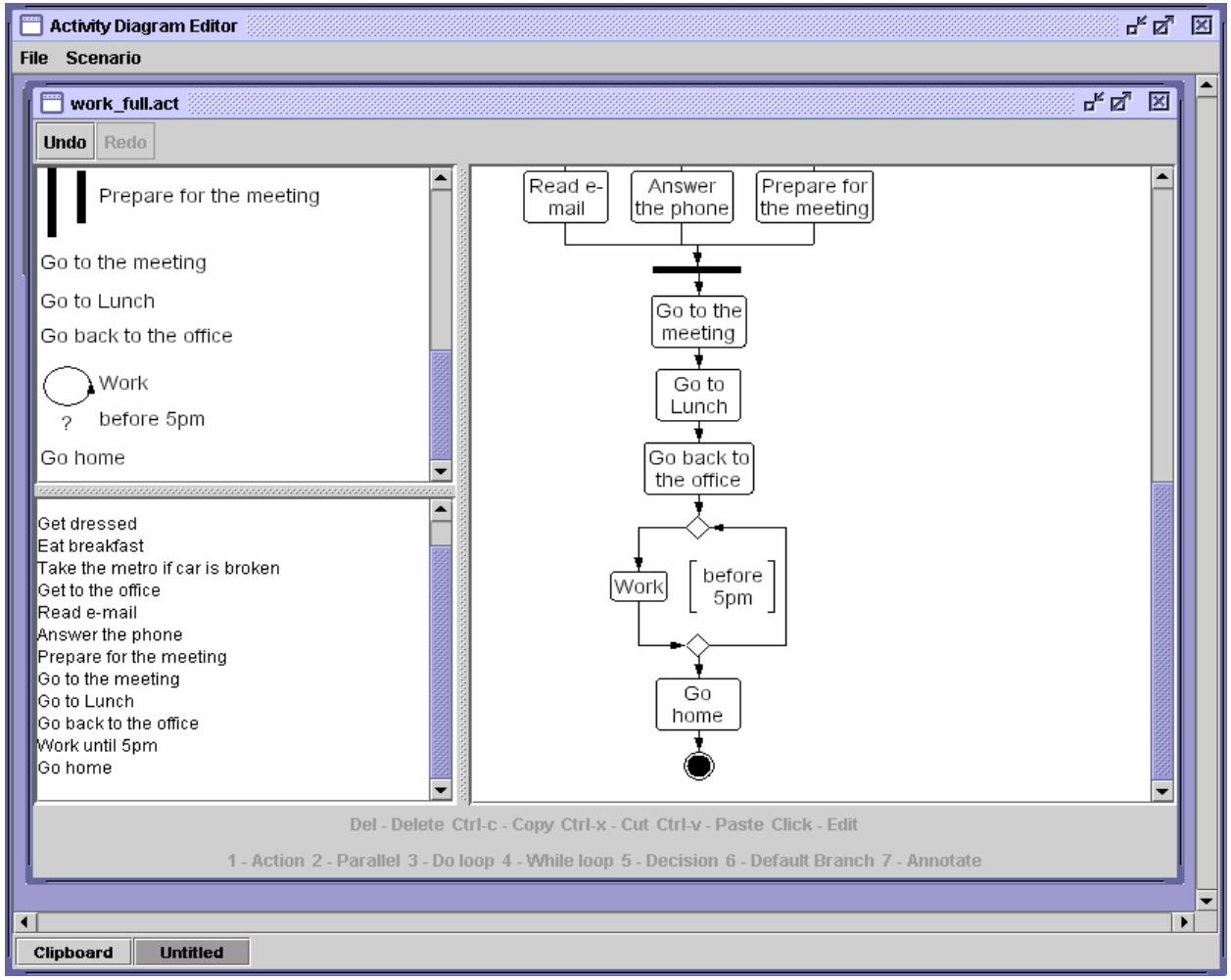
DEMO of SCOUPE Functionality

5. Create Parallel Block with 3 tasks

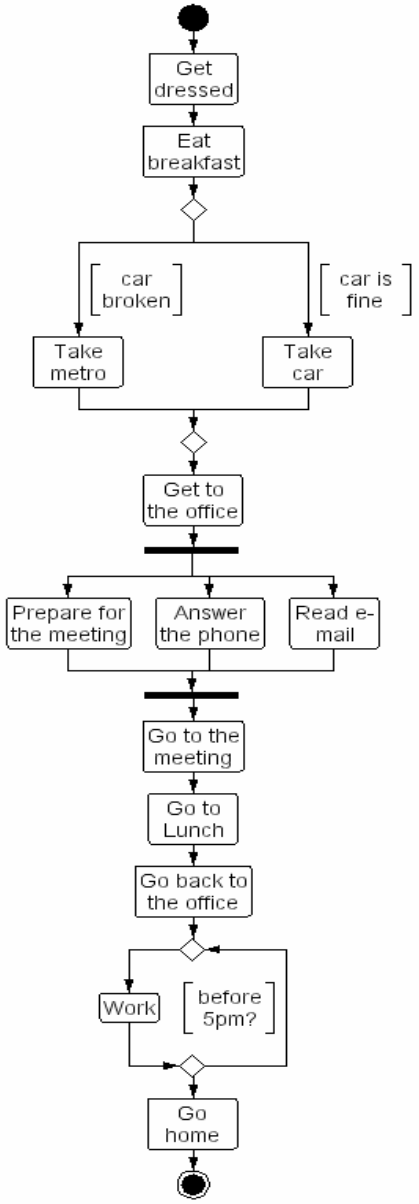


DEMO of SCOUPE Functionality

6. Create more Actions
7. Create a do-while loop "work until 5pm"

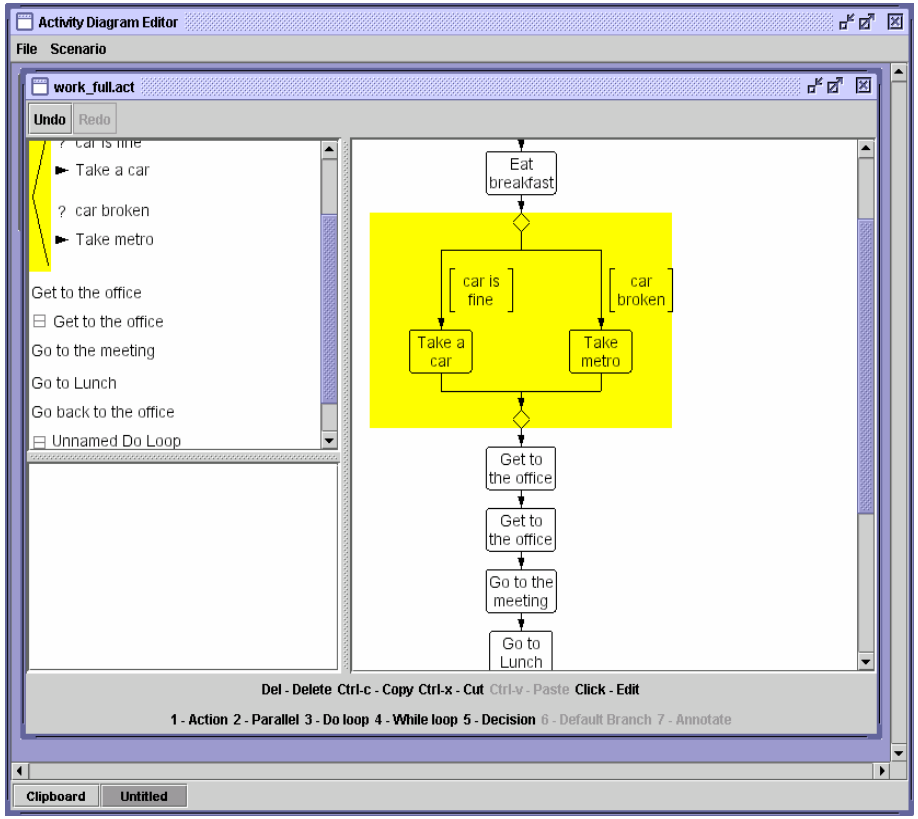


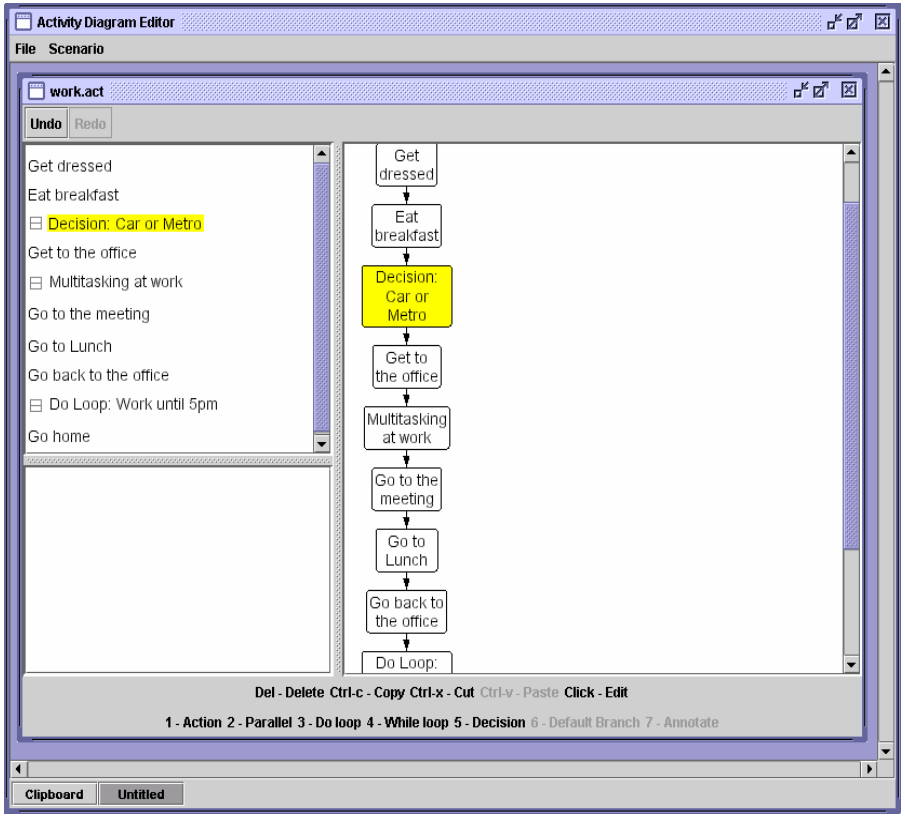
Final Activity Diagram



Other SCOUPE Functionality

Collapsing/Expanding Activity Diagrams





Other SCOUPE Functionality

- **Undo/Redo** – lets the user undo or redo the steps that were made in creating the diagram
- **Copy/Paste** – lets the user copy part of the diagram and paste it into a new diagram or into the existing diagram.
- **Open/Save** – any diagram can be saved for future use.
- **Export Diagram** – any diagram can be exported into a PNG format

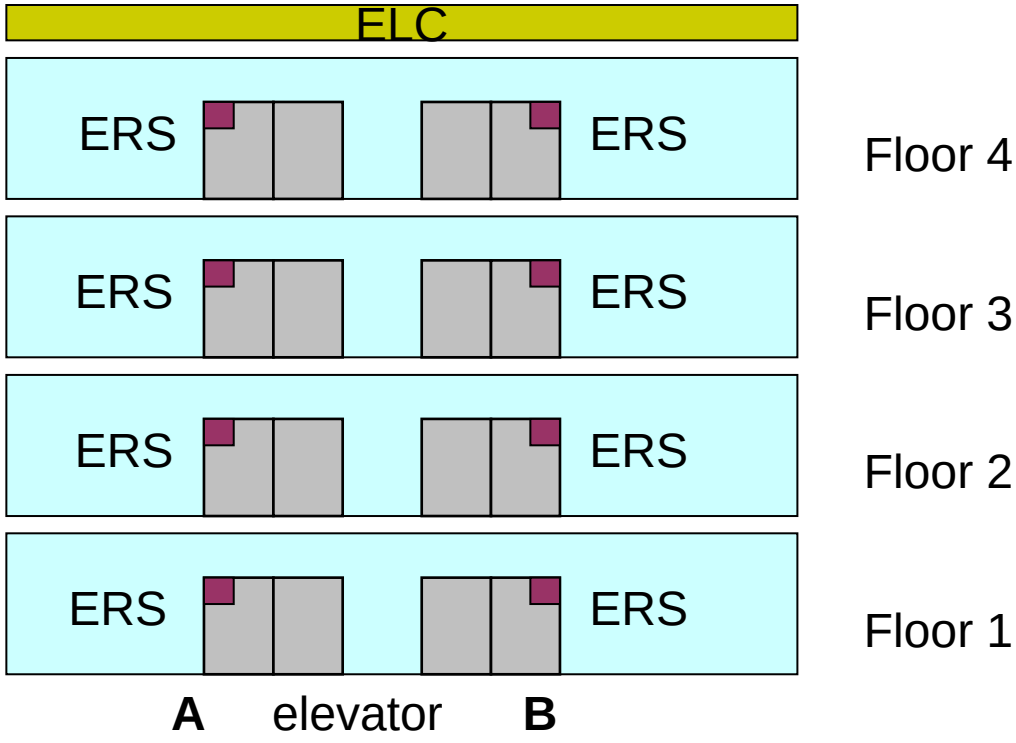
DEMO of Using SCOUPE in a Case Study

Elevator System

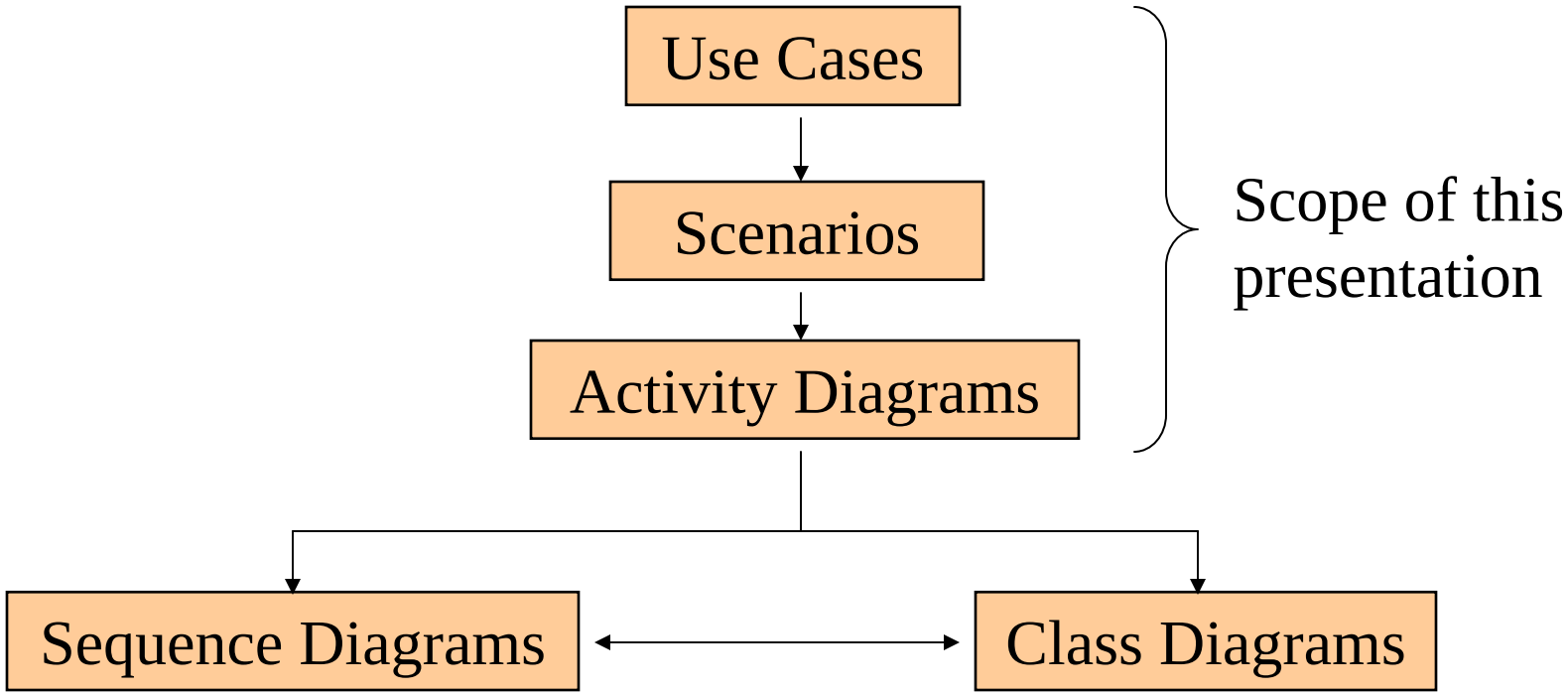
(based on the project of MSSE students Mike Buck and Bonnie Lawson, ENSE 621, Fall 2002)

- Building has two elevators, four floors
- We will call these elevators A and B
- They are controlled by an ELC (Elevator logic controller)
- ELC tells A & B, when to move and in which direction (up or down) and on which floor to stop on the way
- User interacts with this system using different push buttons (Elevator Request System or ERS)

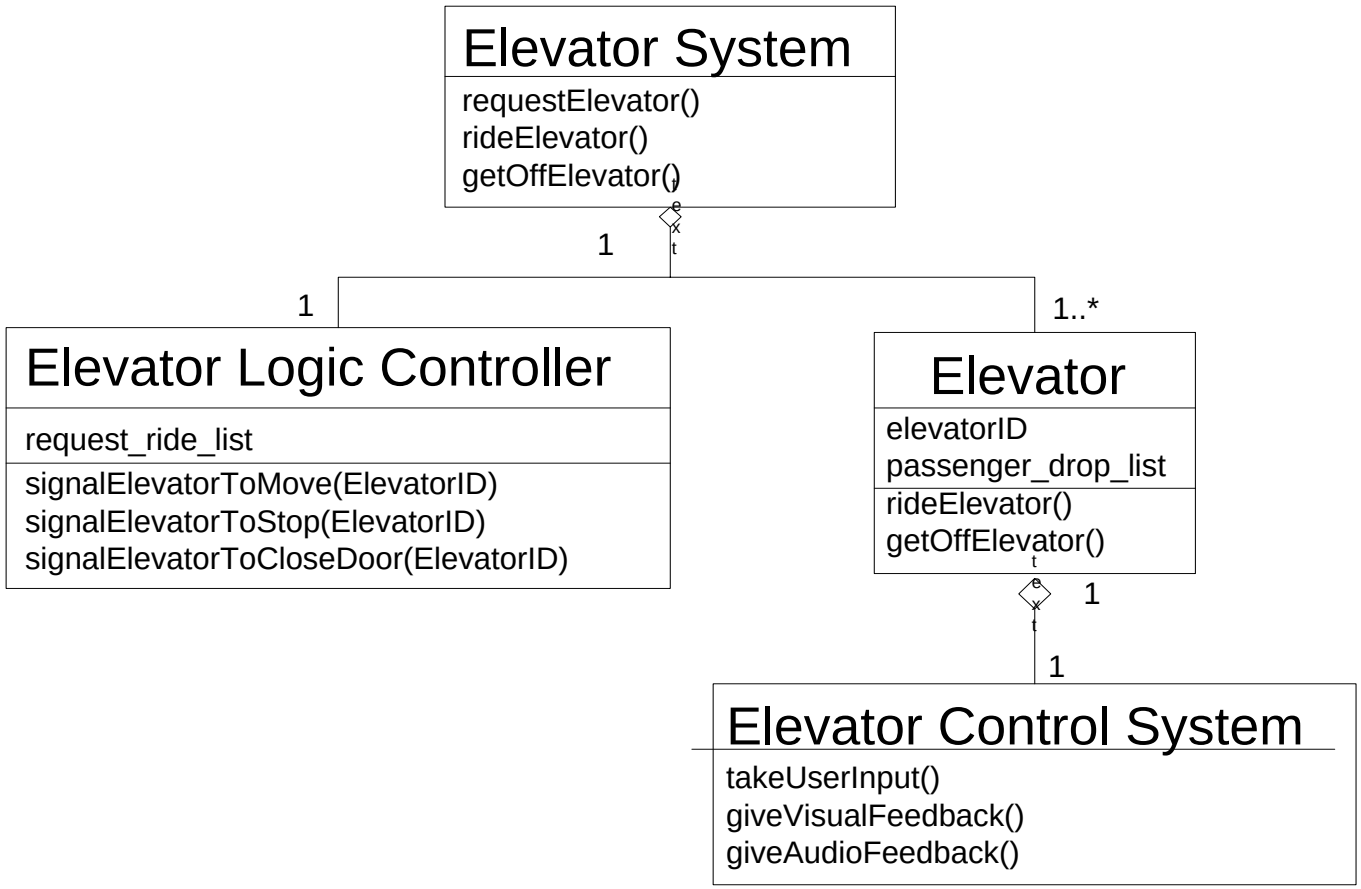
SCOUPE DEMO Developed by Vimal Mayank



Initial Pathway of System Development



Initial Class Diagram



Elevator System Scenarios

Initial State of the System: Elevator A and B are idling at floor X and Y respectively. User arrives at ERS

- Presses Up or Down button signifying direction in which he wants to move.
- ERS sends request to ELC.
- ELC adds request to request_ride_list containing user floor and desired direction.
- While Elevator A gets no signal to move from ELC Elevator A idles
- A Starts moving in the asked direction.

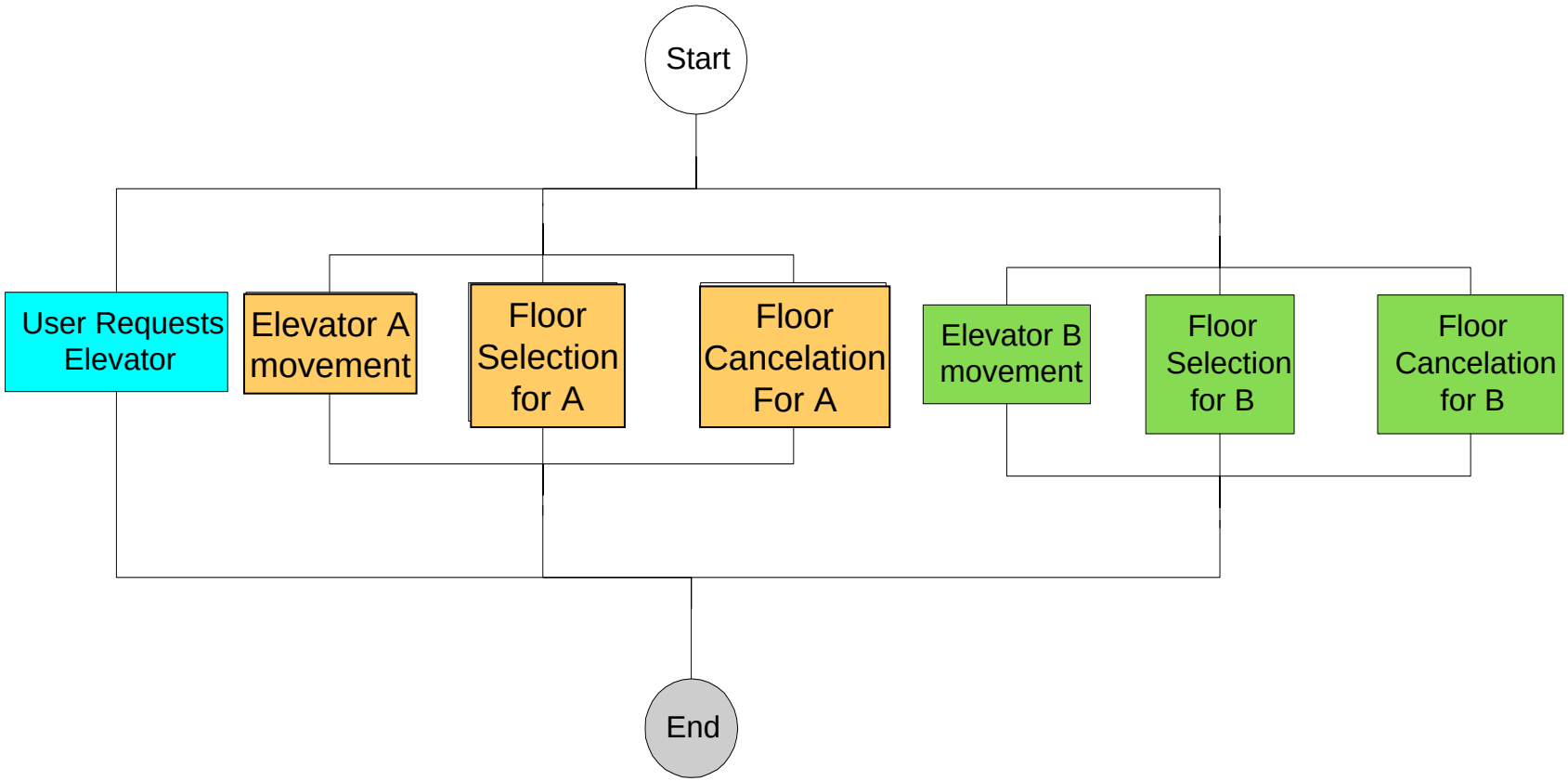
7. While A gets no signal to stop at approaching floor from ELC it continues to move. ELC makes stopping decision based on its request_ride_list and passenger_drop_list_A.
8. If A is asked to stop, A stops at approaching floor.
9. Its door opens.
10. Door remains open until A doesn't get signal to close. It gets signal when user presses close door, or a floor selection, or a preset time is elapsed (signifying passenger left before utilizing the request). On the other hand it can't get that signal if a user inside the elevator presses and holds the open door button

Elevator System Scenarios

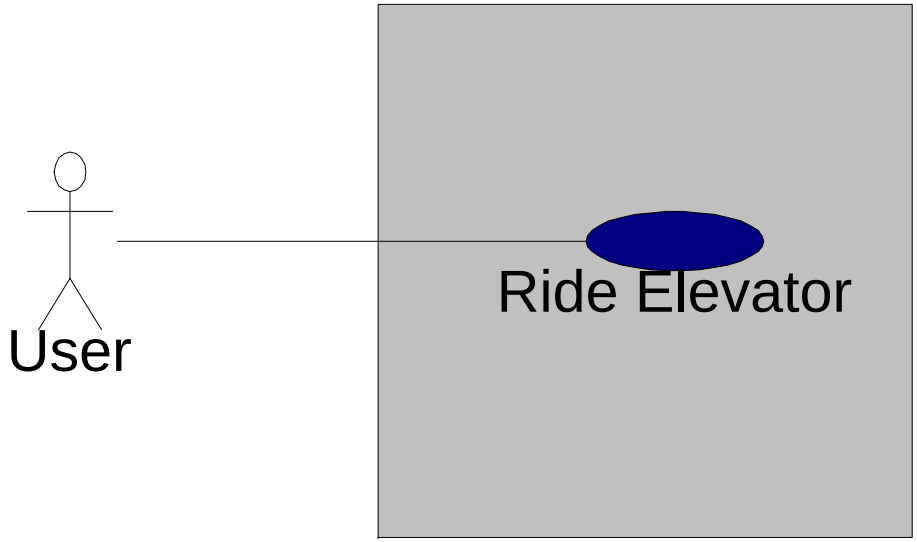
11. Door starts to close
12. Door opens again if there is an obstruction in the path
13. Door remains open if A doesn't get signal to close
14. Door starts to close
15. Door closes
16. ELC updates request_ride_list and passenger_drop_list_A for that floor.
17. Repeat 6 unless Power is switched off
18. If user makes a floor selection his request is added to passenger_drop_list_A
19. If user cancels a floor selection his request is deleted from passenger_drop_list_A
20. Elevator B mimics the scenarios as depicted from 6-19 for elevator A.

Post Condition: Elevator A & B are switched off

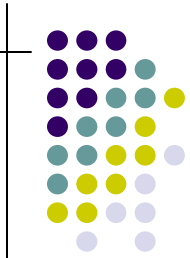
High Level Description of Scenarios



Initial Use Case for Elevator

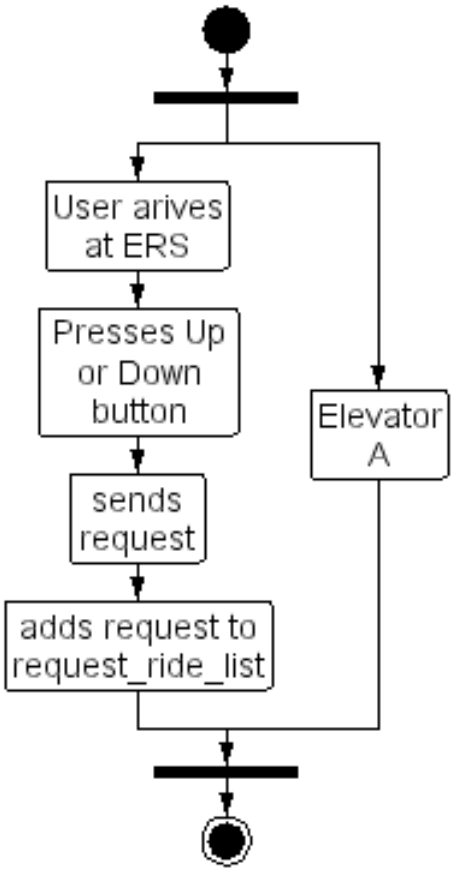


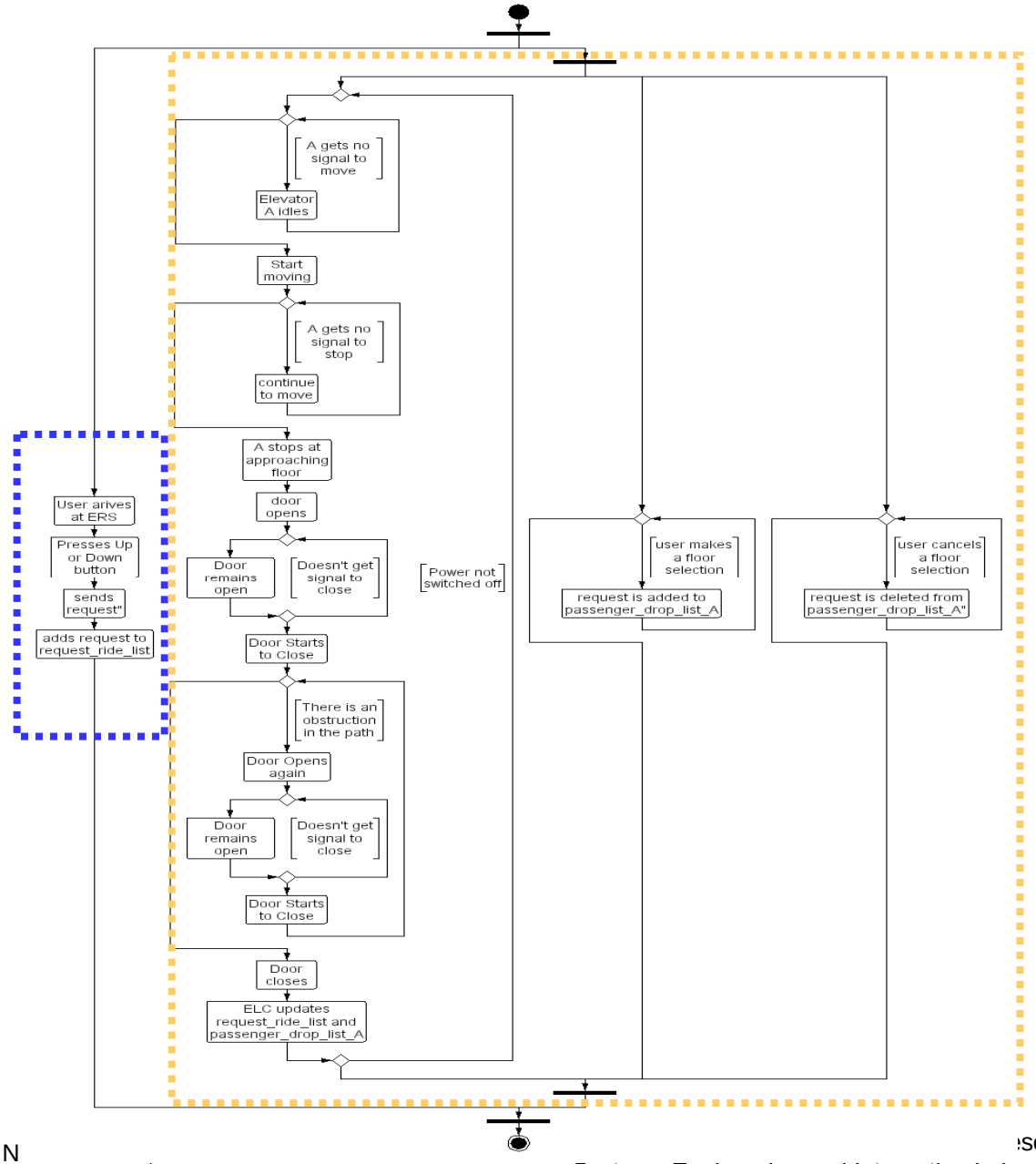
Terminology



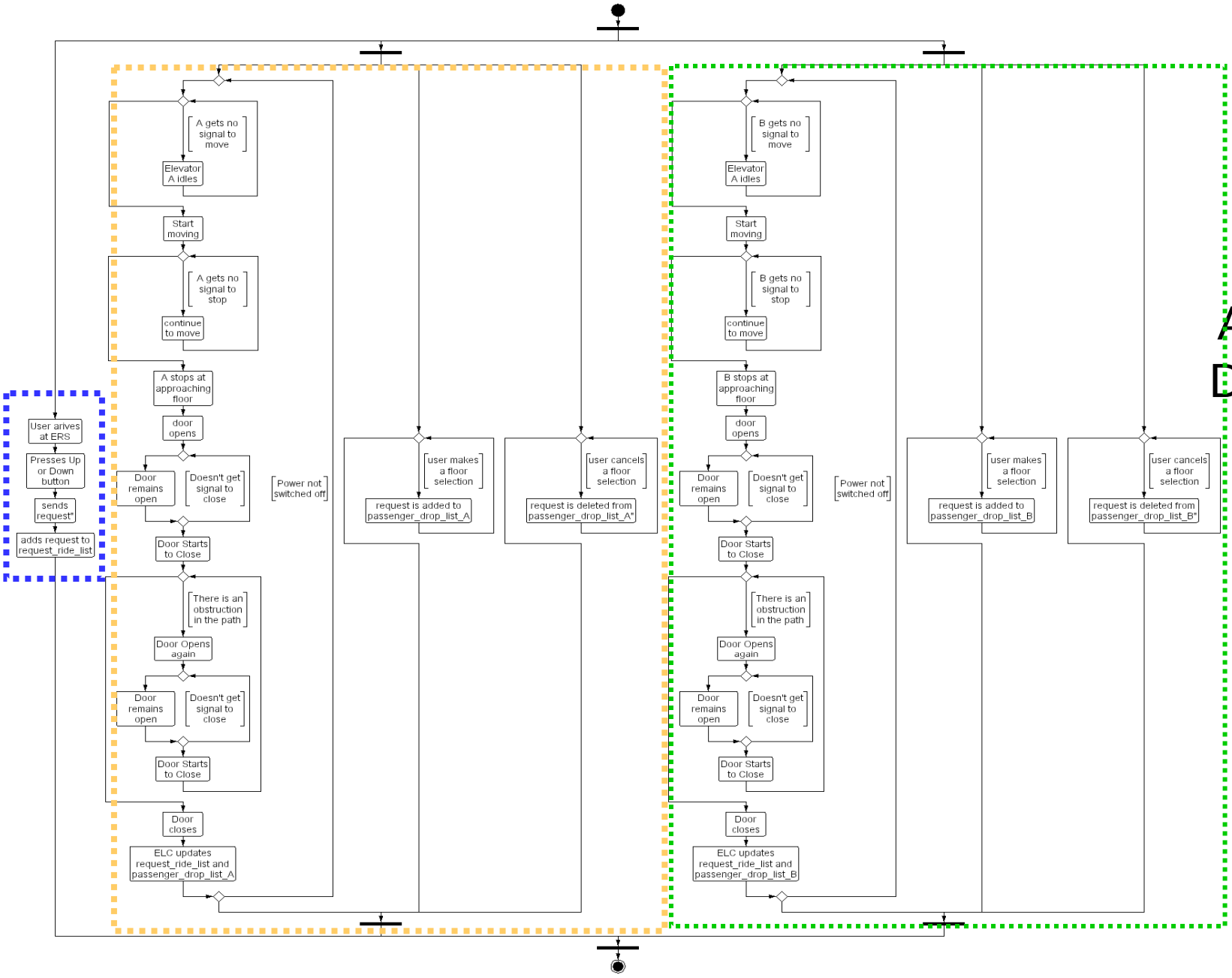
- Request_ride_list: This list is maintained by the ELC, which contains floor from which a user has requested the elevator, and the desired direction (Up or down) in which he wants to move
- Passenger_drop_list_X: This list is maintained separately for each elevator X. This contains the floors at which users will get off from that elevator

Converting to Activity Diagram





Activity Diagram with elevator A Block completed



Full Activity Diagram