

Object-Oriented and Classical Software Engineering

OBJECT-ORIENTED ANALYSIS

- The analysis workflow
- Extracting the entity classes
- The elevator problem case study
 - Object-oriented analysis
 - Functional modeling
 - Entity class modeling
 - Dynamic modeling
- The test workflow: Object-oriented analysis
- Extracting the boundary and control classes

- The MSG Foundation case study
 - The initial functional model
 - The initial class diagram
 - The initial dynamic model
 - Extracting the entity classes
 - Extracting the boundary classes
 - Extracting the control classes
 - Use-case realization
 - Incrementing the class diagram
 - The test workflow

- The specification document in the Unified Process
- More on actors and use cases
- CASE tools for the object-oriented analysis workflow
- Challenges of the object-oriented analysis workflow

- OOA is a semiformal analysis technique for the object-oriented paradigm
 - There are over 60 equivalent techniques
 - Today, the Unified Process is the only viable alternative
- During this workflow
 - The classes are extracted
- Remark
 - The Unified Process assumes knowledge of class extraction

13.1 The Analysis Workflow

Slide 13.7

- The analysis workflow has two aims
 - Obtain a deeper understanding of the requirements
 - Describe them in a way that will result in a maintainable design and implementation

The Analysis Workflow (contd)

Slide 13.8

- There are three types of classes:
 - Entity classes
 - Boundary classes
 - Control classes

The Analysis Workflow (contd)

Slide 13.9

- Entity class
 - Models long-lived information
- Examples:
 - **Account Class**
 - **Investment Class**

The Analysis Workflow (contd)

Slide 13.10

- Boundary class
 - Models the interaction between the product and the environment
 - A boundary class is generally associated with input or output
- Examples:
 - **Investments Report Class**
 - **Mortgages Report Class**

The Analysis Workflow (contd)

Slide 13.11

- Control class
 - Models complex computations and algorithms
- Example:
 - **Estimate Funds for Week Class**

UML Notation for These Three Class Types

Slide 13.12

- Stereotypes (extensions of UML)



Figure 13.1

13.2 Extracting the Entity Classes

Slide 13.13

- Perform the following three steps incrementally and iteratively
 - Functional modeling
 - » Present scenarios of all the use cases (a *scenario* is an instance of a use case)
 - Class modeling
 - » Determine the entity classes and their attributes
 - » Determine the interrelationships and interactions between the entity classes
 - » Present this information in the form of a *class diagram*
 - Dynamic modeling
 - » Determine the operations performed by or to each entity class
 - » Present this information in the form of a *statechart*

A product is to be installed to control n elevators in a building with m floors. The problem concerns the logic required to move elevators between floors according to the following constraints:

1. Each elevator has a set of m buttons, one for each floor. These illuminate when pressed and cause the elevator to visit the corresponding floor. The illumination is canceled when the corresponding floor is visited by the elevator
2. Each floor, except the first and the top floor, has two buttons, one to request an up-elevator, one to request a down-elevator. These buttons illuminate when pressed. The illumination is canceled when an elevator visits the floor, then moves in the desired direction
3. If an elevator has no requests, it remains at its current floor with its doors closed

13.4 Functional Modeling: The Elevator Problem Case Study

Slide 13.15

- A use case describes the interaction between
 - The product, and
 - The actors (external users)

Use Cases

Slide 13.16

- For the elevator problem, there are only two possible use cases
 - Press an Elevator Button, and
 - Press a Floor Button

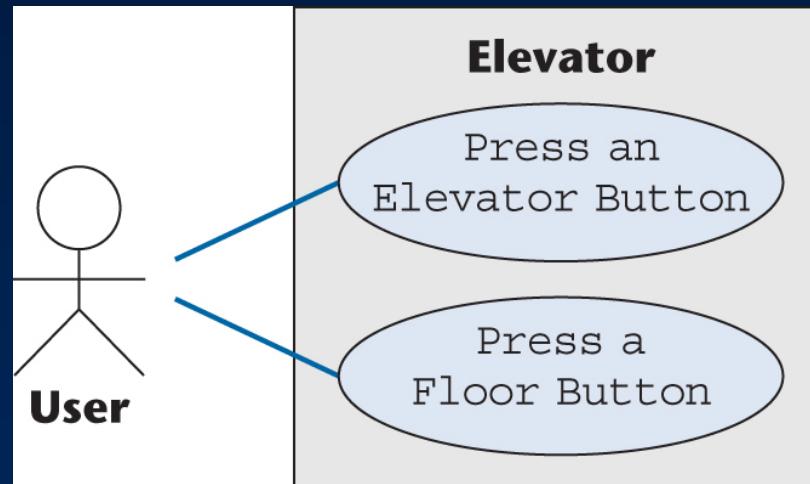


Figure 13.2

Scenarios

Slide 13.17

- A use case provides a generic description of the overall functionality
- A scenario is an instance of a use case
- Sufficient scenarios need to be studied to get a comprehensive insight into the target product being modeled

Normal Scenario: Elevator Problem

Slide 13.18

1. User A presses the Up floor button at floor 3 to request an elevator. User A wishes to go to floor 7.
2. The Up floor button is turned on.
3. An elevator arrives at floor 3. It contains User B, who has entered the elevator at floor 1 and pressed the elevator button for floor 9.
4. The elevator doors open.
5. The timer starts.
User A enters the elevator.
6. User A presses the elevator button for floor 7.
7. The elevator button for floor 7 is turned on.
8. The elevator doors close after a timeout.
9. The Up floor button is turned off.
10. The elevator travels to floor 7.
11. The elevator button for floor 7 is turned off.
12. The elevator doors open to allow User A to exit from the elevator.
13. The timer starts.
User A exits from the elevator.
14. The elevator doors close after a timeout.
15. The elevator proceeds to floor 9 with User B.

Figure 13.3

Exception Scenario: Elevator Problem

Slide 13.19

1. User A presses the Up floor button at floor 3 to request an elevator. User A wishes to go to floor 1.
2. The Up floor button is turned on.
3. An elevator arrives at floor 3. It contains User B, who has entered the elevator at floor 1 and pressed the elevator button for floor 9.
4. The elevator doors open.
5. The timer starts.
User A enters the elevator.
6. User A presses the elevator button for floor 1.
7. The elevator button for floor 1 is turned on.
8. The elevator doors close after a timeout.
9. The Up floor button is turned off.
10. The elevator travels to floor 9.
11. The elevator button for floor 9 is turned off.
12. The elevator doors open to allow User B to exit from the elevator.
13. The timer starts.
User B exits from the elevator.
14. The elevator doors close after a timeout.
15. The elevator proceeds to floor 1 with User A.

- Extract classes and their attributes
 - Represent them using a UML diagram
- One alternative: Deduce the classes from use cases and their scenarios
 - Possible danger: Often there are many scenarios, and hence
 - Too many candidate classes
- Other alternatives:
 - CRC cards (if you have domain knowledge)
 - Noun extraction

13.5.1 Noun Extraction

Slide 13.21

- A two-stage process
- Stage 1. Concise problem definition
 - Describe the software product in single paragraph
 - Buttons in elevators and on the floors control the movement of n elevators in a building with m floors.
Buttons illuminate when pressed to request the elevator to stop at a specific floor; the illumination is canceled when the request has been satisfied. When an elevator has no requests, it remains at its current floor with its doors closed

- Stage 2. Identify the nouns
 - Identify the nouns in the informal strategy
 - Buttons in elevators and on the floors control the movement of n elevators in a building with m floors.
Buttons illuminate when pressed to request the elevator to stop at a specific floor; the illumination is canceled when the request has been satisfied. When an elevator has no requests, it remains at its current floor with its doors closed
- Use the nouns as candidate classes

- Nouns
 - button, elevator, floor, movement, building, illumination, request, door
 - floor, building, door are outside the problem boundary — exclude
 - movement, illumination, request are abstract nouns — exclude (they may become attributes)
- Candidate classes:
 - **Elevator Class and Button Class**
- Subclasses:
 - **Elevator Button Class and Floor Button Class**

First Iteration of Class Diagram

Slide 13.24

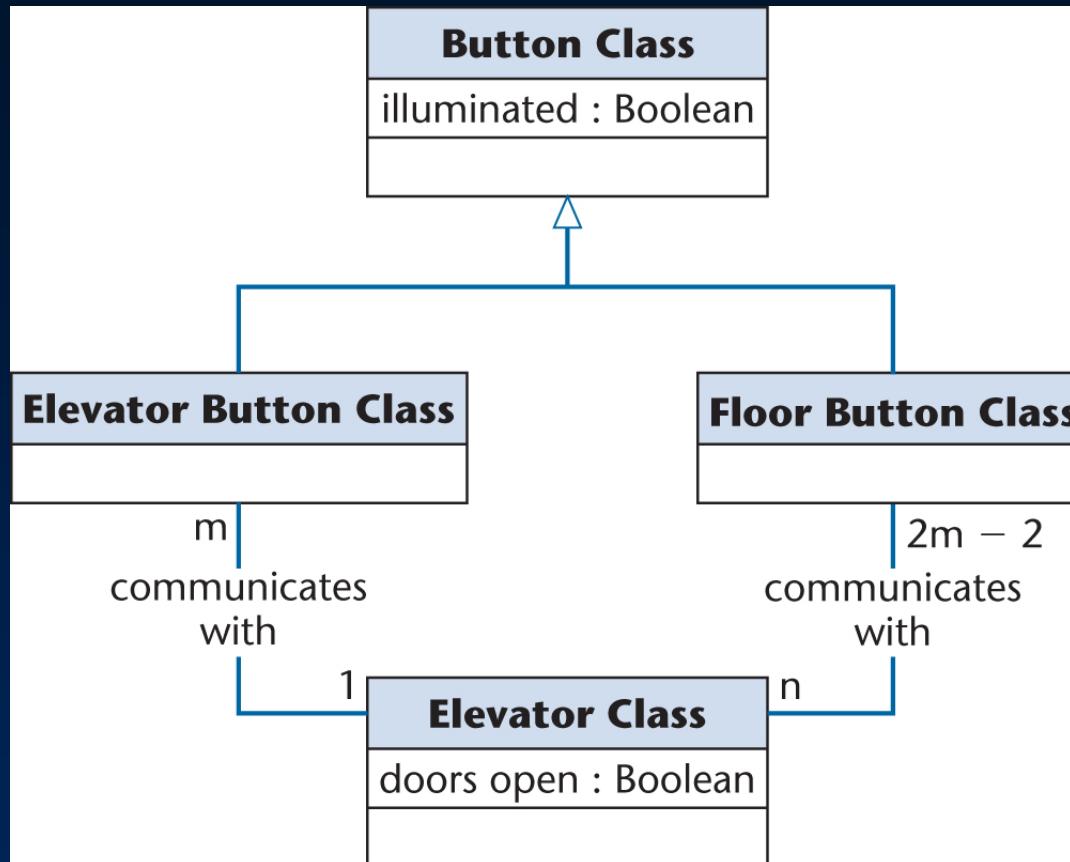


Figure 13.5

- Problem
 - Buttons do not communicate directly with elevators
 - We need an additional class: **Elevator Controller Class**

Second Iteration of Class Diagram

Slide 13.25

- All relationships are now 1-to-n
 - This makes design and implementation easier

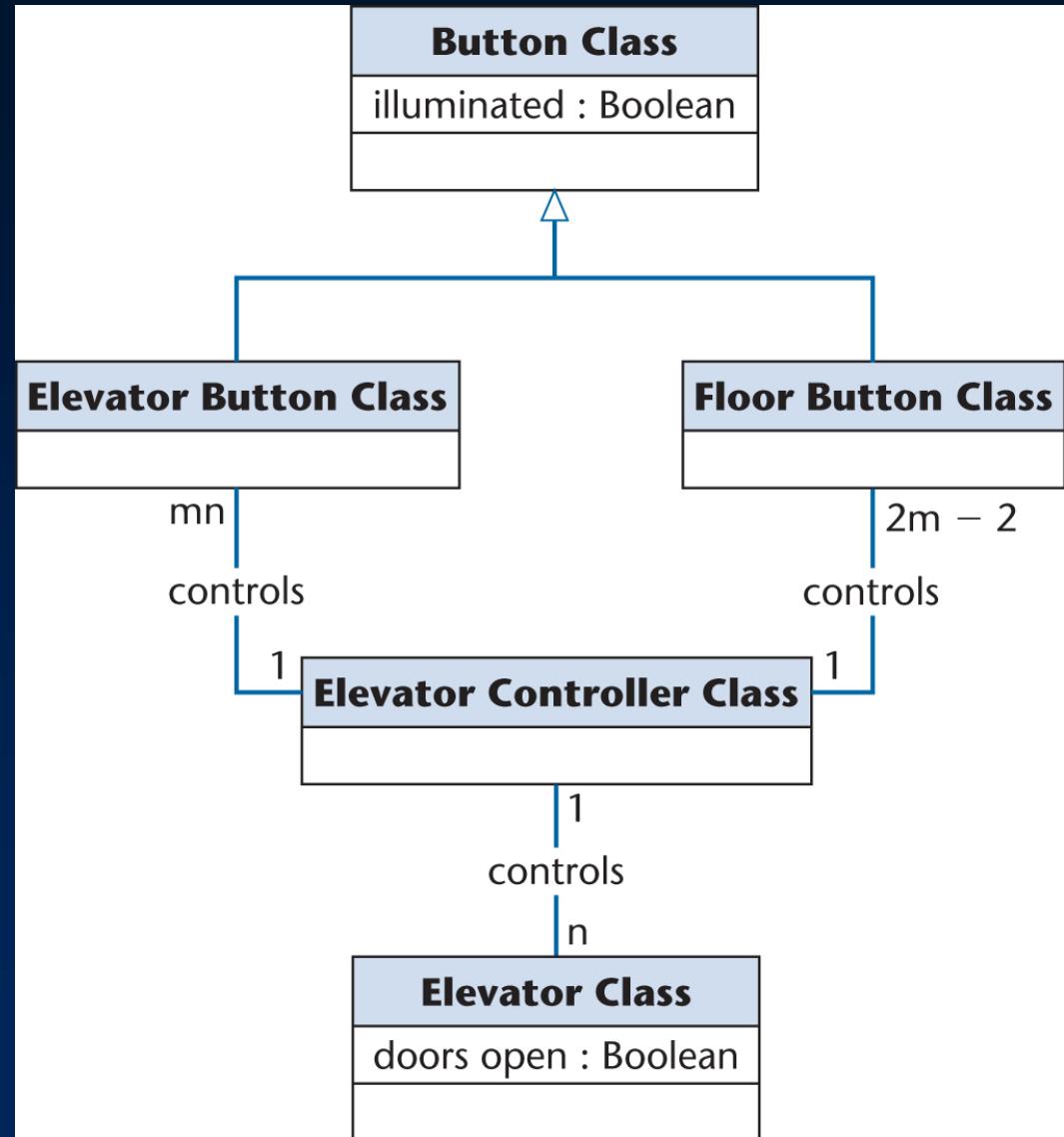


Figure 13.6

13.5.2 CRC Cards

Slide 13.26

- Used since 1989 for OOA
- For each class, fill in a card showing
 - Name of Class
 - Functionality (Responsibility)
 - List of classes it invokes (Collaboration)
- Now CRC cards are automated (CASE tool component)

- Strength
 - When acted out by team members, CRC cards are a powerful tool for highlighting missing or incorrect items
- Weakness
 - If CRC cards are used to identify entity classes, domain expertise is needed

13.6 Dynamic Modeling: The Elevator Problem Case Study

Slide 13.28

- Produce a UML statechart
- State, event, and predicate are distributed over the statechart

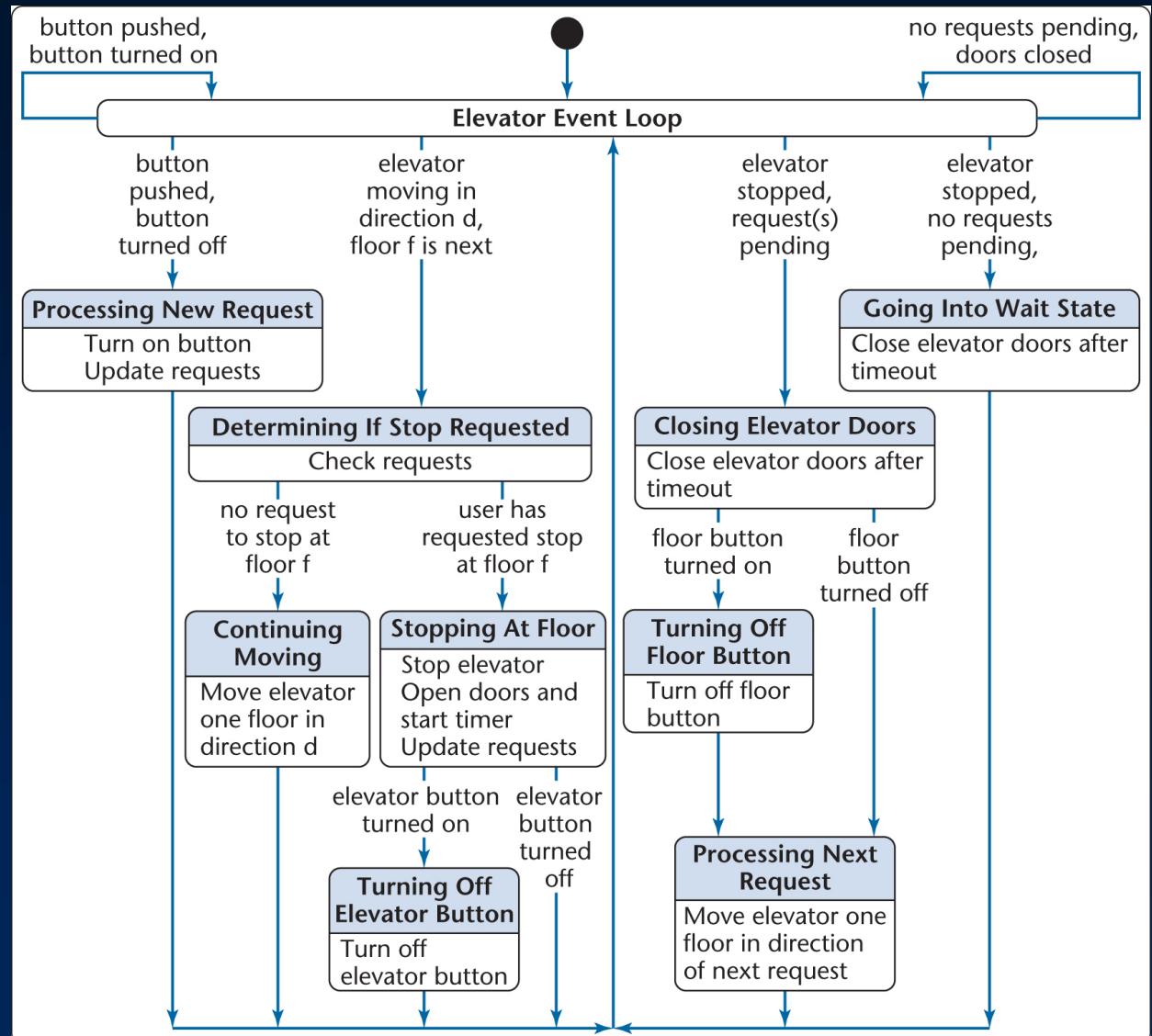


Figure 13.7

Dynamic Modeling: Elevator Problem (contd)

Slide 13.29

- This UML statechart is equivalent to the state transition diagram of Figures 11.15 through 11.17
- This is shown by considering specific scenarios
- In fact, a statechart is constructed by modeling the events of the scenarios

13.7 The Test Workflow: Object-Oriented Analysis

Slide 13.30

- CRC cards are an excellent testing technique

CLASS
Elevator Controller Class
RESPONSIBILITY
1. Turn on elevator button 2. Turn off elevator button 3. Turn on floor button 4. Turn off floor button 5. Move elevator up one floor 6. Move elevator down one floor 7. Open elevator doors and start timer 8. Close elevator doors after timeout 9. Check requests 10. Update requests
COLLABORATION
1. Elevator Button Class 2. Floor Button Class 3. Elevator Class

Figure 13.8

- Consider responsibility
 - 1. Turn on elevator button
- This is totally inappropriate for the object-oriented paradigm
 - Responsibility-driven design has been ignored
 - Information hiding has been ignored
- Responsibility
 - 1. Turn on elevator button

should be

 - 1. Send message to **Elevator Button Class** to turn itself on

- Also, a class has been overlooked
- The elevator doors have a *state* that changes during execution (class characteristic)
 - Add class **Elevator Doors Class**
 - Safety considerations
- Modify the CRC card

Second Iteration of the CRC Card

Slide 13.33

CLASS
Elevator Controller Class
RESPONSIBILITY
1. Send message to Elevator Button Class to turn on button 2. Send message to Elevator Button Class to turn off button 3. Send message to Floor Button Class to turn on button 4. Send message to Floor Button Class to turn off button 5. Send message to Elevator Class to move up one floor 6. Send message to Elevator Class to move down one floor 7. Send message to Elevator Doors Class to open 8. Start timer 9. Send message to Elevator Doors Class to close after timeout 10. Check requests 11. Update requests
COLLABORATION
1. Elevator Button Class (subclass) 2. Floor Button Class (subclass) 3. Elevator Doors Class 4. Elevator Class

Figure 13.9

- Having modified the class diagram, reconsider the
 - Use-case diagram (no change)
 - Class diagram (see the next slide)
 - Statecharts
 - Scenarios (see the slide after the next slide)

Third Iteration of Class Diagram

Slide 13.35

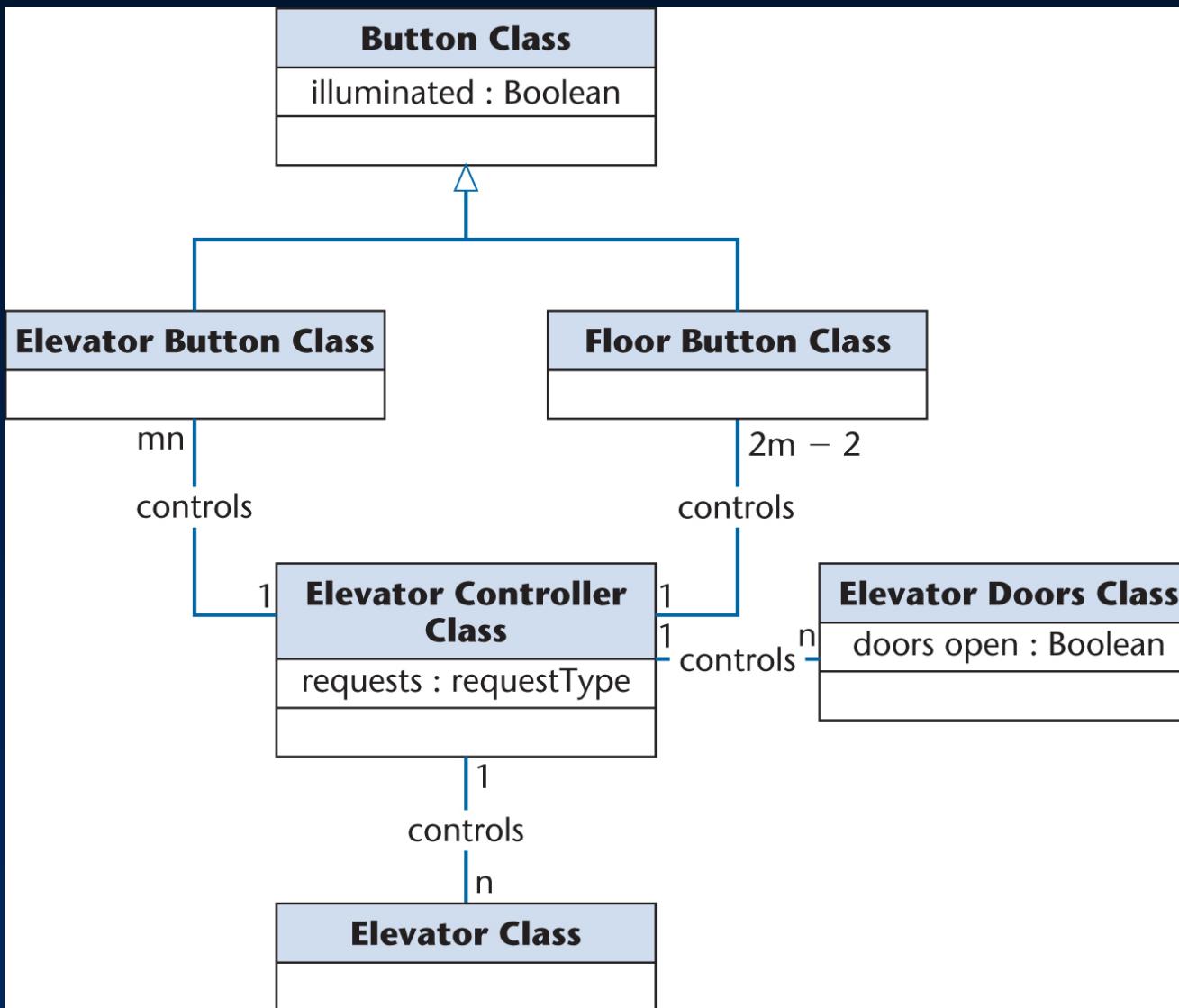


Figure 13.10

Second Iteration of the Normal Scenario:

Slide 13.36

1. User A presses the Up floor button at floor 3 to request an elevator. User A wishes to go to floor 7.
2. The floor button informs the elevator controller that the floor button has been pushed.
3. The elevator controller sends a message to the Up floor button to turn itself on.
4. The elevator controller sends a series of messages to the elevator to move itself up to floor 3. The elevator contains User B, who has entered the elevator at floor 1 and pressed the elevator button for floor 9.
5. The elevator controller sends a message to the elevator doors to open themselves.
6. The elevator controller starts the timer.
User A enters the elevator.
7. User A presses elevator button for floor 7.
8. The elevator button informs the elevator controller that the elevator button has been pushed.
9. The elevator controller sends a message to the elevator button for floor 7 to turn itself on.
10. The elevator controller sends a message to the elevator doors to close themselves after a timeout.
11. The elevator controller sends a message to the Up floor button to turn itself off.
12. The elevator controller sends a series of messages to the elevator to move itself up to floor 7.
13. The elevator controller sends a message to the elevator button for floor 7 to turn itself off.
14. The elevator controller sends a message to the elevator doors to open themselves to allow User A to exit from the elevator.
15. The elevator controller starts the timer.
User A exits from the elevator.
16. The elevator controller sends a message to the elevator doors to close themselves after a timeout.
17. The elevator controller sends a series of messages to the elevator to move itself up to floor 9 with User B.

Figure 13.11

Serious Problem with Third Iteration of Class Diagram

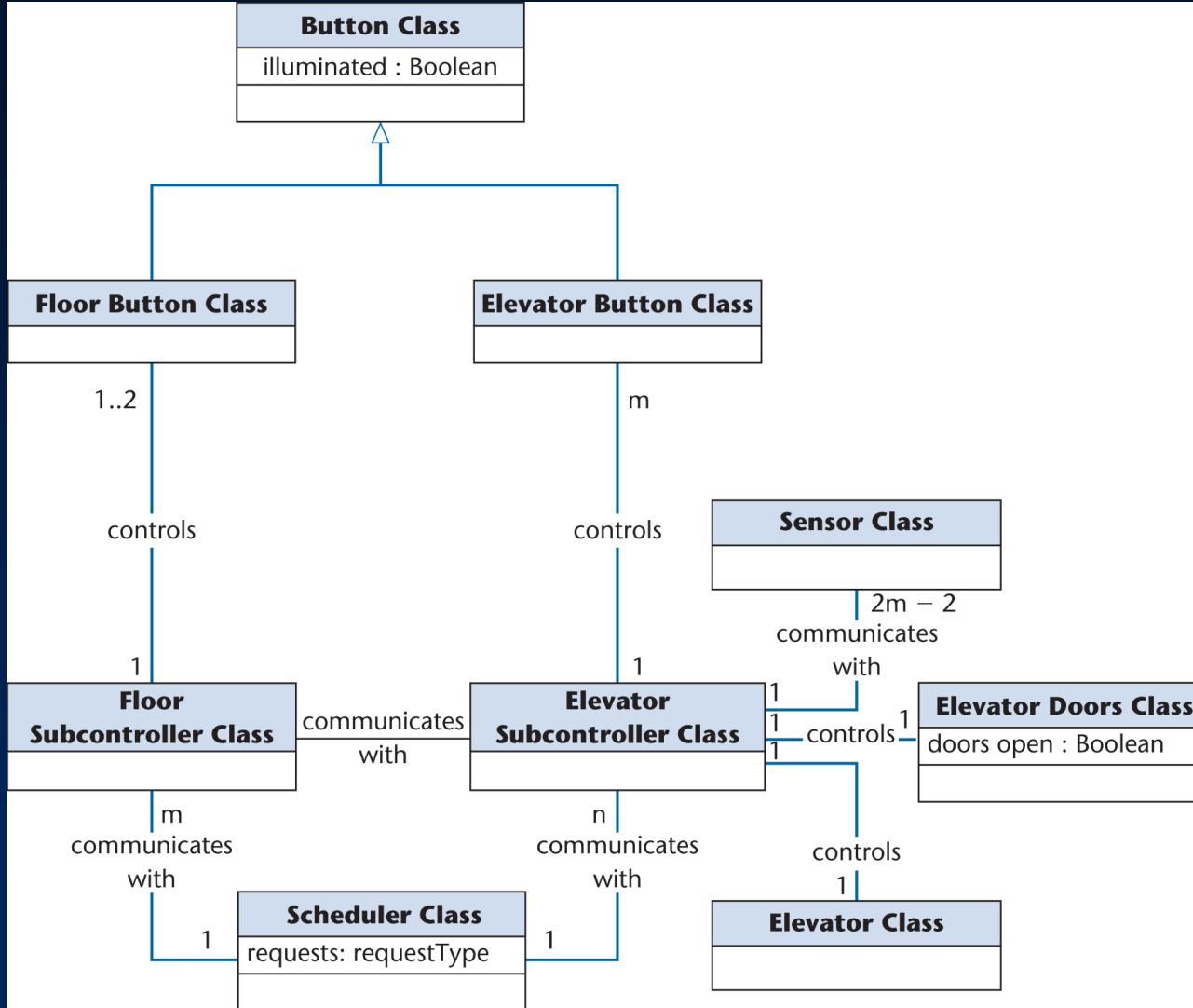
Slide 13.37

- Return to Figure 13.10 (third iteration of class diagram)
- **Elevator Controller Class** is running everything
- This is an example of a so-called “God class”
 - A class that is exposed to too much information, and
 - has too much control
- This is a well-known antipattern

- Distribute the control
 - Instead of having one central elevator controller

Fourth Iteration of Class Diagram

Slide 13.39



- Each of the n elevators now has its own elevator subcontroller
- Each of the m floors now has its own floor subcontroller
- The $(m + n)$ subcontrollers all communicate with a scheduler, which processes requests

- A **Floor Button Class** object is controlled by its corresponding **Floor Subcontroller Class** object
- An **Elevator Button Class** object is controlled by its corresponding **Elevator Subcontroller Class** object

- There is a sensor just above and just below each floor in each elevator shaft
- When an **Elevator Class** object nears or leaves a floor
 - The corresponding **Sensor Class** object informs the corresponding **Elevator Subcontroller Class** object

Distributed Decentralized Architecture (contd)

Slide 13.43

- The UML diagrams now need to be updated to reflect the fourth iteration of the class diagram

First Iteration of Statechart for Elevator Subcontroller Class

Slide 13.44

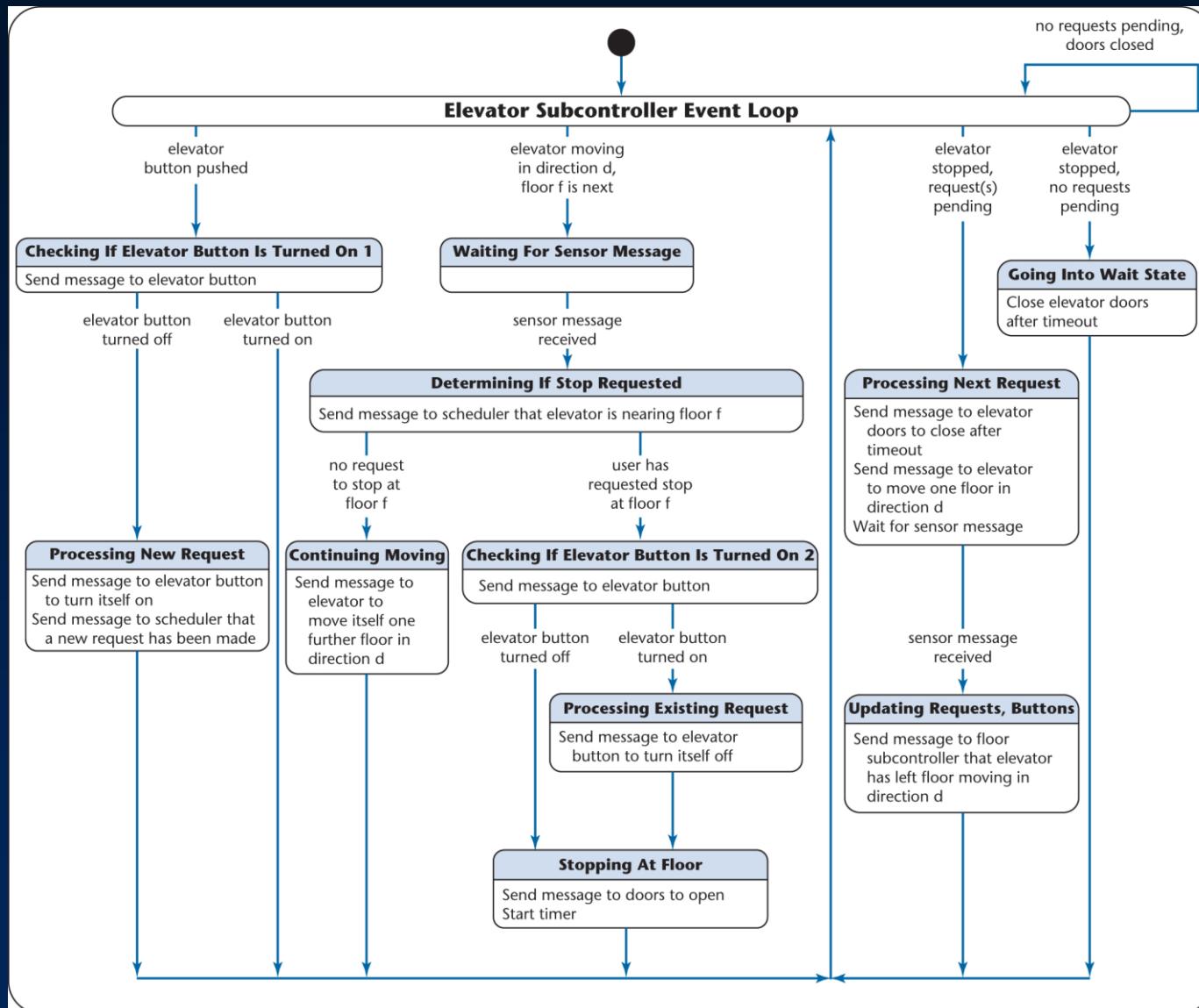


Figure 13.13

First Iteration of CRC Card for **Elevator Subcontroller Class**

Slide 13.45

CLASS
Elevator Subcontroller Class
RESPONSIBILITY
<ol style="list-style-type: none">1. Send message to Elevator Button Class to check if it is turned on2. Send message to Elevator Button Class to turn itself on3. Send message to Elevator Button Class to turn itself off4. Send message to Elevator Doors Class to open themselves5. Start timer6. Send message to Elevator Doors Class to close themselves after timeout7. Send message to Elevator Class to move itself up one floor8. Send message to Elevator Class to move itself down one floor9. Send message to Scheduler Class that a request has been made10. Send message to Scheduler Class that a request has been satisfied11. Send message to Scheduler Class to check if the elevator is to stop at the next floor12. Send message to Floor Subcontroller Class that elevator has left floor
COLLABORATION
<ol style="list-style-type: none">1. Elevator Button Class (subclass)2. Sensor Class3. Elevator Doors Class4. Elevator Class5. Scheduler Class6. Floor Subcontroller Class

Figure 13.14

- The object-oriented analysis is now fine
- We should rather say:
 - The object-oriented analysis is fine *for now*
- We may need to return to the object-oriented analysis workflow during the object-oriented design workflow

13.8 Extracting the Boundary and Control Classes

Slide 13.47

- Each
 - Input screen,
 - Output screen, and
 - Report

is modeled by its own boundary class
- Each nontrivial computation is modeled by a control class

13.9 The Initial Functional Model: MSG Foundation

Slide 13.48

- Recall the seventh iteration of the use-case diagram

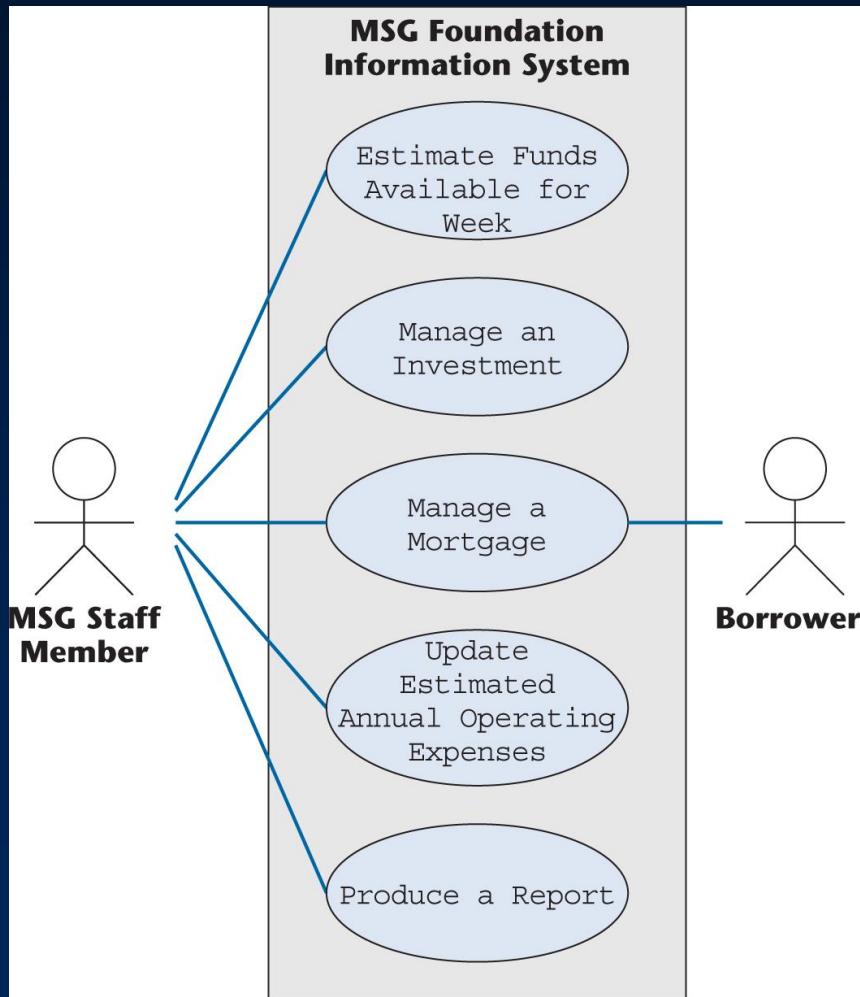


Figure 13.15

- One possible extended scenario

An MSG Foundation staff member wants to update the annual real-estate tax on a home for which the Foundation has provided a mortgage.

1. The staff member enters the new value of the annual real-estate tax.
2. The information system updates the date on which the annual real-estate tax was last changed.

Possible Alternative

- A. The staff member enters the mortgage number incorrectly.

Figure 13.16

- A second extended scenario

There is a change in the weekly income of a couple who have borrowed money from the MSG Foundation. They wish to have their weekly income updated in the Foundation records by an MSG staff member so that their mortgage payments will be correctly computed.

1. The staff member enters the new value of the weekly income.
2. The information system updates the date on which the weekly income was last changed.

Possible Alternatives

- A. The staff member enters the mortgage number incorrectly.
- B. The borrowers do not bring documentation regarding their new income.

Figure 13.17

- One possible scenario

An MSG Foundation staff member wishes to determine the funds available for mortgages this week.

1. For each investment, the information system extracts the estimated annual return on that investment. It sums the separate returns and divides the result by 52 to yield the estimated investment income for the week.
2. The information system then extracts the estimated annual MSG Foundation operating expenses and divides the result by 52.
3. For each mortgage:
 - 3.1 The information system computes the amount to be paid this week by adding the principal and interest payment to $\frac{1}{52}$ nd of the sum of the annual real-estate tax and the annual homeowner's insurance premium.
 - 3.2 It then computes 28 percent of the couple's current gross weekly income.
 - 3.3 If the result of Step 3.1 is greater than the result of Step 3.2, then it determines the mortgage payment for the week as the result of Step 3.2, and the amount of the grant for this week as the difference between the result of Step 3.1 and the result of Step 3.2.
 - 3.4 Otherwise, it takes the mortgage payment for this week as the result of Step 3.1, and there is no grant for the week.
4. The information system sums the mortgage payments of Steps 3.3 and 3.4 to yield the estimated total mortgage payments for the week.
5. It sums the grant payments of Step 3.3 to yield the estimated total grant payments for the week.
6. The information system adds the results of Steps 1 and 4 and subtracts the results of Steps 2 and 5. This is the total amount available for mortgages for the current week.
7. Finally, the software product prints the total amount available for new mortgages during the current week.

Figure 13.18

- One possible scenario

An MSG staff member wishes to print a list of all mortgages.

1. The staff member requests a report listing all mortgages.

Figure 13.19

- Another possible scenario

An MSG staff member wishes to print a list of all investments.

1. The staff member requests a report listing all investments.

Figure 13.20

13.10 The Initial Class Diagram: MSG Foundation

Slide 13.54

- The aim of entity modeling step is to extract the entity classes, determine their interrelationships, and find their attributes
- Usually, the best way to begin this step is to use the two-stage noun extraction method

Noun Extraction: MSG Foundation

Slide 13.55

- Stage 1: Describe the information system in a single paragraph
 - Weekly reports are to be printed showing how much money is available for mortgages. In addition, lists of investments and mortgages must be printed on demand.

Noun Extraction: MSG Foundation (contd)

Slide 13.56

- Stage 2: Identify the nouns in this paragraph
 - Weekly reports are to be printed showing how much money is available for mortgages. In addition, lists of investments and mortgages must be printed on demand.
- The nouns are report, money, mortgage, list, and investment

Noun Extraction: MSG Foundation (contd)

Slide 13.57

- Nouns report and list are not long lived, so they are unlikely to be entity classes (report will surely turn out to be a boundary class)
- money is an abstract noun
- This leaves two candidate entity classes
 - **Mortgage Class** and **Investment Class**

First Iteration of the Initial Class Diagram

Slide 13.58

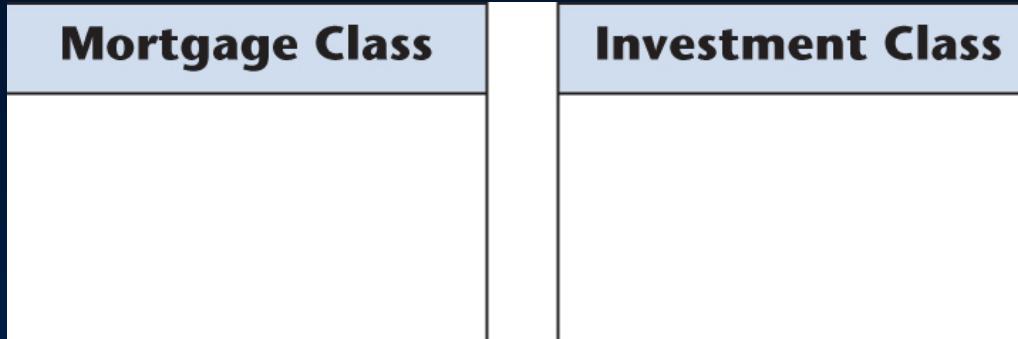


Figure 13.21

Second Iteration of the Initial Class Diagram

Slide 13.59

- Operations performed on the two entity classes are likely to be very similar
 - Insertions, deletions, and modifications
 - All members of both entity classes have to be printed on demand
- **Mortgage Class** and **Investment Class** should be subclasses of a superclass called **Asset Class**

Second Iteration of Initial Class Diagram (contd)

Slide 13.60

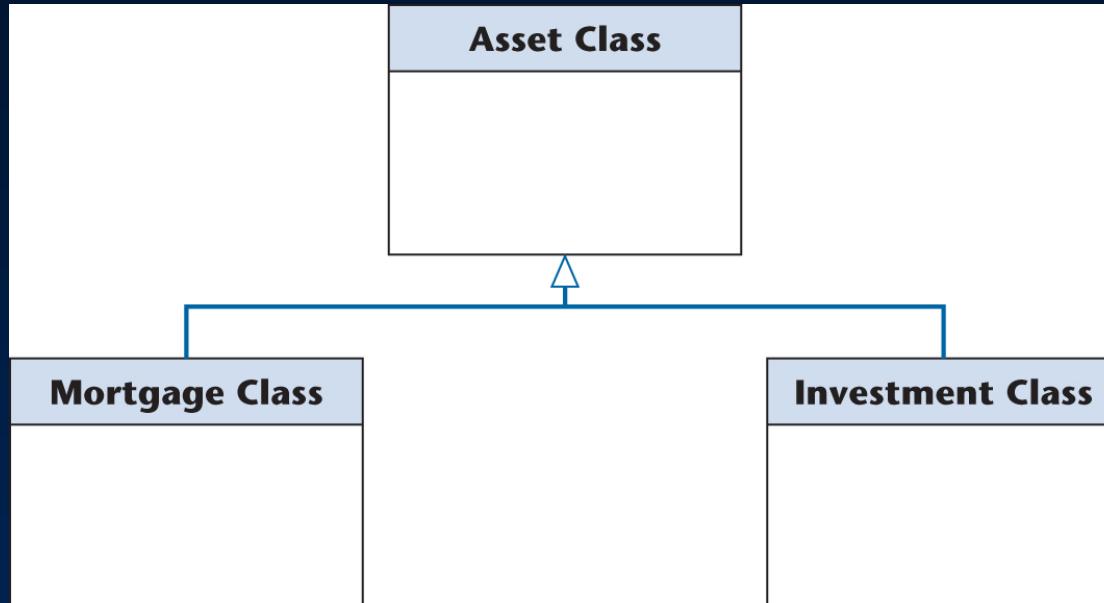


Figure 13.22

Back to the Requirements Workflow

Slide 13.61

- The current five use cases include Manage a Mortgage and Manage an Investment
- These two can now be combined into a single use case, Manage an Asset

Eighth Iteration of the Use-Case Diagram

Slide 13.62

- The new use case is shaded

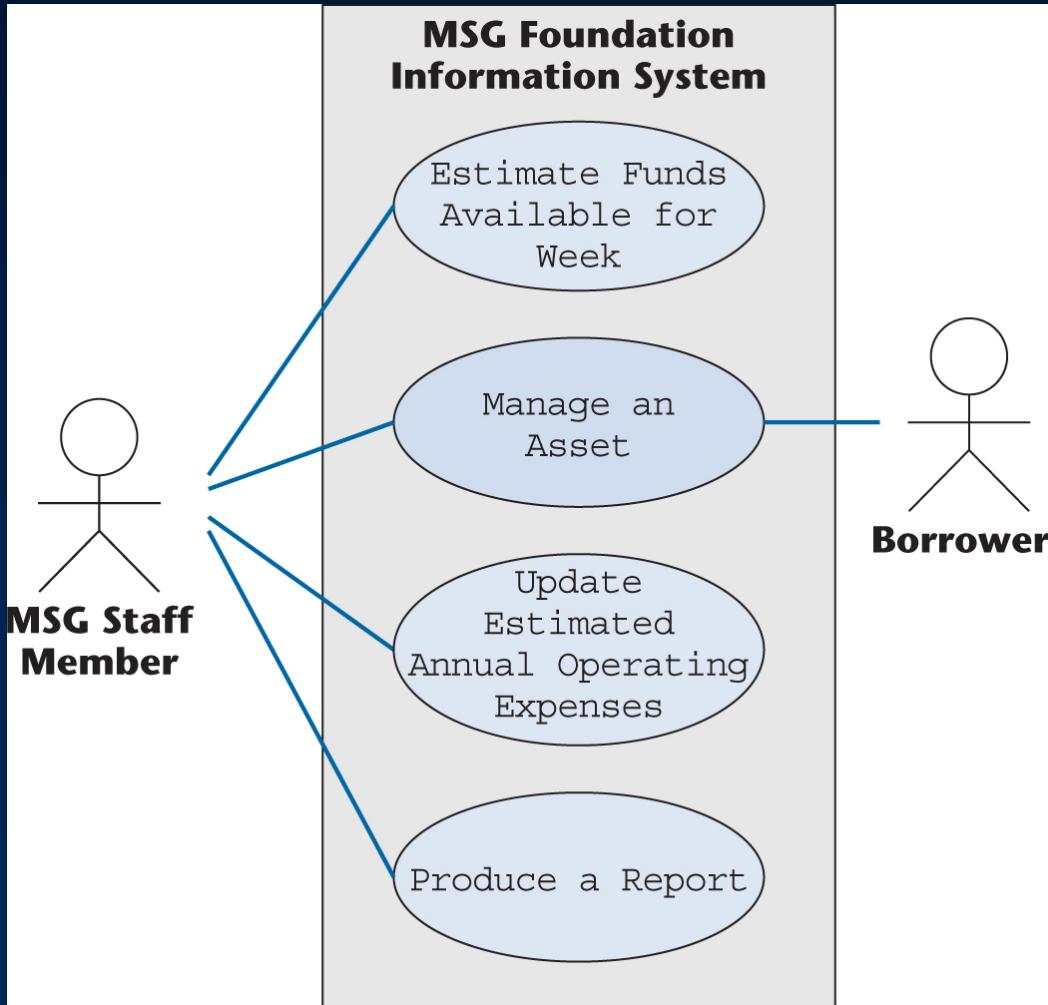


Figure 13.23

Initial Class Diagram: MSG Foundation (contd)

Slide 13.63

- Finally, we add the attributes of each class to the class diagram
 - For the MSG Foundation case study, the result is shown on the next slide
- The empty rectangle at the bottom of each box will later be filled with the operations of that class

Second Iteration of Initial Class Diagram (contd)

Slide 13.64

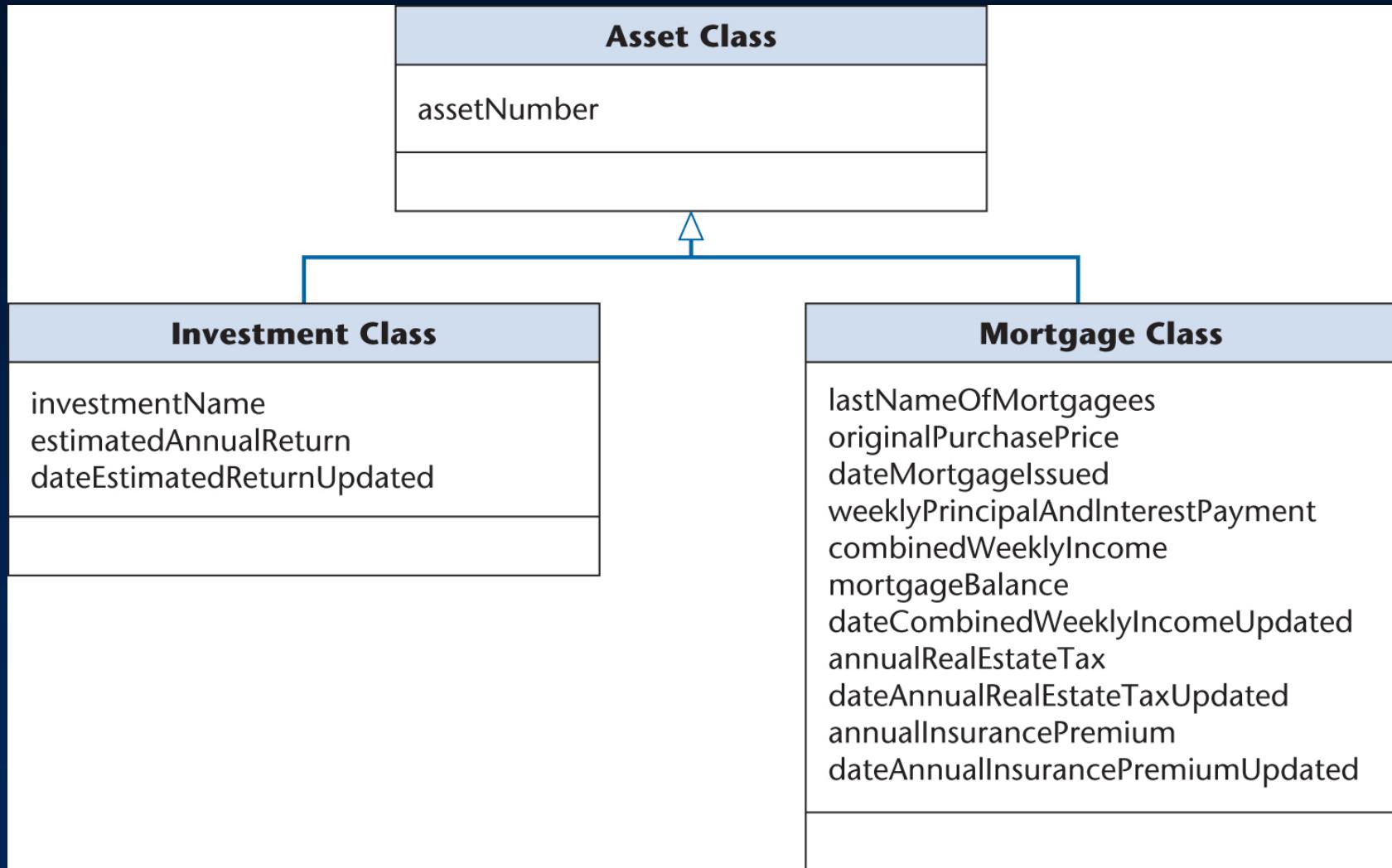


Figure 13.24

Iteration and Incrementation

Slide 13.65

- The phrase “iterate and increment” also includes the possibility of having to decrement what has been developed to date
 - A mistake may have been made, and backtracking is needed
 - As a consequence of reorganizing the UML models, one or more artifacts may have become superfluous

- Dynamic modeling is the third step in extracting the entity classes
- A statechart is constructed that reflects all the operations performed by or to the software product
- The operations are determined from the scenarios

Initial Dynamic Model: MSG Foundation (contd)

Slide 13.67

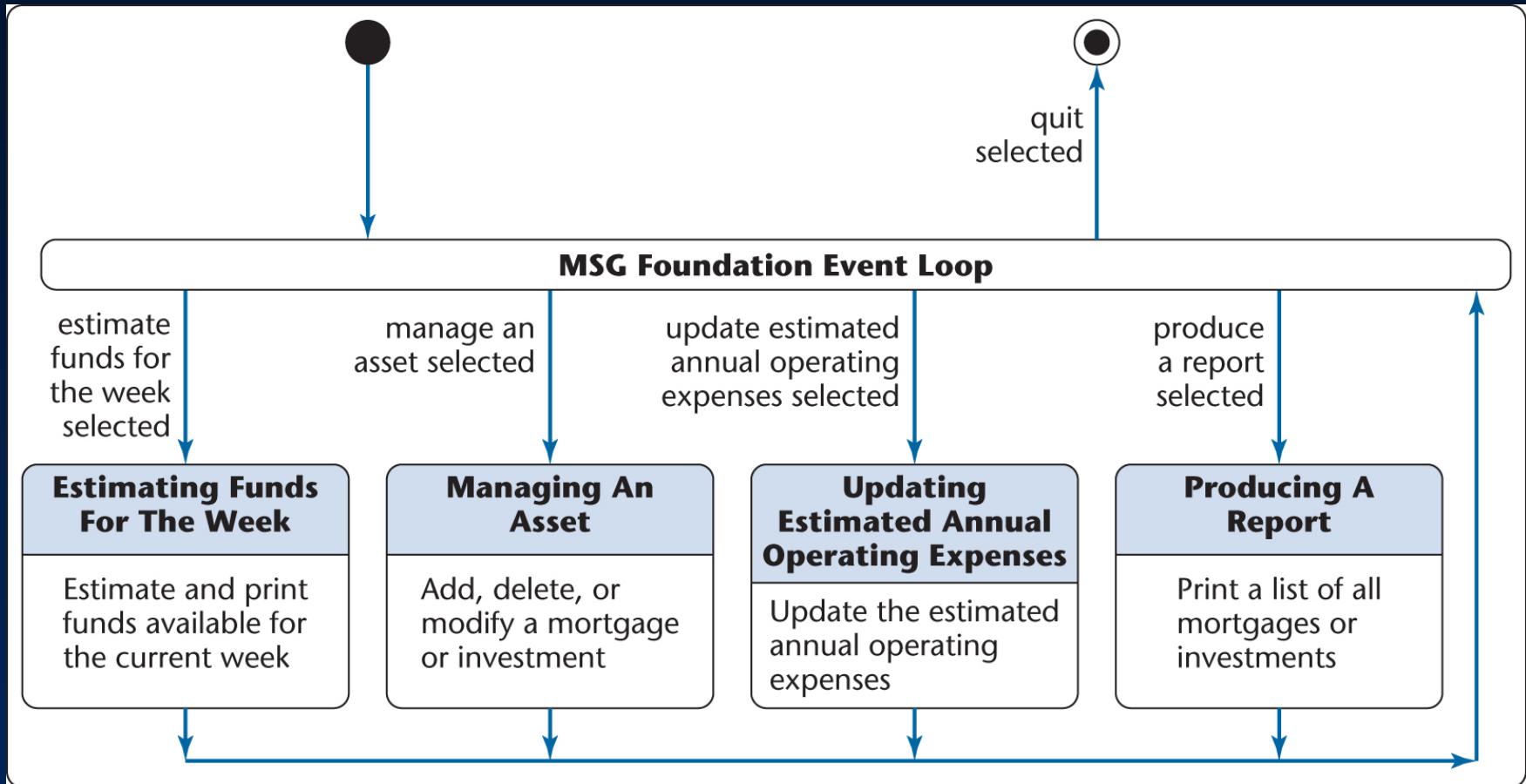


Figure 13.25

Initial Dynamic Model: MSG Foundation (contd)

Slide 13.68

- The statechart reflects the operations of the complete MSG Foundation information system
 - The solid circle on the top left represents the initial state, the starting point of the statechart
 - The white circle containing the small black circle on the top right represents the final state
 - States other than the initial and final states are represented by rectangles with rounded corners
 - The arrows represent possible transitions from state to state

- In state **MSG Foundation Information System Loop**, one of five events can occur
- An MSG staff member can issue one of five commands:
 - estimate funds for the week
 - manage an asset
 - update estimated annual operating expenses
 - produce a report, or
 - quit

Initial Dynamic Model: MSG Foundation (contd)

Slide 13.70

- These possibilities are indicated by the five events
 - estimate funds for the week selected
 - manage an asset selected
 - update estimated annual operating expenses selected
 - produce a report selected, and
 - quit selected
- An event causes a transition between states

Initial Dynamic Model: MSG Foundation (contd)

Slide 13.71

- An MSG staff member selects an option by clicking on the menu

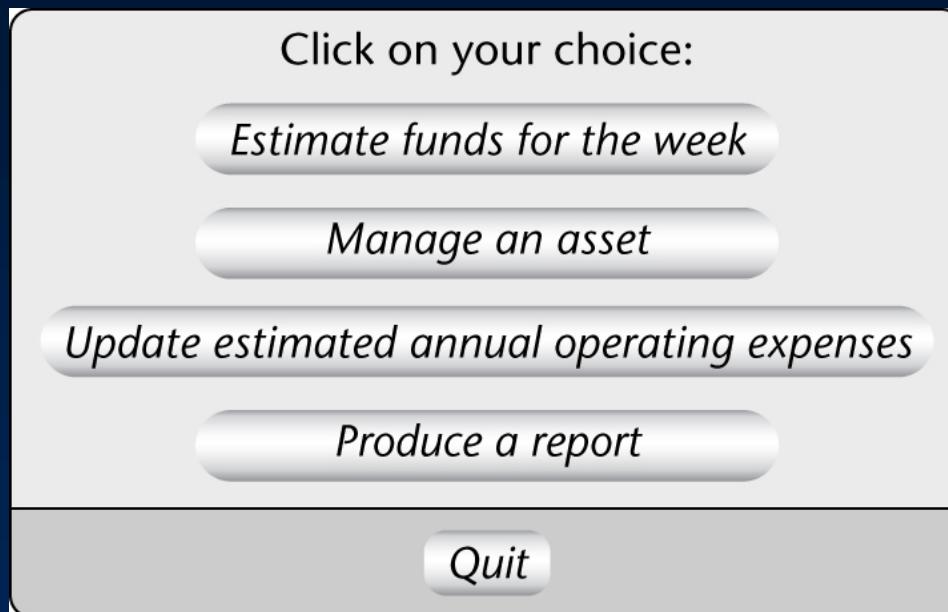


Figure 13.26

- This graphical user interface (GUI) requires special software

Initial Dynamic Model: MSG Foundation (contd)

Slide 13.72

- Equivalent textual user interface that can run on any computer

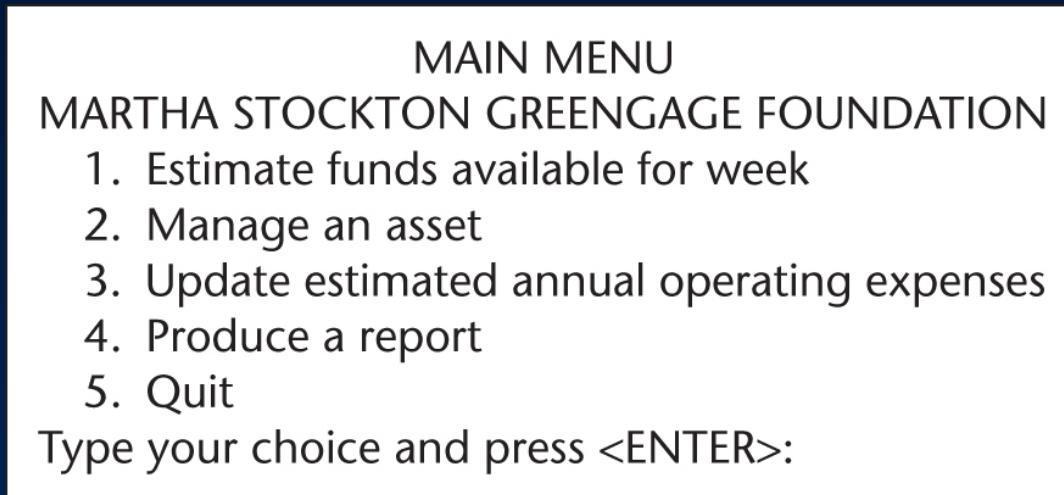


Figure 13.27

- The initial functional model, the initial class diagram, and the initial dynamic model are completed
 - Checking them reveals a fault
- In the initial statechart, consider state **Update Estimated Annual Operating Expenses** with operation `Update the estimated annual operating expenses`
 - This operation has to be performed on the current value of the estimated annual operating expense

- But where is the value of the estimated annual operating expenses to be found?
- Currently there is only one class (**Asset Class**) and its two subclasses
 - Neither is appropriate for storing the estimated annual operating expenses

Revising the Entity Classes: MSG Foundation (contd)

Slide 13.75

- The only way a value can be stored on a long-term basis is as an attribute of an instance of that class or its subclasses
- Another entity class is needed for storing the estimated annual operating expenses
 - **MSG Application Class**

Third Iteration of the Initial Class Diagram: MSG Foundation

Slide 13.76

- **MSG Application Class** has other attributes as well
 - Attributes that do not appertain to the assets

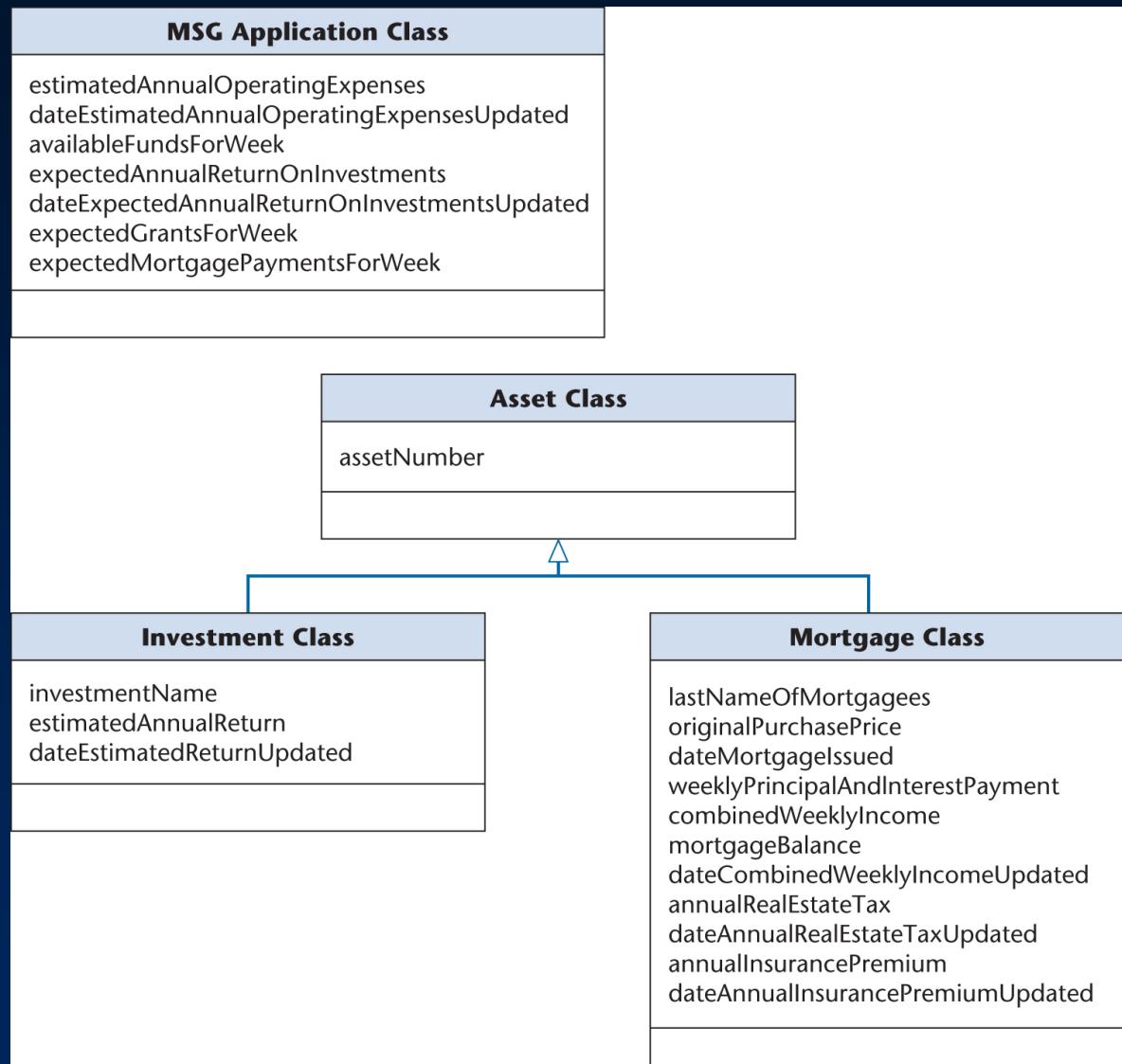


Figure 13.28

Third Iteration of the Initial Class Diagram: MSG Foundation

Slide 13.77

- The class diagram redrawn to show the prototypes

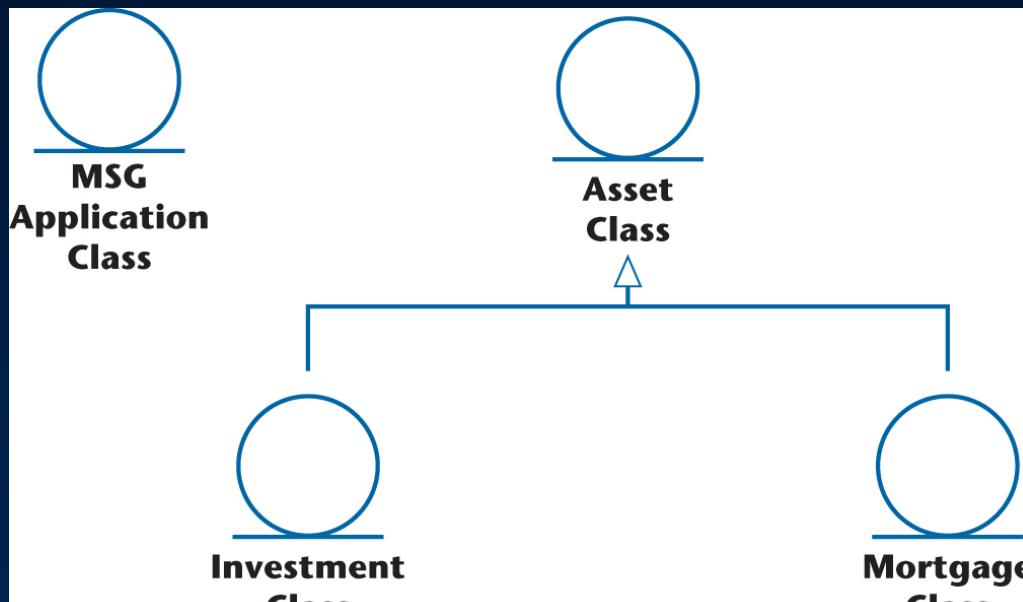


Figure 13.29

- It is usually easy to extract boundary classes
 - Each input screen, output screen, and printed report is generally modeled by a boundary class
- One screen should be adequate for all four MSG Foundation use cases
 - » Estimate Funds Available for Week
 - » Manage an Asset
 - » Update Estimated Annual Operating Expenses
 - » Produce a Report
- Accordingly there is one initial boundary class
 - **User Interface Class**

- Three reports have to be printed
 - The estimated funds for the week report
 - The listing of all mortgages
 - The listing of all investments
- Each of these has to be modeled by a separate boundary class
 - **Estimated Funds Report Class**
 - **Mortgages Report Class**
 - **Investments Report Class**

Extracting Boundary Classes: MSG (contd)

Slide 13.80

- Here are the four initial boundary classes

User Interface Class
Estimated Funds Report Class
Mortgages Report Class
Investments Report Class

Figure 13.30

- There are three reports:
 - The purchases report
 - The sales report
 - The future trends report
- The content of each report is different
 - Each report therefore has to be modeled by a separate boundary class

- Each computation is usually modeled by a control class
- The MSG Foundation case study has just one
 - ▶ Estimate the funds available for the week
- There is one initial control class

Estimate Funds for Week Class

Figure 13.31

Class Extraction (contd)

Slide 13.83

- The description of class extraction is complete
- We now therefore return to the Unified Process

- The process of extending and refining use cases is called *use-case realization*

Use-Case Realization (contd)

Slide 13.85

- The verb “realize” is used at least 3 different ways:
 - Understand (“Harvey slowly began to realize that he was in the wrong classroom”);
 - Receive (“Ingrid will realize a profit of \$45,000 on the stock transaction”); and
 - Accomplish (“Janet hopes to realize her dream of starting a computer company”)
- In the phrase “realize a use case,” the word “realize” is used in this last sense

Use-Case Realization (contd)

Slide 13.86

- The realization of a specific scenario of a use case is depicted using an interaction diagram
 - Either a sequence diagram or collaboration diagram
- Consider use case Estimate Funds Available for Week
- We have previously seen
 - The use case
 - The description of the use case

13.15.1 Estimate Funds Available for Week Use Case

Slide 13.87

- Use-case diagram

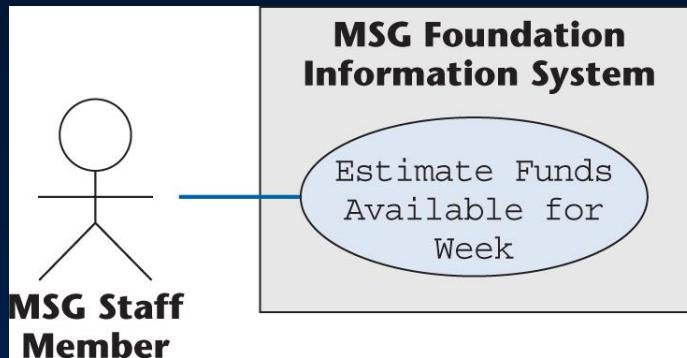


Figure 13.32

- Description of use case

Brief Description

The Estimate Funds Available for Week use case enables an MSG Foundation staff member to estimate how much money the Foundation has available that week to fund mortgages.

Step-by-Step Description

1. For each investment, extract the estimated annual return on that investment. Summing the separate returns and dividing the result by 52 yields the estimated investment income for the week.
2. Determine the estimated MSG Foundation operating expenses for the week by extracting the estimated annual MSG Foundation operating expenses and dividing by 52.
3. For each mortgage:
 - 3.1 The amount to be paid this week is the total of the principal and interest payment and $\frac{1}{52}$ nd of the sum of the annual real-estate tax and the annual homeowner's insurance premium.
 - 3.2 Compute 28 percent of the couple's current gross weekly income.
 - 3.3 If the result of Step 3.1 is greater than the result of Step 3.2, then the mortgage payment for this week is the result of Step 3.2, and the amount of the grant for this week is the difference between the result of Step 3.1 and the result of Step 3.2.
 - 3.4 Otherwise, the mortgage payment for this week is the result of Step 3.1, and there is no grant this week.
4. Summing the mortgage payments of Steps 3.3 and 3.4 yields the estimated total mortgage payments for the week.
5. Summing the grant payments of Step 3.3 yields the estimated total grant payments for the week.
6. Add the results of Steps 1 and 4 and subtract the results of Steps 2 and 5. This is the total amount available for mortgages for the current week.
7. Print the total amount available for new mortgages during the current week.

Figure 13.33

- Class diagram (classes that enter into the use case)

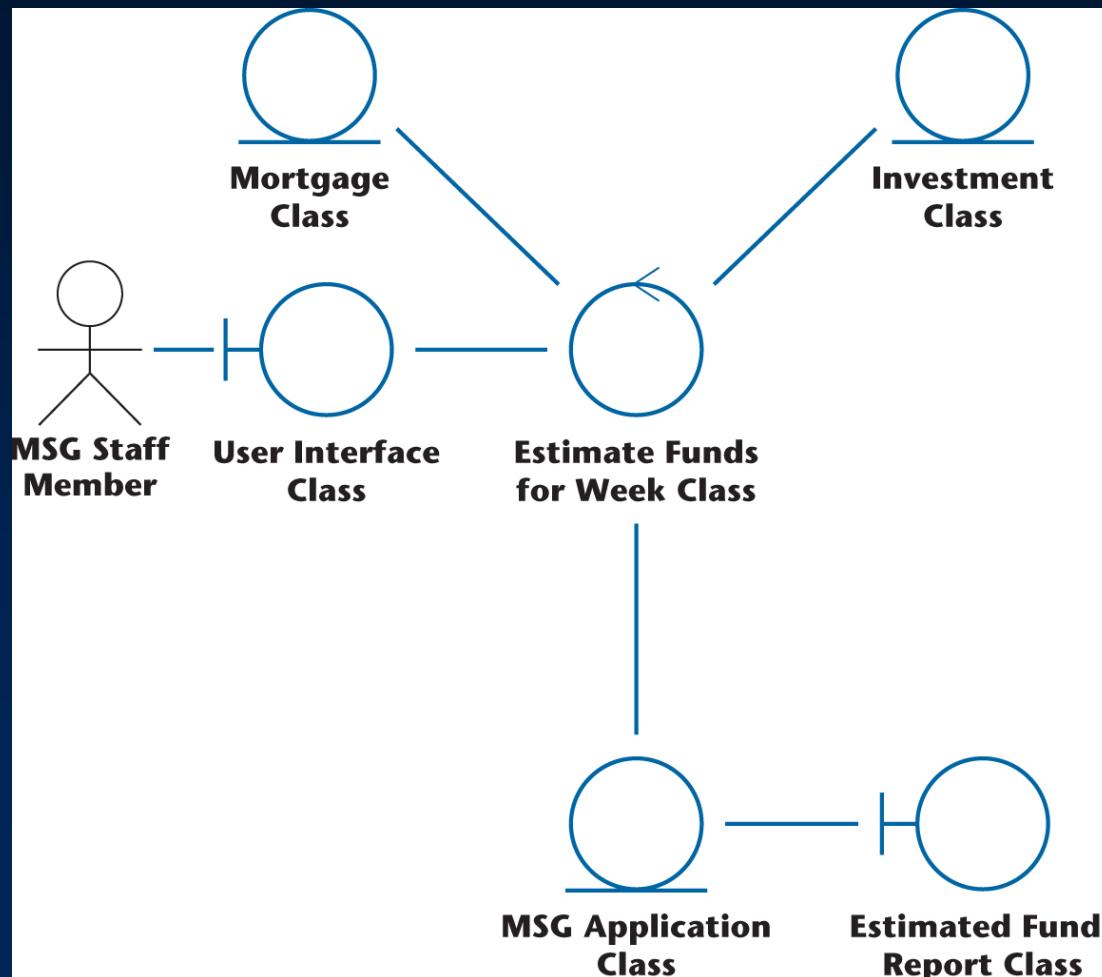


Figure 13.34

- The six classes that enter into this use case are:
 - **User Interface Class**
 - » This class models the user interface
 - **Estimate Funds for Week Class**
 - » This control class models the computation of the estimate of the funds that are available to fund mortgages during that week
 - **Mortgage Class**
 - » This class models the estimated grants and payments for the week

- **Investment Class**
 - » This class models the estimated return on investments for the week
- **MSG Application Class**
 - » This class models the estimated return on investments for the week
- **Estimated Funds Report Class**
 - » This class models the printing of the report

- Scenario (one possible instance of the use case)

An MSG Foundation staff member wishes to determine the funds available for mortgages this week.

1. For each investment, the information system extracts the estimated annual return on that investment. It sums the separate returns and divides the result by 52 to yield the estimated investment income for the week.
2. The information system then extracts the estimated annual MSG Foundation operating expenses and divides the result by 52.
3. For each mortgage:
 - 3.1 The information system computes the amount to be paid this week by adding the principal and interest payment to $\frac{1}{52}$ nd of the sum of the annual real-estate tax and the annual homeowner's insurance premium.
 - 3.2 It then computes 28 percent of the couple's current gross weekly income.
 - 3.3 If the result of Step 3.1 is greater than the result of Step 3.2, then it determines the mortgage payment for the week as the result of Step 3.2, and the amount of the grant for this week as the difference between the result of Step 3.1 and the result of Step 3.2.
 - 3.4 Otherwise, it takes the mortgage payment for this week as the result of Step 3.1, and there is no grant for the week.
4. The information system sums the mortgage payments of Steps 3.3 and 3.4 to yield the estimated total mortgage payments for the week.
5. It sums the grant payments of Step 3.3 to yield the estimated total grant payments for the week.
6. The information system adds the results of Steps 1 and 4 and subtracts the results of Steps 2 and 5. This is the total amount available for mortgages for the current week.
7. Finally, the software product prints the total amount available for new mortgages during the current week.

Figure 13.35

- A working information system uses objects, not classes
 - Example: A specific mortgage cannot be represented by **Mortgage Class** but rather by an object, a specific instance of **Mortgage Class**
- Such an object is denoted by : **Mortgage Class**

- A class diagram shows the classes in the use case and their relationships
 - It does not show the objects nor the sequence of messages as they are sent from object to object
- Something more is needed

- Collaboration diagram (of the realization of the scenario of the use case)

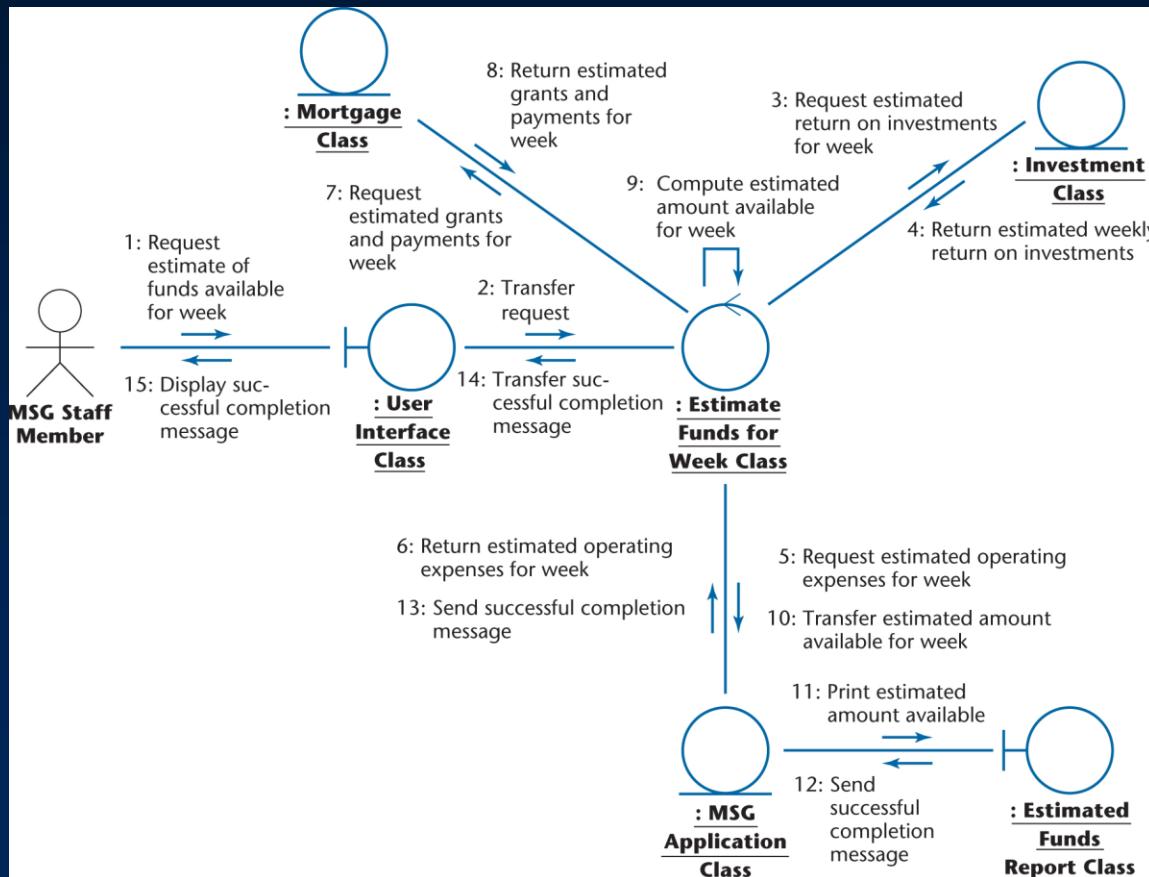


Figure 13.36

- The collaboration diagram shows the objects as well as the messages, numbered in the order in which they are sent in the specific scenario

- Item 1:
 - The staff member wants to compute the funds available for the week
 - In the collaboration diagram, this is modeled by message
 - » 1: Request estimate of funds available for week from **MSG Staff Member** to : User Interface Class, an instance of **User Interface Class**

- Item 2
 - This request is passed on to : **Estimate Funds for Week Class**, an instance of the control class that actually performs the calculation
 - This is modeled by message
 - » 2: Transfer request
- Four separate financial estimates are now determined by : **Estimate Funds for Week Class**

- Item 3

- In Step 1 of the scenario, the estimated annual return on investments is summed for each investment and the result divided by 52
 - This extraction of the estimated weekly return is modeled by message
 - » 3: Request estimated return on investments for week from : **Estimate Funds for Week Class** to : **Investment Class** followed by message
 - » 4: Return estimated weekly return on investments in the other direction

- Item 4

- In Step 2 of the scenario, the weekly operating expenses are estimated by taking the estimated annual operating expenses and dividing by 52
 - This extraction of the weekly expenses is modeled by message
 - » 5: Request estimated operating expenses for week

from : **Estimate Funds for Week Class** to : **MSG Application Class** followed by message

- » 6: Return estimated operating expenses for week in the other direction

- Item 5

- In Steps 3, 4, and 5 of the scenario, two estimates are determined
 - » the estimated grants for the week, and
 - » the estimated payments for the week
 - This is modeled by message
 - » 7: Request estimated grants and payments for week

from : **Estimate Funds for Week Class** to : **Mortgage Class**, and by message

- » 8: Return estimated grants and payments for week in the other direction

- Item 6

- Now the arithmetic computation of Step 6 of the scenario is performed
- This is modeled by message
 - » 9: Compute estimated amount available for week
- This is a self call
- **: Estimate Funds for Week Class** tells itself to perform the calculation
- The result of the computation is stored in **: MSG Application Class** by message
 - » 10: Transfer estimated amount available for week

- Item 7
 - The result is printed in Step 7 of the scenario
 - This is modeled by message
 - » 11: Print estimated amount available
 - from : **MSG Application Class** to : **Estimated Funds Report Class**

- Item 8

- Finally, an acknowledgment is sent to the MSG staff member that the task has been successfully completed
- This is modeled by messages
 - » 12: Send successful completion message
 - » 13: Send successful completion message
 - » 14: Transfer successful completion message, and
 - » 15: Display successful completion message

- No client will approve the specification document without understanding it
- Accordingly, a written description of the collaboration diagram is needed, the flow of events

- The flow of events of the collaboration diagram of the realization of the scenario of the use case

An MSG staff member requests an estimate of the funds available for mortgages for the week (1, 2). The information system estimates the return on investments for the week (3, 4), the operating expenses for the week (5, 6), and the grants and payments for the week (7, 8). Then it estimates (9), stores (10), and prints out (11–15) the funds available for the week.

Figure 13.37

- Sequence diagram equivalent to the collaboration diagram of the realization of the scenario of the use case)

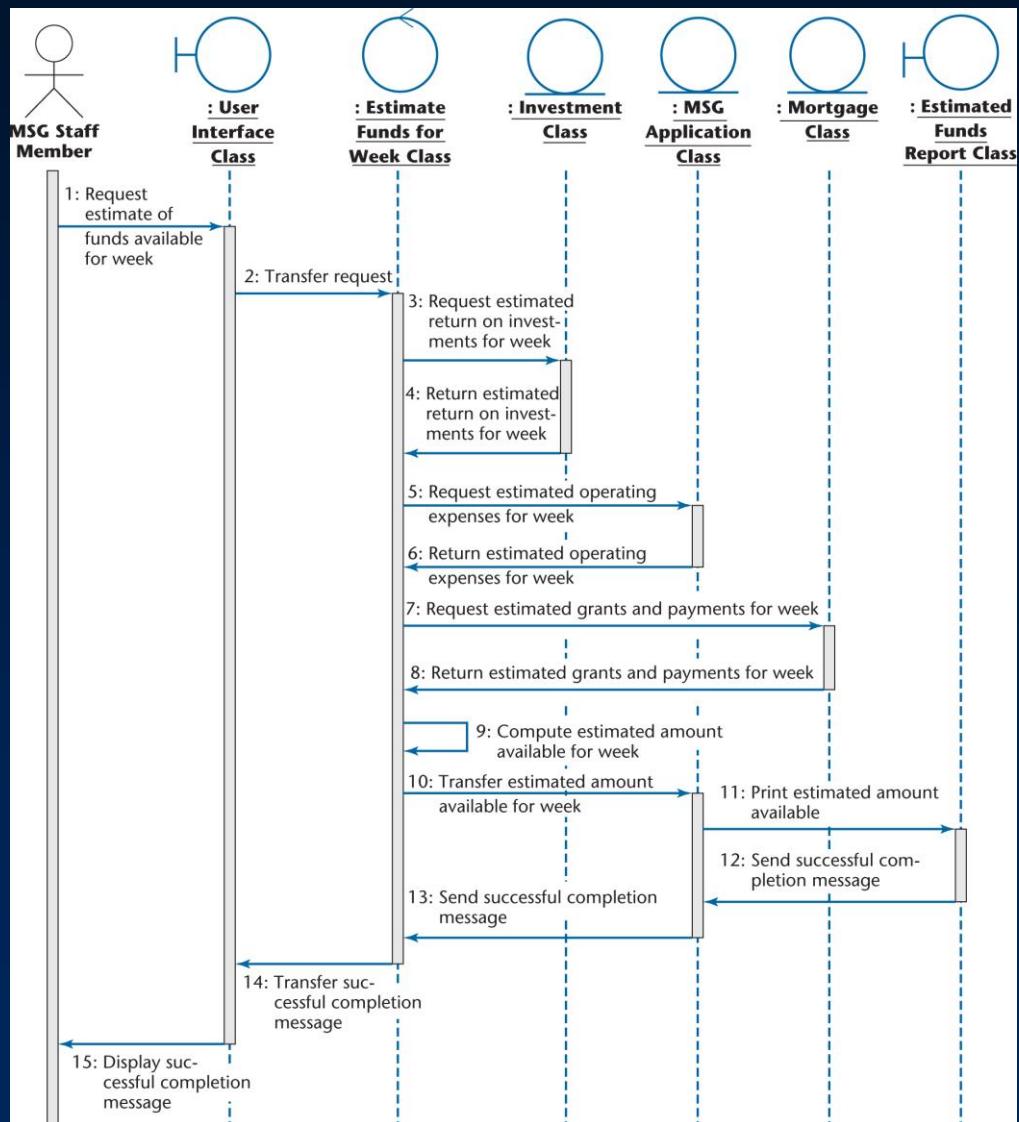


Figure 13.38

Interaction Diagrams

Slide 13.108

- The strength of a sequence diagram is that it shows the flow of messages and their order unambiguously
 - When transfer of information is the focus of attention, a sequence diagram is superior to a collaboration diagram
- A collaboration diagram is similar to a class diagram
 - When the developers are concentrating on the classes, a collaboration diagram is more useful than the equivalent sequence diagram

- Figures 13.32 through 13.38 do not depict a random collection of UML artifacts
- Instead, these figures depict a use case and artifacts derived from that use case
- In more detail (see next slide):

- Figure 13.32 depicts the use case Estimate Funds Available for Week
- The figure models
 - All possible sets of interactions
 - Between the actor **MSG Staff Member** (external to the software product) and the MSG Foundation software product itself
 - That relate to the action of estimating funds available for the week

- Figure 13.33 is the description of that use case
- The figure provides a written account of the details of the Estimate Funds Available for Week use case of Figure 13.32

- Figure 13.34 is a class diagram showing the classes that realize the Estimate Funds Available for Week use case
- The figure depicts
 - The classes that are needed to model all possible scenarios of the use case
 - Together with their interactions

- Figure 13.35 is a scenario
- It depicts one specific instance of the use case of Figure 13.32

- Figure 13.36 is a collaboration diagram of the realization of the scenario of Figure 13.35
- The figure depicts the objects and the messages sent between them in the realization of that one specific scenario

- Figure 13.37 is the flow of events of the collaboration diagram of the realization of the scenario of Figure 13.35
- Figure 13.37 is a written description of the realization of the scenario of Figure 13.35
 - (Compare: Figure 13.33 is a written description of the Estimate Funds Available for Week use case of Figure 13.32)

- Figure 13.38 is the sequence diagram that is fully equivalent to the collaboration diagram of Figure 13.36
- The sequence diagram depicts the objects and the messages sent between them in the realization of the scenario of Figure 13.35
- Its flow of events is therefore also shown in Figure 13.37

13.15.2 Manage an Asset Use Case

Slide 13.117

- Use case

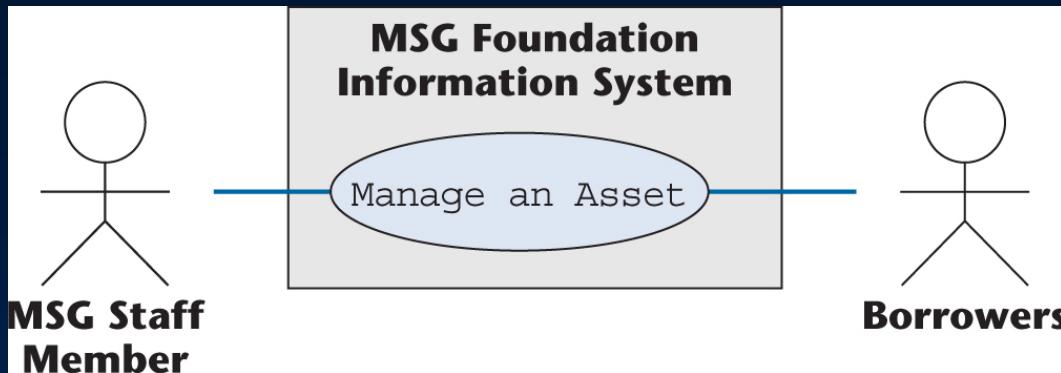


Figure 13.39

- Description of use case

Brief Description

The Manage an Asset use case enables an MSG Foundation staff member to add and delete assets and manage the portfolio of assets (investments and mortgages). Managing a mortgage includes updating the weekly income of a couple who have borrowed money from the Foundation.

Step-by-Step Description

1. Add, modify, or delete an investment or mortgage, or update the borrower's weekly income.

Figure 13.40

Manage an Asset Use Case (contd)

Slide 13.119

- Class diagram showing the classes that realize the use case

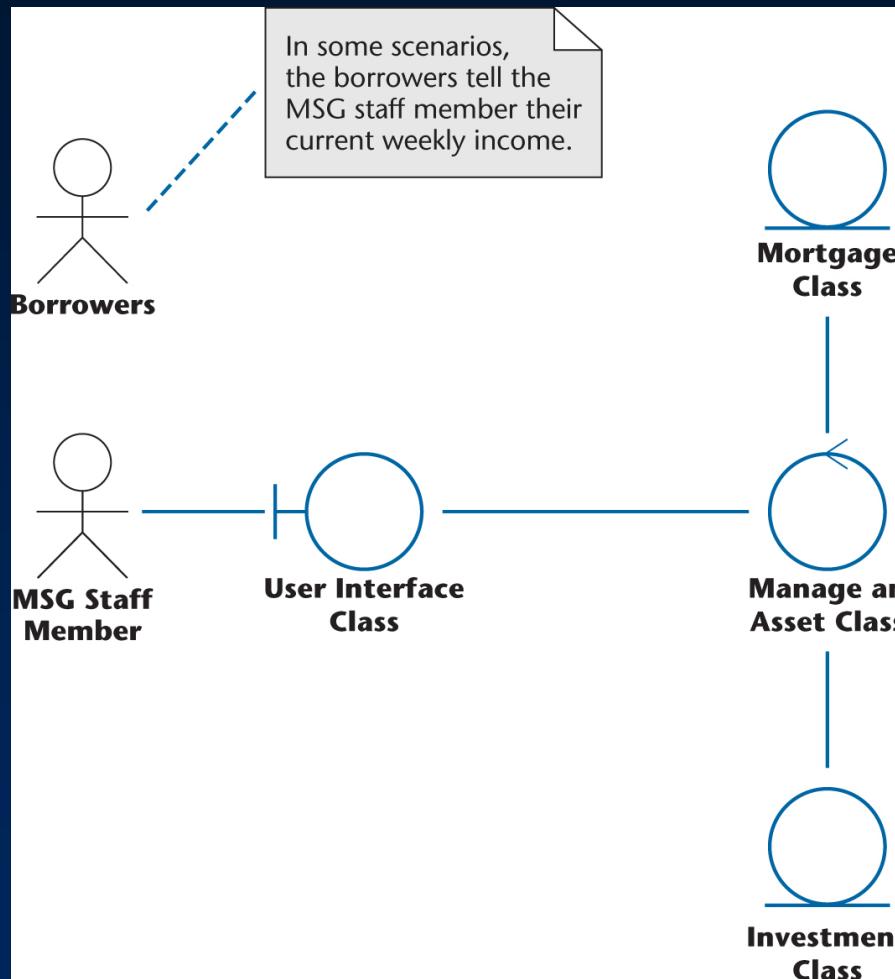


Figure 13.41

- One scenario of the use case

An MSG Foundation staff member wants to update the annual real-estate tax on a home for which the Foundation has provided a mortgage.

1. The staff member enters the new value of the annual real-estate tax.
2. The information system updates the date on which the annual real-estate tax was last changed.

Figure 13.42

Manage an Asset Use Case (contd)

Slide 13.121

- Collaboration diagram of the realization of the scenario of the use case

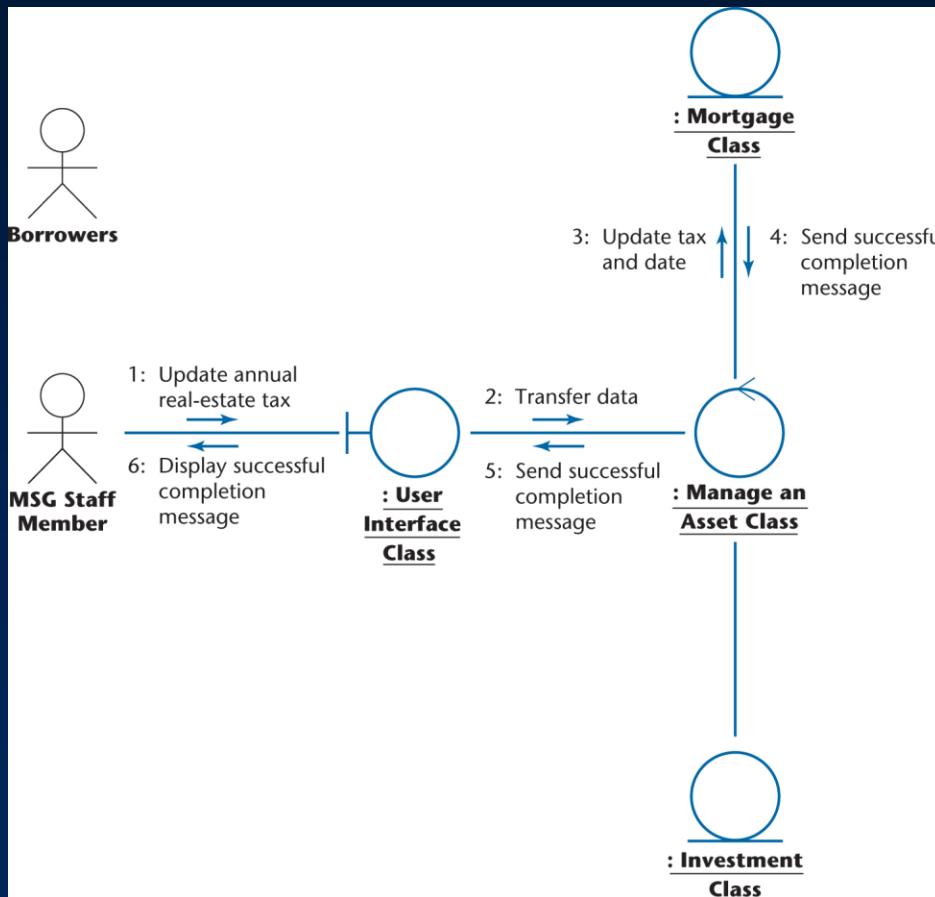


Figure 13.43

- Object : **Investment Class** does not play an active role in this collaboration diagram
 - This scenario does not involve an investment, only a mortgage
- Actor **Borrowers** does not play a role in this use case, either

Manage an Asset Use Case (contd)

Slide 13.123

- Sequence diagram equivalent to the collaboration diagram (of the realization of the scenario of the use case)

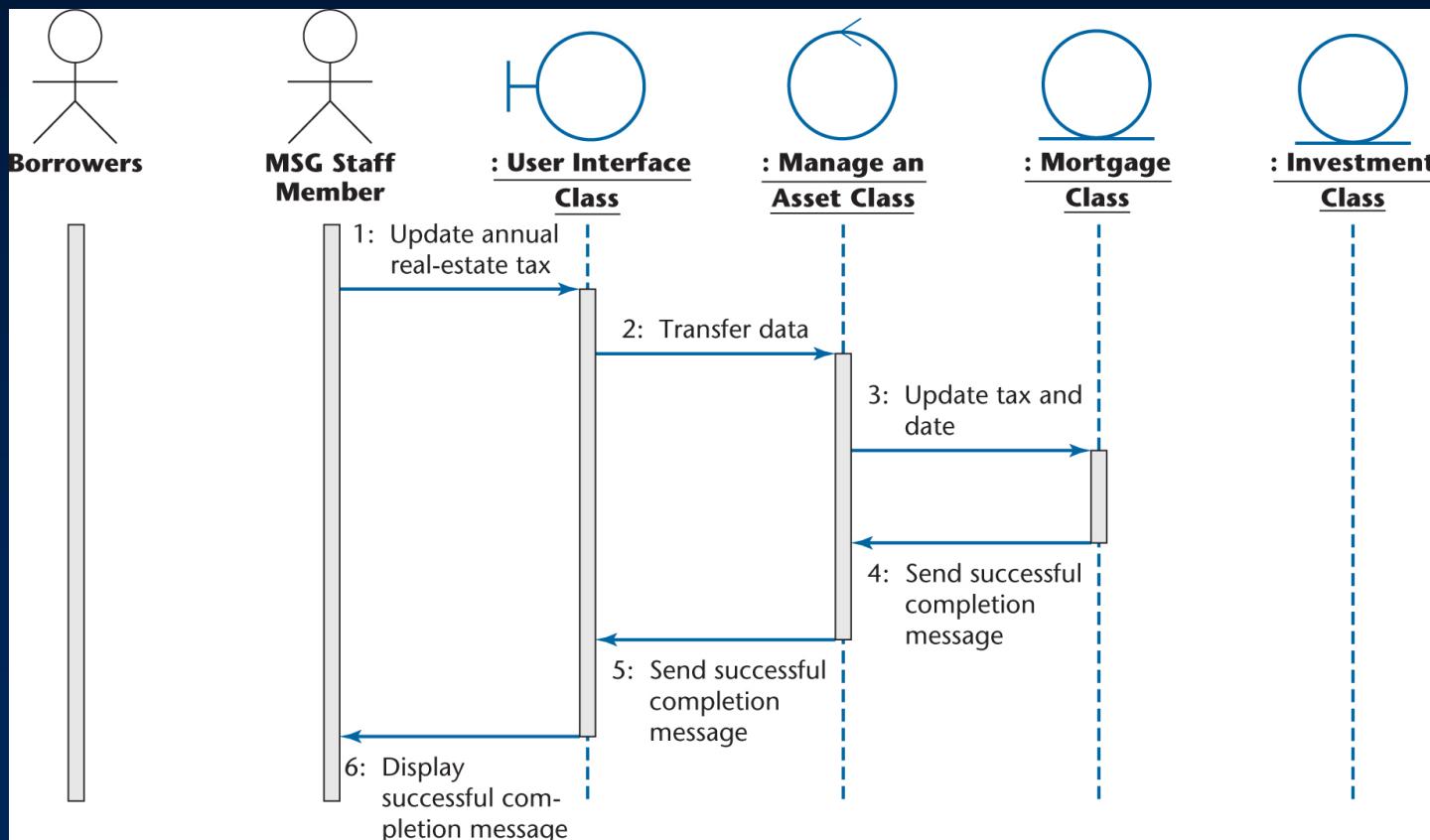


Figure 13.44

- A different scenario of the use case

There is a change in the weekly income of a couple who have borrowed money from the MSG Foundation. They wish to have their weekly income updated in the Foundation records by an MSG staff member so that their mortgage payments will be correctly computed.

1. The staff member enters the new value of the weekly income.
2. The information system updates the date on which the weekly income was last changed.

Figure 13.45

Manage an Asset Use Case (contd)

Slide 13.125

- Collaboration diagram of the realization of the scenario of the use case

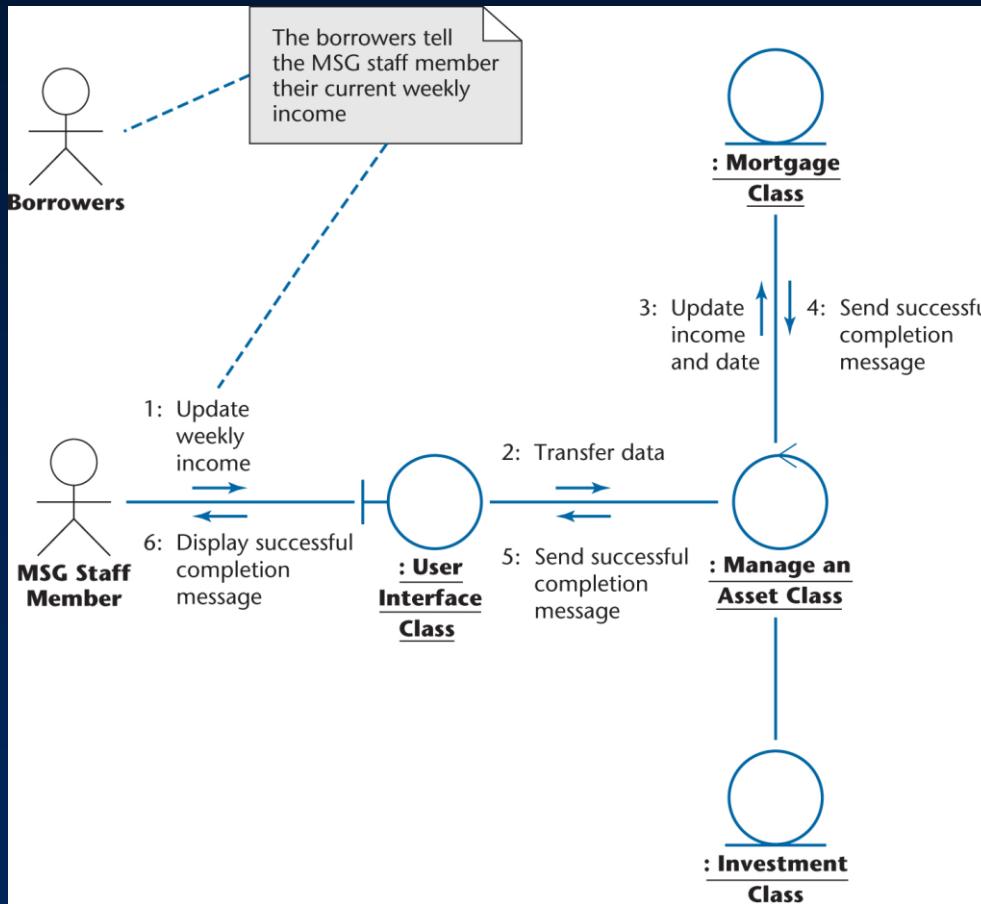


Figure 13.46

- At the request of the borrowers, the MSG staff member updates the weekly income of a couple
- The scenario is initiated by the **Borrowers**
- Their data are entered into the software product by the **MSG Staff Member**
 - This is stated in the note in the collaboration diagram

Manage an Asset Use Case (contd)

Slide 13.127

- Sequence diagram equivalent to the collaboration diagram (of the realization of the scenario of the use case)

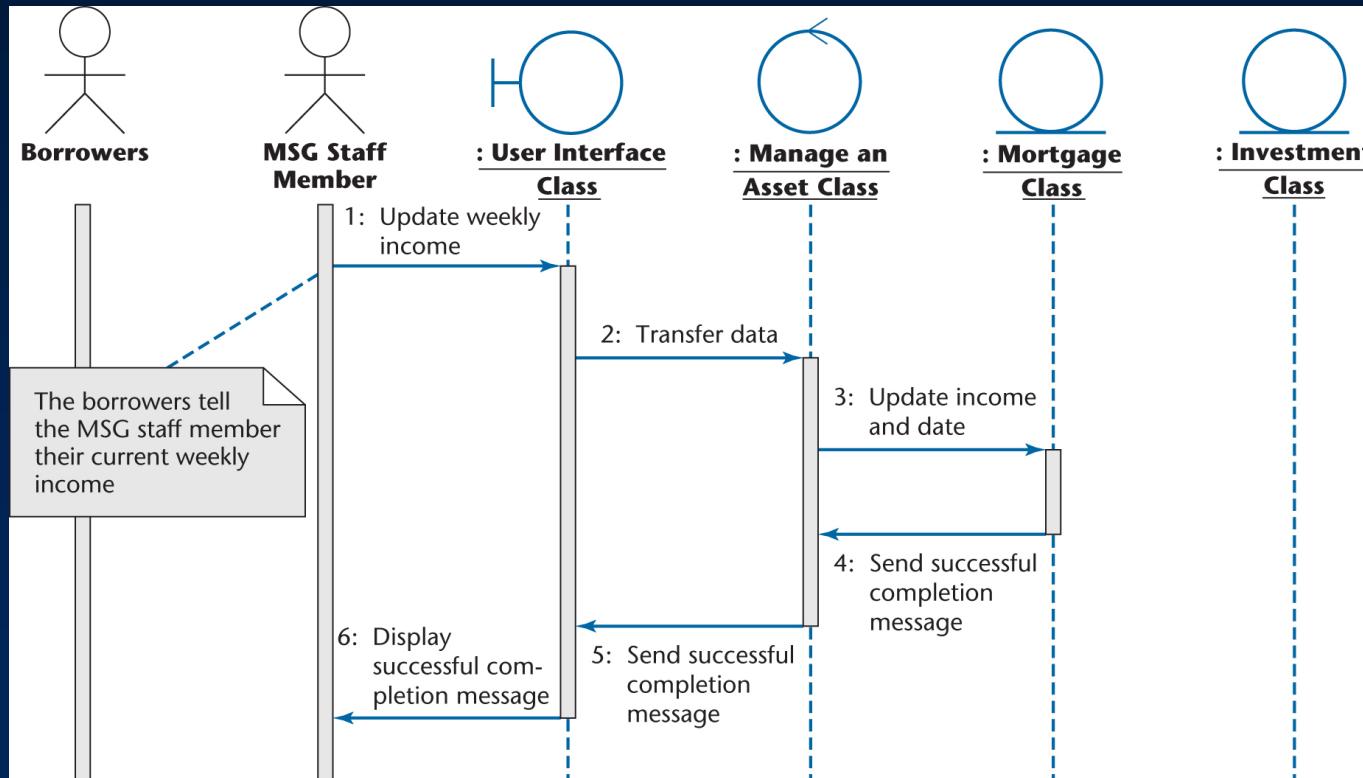


Figure 13.47

- Two different scenarios of the same use case have been presented
- The use case is the same
 - The class diagram is therefore the same
- However, the collaboration (and sequence) diagrams reflect the differences between the two scenarios

- Boundary class **User Interface Class** appears in all the realizations
 - The same screen will be used for all commands of the information system

- Revised menu

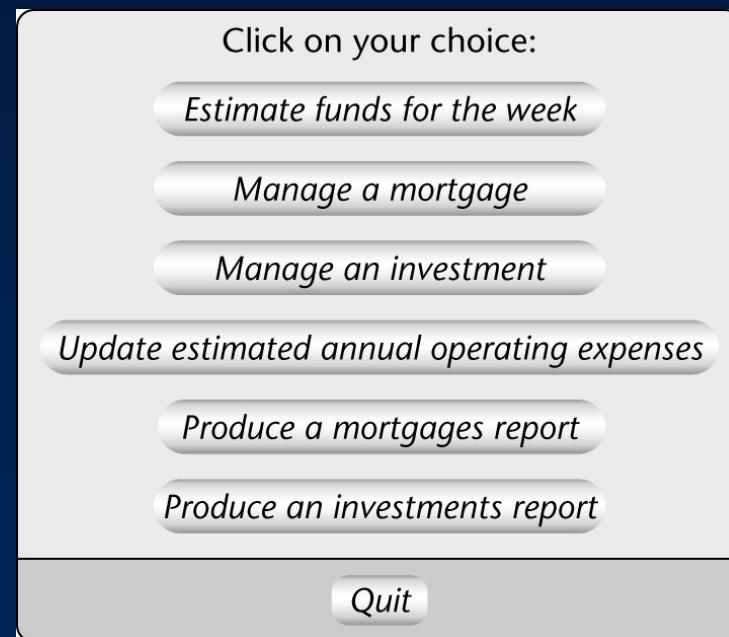


Figure 13.48

- Corresponding textual interface

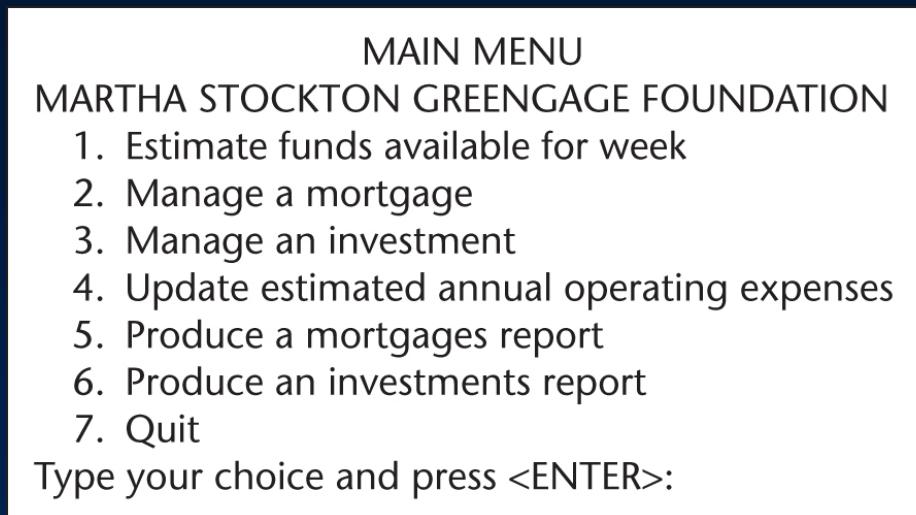


Figure 13.49

13.15.3 Update Annual Operating Expenses Use Case

Slide 13.131

- Class diagram

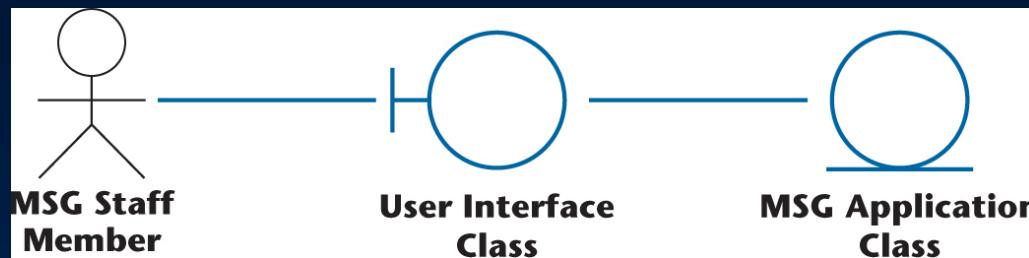


Figure 13.50

Update Annual Operating Expenses Use Case (contd)

Slide 13.132

- Collaboration diagram of a realization of a scenario of the use case

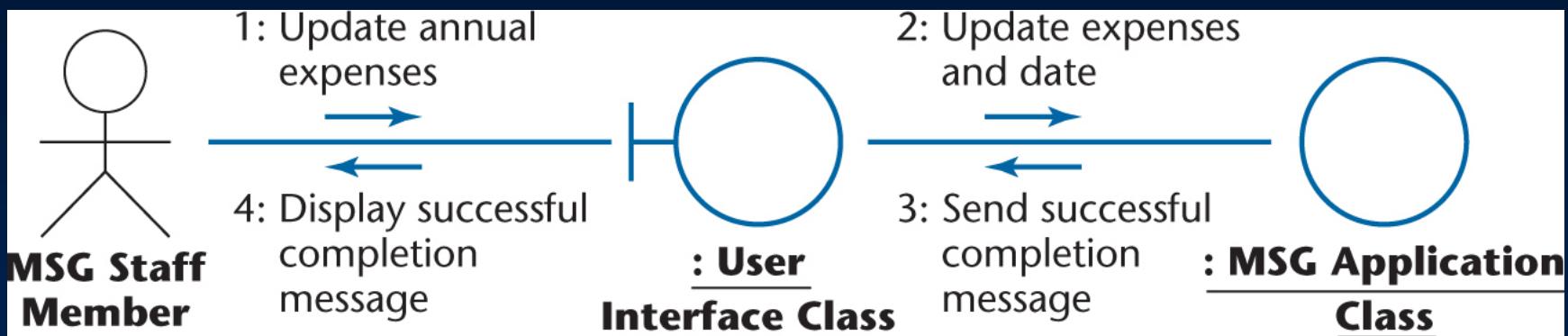


Figure 13.51

Update Annual Operating Expenses Use Case (contd)

Slide 13.133

- Equivalent sequence diagram

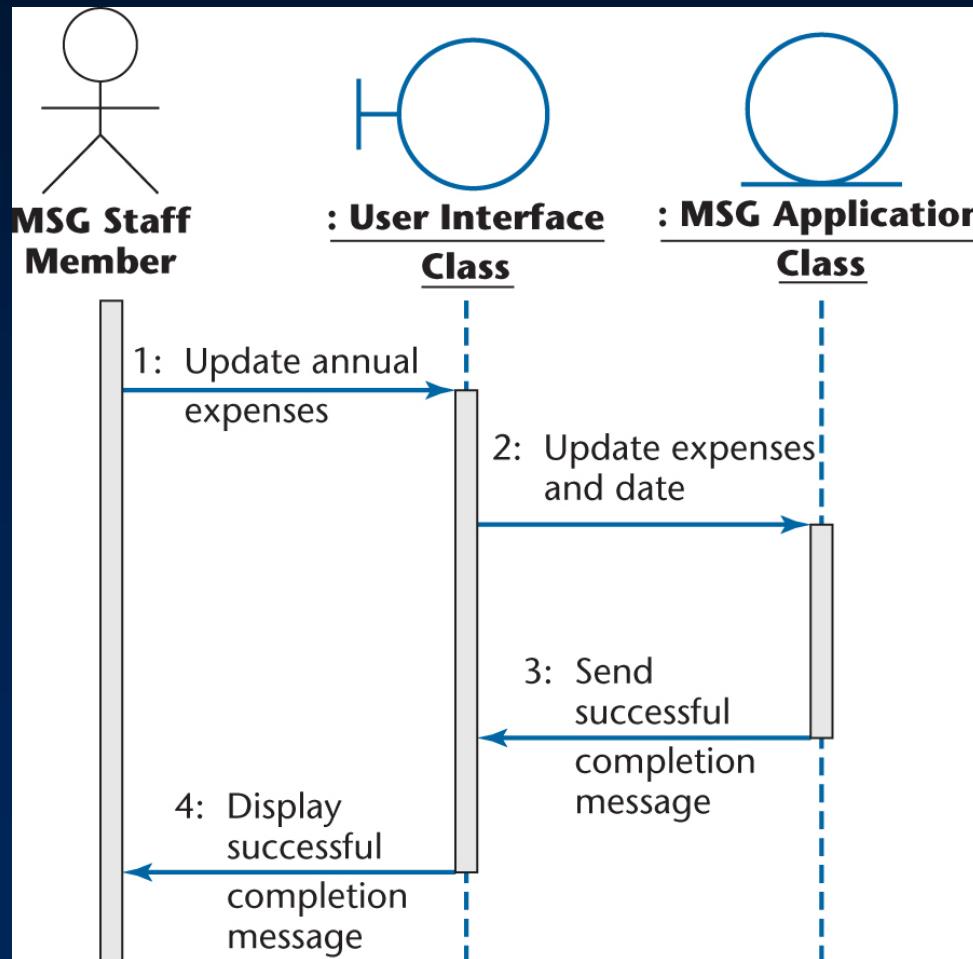


Figure 13.52

13.15.4 Produce a Report Use Case

Slide 13.134

- Use case

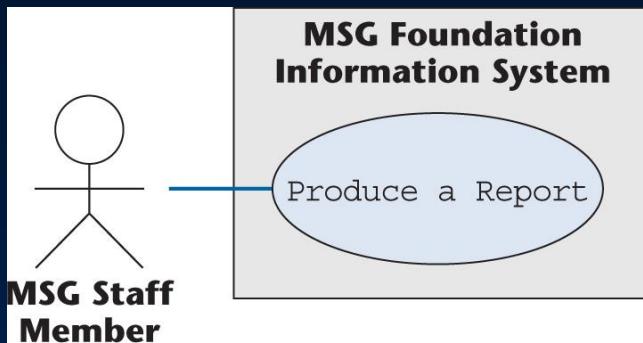


Figure 13.53

- Description of use case

Brief Description

The Produce a Report use case enables an MSG Foundation staff member to print a listing of all investments or all mortgages.

Step-by-Step Description

1. The following reports must be generated:

- 1.1 Investments report—printed on demand:

The information system prints a list of all investments. For each investment, the following attributes are printed:

Item number

Item name

Estimated annual return

Date estimated annual return was last updated

- 1.2 Mortgages report—printed on demand:

The information system prints a list of all mortgages. For each mortgage, the following attributes are printed:

Account number

Name of mortgagees

Original price of home

Date mortgage was issued

Principal and interest payment

Current combined gross weekly income

Date current combined gross weekly income was last updated

Annual real-estate tax

Date annual real-estate tax was last updated

Annual homeowner's insurance premium

Date annual homeowner's insurance premium was last updated

Figure 13.54

Produce a Report Use Case (contd)

Slide 13.136

- Class diagram

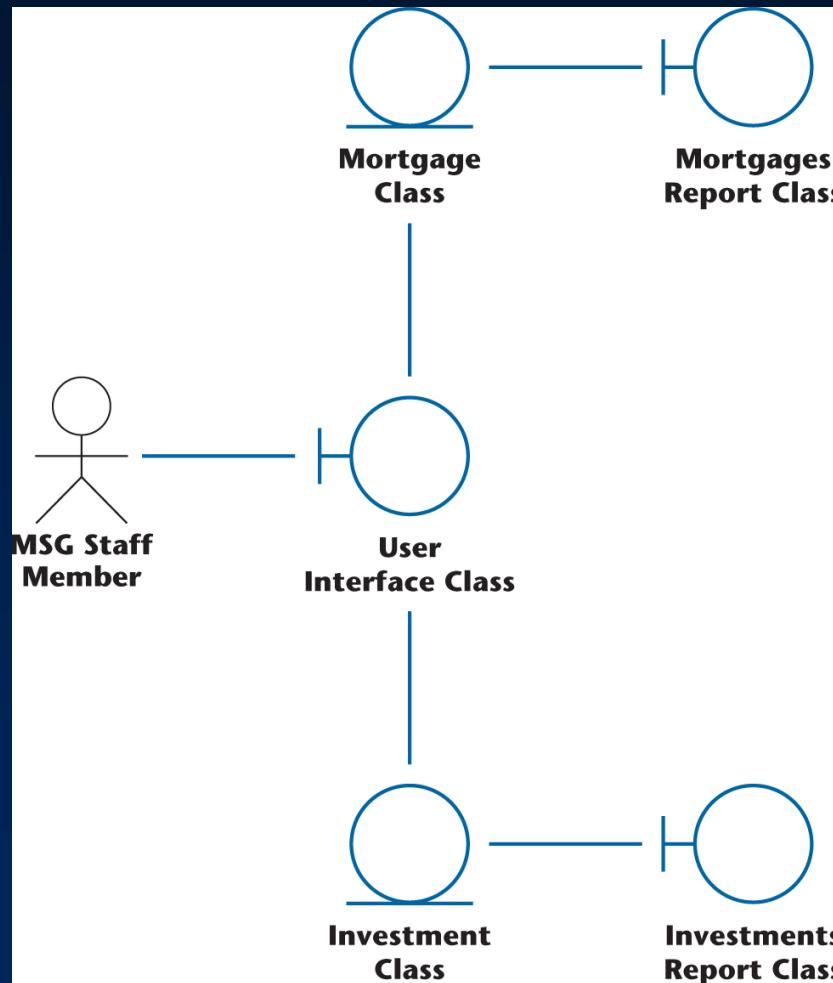


Figure 13.55

- One scenario of the use case

An MSG staff member wishes to print a list of all mortgages.

1. The staff member requests a report listing all mortgages.

Figure 13.56

Produce a Report Use Case (contd)

Slide 13.138

- Collaboration diagram
 - Mortgages (but not investments) are involved

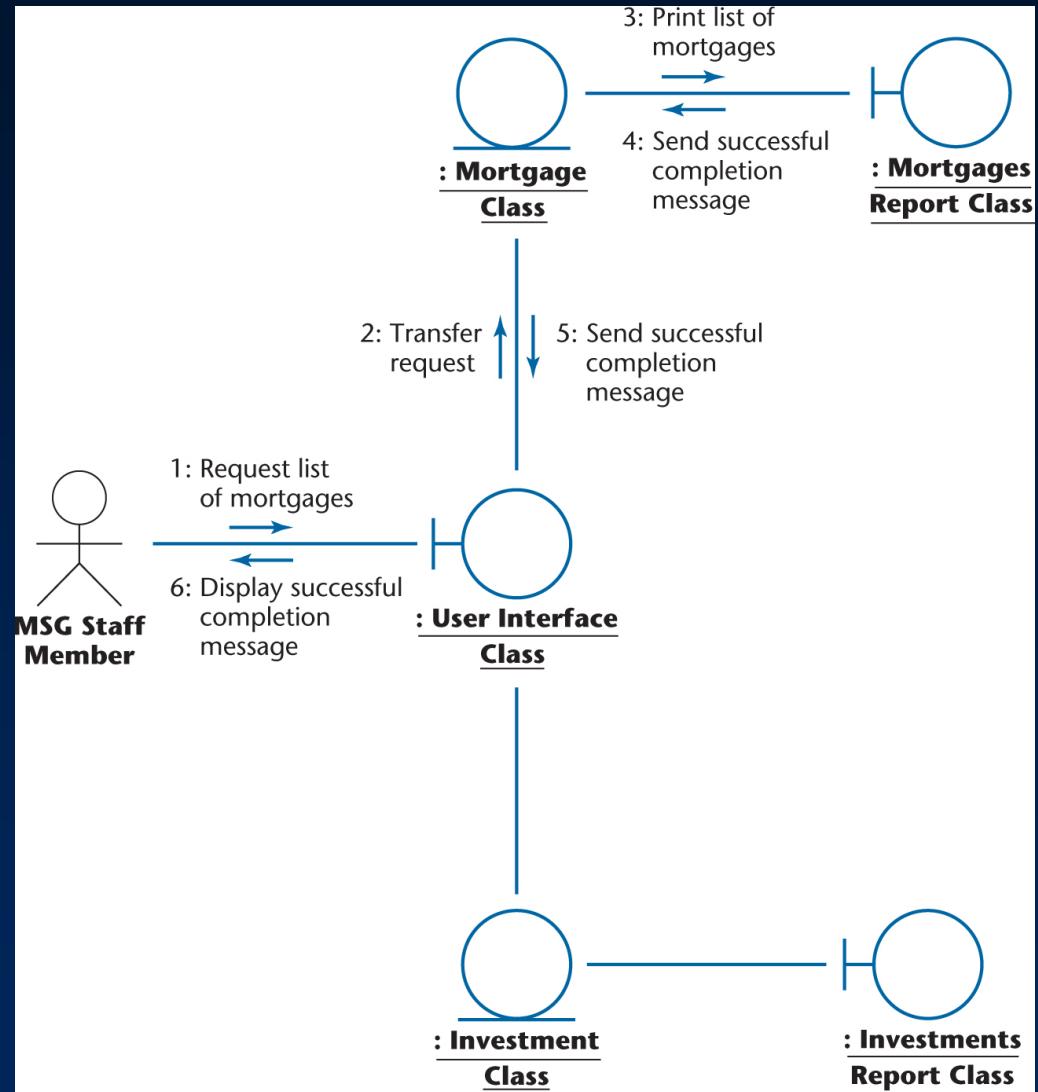


Figure 13.57

Produce a Report Use Case (contd)

Slide 13.139

- Sequence diagram

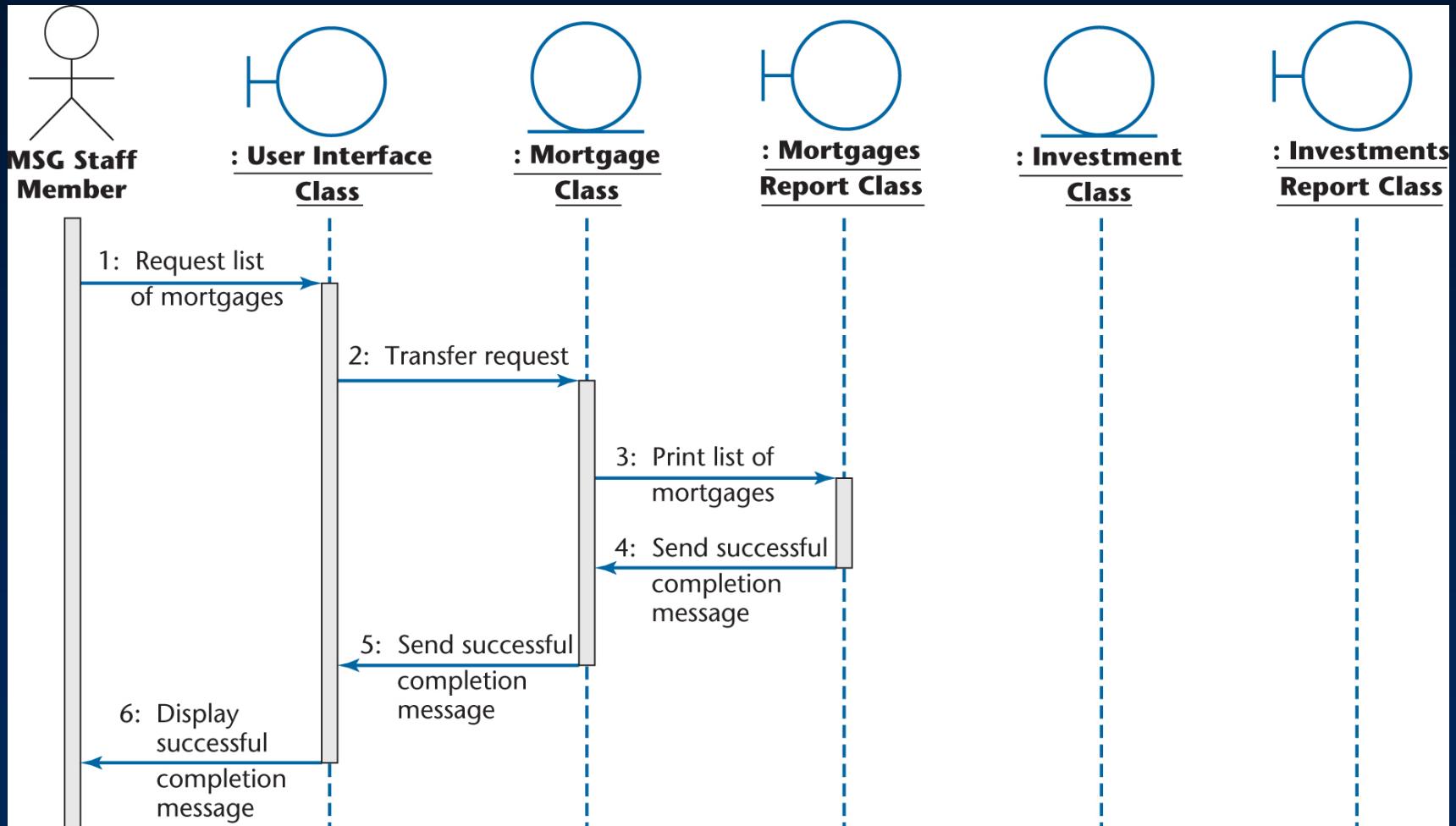


Figure 13.58

- A second scenario (listing all investments) of the use case

An MSG staff member wishes to print a list of all investments.

1. The staff member requests a report listing all investments.

Figure 13.59

Produce a Report Use Case (contd)

Slide 13.141

- Collaboration diagram for second scenario
 - This time, investments (but not mortgages) are involved

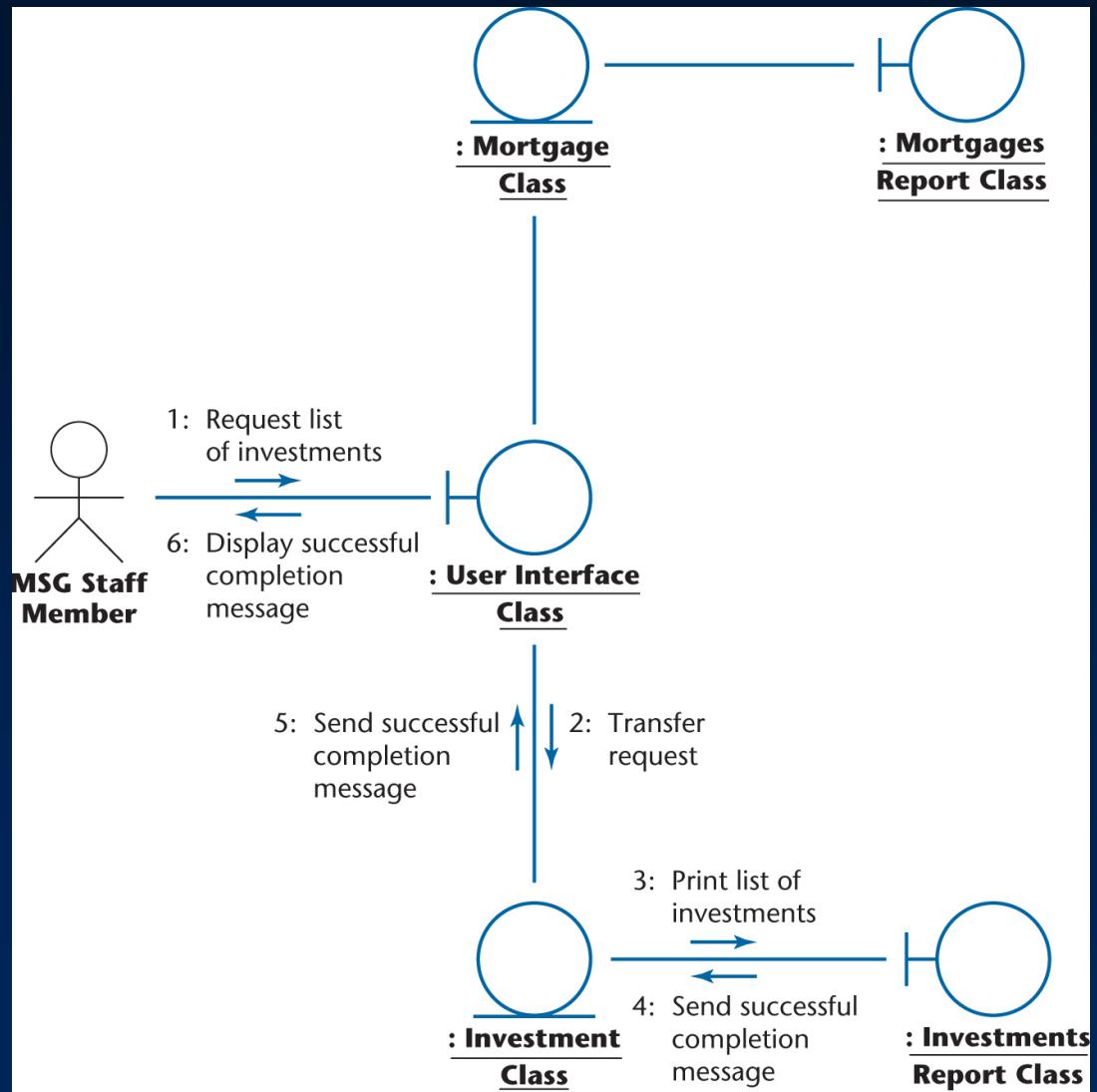


Figure 13.60

Produce a Report Use Case (contd)

Slide 13.142

- Sequence diagram for second scenario

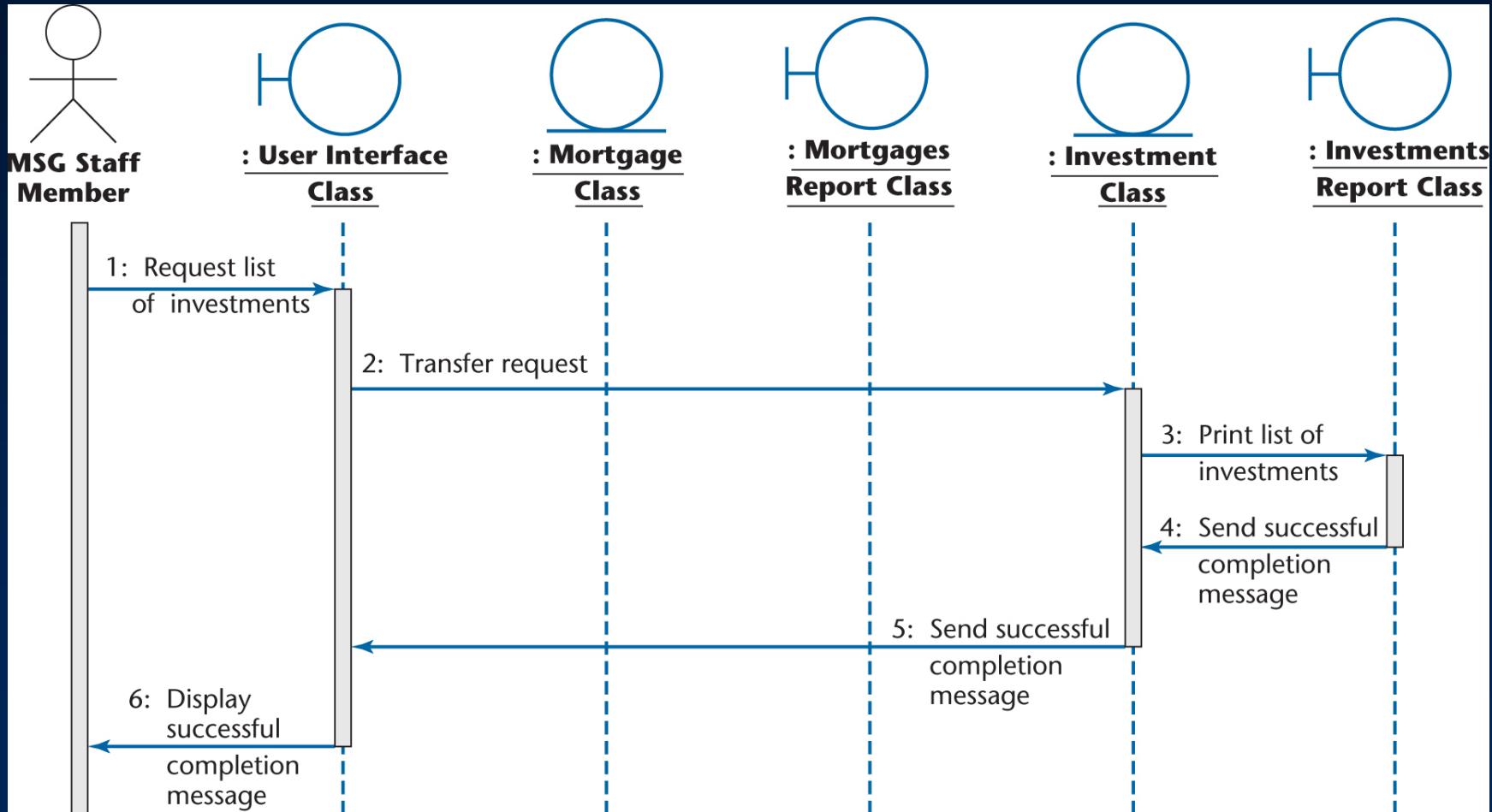


Figure 13.61

13.16 Incrementing the Class Diagram: The MSG Foundation

Slide 13.143

- In the course of realizing the various use cases
 - Interrelationships between classes become apparent
- Accordingly, we now combine the realization class diagrams

Combining the Realization Class Diagrams

Slide 13.144

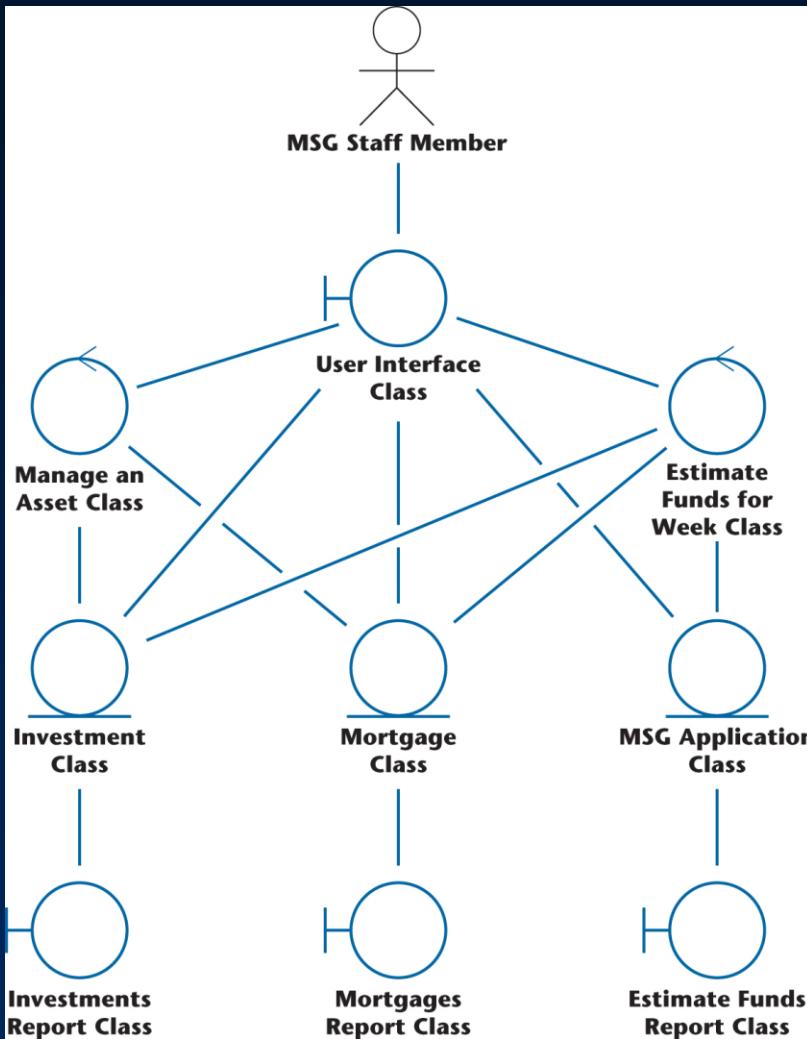


Figure 13.62

Fourth Iteration of the Class Diagram

Slide 13.145

- Fifth iteration + realization class diagram

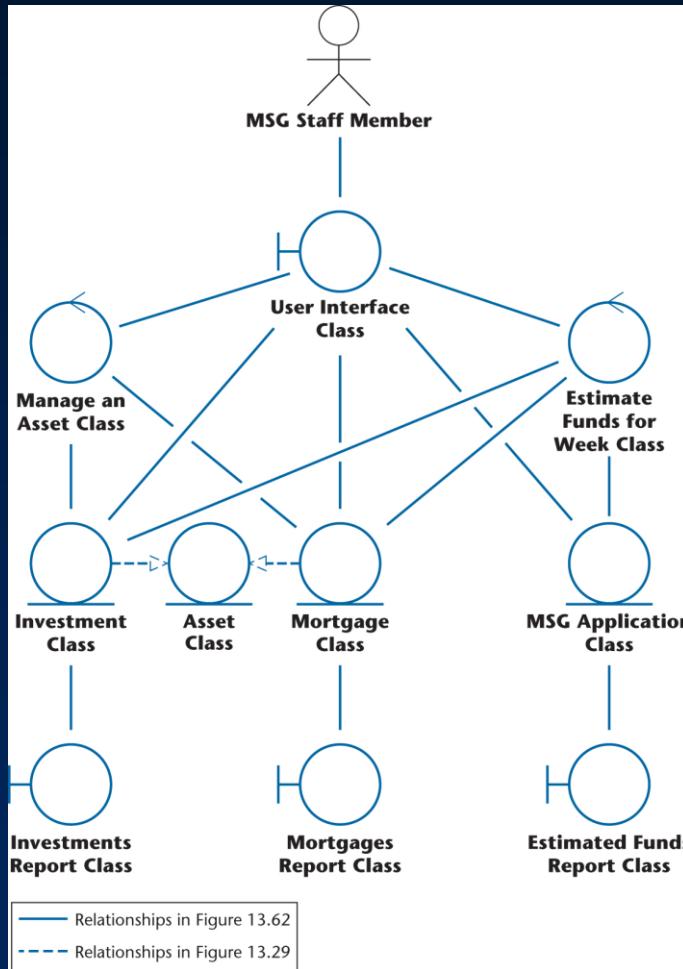


Figure 13.63

Software Project Management Plan

Slide 13.146

- As with the classical paradigm, the SPMP is drawn up at this point
 - It appears in Appendix F
 - The plan conforms to the IEEE SPMP format

13.17 The Test Workflow: MSG Foundation

Slide 13.147

- CRC cards are used to check the entity classes
- All the artifacts are then inspected

- The Unified Process is use-case driven
 - The use cases and the artifacts derived from them replace the traditional textual specification document
- The client must be shown each use case and associated artifacts, both diagrammatic and textual
 - These UML diagrams convey to the client more information more accurately than the traditional specification document
 - The set of UML diagrams can also play the same contractual role as the traditional specification document

The Specification Document (contd)

Slide 13.149

- A scenario is a specific execution sequence
- The client can therefore appreciate how the product works equally well from
 - A use case together with its scenarios, or
 - A rapid prototype
- The difference is
 - The use cases are successively refined, with more information added each time, whereas
 - The rapid prototype is discarded

The Specification Document (contd)

Slide 13.150

- However, a rapid prototype of the user interface is required
 - Specimen screens and reports are needed (not a complete rapid prototype)

13.19 More on Actors and Use Cases

Slide 13.151

- To find the actors, consider every role in which an individual can interact with the software product
 - Example: **Applicants, Borrowers**
- Actors are not individuals
 - They are roles played by those individuals
- Find all the different roles played by each user
 - From the list of roles, extract the actors

- In the Unified Process
 - The term *worker* is used to denote a role played by an individual
 - In the Unified Process, **Applicants** and **Borrowers** are two different workers
- In common parlance
 - The word “worker” usually refers to an employee
- In this book, the word “role” is used in place of “worker”

More on Actors and Use Cases (contd)

Slide 13.153

- Within a business context, finding the roles is easy
 - They are displayed within the use-case business model
- To find the actors
 - Find the subset of the use-case business model that corresponds to the use-case model of the requirements

More on Actors and Use Cases (contd)

Slide 13.154

- To find the actors (in more detail):
 - Construct the use-case business model
 - Consider only those parts of the business model that correspond to the proposed software product
 - The actors in this subset are the actors we seek

More on Actors and Use Cases (contd)

Slide 13.155

- Within a business context, finding use cases is easy
- For each role, there will be one or more use cases
 - Find the actors (see previous slide)
 - The use cases then follow

- Diagrams play a major role in object-oriented analysis
- Diagrams often change
 - We need a diagramming tool
 - Many tools go further
- All modern tools support UML
 - Commercial examples
 - » IBM Rational Rose
 - » Together
 - Open-source example
 - » ArgoUML

- Do not cross the boundary into object-oriented design
- Do not allocate methods to classes yet
 - Relocating methods to classes during stepwise refinement is wasted effort

- As with the other core workflows
 - It is essential to measure the five fundamental metrics: size, cost, duration, effort, and quality
 - It is essential to keep accurate fault statistics
- A measure of size of the object-oriented analysis
 - Number of pages of UML diagrams

Overview of the MSG Foundation Case Study

Slide 13.159

Initial functional model	Section 13.9
Seventh iteration of the use-case diagram	Figure 13.15
Initial class diagram	Section 13.10
First iteration of the class diagram	Figure 13.21
Second iteration of the class diagram	Figure 13.22
Eighth iteration of the use-case diagram	Figure 13.23
Second iteration of the class diagram, with attributes added	Figure 13.24
Initial dynamic model	Section 13.11
Initial statechart	Figure 13.25
Revising the entity classes	Section 13.12
Third iteration of the class diagram	Figure 13.28
Extracting the boundary classes	Section 13.13
Extracting the control classes	Section 13.14
Use-case realization	Section 13.15
Estimate Funds Available for Week use case	Section 13.15.1
Manage an Asset use case	Section 13.15.2
Update Estimated Annual Operating Expenses use case	Section 13.15.3
Produce a Report use case	Section 13.15.4
Incrementing the class diagram	Section 13.16
Fourth iteration of the class diagram	Figure 13.63
	Figure 13.64

Overview of the Elevator Problem Case Study

Slide 13.160

Object-oriented analysis	Section 13.3
Functional modeling	Section 13.4
Entity class modeling	Section 13.5
First iteration of the class diagram	Figure 13.5
Second iteration of the class diagram	Figure 13.6
Dynamic modeling	Section 13.6
First iteration of the statechart for the elevator controller	Figure 13.7
Test workflow	Section 13.7
Third iteration of the class diagram	Figure 13.10
Fourth iteration of the class diagram	Figure 13.12
First iteration of the statechart for the elevator subcontroller	Figure 13.13

Figure 13.65