Object Oriented Systems Analysis and Design Using UML

Simon Bennett, Steve McRobb and Ray Farmer

Requirements Analysis 2: Realizing Use Cases

Based on Chapter 7 of Bennett, McRobb and Farmer:

Object Oriented Systems Analysis and Design Using UML, (4th Edition), McGraw Hill, 2010.

In This Lecture You Will Learn:

- What is meant by use case realization
- Two approaches for realizing use cases:
  - Robustness analysis combined with communication diagrams
  - Class-Responsibility-Collaboration (CRC)
- How to combine use case class diagrams into a single analysis class model

From Requirements to Classes

- Requirements (use cases) are usually expressed in user language
- Use cases are units of development, but they are not structured like software
- The software we will implement consists of classes
- We need a way to translate requirements into classes

Goal of Realization

- An analysis class diagram is only an interim product
- This in turn will be realized as a design class diagram
- The ultimate product of realization is the software implementation of that use case

Communication Diagram Approach

- Analyse one use case at a time
- Identify likely classes involved (the use case collaboration)
  - These may come from a domain model
- Draw a communication diagram that fulfils the needs of the use case
- Translate this into a use case class diagram
- Repeat for other use cases
- Assemble the use case class diagrams into a single analysis class diagram

Robustness Analysis

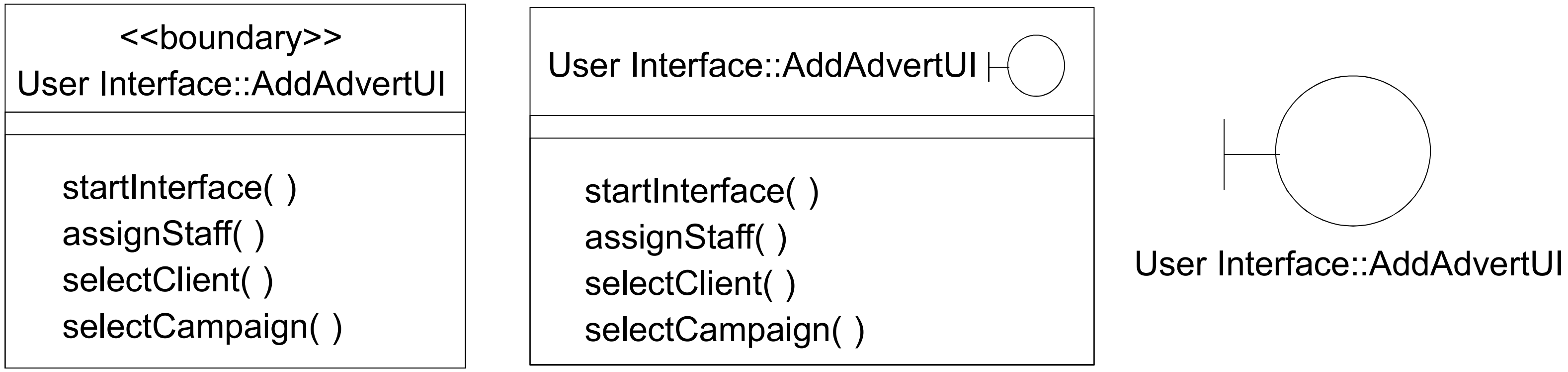
- Aims to produce set of classes robust enough to meet requirements of a use case
- Makes some assumptions about the interaction:
  - Assumes some class or classes are needed to handle the user interface
  - Abstracts logic of the use case away from *entity* classes (that store persistent data)

Robustness Analysis: Class Stereotypes

- Class stereotypes differentiate the roles objects can play:
  - Boundary objects model interaction between the system and actors (and other systems)
  - Control objects co-ordinate and control other objects
  - Entity objects represent information and behaviour in the application domain
  - Entity classes may be imported from domain model
  - Boundary and control classes are more likely to be unique to one application

Boundary Class Stereotype

- Boundary classes represent interaction with the user - likely to be unique to the use case but inherited from a library
- Alternative notations:



<<boundary>>
User Interface::AddAdvertUI

startInterface( )
assignStaff( )
selectClient( )
selectCampaign( )

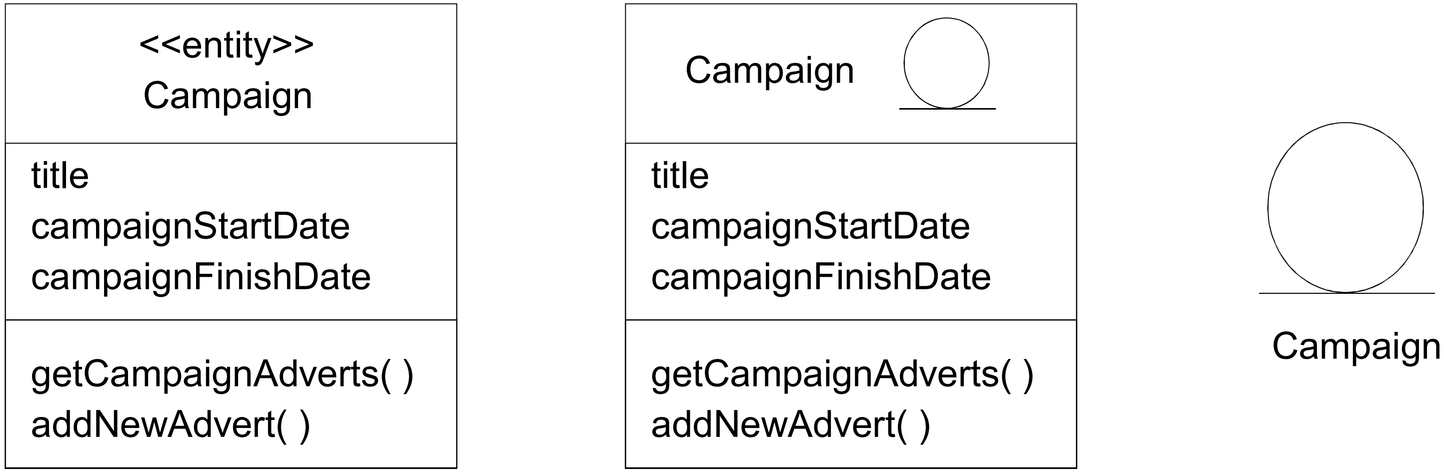
User Interface::AddAdvertUI

startInterface( )
assignStaff( )
selectClient( )
selectCampaign( )

User Interface::AddAdvertUI

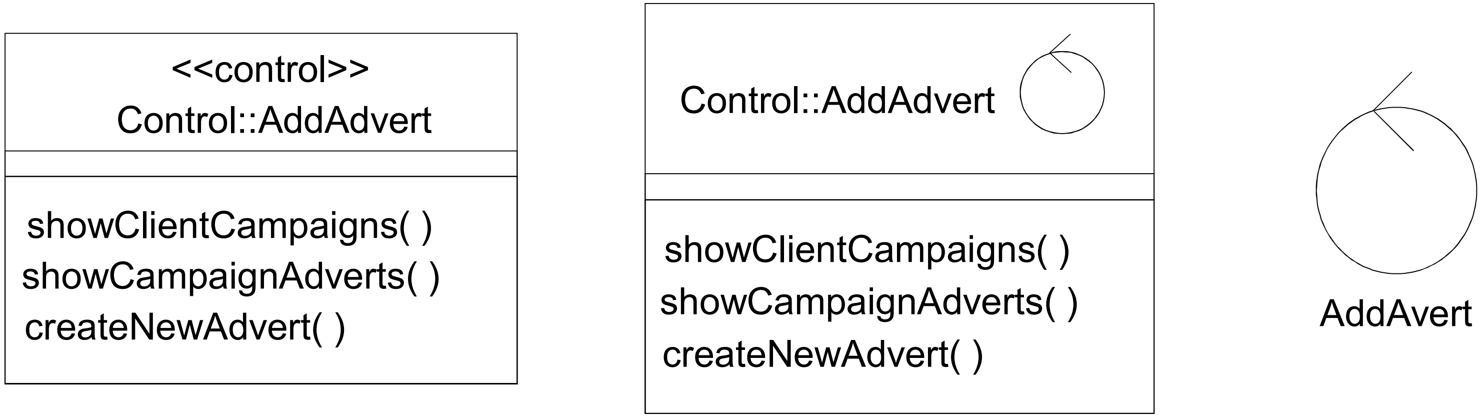
Entity Class Stereotype

- Entity classes represent persistent data and common behaviour likely to be used in more than one application system
- Alternative notations :



Control Class Stereotype

- Control classes encapsulate unique behaviour of a use case
- Specific logic kept separate from the common behaviour of entity classes
- Alternative notations:



<<control>>
Control::AddAdvert

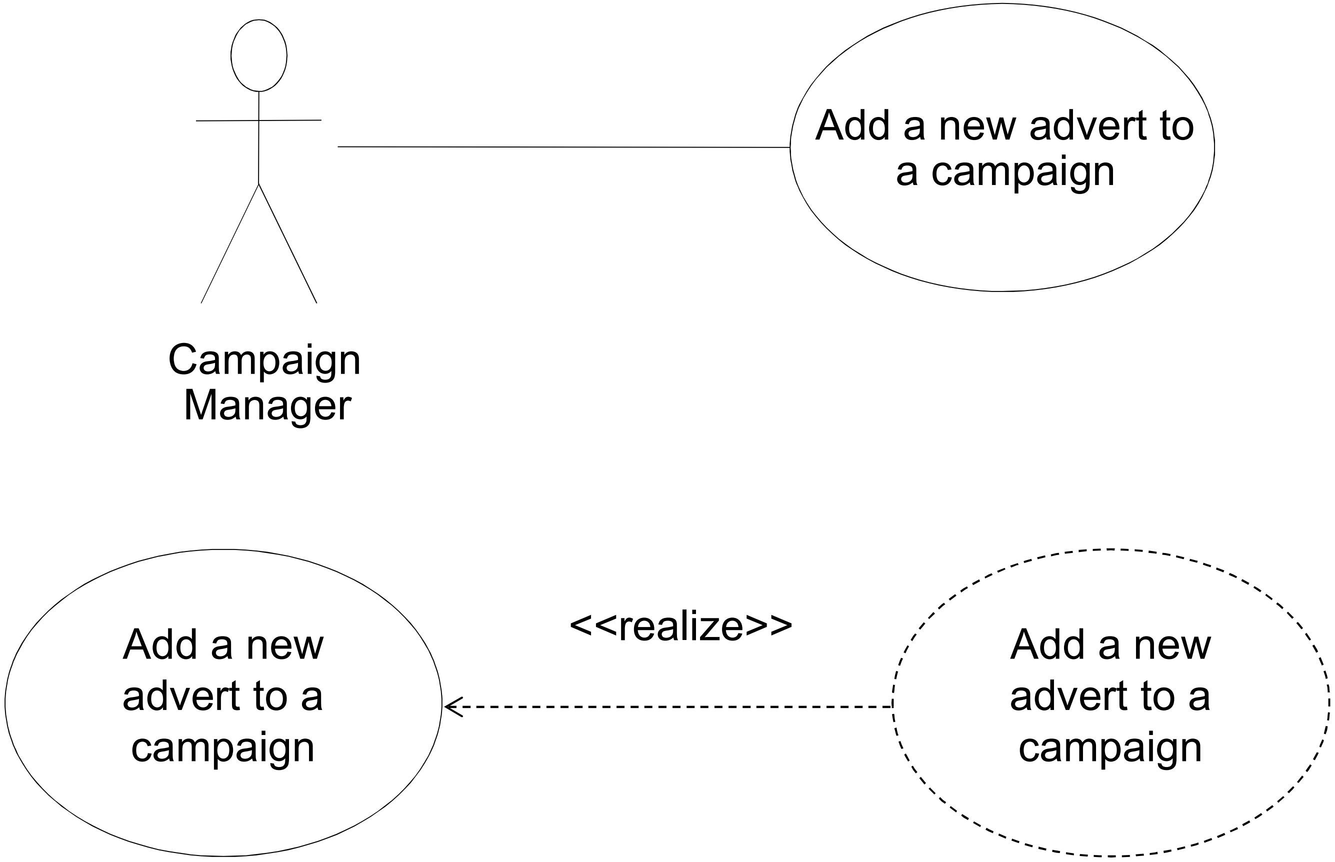
showClientCampaigns( )
showCampaignAdverts( )
createNewAdvert( )

Control::AddAdvert

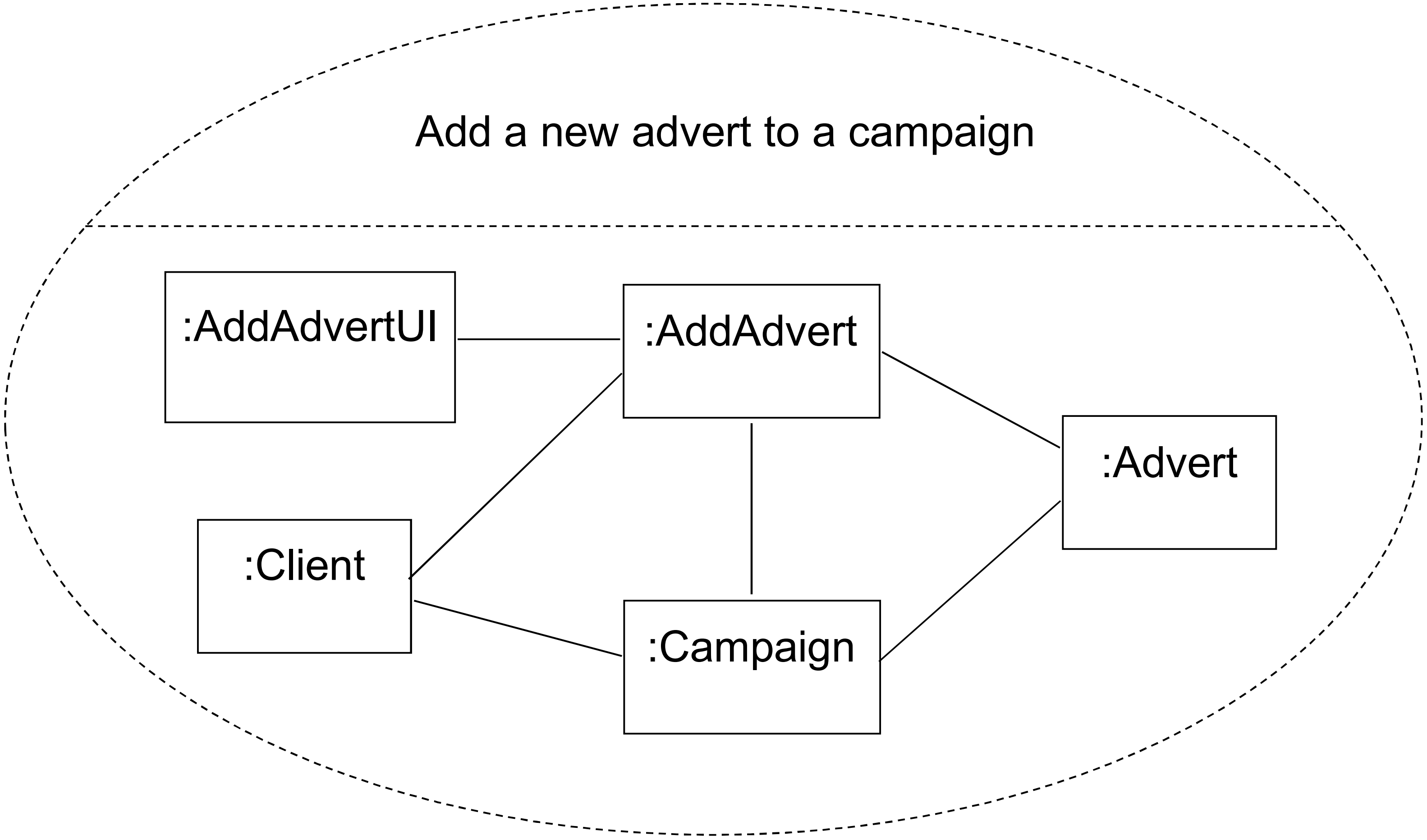
showClientCampaigns( )
showCampaignAdverts( )
createNewAdvert( )

AddAvert

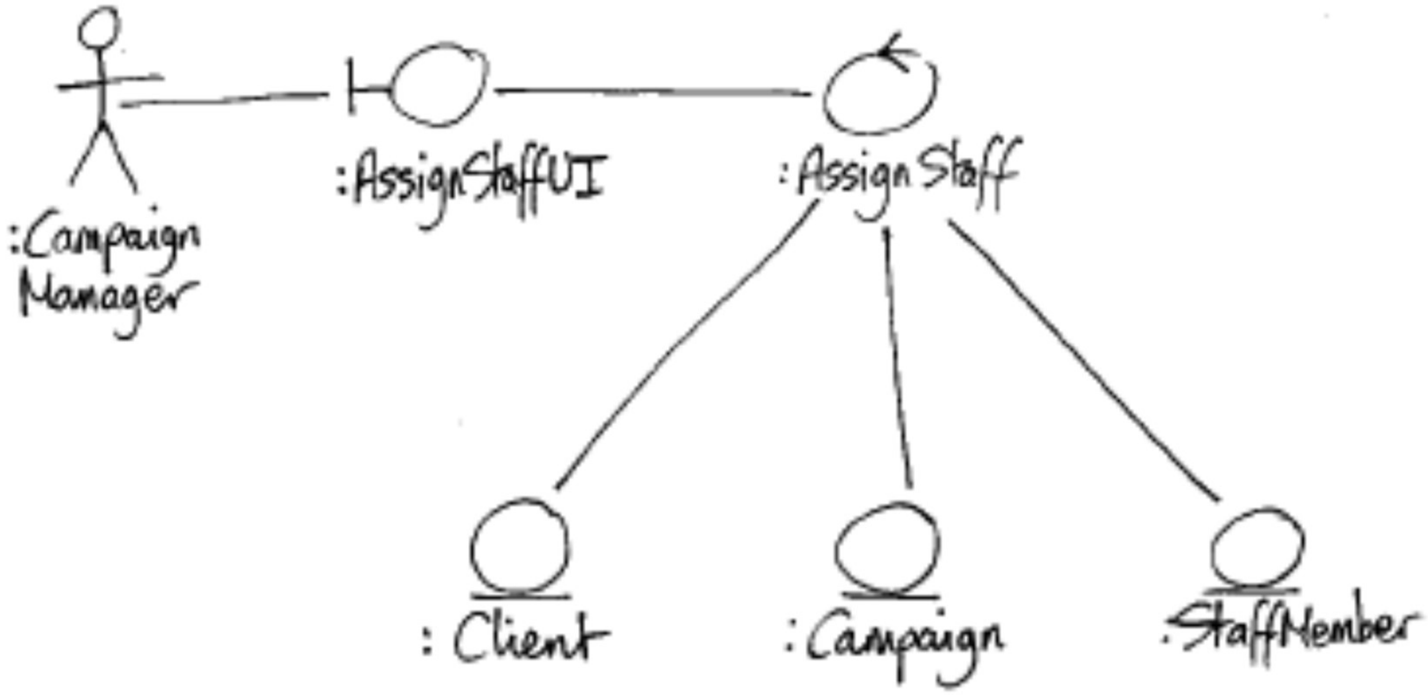
Use Case and Collaboration


Add a new advert to a campaign
Campaign Manager
Add a new advert to a campaign
<<realize>>
Add a new advert to a campaign

A Possible Collaboration

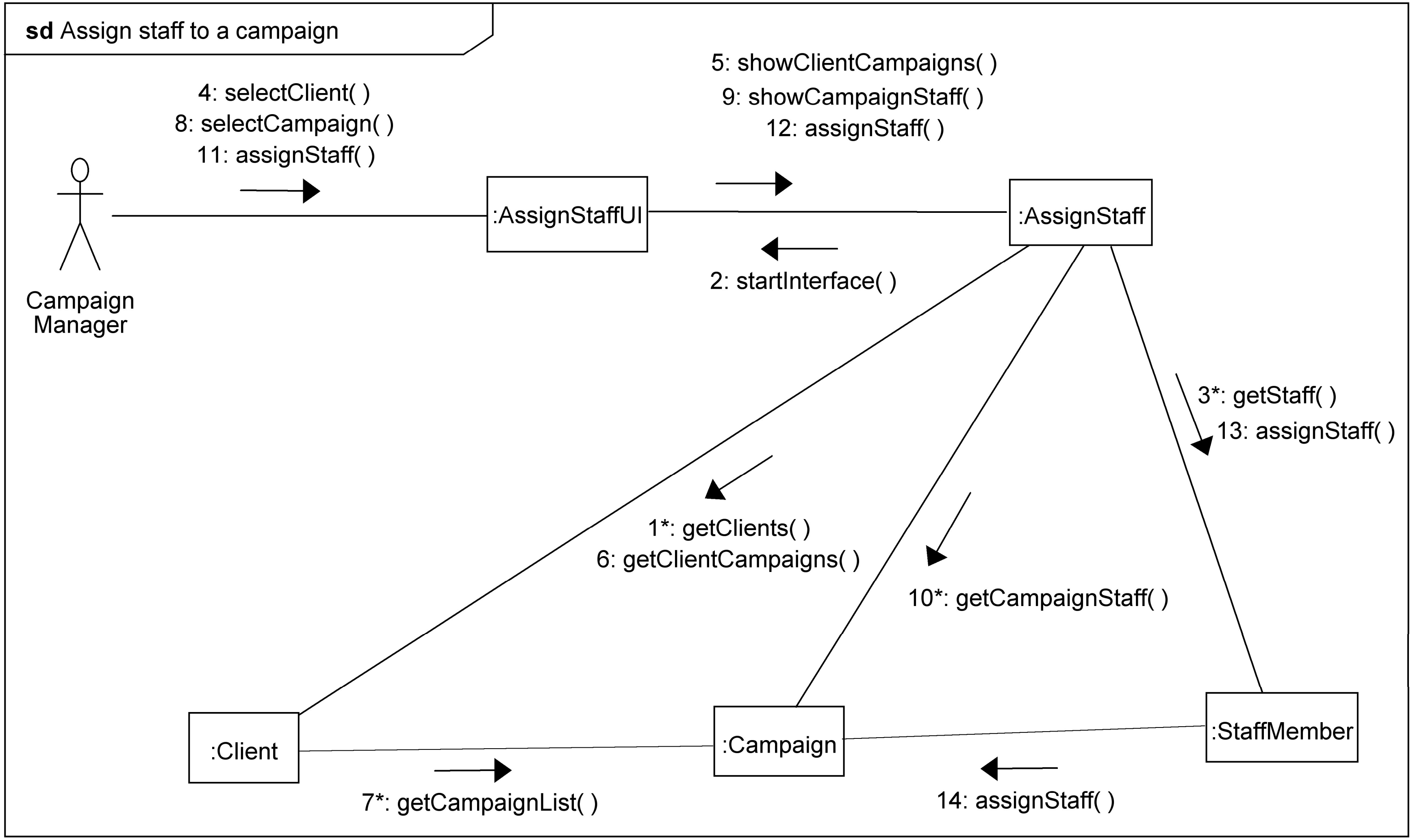


Early Draft Communication Diagram

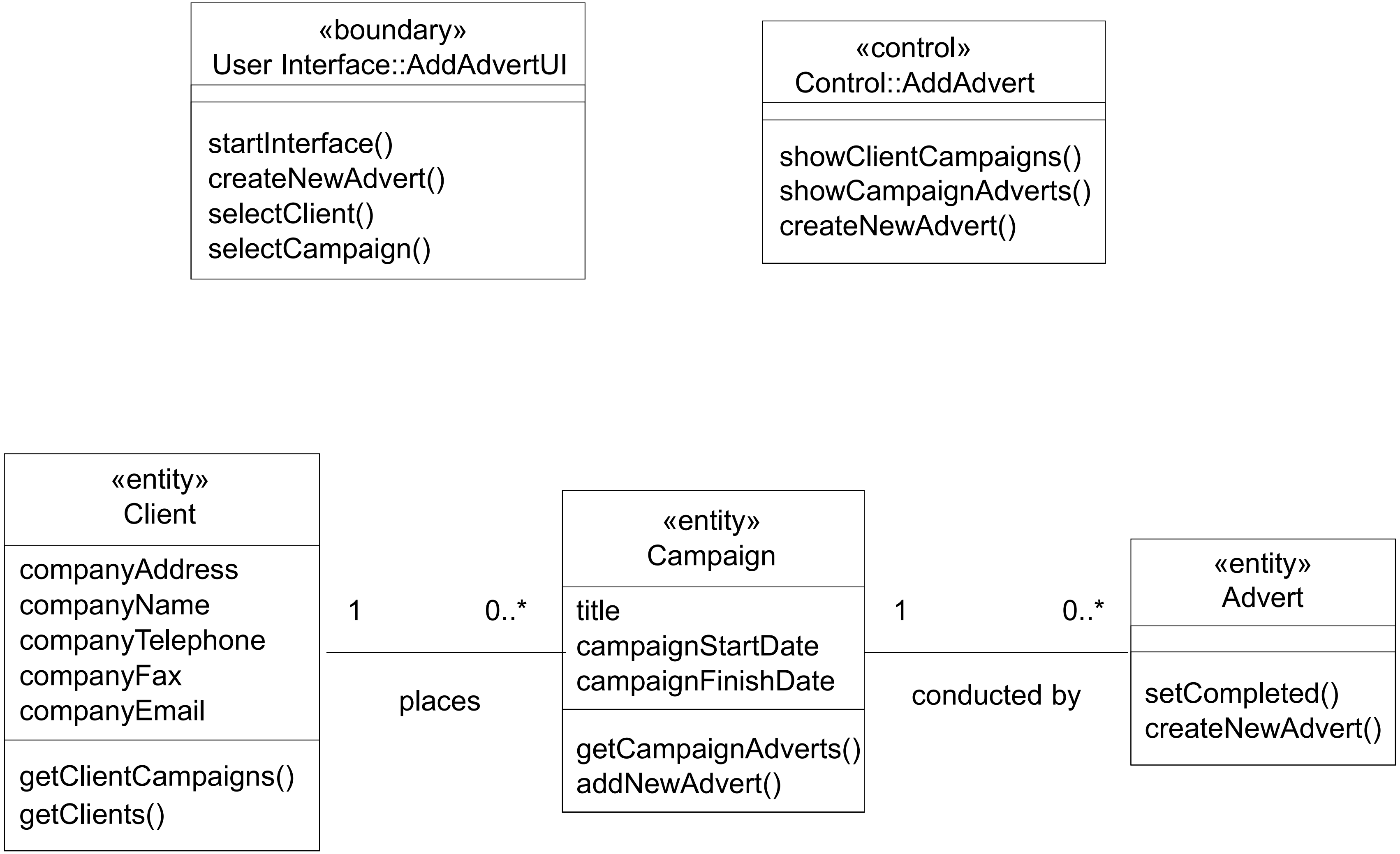


© 2010 Bennett, McRobb and Farmer

More Developed Communication Diagram



Resulting Class Diagram



Reasonability Checks for Candidate Classes

- A number of tests help to check whether a candidate class is reasonable
  - – Is it beyond the scope of the system?
  - – Does it refer to the system as a whole?
  - – Does it duplicate another class?
  - – Is it too vague?

  (More on next slide)

Reasonability Checks for Candidate Classes (cont'd)

  - Is it too tied up with physical inputs and outputs?
  - Is it really an attribute?
  - Is it really an operation?
  - Is it really an association?
- If any answer is 'Yes', consider modelling the potential class in some other way (or do not model it at all)

CRC Cards

- Class–Responsibility–Collaboration cards help to model interaction between objects
- Used as a way of:
  - Identifying classes that participate in a scenario
  - Allocating responsibilities - both operations and attributes (*what can I do?* and *what do I know?*)
- For a given scenario (or use case):
  - Brainstorm the objects
  - Allocate to team members
  - Role play the interaction

CRC Cards

Class Name:	
Responsibilities	Collaborations
Responsibilities of a class are listed in this section.	*Collaborations with other classes are listed here, together with a brief description of the purpose of the collaboration.*

Class Name	Client
Responsibilities	Collaborations
Provide client information.	
Provide list of campaigns.	Campaign provides campaign details.

Class Name	Campaign
Responsibilities	Collaborations
Provide campaign information.	
Provide list of adverts.	Advert provides advert details.
Add a new advert.	Advert constructs new object.

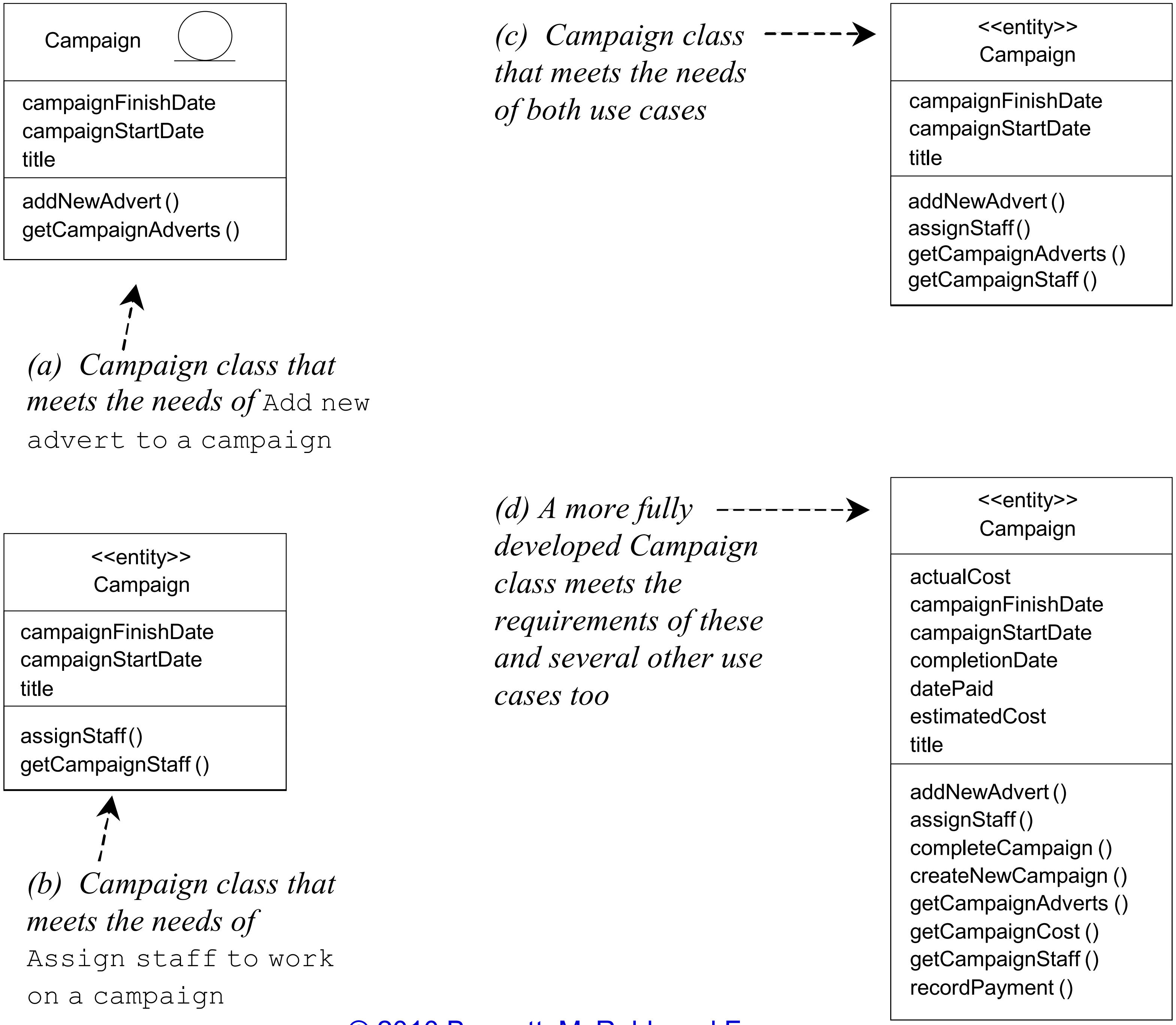
Class Name	Advert
Responsibilities	Collaborations
Provide advert details.	
Construct adverts.	

CRC Cards

- Effective role play depends on an explicit strategy for distributing responsibility among classes
- For example:
  - Each role player tries to be lazy
  - Persuades other players *their* class should accept responsibility for a given task
- May use ‘Paper CASE’ to document the associations and links

Assembling the Class Diagram

- However individual use cases are analysed, the aim is to produce a single analysis class diagram
- This models the application as a whole
- The concept is simple:
  - A class in the analysis model needs *all* the details required for that class in each separate use case



Campaign

campaignFinishDate
campaignStartDate
title

addNewAdvert ()
getCampaignAdverts ()

(a) Campaign class that meets the needs of `Add new advert to a campaign`

<<entity>>
Campaign

campaignFinishDate
campaignStartDate
title

assignStaff ()
getCampaignStaff ()

(b) Campaign class that meets the needs of `Assign staff to work on a campaign`

(c) Campaign class that meets the needs of both use cases

<<entity>>
Campaign

campaignFinishDate
campaignStartDate
title

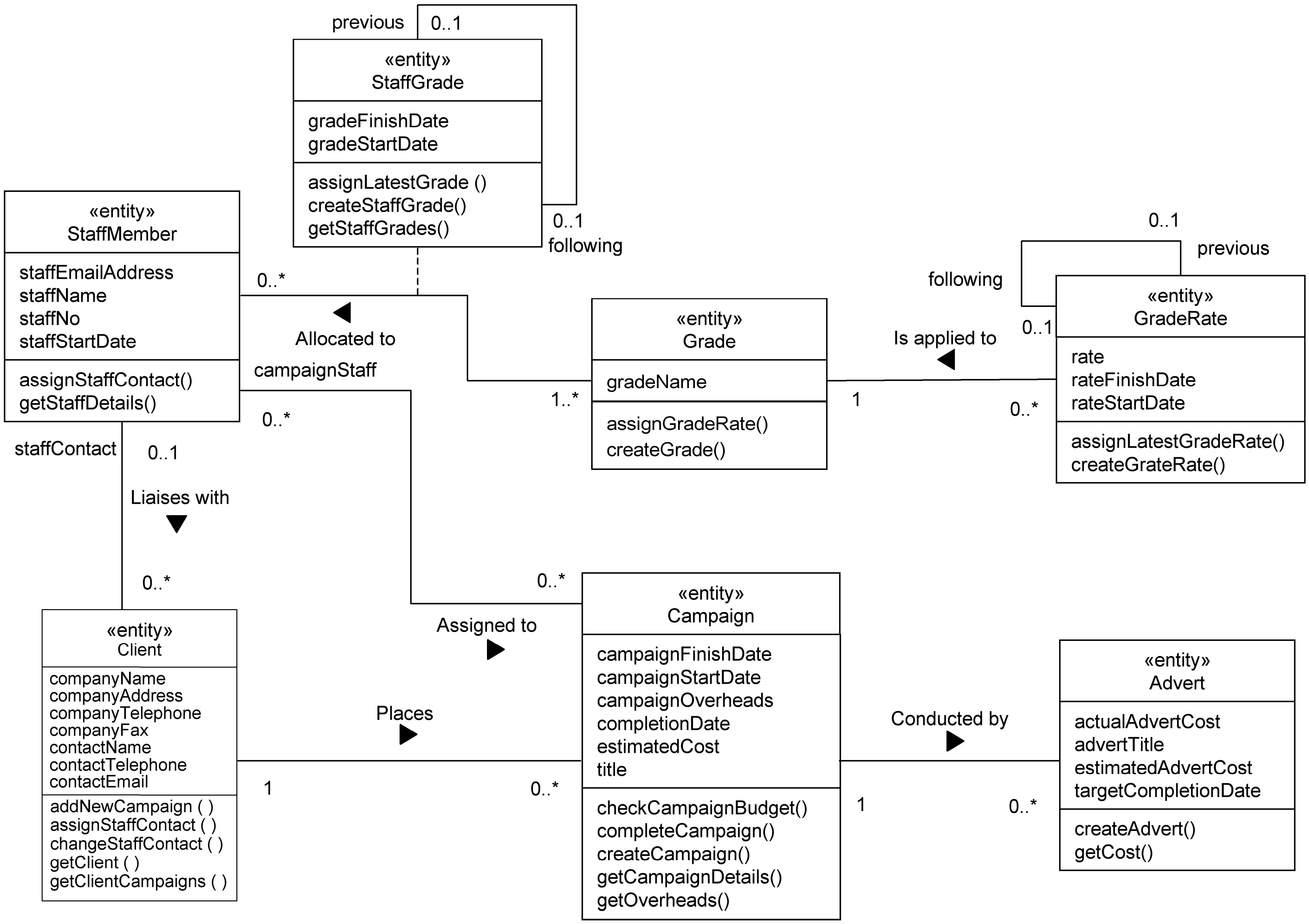
addNewAdvert ()
assignStaff ()
getCampaignAdverts ()
getCampaignStaff ()

(d) A more fully developed Campaign class meets the requirements of these and several other use cases too

<<entity>>
Campaign

actualCost
campaignFinishDate
campaignStartDate
completionDate
datePaid
estimatedCost
title

addNewAdvert ()
assignStaff ()
completeCampaign ()
createNewCampaign ()
getCampaignAdverts ()
getCampaignCost ()
getCampaignStaff ()
recordPayment ()


previous
0..1
«entity»
StaffGrade
gradeFinishDate
gradeStartDate
assignLatestGrade ()
createStaffGrade()
getStaffGrades()
0..1
following
«entity»
StaffMember
staffEmailAddress
staffName
staffNo
staffStartDate
assignStaffContact()
getStaffDetails()
0..*
Allocated to
campaignStaff
1..*
«entity»
Grade
gradeName
assignGradeRate()
createGrade()
0..1
previous
following
«entity»
GradeRate
rate
rateFinishDate
rateStartDate
assignLatestGradeRate()
createGrateRate()
Is applied to
0..1
1
0..*
0..*
staffContact
0..1
Liaises with
0..*
0..*
«entity»
Client
companyName
companyAddress
companyTelephone
companyFax
contactName
contactTelephone
contactEmail
addNewCampaign ( )
assignStaffContact ( )
changeStaffContact ( )
getClient ( )
getClientCampaigns ( )
Assigned to
Places
1
0..*
«entity»
Campaign
campaignFinishDate
campaignStartDate
campaignOverheads
completionDate
estimatedCost
title
checkCampaignBudget()
completeCampaign()
createCampaign()
getCampaignDetails()
getOverheads()
Conducted by
1
0..*
«entity»
Advert
actualAdvertCost
advertTitle
estimatedAdvertCost
targetCompletionDate
createAdvert()
getCost()

Summary

In this lecture you have learned:

- What is meant by *use case realization*
- How to realize use cases with robustness analysis and communication diagrams
- How the CRC technique helps identify classes and allocate responsibilities
- How to assemble the analysis class diagram

References

- Wirfs-Brock (1990) gives a good exposition of CRC cards

  (For full bibliographic details, see Bennett, McRobb and Farmer)