Lecture 15: Modelling System Interactions

- → **Interactions with the new system**
  - ↳ How will people interact with the system?
  - ↳ When/Why will they interact with the system?
- → **Use Cases**
  - ↳ introduction to use cases
  - ↳ identifying actors
  - ↳ identifying cases
  - ↳ Advanced features
- → **Sequence Diagrams**
  - ↳ Temporal ordering of events involved in a use case

© 2004-5 Steve Easterbrook. This presentation is available free for non-commercial use with attribution under a creative commons license.

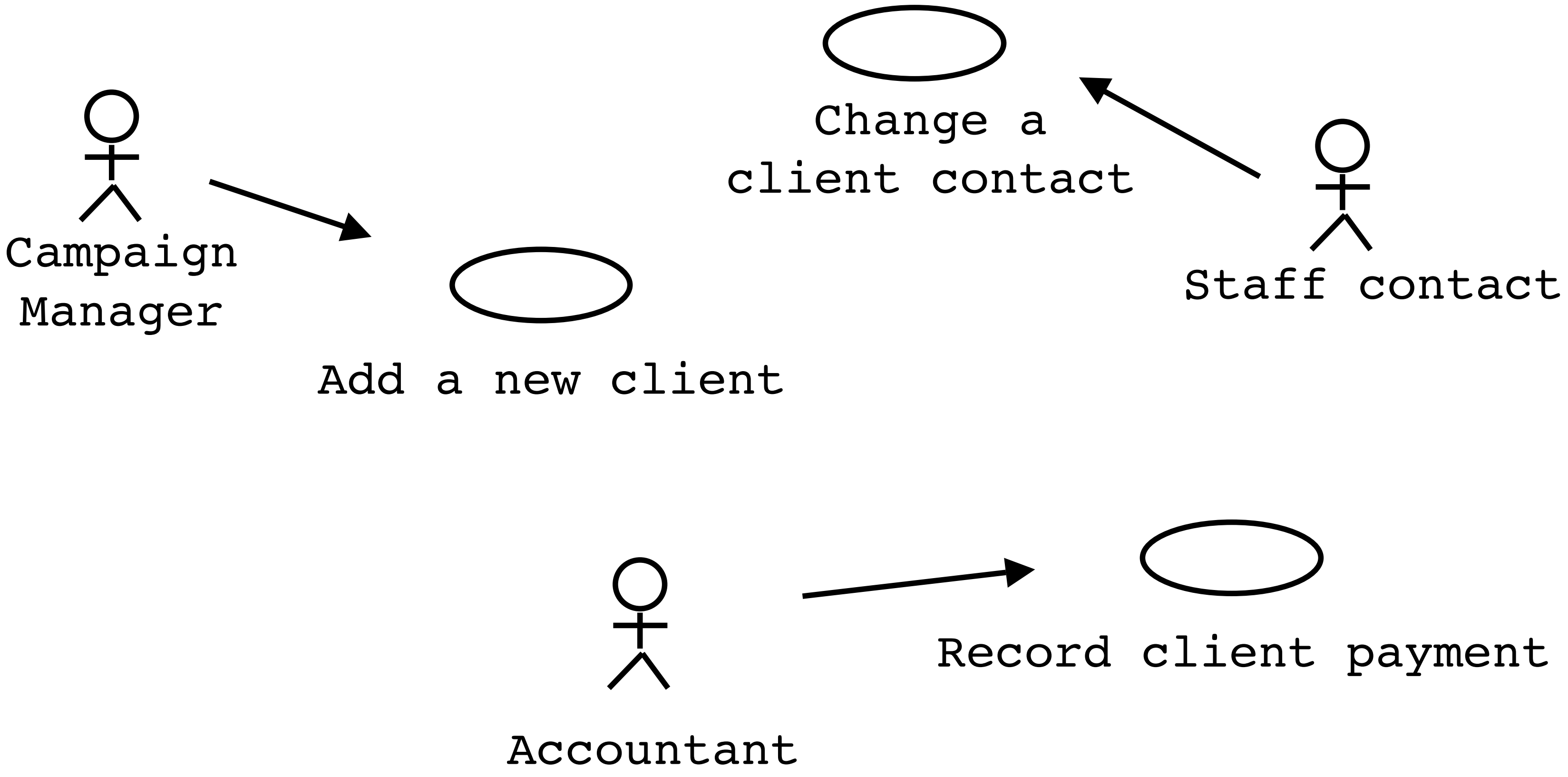
Moving towards specification

- What functions will the new system provide?
  - How will people interact with it?
  - Describe functions from a user's perspective
- UML Use Cases
  - Used to show:
    - the **functions** to be provided by the system
    - which **actors** will use which functions
  - Each Use Case is:
    - a pattern of behavior that the new system is required to exhibit
    - a sequence of related actions performed by an actor and the system via a dialogue.
- An actor is:
  - anything that needs to interact with the system:
    - a person
    - a role that different people may play
    - another (external) system.

© 2004-5 Steve Easterbrook. This presentation is available free for non-commercial use with attribution under a creative commons license.

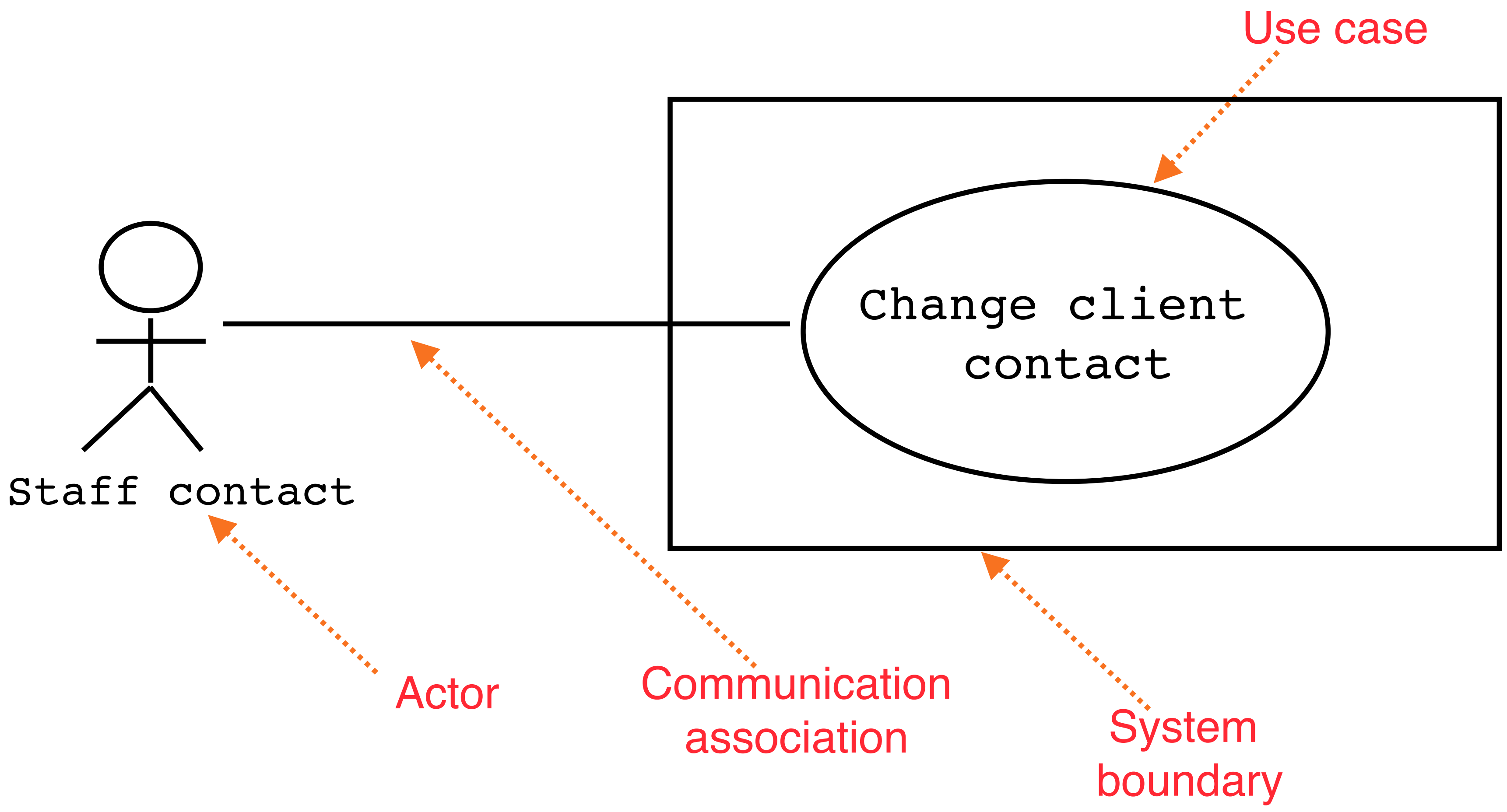
Use Case Diagrams

- → Capture the relationships between actors and Use Cases



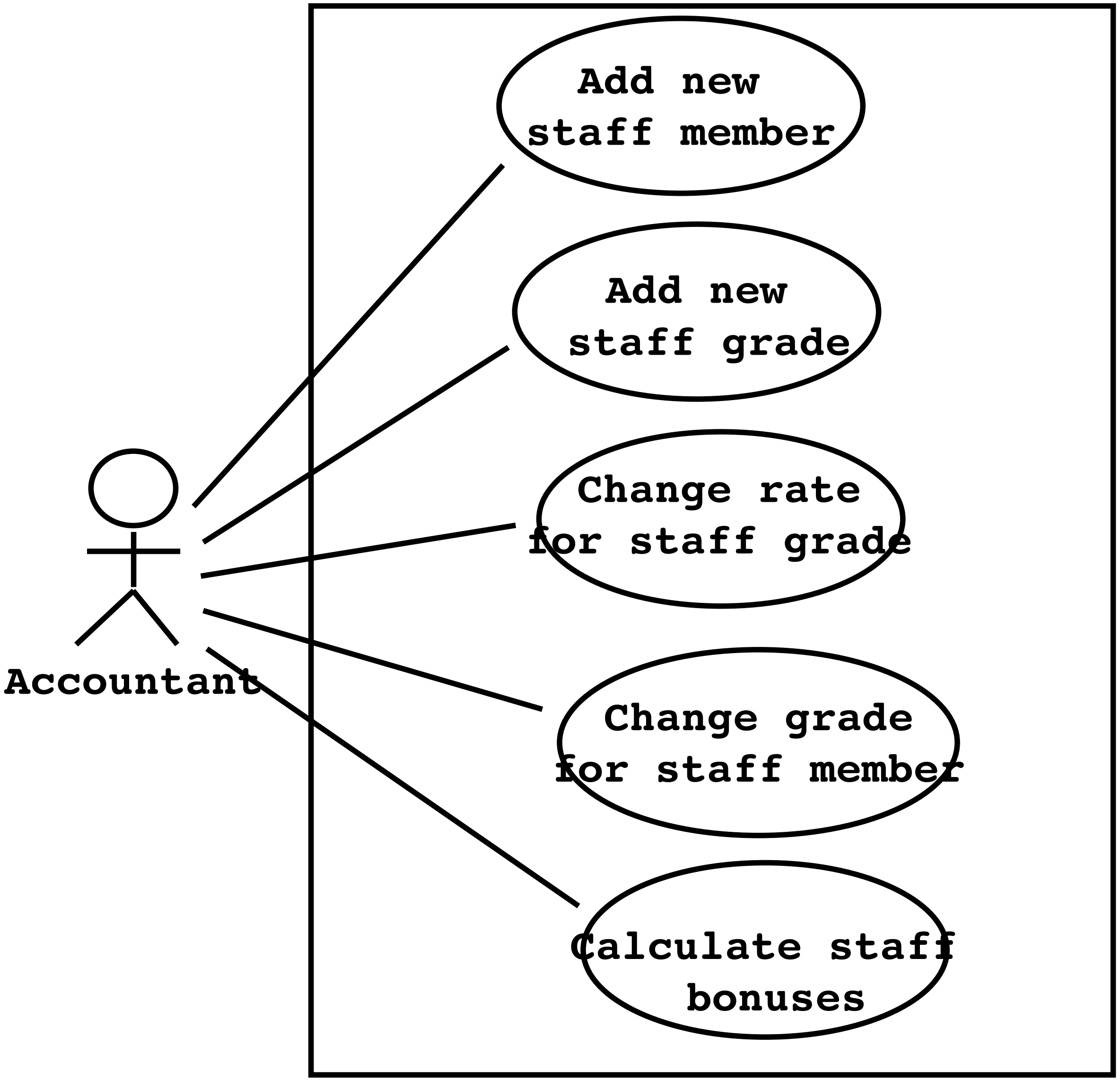
© 2004-5 Steve Easterbrook. This presentation is available free for non-commercial use with attribution under a creative commons license.

Notation for Use Case Diagrams



© 2004-5 Steve Easterbrook. This presentation is available free for non-commercial use with attribution under a creative commons license.

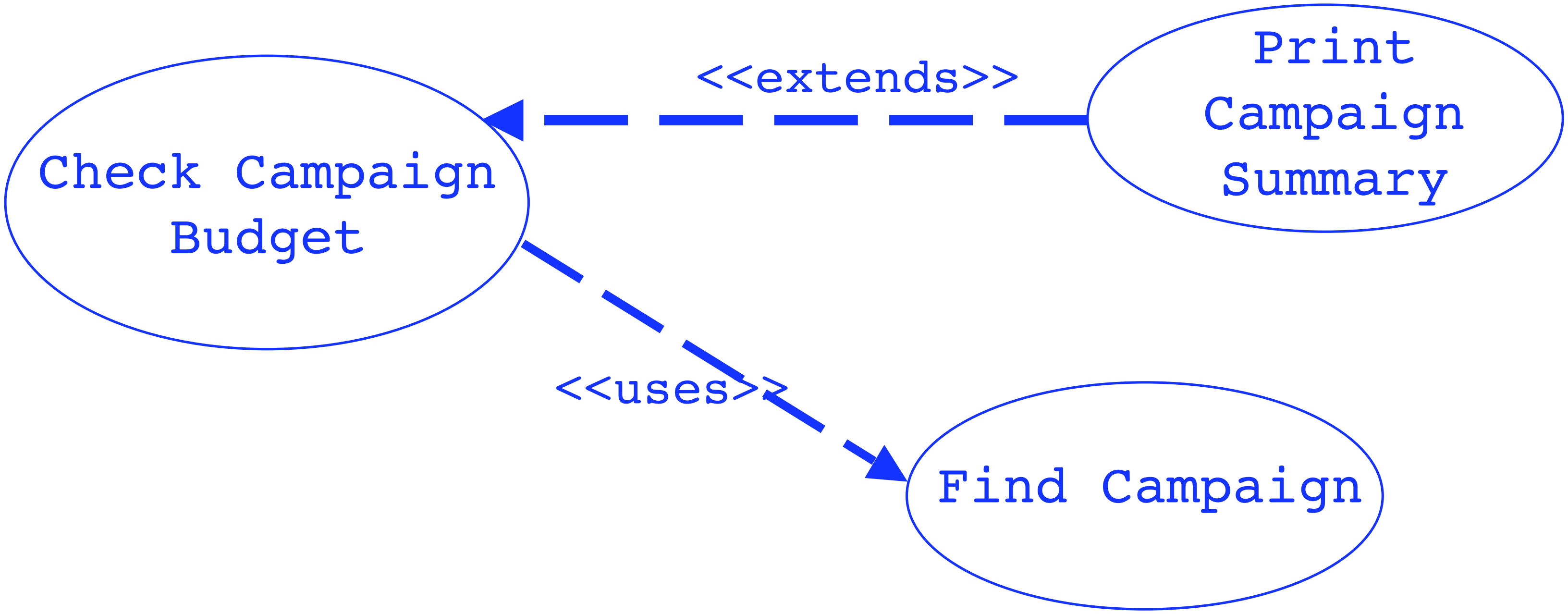
Example



© 2004-5 Steve Easterbrook. This presentation is available free for non-commercial use with attribution under a creative commons license.

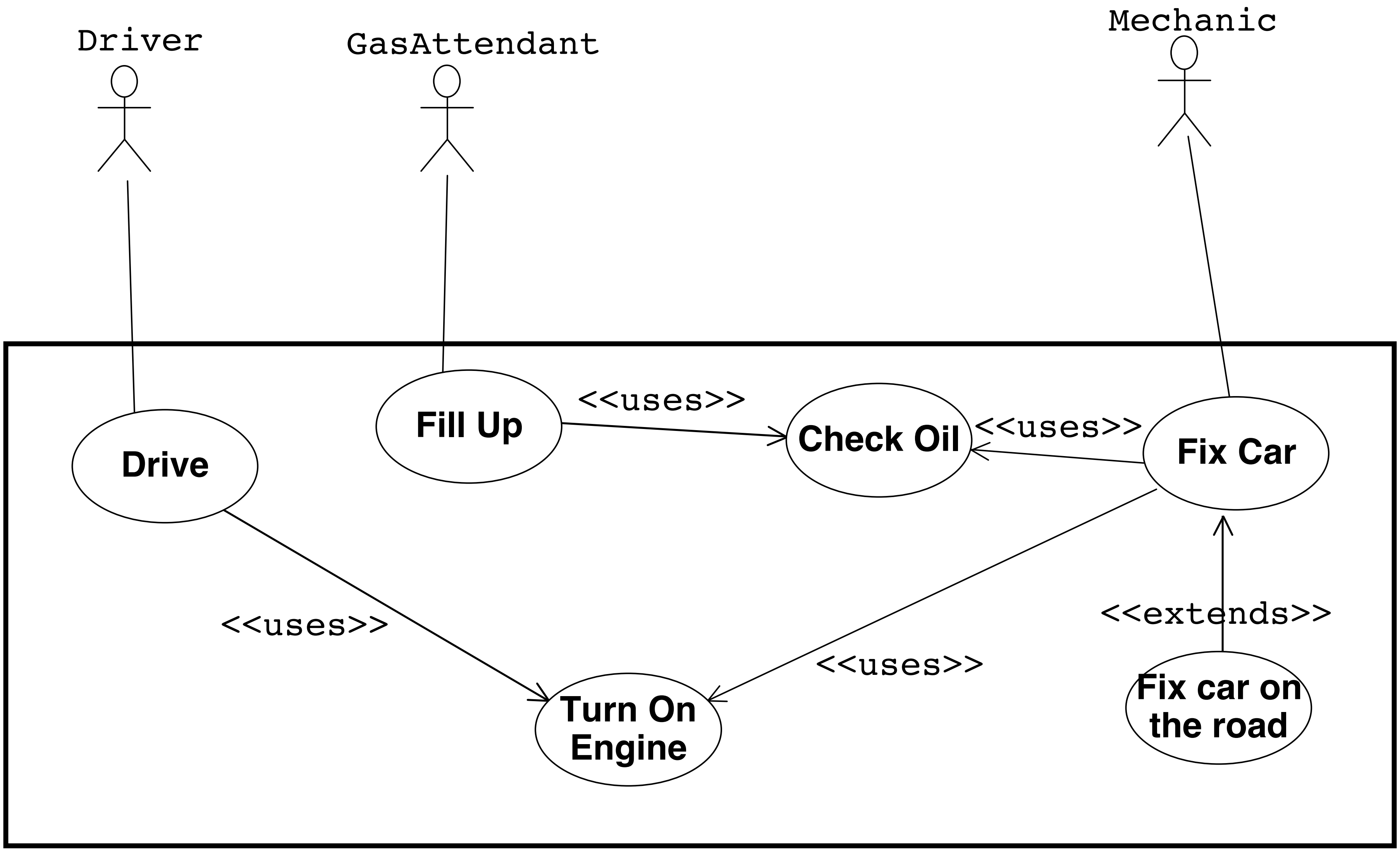
<<extends>> and <<uses>>

- → <<extends>> when one use case adds behaviour to a base case
  - ↳ used to model a part of a use case that the user may see as optional system behavior;
  - ↳ also models a separate sub-case which is executed conditionally.
- → <<uses>>: one use case invokes another (like a procedure call);
  - ↳ used to avoid describing the same flow of events several times
  - ↳ puts the common behavior in a use case of its own.



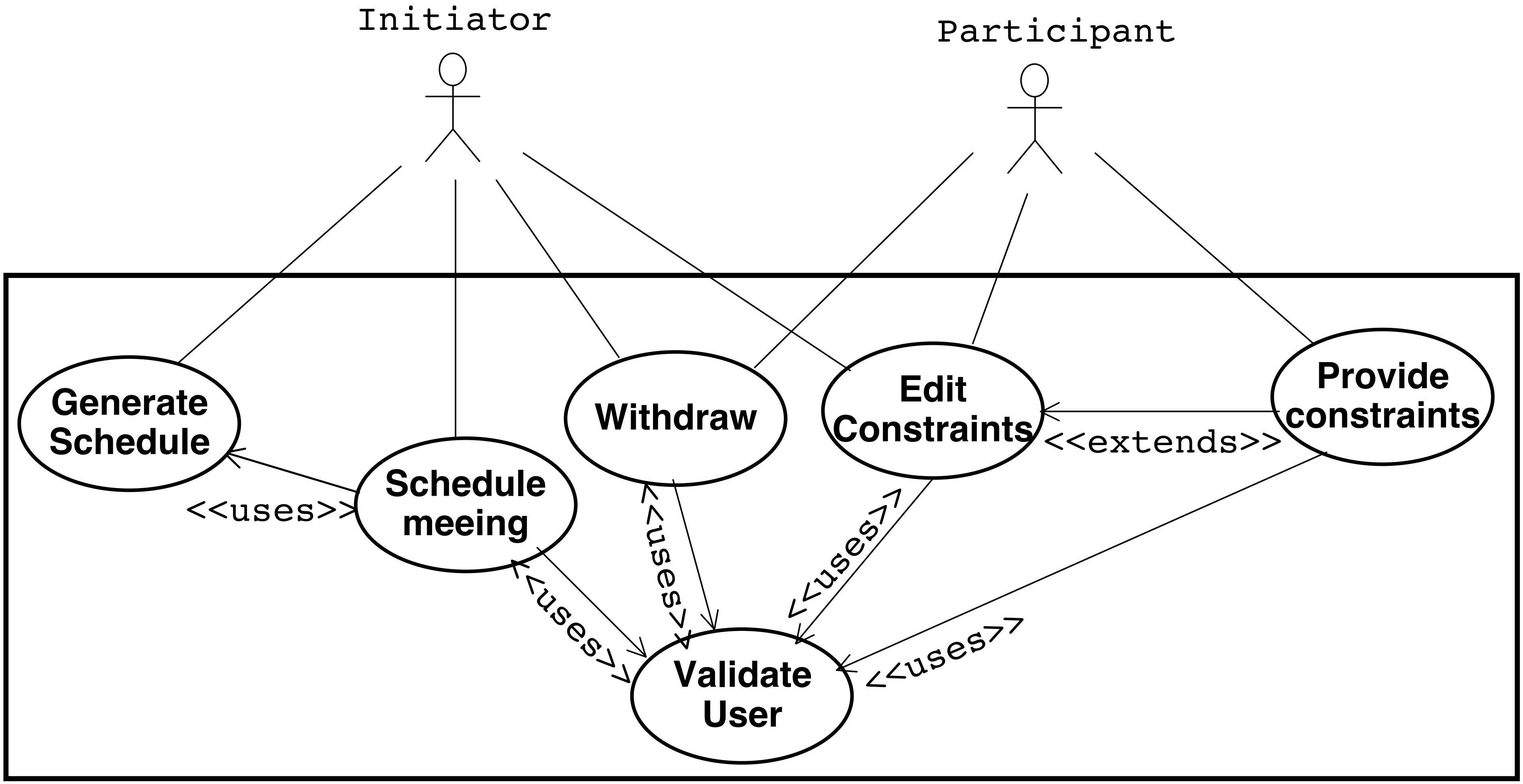
© 2004-5 Steve Easterbrook. This presentation is available free for non-commercial use with attribution under a creative commons license.

Sample use cases for a car



© 2004-5 Steve Easterbrook. This presentation is available free for non-commercial use with attribution under a creative commons license.

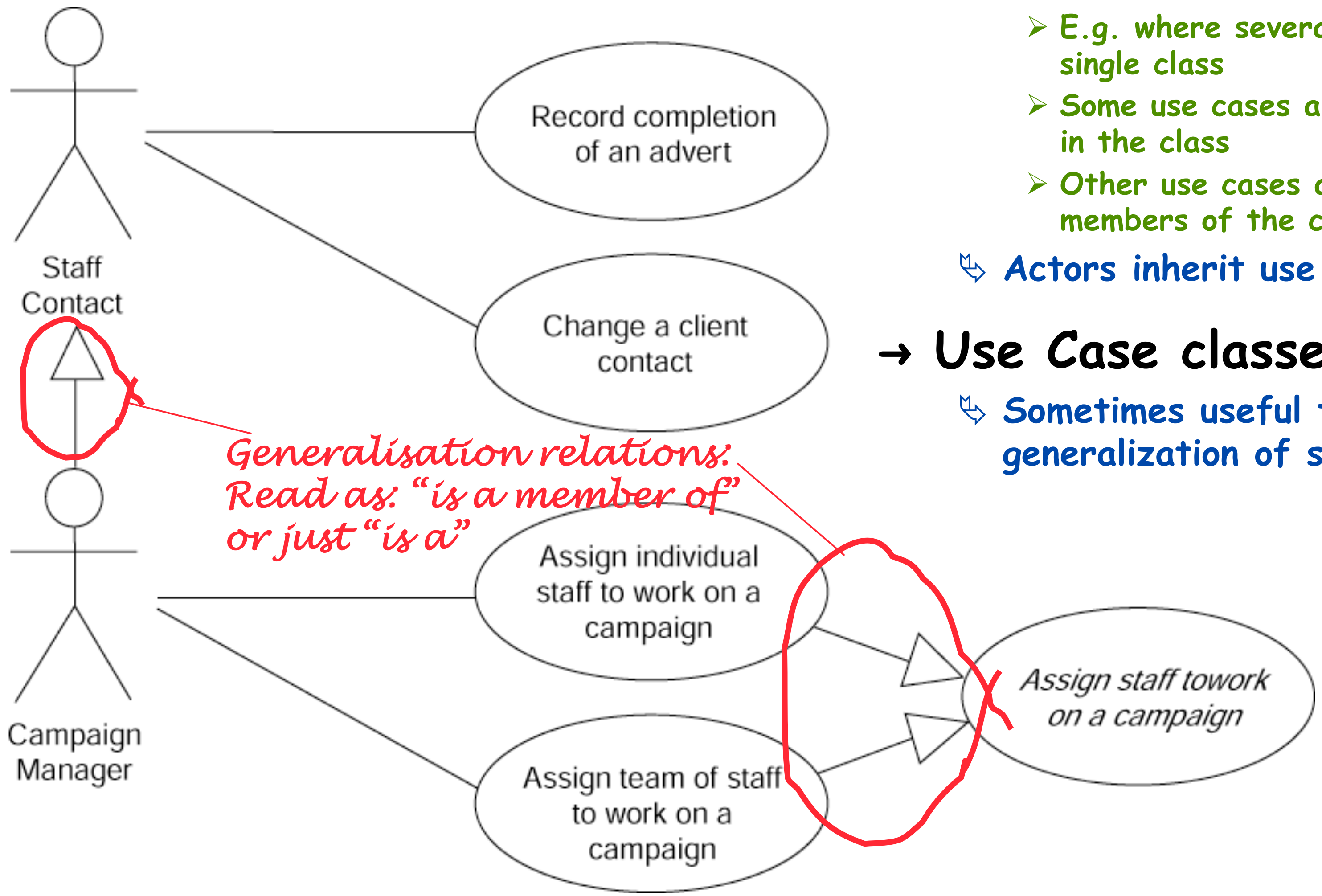
Meeting Scheduler Example



© 2004-5 Steve Easterbrook. This presentation is available free for non-commercial use with attribution under a creative commons license.

Generalizations

- Actor classes
  - It’s sometimes useful to identify classes of actor
    - E.g. where several actors belong to a single class
    - Some use cases are needed by all members in the class
    - Other use cases are only needed by some members of the class
  - Actors inherit use cases from the class
- Use Case classes
  - Sometimes useful to identify a generalization of several use cases


Staff Contact
Record completion of an advert
Change a client contact
Campaign Manager
Assign individual staff to work on a campaign
Assign team of staff to work on a campaign
Assign staff towork on a campaign

Generalisation relations: Read as: “is a member of” or just “is a”

© 2004-5 Steve Easterbrook. This presentation is available free for non-commercial use with attribution under a creative commons license.

Identifying Actors

- Ask the following questions:
  - Who will be a primary user of the system? (primary actor)
    - Who will need support from the system to do her daily tasks?
    - Who or what has an interest in the results that the system produces ?
  - Who will maintain, administrate, keep the system working? (secondary actor)
  - Which hardware devices does the system need?
  - With which other systems does the system need to interact with?
- Look for:
  - the users who directly use the system
  - also others who need services from the system

SOME RIGHTS RESERVED © 2004-5 Steve Easterbrook. This presentation is available free for non-commercial use with attribution under a creative commons license.

Finding Use Cases

→ For each actor, ask the following questions:

- Which functions does the actor require from the system?
- What does the actor need to do ?
- Does the actor need to read, create, destroy, modify, or store some kinds of information in the system ?
- Does the actor have to be notified about events in the system?
- Does the actor need to notify the system about something?
- What do those events require in terms of system functionality?
- Could the actor's daily work be simplified or made more efficient through new functions provided by the system?

SOME RIGHTS RESERVED © 2004-5 Steve Easterbrook. This presentation is available free for non-commercial use with attribution under a creative commons license.

Documenting Use Cases

- For each use case:
  - prepare a “flow of events” document, written from an actor’s point of view.
  - describe what the system must provide to the actor when the use case is executed.
- Typical contents
  - How the use case starts and ends;
  - Normal flow of events;
  - Alternate flow of events;
  - Exceptional flow of events;
- Documentation style:
  - Choice of how to represent the use case:
    - English language description
    - Activity Diagrams - good for business process
    - Collaboration Diagrams - good for high level design
    - Sequence Diagrams - good for detailed design

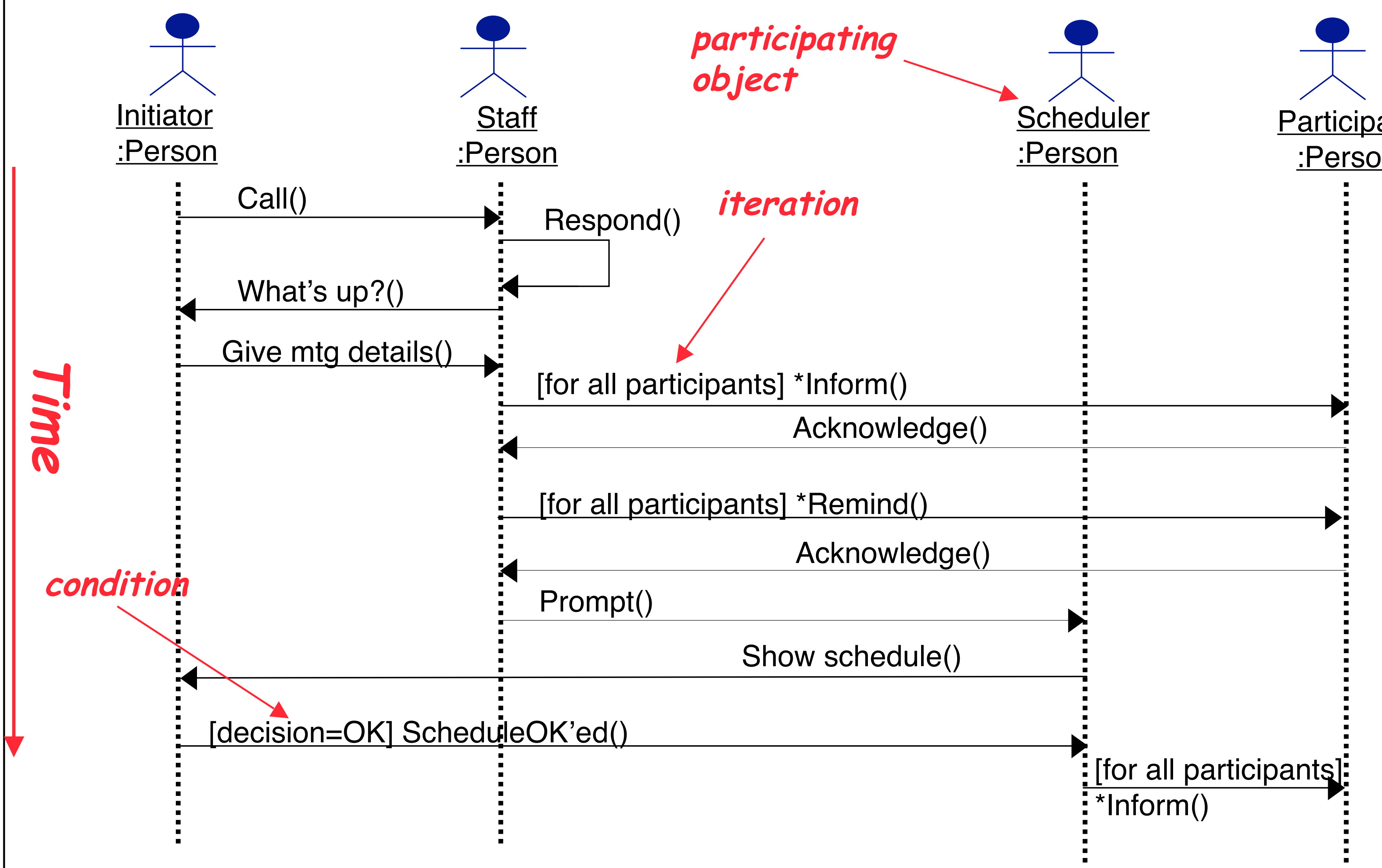
© 2004-5 Steve Easterbrook. This presentation is available free for non-commercial use with attribution under a creative commons license.

Modelling Sequences of Events

- Objects "own" information and behaviour
  - they have attributes and operations relevant to their *responsibilities*.
  - They don't "know" about other objects' information, but can ask for it.
  - To carry out business processes, objects have to collaborate.
    - …by sending messages to one another to invoke each others' operations
  - Objects can only send messages to one another if they "know" each other
    - I.e. if there is an association between them.
- Describe a Use Case using Sequence Diagrams
  - Sequence diagrams show step-by-step what's involved in a use case
    - Which objects are relevant to the use case
    - How those objects participate in the function
  - You may need several sequence diagrams to describe a single use case.
    - Each sequence diagram describes one possible scenario for the use case
  - Sequence diagrams…
    - …should remain easy to read and understand.
    - …do not include complex control logic

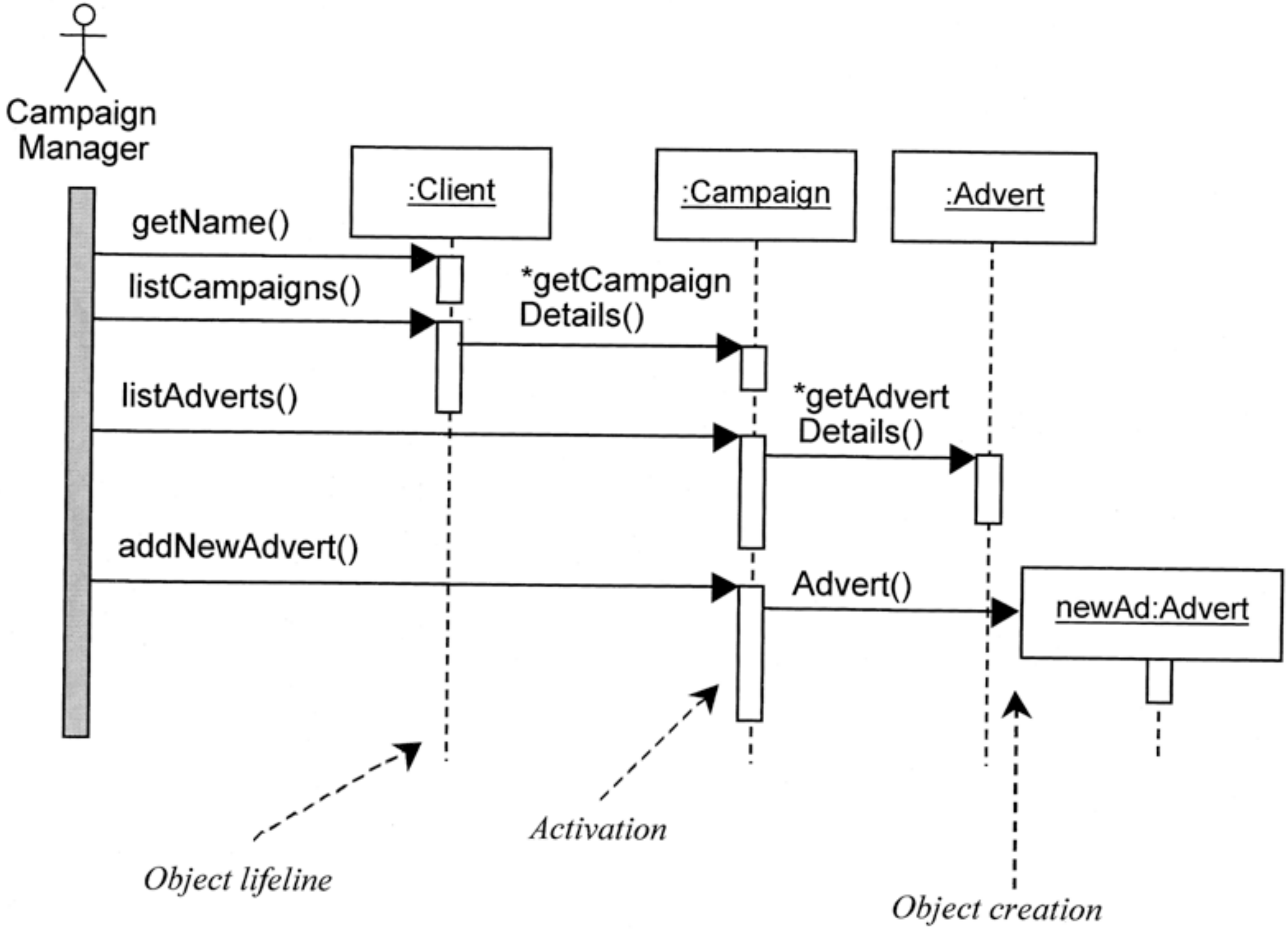
© 2004-5 Steve Easterbrook. This presentation is available free for non-commercial use with attribution under a creative commons license.

Example Sequence Diagram



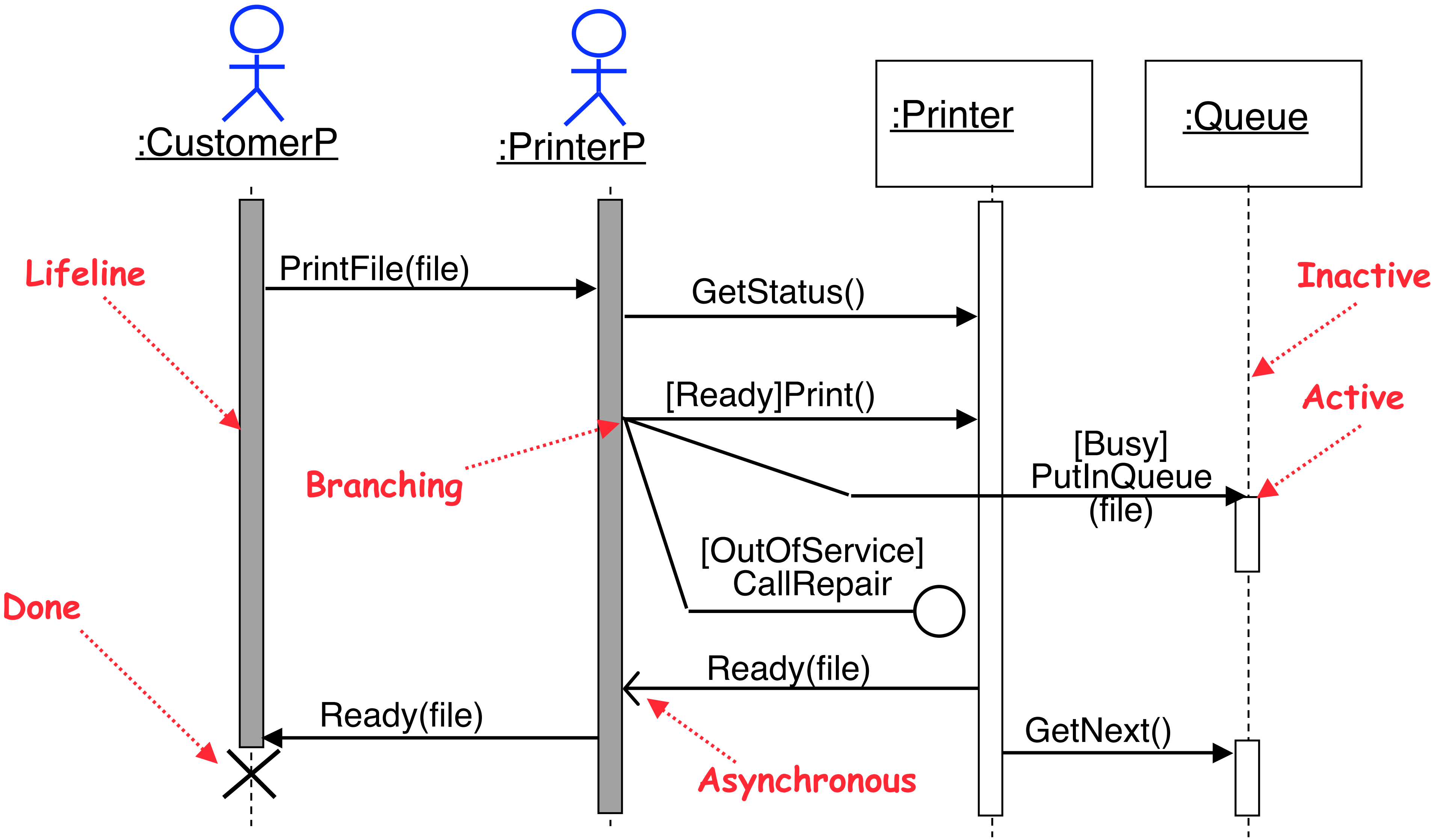
© 2004-5 Steve Easterbrook. This presentation is available free for non-commercial use with attribution under a creative commons license.

Another Example



© 2004-5 Steve Easterbrook. This presentation is available free for non-commercial use with attribution under a creative commons license.

Branching messages, etc



SOME RIGHTS RESERVED © 2004-5 Steve Easterbrook. This presentation is available free for non-commercial use with attribution under a creative commons license.

Don't forget what we're modelling

- During analysis
  - we want to know about the application domain and the requirements
  - …so we develop a course-grained model to show where responsibilities are, and how objects interact
    - Our models show a message being passed, but we don't worry too much about the contents of each message
    - To keep things clear, use icons to represent external objects and actors, and boxes to represent system objects.
- During design
  - we want to say how the software should work
  - … so we develop fine-grained models to show exactly what will happen when the system runs
    - E.g. show the precise details of each method call.

SOME RIGHTS RESERVED © 2004-5 Steve Easterbrook. This presentation is available free for non-commercial use with attribution under a creative commons license.