Software component interactions and sequence diagrams

Nigel Goddard

School of Informatics
University of Edinburgh

What do we need to know? Recap

Recall that this is an *overview* of software engineering, dipping into some aspects. We've discussed:

- how to analyse requirements and summarise them in a use case diagram;

What do we need to know? Recap

Recall that this is an *overview* of software engineering, dipping into some aspects. We've discussed:

- how to analyse requirements and summarise them in a use case diagram;
- how to tell good design from bad;

What do we need to know? Recap

Recall that this is an *overview* of software engineering, dipping into some aspects. We've discussed:

- how to analyse requirements and summarise them in a use case diagram;
- how to tell good design from bad;
- how to record basics of the static structure of our designed system in a class diagram;

What do we need to know? Recap

Recall that this is an *overview* of software engineering, dipping into some aspects. We've discussed:

- how to analyse requirements and summarise them in a use case diagram;
- how to tell good design from bad;
- how to record basics of the static structure of our designed system in a class diagram;
- how to get started with choosing an appropriate static structure.

What do we need to know? Recap

Recall that this is an *overview* of software engineering, dipping into some aspects. We've discussed:

- how to analyse requirements and summarise them in a use case diagram;
- how to tell good design from bad;
- how to record basics of the static structure of our designed system in a class diagram;
- how to get started with choosing an appropriate static structure.

What do we need to know? Recap

Recall that this is an *overview* of software engineering, dipping into some aspects. We've discussed:

- how to analyse requirements and summarise them in a use case diagram;
- how to tell good design from bad;
- how to record basics of the static structure of our designed system in a class diagram;
- how to get started with choosing an appropriate static structure.

We have not discussed dynamic aspects of design: what operations should our classes have, and what should they do?

Dynamic aspects of design

Suppose that we have decided what classes should be in our system, provisionally. What next? Well, we have to meet the requirements...

Dynamic aspects of design

Suppose that we have decided what classes should be in our system, provisionally. What next? Well, we have to meet the requirements...

In the end, we need to know what operations they have, and what each method should do.

Dynamic aspects of design

Suppose that we have decided what classes should be in our system, provisionally. What next? Well, we have to meet the requirements...

In the end, we need to know what operations they have, and what each method should do.

Two ways of looking at this:

1. inter-object behaviour: who sends which messages to whom?

Dynamic aspects of design

Suppose that we have decided what classes should be in our system, provisionally. What next? Well, we have to meet the requirements...

In the end, we need to know what operations they have, and what each method should do.

Two ways of looking at this:

1. inter-object behaviour: who sends which messages to whom?
2. intra-object behaviour: what state changes does each object undergo as it receives messages, and how do they affect its behaviour?

Dynamic aspects of design

Suppose that we have decided what classes should be in our system, provisionally. What next? Well, we have to meet the requirements...

In the end, we need to know what operations they have, and what each method should do.

Two ways of looking at this:

1. inter-object behaviour: who sends which messages to whom?
2. intra-object behaviour: what state changes does each object undergo as it receives messages, and how do they affect its behaviour?

Dynamic aspects of design

Suppose that we have decided what classes should be in our system, provisionally. What next? Well, we have to meet the requirements...

In the end, we need to know what operations they have, and what each method should do.

Two ways of looking at this:

1. inter-object behaviour: who sends which messages to whom?
2. intra-object behaviour: what state changes does each object undergo as it receives messages, and how do they affect its behaviour?

Complementary: but in this course, we only consider 1. For 2, UML provides an enhanced variant on the FSMs you saw last year.

For more info, do SEOC next year, and/or read the recommended texts.

Thinking about inter-object behaviour

There's no algorithm for constructing a good design. Create one that's good according to the design principles...

1. Your classes should, as far as possible, correspond to domain concepts.

Thinking about inter-object behaviour

There's no algorithm for constructing a good design. Create one that's good according to the design principles...

1. Your classes should, as far as possible, correspond to domain concepts.
2. The data encapsulated in the classes is usually pretty easy to define using the real world as a model.

Thinking about inter-object behaviour

There's no algorithm for constructing a good design. Create one that's good according to the design principles...

1. Your classes should, as far as possible, correspond to domain concepts.
2. The data encapsulated in the classes is usually pretty easy to define using the real world as a model.
3. Then look at the scenarios in the use cases, and work out where to put what operations to get them done.

Thinking about inter-object behaviour

There's no algorithm for constructing a good design. Create one that's good according to the design principles...

1. Your classes should, as far as possible, correspond to domain concepts.
2. The data encapsulated in the classes is usually pretty easy to define using the real world as a model.
3. Then look at the scenarios in the use cases, and work out where to put what operations to get them done.

Some of this is easy. Hard parts are usually when several objects have to collaborate and it isn't clear which should take overall responsibility.

Interaction diagrams

describe the *dynamic* interactions between objects in the system, i.e. the pattern of message-passing.

Interaction diagrams

describe the *dynamic* interactions between objects in the system, i.e. the pattern of message-passing.

Two main uses:

- Showing how the system realises [part of] a use case

Interaction diagrams

describe the *dynamic* interactions between objects in the system, i.e. the pattern of message-passing.

Two main uses:

- Showing how the system realises [part of] a use case
- Showing how an object reacts to some message

Interaction diagrams

describe the *dynamic* interactions between objects in the system, i.e. the pattern of message-passing.

Two main uses:

- Showing how the system realises [part of] a use case
- Showing how an object reacts to some message

Particularly useful where the flow of control is complicated, since this can't be deduced from the class model, which is static.

Interaction diagrams

describe the *dynamic* interactions between objects in the system, i.e. the pattern of message-passing.

Two main uses:

- Showing how the system realises [part of] a use case
- Showing how an object reacts to some message

Particularly useful where the flow of control is complicated, since this can't be deduced from the class model, which is static.

UML has two sorts, *sequence* and *communication* diagrams – the differences are subtle, and we'll only talk about sequence diagrams.

Developing an interaction diagram

1. Decide exactly what behaviour to model.

Developing an interaction diagram

1. Decide exactly what behaviour to model.
2. Check that you know how the system provides the behaviour: are all the necessary classes and relationships in the class model?

Developing an interaction diagram

1. Decide exactly what behaviour to model.
2. Check that you know how the system provides the behaviour: are all the necessary classes and relationships in the class model?
3. Name the objects which are involved.

Developing an interaction diagram

1. Decide exactly what behaviour to model.
2. Check that you know how the system provides the behaviour: are all the necessary classes and relationships in the class model?
3. Name the objects which are involved.
4. Identify the sequence of messages which the objects send to one another.

Developing an interaction diagram

1. Decide exactly what behaviour to model.
2. Check that you know how the system provides the behaviour: are all the necessary classes and relationships in the class model?
3. Name the objects which are involved.
4. Identify the sequence of messages which the objects send to one another.
5. Record this in the syntax of a sequence diagram.

Developing an interaction diagram

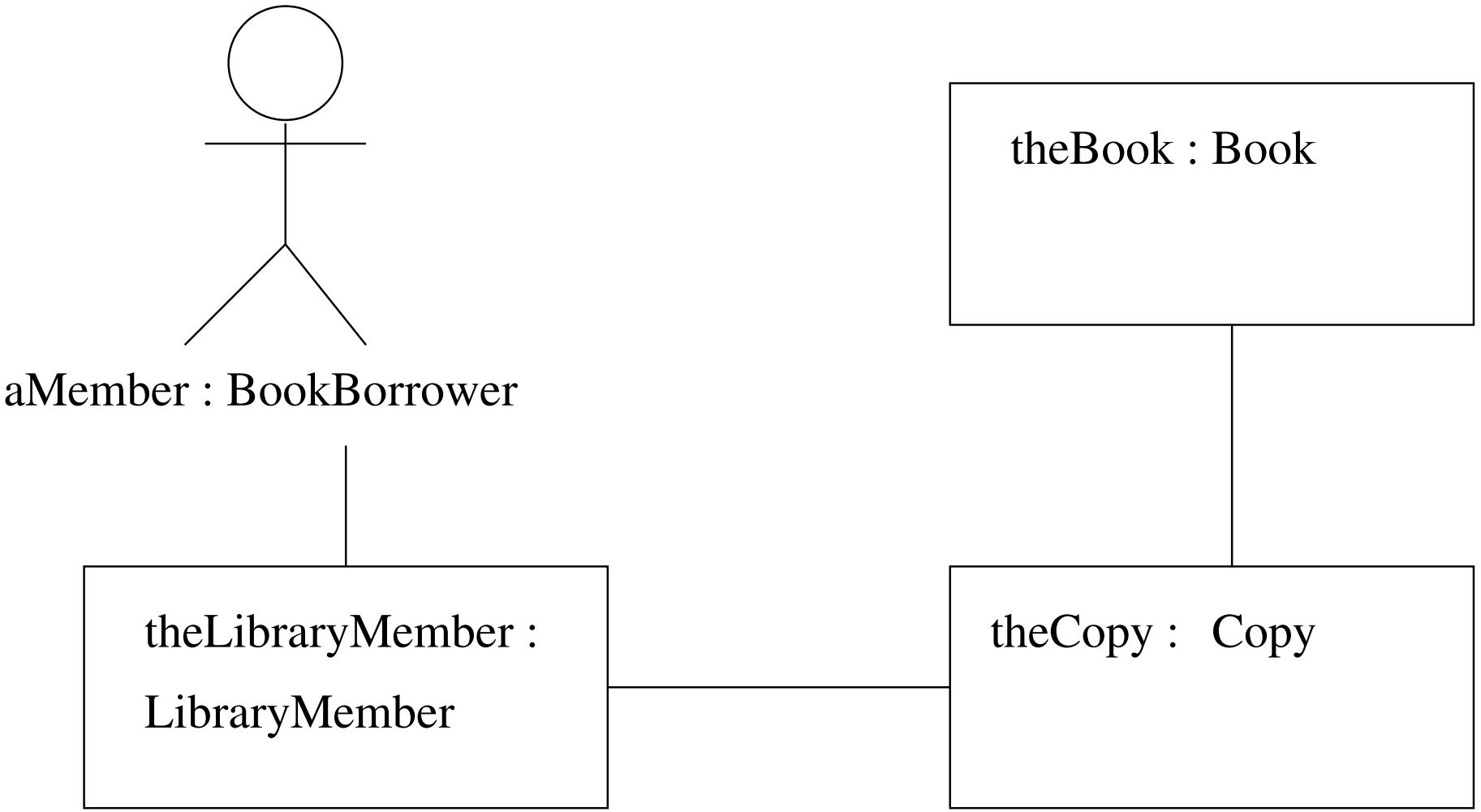
1. Decide exactly what behaviour to model.
2. Check that you know how the system provides the behaviour: are all the necessary classes and relationships in the class model?
3. Name the objects which are involved.
4. Identify the sequence of messages which the objects send to one another.
5. Record this in the syntax of a sequence diagram.

Developing an interaction diagram

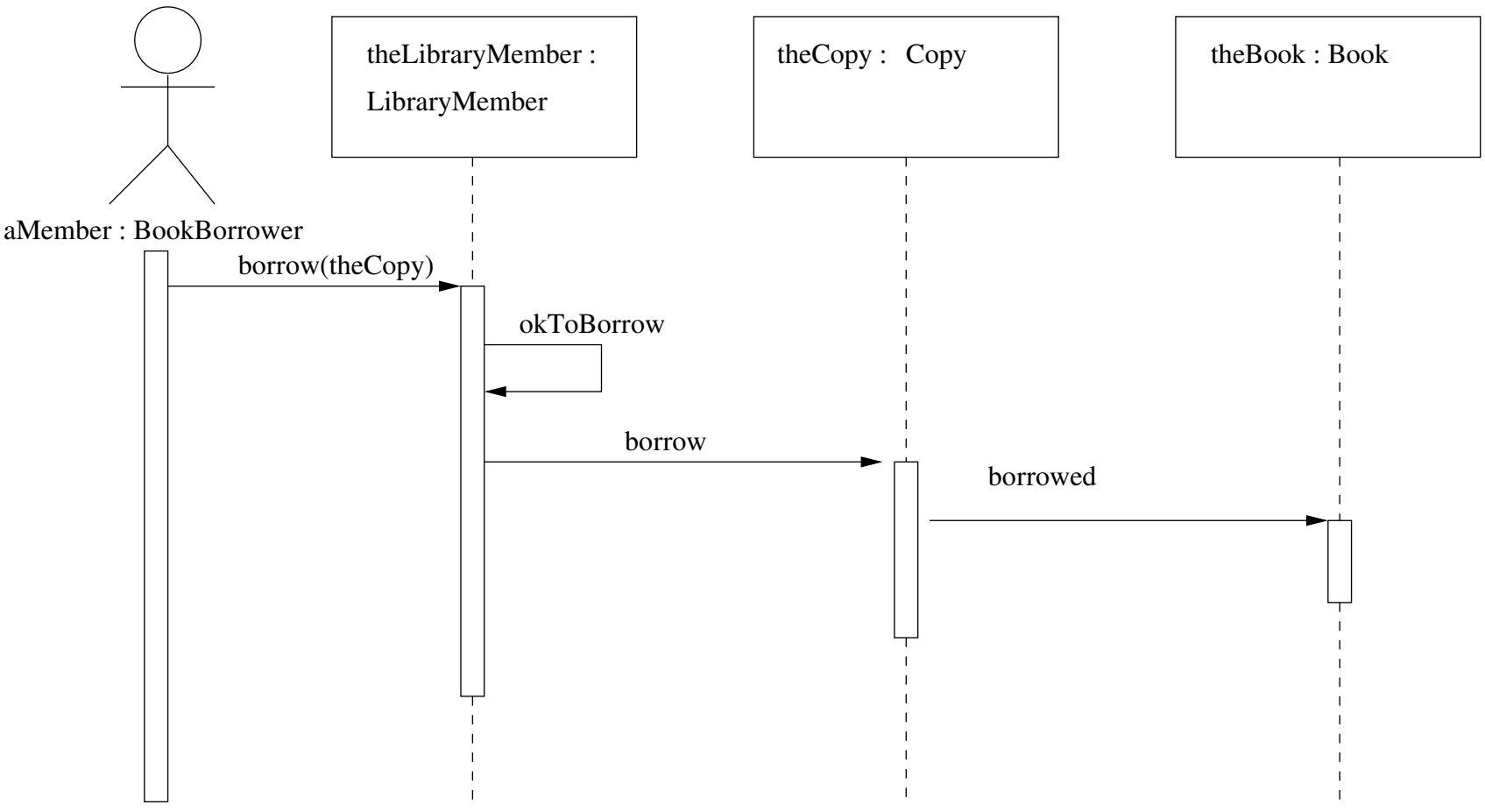
1. Decide exactly what behaviour to model.
2. Check that you know how the system provides the behaviour: are all the necessary classes and relationships in the class model?
3. Name the objects which are involved.
4. Identify the sequence of messages which the objects send to one another.
5. Record this in the syntax of a sequence diagram.

Simple :-)

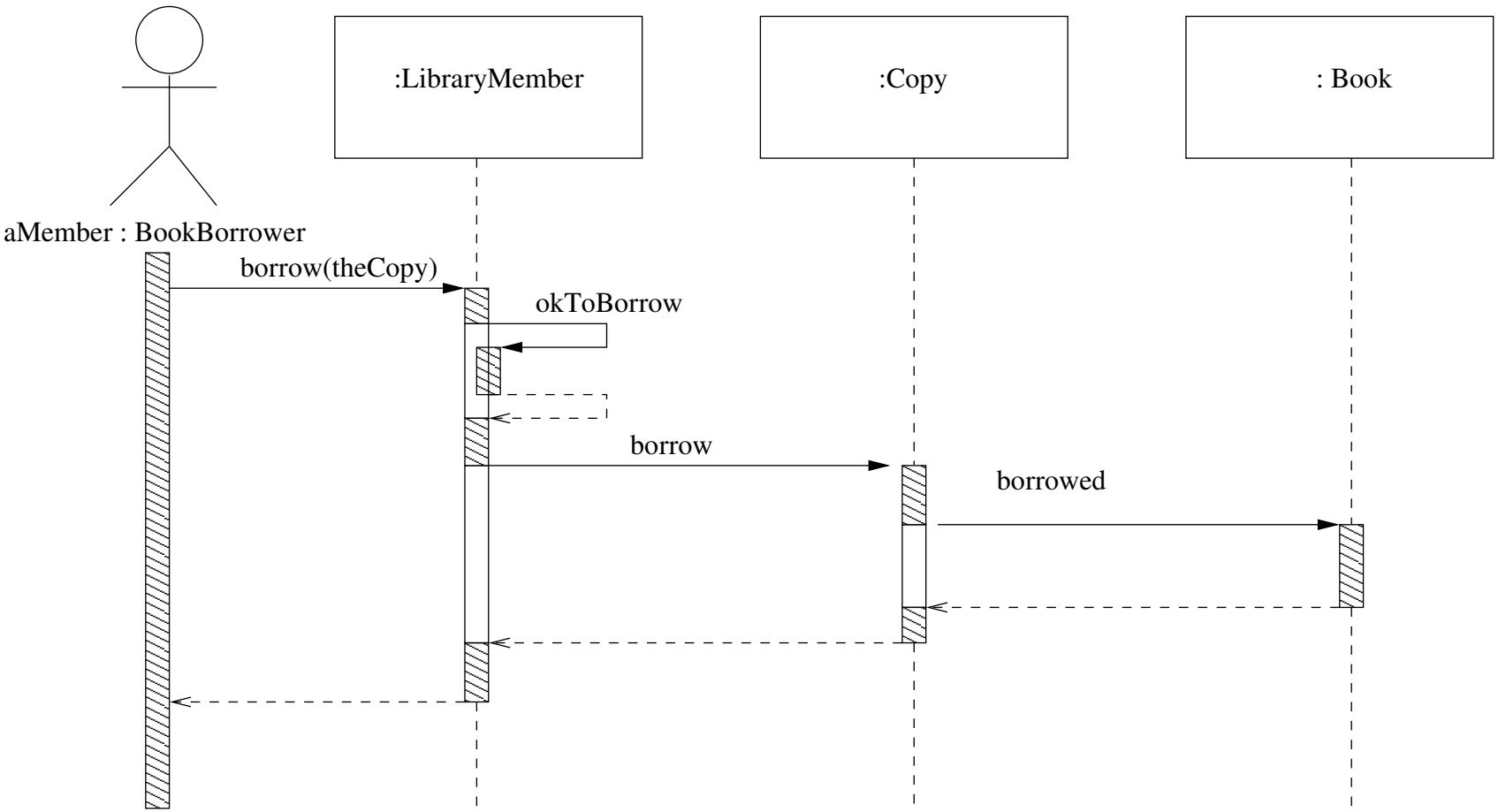
A collaboration



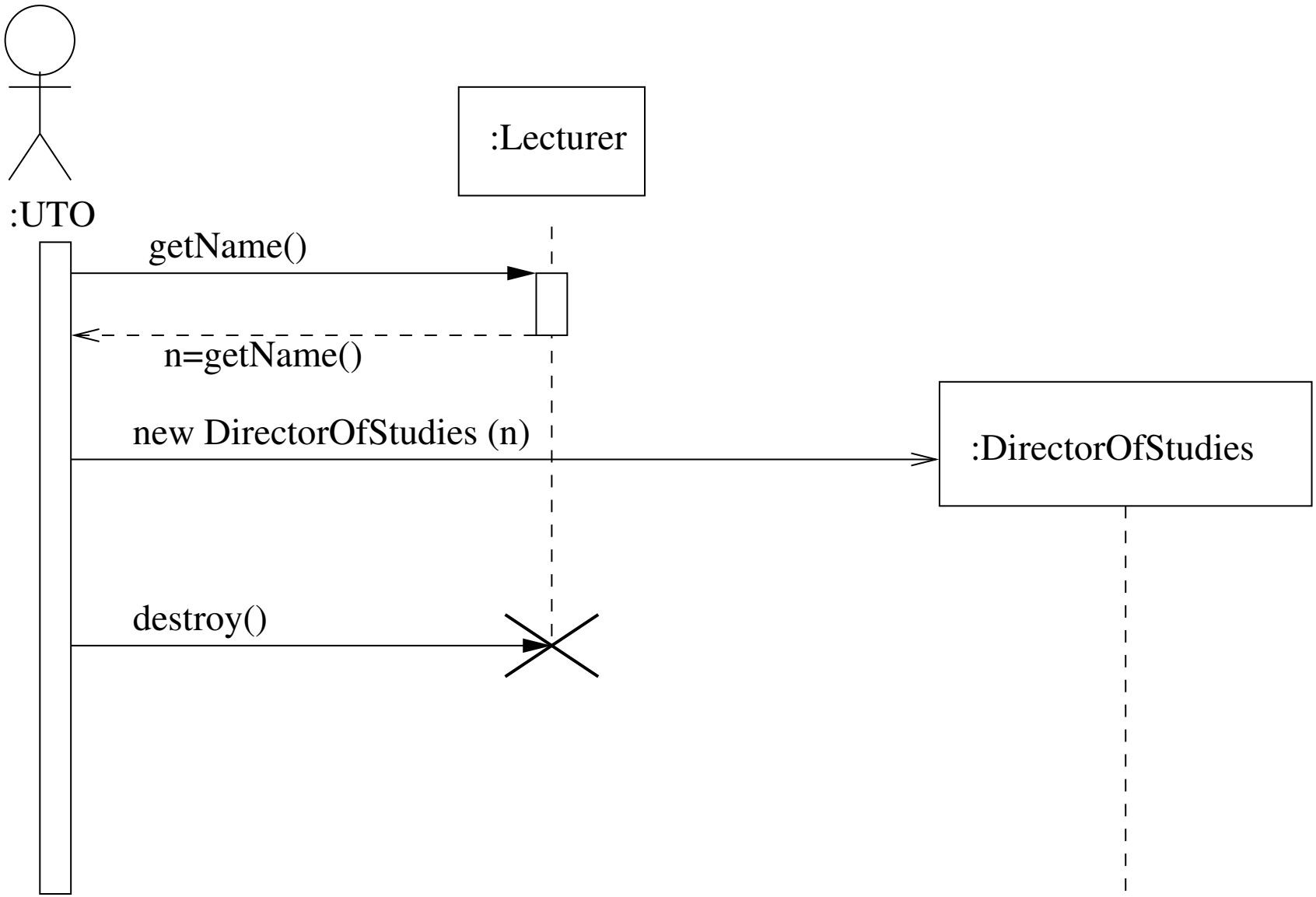
Sequence diagram



Showing more detail



Creation/deletion in sequence diagram



What is a good interaction pattern?

In designing an interaction, your first aim is obviously to design *some* collection of operations that can work together to achieve the aim.

Next, consider:

- conceptual coherence: does it make sense for this class to have that operation?

What is a good interaction pattern?

In designing an interaction, your first aim is obviously to design *some* collection of operations that can work together to achieve the aim.

Next, consider:

- conceptual coherence: does it make sense for this class to have that operation?
- maintainability: which aspects might change, and how hard will it be to change the interaction accordingly?

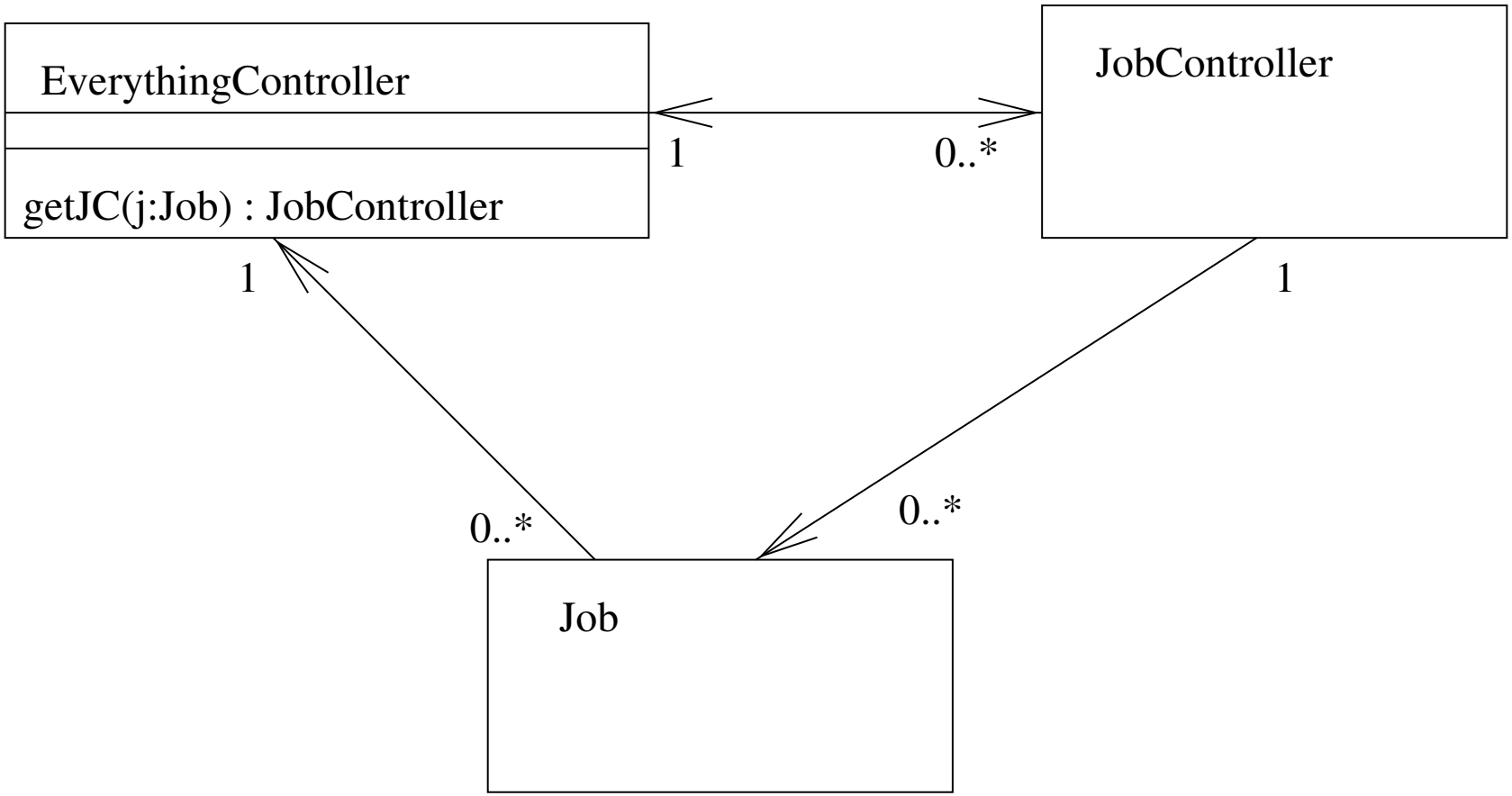
What is a good interaction pattern?

In designing an interaction, your first aim is obviously to design *some* collection of operations that can work together to achieve the aim.

Next, consider:

- conceptual coherence: does it make sense for this class to have that operation?
- maintainability: which aspects might change, and how hard will it be to change the interaction accordingly?
- performance: is all the work being done necessary?

Designing interactions



Problems?

Law of Demeter

in response to a message m, an object O should send messages *only* to the following objects:

1. O itself

Law of Demeter

in response to a message m, an object O should send messages *only* to the following objects:

1. O itself
2. objects which are sent as arguments to the message m

Law of Demeter

in response to a message m, an object O should send messages *only* to the following objects:

1. O itself
2. objects which are sent as arguments to the message m
3. objects which O creates as part of its reaction to m

Law of Demeter

in response to a message m, an object O should send messages *only* to the following objects:

1. O itself
2. objects which are sent as arguments to the message m
3. objects which O creates as part of its reaction to m
4. objects which are *directly* accessible from O, that is, using values of attributes of O.

More complex sequence diagrams

We've only discussed very simple sequence diagrams. UML provides notation for reusing pieces of interactions, conditional or iterative behaviour, asynchronous messages, etc. etc.

Reading

Suggested: The original paper on CRC cards, a technique for designing interactions: *A Laboratory for Object-Oriented Thinking*, by Kent Beck and Ward Cunningham. See web page.

MCQ from 05/06 exam

When does CVS check for the existence of potentially conflicting changes to a version-controlled file?

A at *checkout*

B at *update*

C at *commit* (checkin)

D at more than one of the above

E at none of the above

Answer:

MCQ from 05/06 exam

When does CVS check for the existence of potentially conflicting changes to a version-controlled file?

A at *checkout*

B at *update*

C at *commit* (checkin)

D at more than one of the above

E at none of the above

Answer: D