From requirements to modelling 2: Robustness analysis, CRC cards

Perdita Stevens

School of Informatics
University of Edinburgh

Allocating data and behaviour to classes

Students often have trouble with building a conceptual class model, especially with going beyond class names to allocate data and behaviour to classes.

Techniques worth considering:

- robustness analysis;
- using CRC cards.

Once you understand behaviour you can record it in sequence diagrams to explain how the scenarios in the use cases play out.

Robustness analysis

systematises the essential process of making the use case text and the conceptual class diagram consistent, and moving the conceptual class diagram on to a design class diagram.
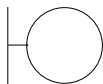
For each use case,

build a robustness diagram*

revising the use case text and class model as necessary.

* *not* part of UML

"A robustness diagram is an object picture of a use case"

It contains:

1. boundary objects (e.g., screens) [N]

2. entity objects (instances of your conceptual classes) [N]

3. controllers (typically messages in the end) [V]

You draw the use case by connecting these with no two Ns adjacent. One use case sentence at a time.

boundary and entity objects $\longleftrightarrow$ nouns [N]

controllers $\longleftrightarrow$ verbs [V]

Miniature example I

From the student detail page, the lecturer clicks on the
''Add courses'' button and the system displays the list of
courses. The lecturer selects the name of a course and
presses the ''Register'' button. The system registers the
student for the course.
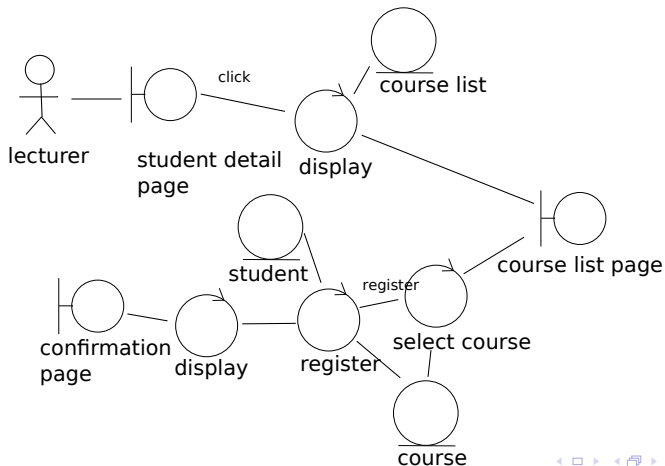
Miniature example II

From the student detail page, the lecturer clicks on the ''Add courses'' button and the system displays the list of courses. The lecturer selects the name of a course and presses the ''Register'' button. If the student satisfies the course's prerequisites and the course is not full then the system registers the student for the course and displays a success message. Otherwise it displays an explanation.

Over to you...

Things to note

Robustness diagrams are *informal* and not part of UML.

Don't worry about the detail: they are just one way to get to:

- ▶ clear, unambiguous use case descriptions
- ▶ a reasonable, complete-enough conceptual class diagram
- ▶ a list of screens/web pages needed
- ▶ beginning design of functionality.

The next stage can be harder...

Which class should contain which behaviour?

In the ICONIX approach behaviour (controllers) is initially separate from entity objects, i.e. it tends to put data first, before behaviour.

After robustness analysis we'll know (mostly) what data there is, where it is, what's connected to what;

and we'll know what behaviour there is;

but not, yet, which class is responsible for each behaviour.

Typically a controller becomes a method – but of which object?

(Complex controllers may become controller objects... this depends partly on your technical architecture/platform.)

Allocating behaviour; flow of control

Basic fact: an object that *receives* a message must have an appropriate method! Rules of thumb:

- behaviour usually lives with the data it works on;
- anthropomorphise! If the object were a person, would it be reasonable to ask it to do that?

Making these decisions is an important part of design: often easy, sometimes not. When it's not, design principles and patterns can help (more later).

Once you know how the behaviour is allocated between objects, you can record it

1. (statically) by adding operations to the domain model
2. (dynamically) by drawing a sequence diagram.

Beyond conceptual modelling...

At this stage we have domain classes with both data and behaviour.

Detailed design requires choices about technical architecture/platform, e.g., what Java UI/persistence/etc. frameworks to use; some classes and methods are dictated by them.

Eventually can draw real UML sequence diagrams that relate precisely to an implementation. But should you?

Ways to use sequence diagrams

1. To show *example* behaviour: what happens in some particular situation (typical? problematic? under discussion?)
2. To show *complete* behaviour, i.e. all the traces that can result from some starting configuration, e.g. the whole of a use case or method implementation.

1) much more useful: pseudocode usually easier than diagram for 2)!

Be clear which it is in each case and if it's an example, say exactly what the assumptions are ("This is what happens if a lecturer tries to register a student for a course that is full").

Alternatively: CRC cards

CRC cards are another way of getting from an initial understanding of the domain plus an initial understanding of the requirements to solid class model with data and behaviour.

More behaviour/responsibility-oriented than ICONIX.

Tends to abstract away GUI screens/pages entirely.

Which approach you prefer is really a matter of taste.

CRC cards

Class, **R**esponsibilities, **C**ollaborations

Originally introduced by Kent Beck and Ward Cunningham as an aid to getting non-OO programmers (in Tektronix) to "think objects".

Also useful for validating the chosen set of classes (or class model) against the required behaviour (or use case model).

CRC cards are an aid to clear thought, not a formal part of the design process – though UML does permit you to record the information from them in the class model, if you wish.

Examples

LibraryMember	
Responsibilities	**Collaborators**
Know what copies are currently borrowed	
Meet requests to borrow and return copies	Copy

Copy	
Responsibilities	**Collaborators**
Know what Book this is a copy of	
Inform corresponding Book when borrowed/returned	Book

C, R and C

Class: a well chosen name capturing the essence of the class

Responsibility: what services is this class supposed to provide?
(Perhaps at a more abstract level than operations; check for
coherence and cohesion.)

Collaborators: what services does this class need in order to fulfil
its responsibilities? (Again, at a more abstract level than message
passing: may leave protocol undecided, but check for feasibility
and coupling.)

How to use CRC cards (1)

1. Need a requirements document, or equivalent knowledge, before you start
2. Group of 5-6 people, including domain expert(s)
3. Work on a "reasonable size" part of the problem (subsystem?)
4. Brainstorm possible classes
5. Discuss and filter to likely set of candidates
6. Share the classes between the people
7. Each person writes a card for the class(es) they've been assigned: name on the front, short precise description on the back
8. Read out descriptions to make sure everyone understands
9. Add the totally obvious responsibilities and attributes, only
10. Start playing scenarios...

How to use CRC cards (2)

Designate a scribe (optional, but usually advisable)

Pick a scenario. It can be end-to-end or an "inside" behaviour – must involve some collaboration!

Make it really specific. E.g. consider what happens when "Perdita Stevens, who has no outstanding fines and nothing else on loan, returns *Using CRC Cards* by Nancy Wilkinson".

Decide where does the initial request comes in. Does that class have an appropriate responsibility? If not, add one. Owner holds up that card.

What help does this object need to carry out that responsibility? Check or add collaborator.

Does the collaborating class have an appropriate responsibility?

Points to note

When there's a choice, consider trying it both ways.

Expect to make mistakes and need to change things.

Keep it simple.

From CRC cards to sequence diagrams

Straightforward: a CRC card scenario can be recorded directly in a sequence diagram and the logic of the game takes care of the message direction and causality.

Be careful if more than one object of the same class is involved.

Refinements of CRC card use

Some people like to use more than the basic C, R, C, e.g. showing:

- sub- and super-classes under the class's name;
- emerging attributes and other parts on the back of the card;
- a concise definition of the concept represented by the class on the back of the card.

Yes, there are computer-based CRC card tools. But in fact there's value in using the physical cards.

Summary

These slides discussed two ways of getting from:

- ▶ a clear understanding of the functional requirements of the system, expressed as use cases; and
- ▶ a conceptual class model

to an allocation of behaviour to the classes that works with both.

From your reading and the labs you know how to document this behaviour with sequence, state and activity diagrams.