Information Systems Concepts

Specifying Operations

Roman Kontchakov
Birkbeck, University of London

Based on Chapter 10 of Bennett, McRobb and Farmer:
Outline

- Specifying Operations
  - Section 10.4 pp. 295–304
Why we specify operations

- From analysis perspective:
  - ensure users’ needs are understood

- From design perspective:
  - guide programmer to an appropriate implementation (i.e., method)

- From test perspective:
  - verify that the method does what was originally intended
Types of operations and their effects

- Operations without side-effects are pure queries that
  - request data but do not change anything
  - carry out calculations

- Operations with side-effects may
  - create or destroy object instances
  - set attribute values
  - form or break links with other objects
  - send messages or events to other objects
  - any combination of these
Services among objects

- When objects collaborate, one object typically provides a service to another for example,
  - A Client object might ask a Campaign object for its details
  - The same Client object might then ask a boundary object to display the related campaign details to the user
Contracts: an approach to defining services

- A service can be defined as a contract between the participating objects.
- Contracts focus on inputs and outputs.
- Intervening process is seen as a black box.
- Irrelevant details are hidden.
- This emphasizes service delivery, and ignores implementation.
Contract-style operation specification

- intent or purpose of the operation
- operation signature, including return type
- description of the logic
- other operations called
- events transmitted to other objects
- any attributes set
- response to exceptions (e.g., an invalid parameter)
- non-functional requirements
Types of logic specification

- Logic description is probably the most important element.

- Two main categories:
  - *algorithmic* specifications are white box — they focus on how the operation might work.
  - *non-algorithmic* specifications are black box — they focus on what the operation should achieve.
Non-algorithmic techniques

- appropriate where correct result matters more than method to arrive at it

- decision trees:
  complex decisions, multiple criteria and steps
  (not described further here)

- decision tables:
  similar applications to decision tree

- pre- and post-condition pairs:
  suitable where precise logic is unimportant or uncertain
## Decision tables: example

<table>
<thead>
<tr>
<th>Conditions and actions</th>
<th>Rule 1</th>
<th>Rule 2</th>
<th>Rule 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Conditions</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Is budget likely to be overspent?</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Is overspend likely to exceed 2%?</td>
<td>–</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td><strong>Actions</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No action</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Send letter</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Set up meeting</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Pre- and post-condition pairs

`CreativeStaff.changeGrade(gradeObj, gradeChangeDate)`

**pre-conditions:**
- `creativeStaff` object is valid
- `gradeObj` is valid
- `gradeChangeDate` is a valid date
- `gradeChangeDate` is greater than or equal to today’s date

**post-conditions:**
- a new `staffGradeObj` exists
- new `staffGradeObj` is linked to the `creativeStaff` object
- new `staffGradeObj` is linked to the previous one
- value of previous `staffGradeObject.gradeFinishDate` set equal to `gradeChangeDate – 1 day`
Algorithmic techniques

- suitable where users understand the **procedure** for arriving at a result
- can be constructed top-down, to handle arbitrarily complex functionality
- examples:
  - Structured English
  - Activity Diagrams
Algorithmic techniques:
Structured English

- commonly used, easy to learn
- three types of control structure, derived from structured programming:
  - sequences of instructions
  - selection of alternative instructions (or groups of instructions)
  - iteration (repetition) of instructions (or groups of instructions)
Sequence in Structured English

each instruction executed in turn, one after another

get client contact name

sale cost = item cost * (1 - discount rate)

calculate total bonus

description = new description
Selection in Structured English

one or other alternative course is followed, depending on result of a test:

if client contact is 'Sushila'
    set discount rate to 5%
else
    set discount rate to 2%
end if
Iteration in Structured English

instruction or block of instructions is repeated
 can be a set number of repeats or until some test is satisfied:

```
do while there are more staff in the list
    calculate staff bonus
    store bonus amount
end do
```

```
repeat
    allocate member of staff to campaign
    increment count of allocated staff
until count of allocated staff = 10
```
Algorithmic techniques: Activity Diagrams

- are part of UML notation set
- can be used for operation logic specification, among many other uses
- are easy to learn and understand
- have the immediacy of graphical notation
- bear some resemblance to old-fashioned flowchart technique
Activity Diagram: example

Use Case: check campaign budget

get Client → show Campaign

show Campaign
- [incorrect Campaign]
- [correct Campaign]

get Advert cost

calculate overheads

[no more Adverts]
[more Adverts]
Object Constraint Language

- Most OCL statements consist of: Context, Property and Operation

- Context
  - defines domain within which expression is valid
  - instance of a type, e.g. object in class diagram
  - link (association instance) may be a context

- A property of that instance
  - often an attribute, association-end or query operation
OCL operations

- Operation is applied to the property
  - arithmetical operators *+, – and /
  - set operators such as size, isEmpty and select
  - type operators such as oclIsTypeOf
OCL expressions: examples

- **context** Person
  
  self.gender

  In the context of a specific person, the value of the property ‘gender’ of that person.

- **context** Person
  
  **inv:** self.savings >= 500

  The property ‘savings’ of the person under consideration must be greater than or equal to 500.

- **context** Person
  
  **inv:** self.husband->notEmpty() **implies**

  self.husband.gender = Gender::male

  If the set ‘husband’ associated with a person is not empty, then the value of the property ‘gender’ of the husband must be male.
OCL expressions: examples (cont.)

- **context** Company  
  **inv:** self.CEO->size() <= 1

  The size of the set of the property `CEO` of a company must be less than or equal to 1.

- **context** Company  
  **inv:** self.employee->select(age < 60)->notEmpty()

  The set of employees of a company whose age is less than 60 is never empty.
context CreativeStaff::changeGrade
    (grade:Grade, gradeChangeDate:Date)

pre:
    grade oclIsTypeOf(Grade)
    gradeChangeDate >= today

post:
    self.staffGrade->exists() and
    self.staffGrade[previous]->notEmpty() and
    self.staffGrade.gradeStartDate = gradeChangeDate and
    self.staffGrade.previous.gradeFinishDate =
    gradeChangeDate - 1 day
Take Home Messages

- The role of operation specifications
- What is meant by ‘Contracts’
- Algorithmic and non-algorithmic techniques, and how they differ

How to use:

- Decision Tables,
- Pre- and Post-condition pairs,
- Structured English,
- Activity Diagrams and
- Object Constraint Language