Teach Z by Reverse Engineering Specifications From Real-Life Implementations

Ian Bayley,
School of Technology,
Oxford Brookes University

ibayley@brookes.ac.uk

Abstract

This paper describes a different approach to teaching formal methods and Z in particular. A specification is developed interactively with the class but unusually, a working implementation exists and is made available for exploratory testing as part of the lecture. The domain chosen for the application is a simple dynamic content website.

Keywords: formal methods, education, reverse engineering, website specification

1. INTRODUCTION

One challenge in the teaching of formal specification is to give the student confidence that the technique can be applied to sizeable problems in the real world. Perhaps the huge gap between specification and implementation is one reason for this doubt. If so, then the problem can be ameliorated with lightweight formal methods, where the two co-exist harmoniously. But for those of us who still wish to teach a high-level unexecutable specification language like Z[1], this paper provides a new way to bridge the gap. Instead of turning specifications into code just to show it can be done, a lengthy tedious task that would add much supporting non-functional code, why not take real-life working software with simple observable behaviour and, without exposing the code, show that this behaviour can be specified formally. If every aspect of the behaviour is captured then you would have shown that formal specification could have been used to produce the artefact the students see before them.

The example case study considered here is the CMS (Computing and Mathematical Sciences) website used to hold the teaching materials for most of our Computing modules at Oxford Brookes University. Students can only view the uploaded documents but lecturers can add, edit and delete them as well. So CMS does nothing more than provide a view of a database together with operations for manipulating it. Here now is a lecture plan for developing a Z specification of CMS interactively with your students, a plan that you can easily adapt to the dynamic content (eg PHP/MySQL) website of your choice.

2. LECTURE PLAN

Start by showing the students around the website for a minute or so, including the lecturer view which they have not seen before. Ask them to spot and shout out as many as possible of the operations supported. Answers will include operations to view, add, delete and update the lectures, practicals, assignments etc. In this way, they have just identified all the derived types that they will need. The next step is to define these types in terms of the basic and free types. Lecturers appear to be indexed not only by module no. and week no. but also by semester no., since some core modules run twice a year to let the students who failed the module retake it.

This will be the domain of a partial function, the range of which, *LECTINFO*, is constructed by noting all the other spaces on the web form for editing a lecture, below.

The screenshot shows a web form titled "Update Lecture" with the following fields:

- Module Number:** A dropdown menu with "U08039" selected.
- Semester:** A dropdown menu with "2" selected.
- Week:** A dropdown menu with "8" selected.
- Date:** Three dropdown menus for day, month, and year, showing "7", "April", and "2006" respectively.
- Time:** Two dropdown menus for hour and minute, showing "09" and "00" respectively.
- Location:** A text input field containing "unspecified".
- Lecturer:** A dropdown menu with "Mr David Lightfoot" selected.
- Lecture Title:** A text input field containing "Eurovision".
- Check the box to remove a file:** A checkbox followed by the text "Eurovision".

2.1. A neat specification

So some of the types needed are as follows.

$[STRING, MODULENO, SEMESTER, WEEK, DATE, TIME, LECTURER, FILE]$

$ROOM == STRING$

$TITLE == STRING$

$URL == STRING$

$WEBLINK == TITLE \times URL$

$NOTES == TITLE \rightarrow FILE$

$LECTINFO == DATE \times TIME \times ROOM \times LECTURER \times TITLE \times WEBLINK$

It is worth pointing out now, to anticipate a common distraction, that some types like *DATE* and *TIME* are being left deliberately opaque at this point, even though they can easily be refined, simply because the details are not relevant for the rest of the specification.

Now give the students the exercise of defining types analogous to *LECTINFO* for announcements, practicals, assignments, links, past exam papers and resources. The type *INTRO*, used to

represent the introduction to each module, specifies both the module leader and the lecturers for the module.

$$INTRO == TITLE \times DEPT \times TEXT \times LECTURER \times \mathbb{P} LECTURER$$

As an aside, it is unclear whether the set of lecturers should include or exclude the module leader. In any case, no such checking is done.

Given these types, we can represent the information about modules as a function from *MODULENO* to the type

$$INTRO \times \text{seq } ANNOUNCEMENT \times \mathbb{P}(WEBLINK \times TEXT) \times \mathbb{P} NOTES \times \mathbb{P} RESINFO \times \\ (SEMESTER \times WEEK \leftrightarrow LECTINFO \times NOTES \times PRACINFO \times NOTES) \times \\ (SEMESTER \times \mathbb{N}_1 \leftrightarrow ASSIGINFO \times NOTES)$$

2.2. The not-so-neat specification

Unfortunately, specifying operations for this datatype is a little tricky. Editing a lecture, for example, involves extracting a data item of type *LECTINFO*, modifying it, and replacing it. This operation is much easier to specify if the information is stored in several functions as below:

CMS

```

modules :  $\mathbb{P} MODULENO$ 
intros :  $MODULENO \rightarrow INTRO$ 
announcements :  $MODULENO \rightarrow \text{seq } ANNOUNCEMENT$ 
lectures :  $MODULENO \times SEMESTER \times WEEK \rightarrow LECTINFO$ 
lectureNotes, practicalNotes :  $MODULENO \times SEMESTER \times WEEK \rightarrow NOTES$ 
practicals :  $MODULENO \times SEMESTER \times WEEK \rightarrow PRACINFO$ 
assigs :  $MODULENO \times SEMESTER \times \mathbb{N}_1 \rightarrow ASSIGINFO$ 
assigNotes :  $MODULENO \times SEMESTER \times \mathbb{N}_1 \rightarrow NOTES$ 
links :  $MODULENO \rightarrow \mathbb{P}(WEBLINK \times TEXT)$ 
papers :  $MODULENO \rightarrow \mathbb{P} NOTES$ 
resources :  $MODULENO \rightarrow \mathbb{P} RESINFO$ 

```

```

dom lectures = dom lectureNotes
dom practicals = dom practicalNotes
dom assigs = dom assigNotes
dom intros  $\cup$  dom announcements  $\cup$  dom links  $\cup$  dom papers  $\cup$  dom resources  $\subseteq$  modules
{ m : MODULENO |  $\exists s : SEMESTER; n : \mathbb{N}_1 \bullet (m, s, n) \in \text{dom assigs}$  }  $\subseteq$  modules
{ m : MODULENO |  $\exists s : SEMESTER; w : WEEK \bullet$ 
  (m, s, w)  $\in$  dom lectures  $\cup$  dom practicals }  $\subseteq$  modules

```

The longwinded predicate is now needed to ensure that no information is stored for some modules and not for others. The lecture notes are kept separately from the rest of the lecture information to make the operations easier to write.

Now we can define a few basic operations, the first of which *GetLecture* is used by the script both to display the lecture in the student view and also to fill the form with values for each of the fields, every time the link for Add a Lecture or Edit a Lecture is clicked.

GetLecture $\exists \text{CMS}$ $m? : \text{MODULENO}$ $s? : \text{SEMESTER}$ $w? : \text{WEEK}$ $l! : \text{LECTINFO}$ $n! : \text{NOTES}$ $(m?, s?, w?) \in \text{dom } \textit{lectures}$ $l! = \textit{lectures}(m?, s?, w?)$ $n! = \textit{lectureNotes}(m?, s?, w?)$ AddLecture ΔCMS $m? : \text{MODULENO}$ $s? : \text{SEMESTER}$ $w? : \text{WEEK}$ $l? : \text{LECTINFO}$ $n? : \text{NOTES}$ $m? \in \textit{modules}$ $(m?, s?, w?) \notin \text{dom } \textit{lectures}$ $\#(\text{dom } n?) \leq 3$ $\textit{lectures}' = \textit{lectures} \oplus \{((m?, s?, w?) \mapsto l?)\}$ $\textit{lectureNotes}' = \textit{lectureNotes} \oplus \{((m?, s?, w?) \mapsto n?)\}$ DeleteLecture ΔCMS $m? : \text{MODULENO}$ $s? : \text{SEMESTER}$ $w? : \text{WEEK}$ $(m?, s?, w?) \in \text{dom } \textit{lectures}$ $\textit{lectures}' = \{ (m?, s?, w?) \} \triangleleft \textit{lectures}$ $\textit{lectureNotes}' = \{ (m?, s?, w?) \} \triangleleft \textit{lectureNotes}$

Point out that the pre-conditions are enforced by the web interface. In the case of *GetLecture* for example, the week is selected on the Admin page for that module, from a drop-down list assembled from the domain of *lectures* alone. The size pre-condition $\#(\text{dom } n?) \leq 3$ for *AddLecture* is a consequence of there only being three spaces on the form through which files can be uploaded. To upload more files, the user must select Edit a Lecture where, on an almost identical form, (s)he can not only specify up to three more files at a time, but also click any number of files that (s)he does not want any more.

EditLecture ΔCMS $m? : \text{MODULENO}$ $s? : \text{SEMESTER}$ $w? : \text{WEEK}$ $l? : \text{LECTINFO}$ $n? : \text{NOTES}$ $ts? : \mathbb{P} \textit{TITLE}$ $(m?, s?, w?) \in \text{dom } \textit{lectures}$ $\#(\text{dom } n?) \leq 3$ $\textit{lectures}' = \textit{lectures} \oplus \{((m?, s?, w?) \mapsto l?)\}$ $\textit{lectureNotes}' = \textit{lectureNotes} \oplus \{((m?, s?, w?) \mapsto ts? \triangleleft (\textit{lectureNotes}(m?, s?, w?) \oplus n?))\}$

2.3. The even-less-neat implementation

As a matter of fact, the act of specification uncovers a surprising number of possible bugs in the implementation itself. The first of these is spotted when we consider the domain membership test in the operation *AddLecture*. In reality, no such check is made and the user can easily end up with two versions of the lectures for each week. Perhaps more surprisingly, the module number itself is changeable on the form so a lecture can disappear from one module and reappear in another. So, although we do not have the actual code to hand, we can hypothesise that the actual representation of the variables *lectures* and *lectureNotes* is something like this:

lectures : seq(*DATE* × *TIME* × *ROOM* × *LECTURER* × *TITLE* × *WEBLINK*)
lectureNotes : seq *NOTES*

If so, the database tables would most likely contain an auto-increment numeric primary key that need not be specified by the SQL INSERT commands.

When adding a lecture, the lecturer is specified by selecting his name from a drop down box, but this is taken from the list of all the lecturers available rather than the list of those teaching the module. Also not included is basic timetabling information, and since the date is in the format dd/mm/yy without the weekday specified, it is very easy to miscalculate the day number and cause confusion to the students.

As a matter of fact, it would be more succinct to say simply that there is no data integrity checking at all, either client-side or server-side. So much can be done to improve the software. Not only can Z be used to specify these improvements but it could also have been used to get the system right in the first place.

3. OBSERVATION

Reverse engineering to a specification is a common activity in industry, typically undertaken to improve documentation. The specification in question is nearly always an informal one though, but with instructive examples like this one and those of the ESPIRIT REDO project [2], there is no reason why this should be the case any more.

4. FUTURE PLANS

The lecture described above has not been delivered yet to its target audience of mostly second-year students so there is no success or failure to report. However, the hope is that some of the students will become more adventurous in their courseworks, where they must specify a system of their choice in groups. Since the application described above is similar to that of some distinctly unambitious final-year projects, it is just conceivable that Z may be applied here too, or even in industry, in compliance with the ultimate wish of us all.

5. ACKNOWLEDGEMENTS

I would like to thank David Duce and my partners of the Formal Specification module, Clare Martin and David Lightfoot, for their willingness to discuss with me their thoughts on this particular topic and on education in general. I also appreciate the diligence of the anonymous reviewers, especially the one who came back with a whole page of thought-provoking comments.

REFERENCES

- [1] Mike Spivey (1989) *The Z Notation: A Reference Manual* Prentice Hall
- [2] HJ von Zuylen *The REDO Compendium: Reverse Engineering for Software Maintenance* John Wiley & Sons, 1993