Using Z as a Specification Calculus for Object-Oriented Systems

Anthony Hall
Praxis Systems plc

ABSTRACT

One of the useful features of the Z notation is that it offers a calculus for building large specifications from smaller components. So far most Z specifications have followed a single paradigm in which the system as a whole is treated as a state machine and parts of the specification define parts of the state and operations on these parts. A more recent paradigm for system structuring is the object-oriented approach, in which the system is divided into objects each of which has its own set of operations. We have built a system based on the object-oriented approach and we have used Z to specify it. This paper reports on the methods we used to develop an object-oriented Z specification, defines some extensions to the Z library which we developed and suggests some conventions and extensions which would help to make such specifications more straightforward.

1. INTRODUCTION

We have recently developed a substantial piece of software using formal specification in Z in conjunction with implementation in an object-oriented language, Objective C. In carrying out this development, we used the object-oriented approach from an early stage to guide our overall system architecture and the structure of the specification. When used in this way, the object-oriented approach leads to a way of thinking about the specification which is different from the way that Z specifications are usually written. We therefore had to develop new conventions for using Z in this context. We found that the combination of an object-oriented approach with formal specification was very effective, and this paper therefore reports on the methods we used so that others who wish to use this combination of techniques can learn from our experience.

1.1 Z

The language Z is a notation for writing specifications. It is based on typed set theory and first order logic. In addition it contains a number of constructs for structuring specifications, notably the schema. The use of schemas offers a calculus of specifications, whereby specifications of large systems can be built up from smaller parts. The language has been developed over a number of years largely through case studies and industrial experience, and during this development a number of conventions have been developed for using the schema calculus to build up specifications. Most of these case studies have used a common paradigm: the system has been viewed as a state machine. The schema calculus has been used to build up a model of the internal state in terms of more or less separate components; operations have been defined in terms of changes to the overall state, and again the schema calculus has been used to specify these operations by merging
operations on the components of the state.

The existence of this calculus is one of the main distinguishing features of Z and it is enormously important in the practical development of large scale specifications. First, it makes the job of the specifiers easier by allowing them to develop specifications incrementally. Second, it makes large Z specifications relatively easy to read, by structuring them into comprehensible units. Third, it helps with the development of proofs and possibly refinement steps in a structured way.

1.2 The Object-Oriented Approach

The object-oriented approach to system design is becoming increasingly popular, and we believe that this popularity is based on sound technical reasons. The fundamental characteristics of the object-oriented approach are:

- The structure of a system is based on the major objects which are of interest.
- The behaviour of each object is defined as a set of operations whose implementation is internal to the object.
- The behaviour of the system as a whole is defined in terms of the behaviour of its component objects.

In addition, a number of other techniques are commonly associated with the object-oriented approach, although not a necessary part of it. These techniques are primarily concerned with classifying the objects in the system. Typically, objects are grouped into classes, where all instances of a class have a common behaviour. Commonly these classes themselves are related to each other in an inheritance hierarchy, whereby members of a particular class have all the properties of the superclass as well as any properties special to their own class.

The object-oriented approach offers one of the most promising ways of structuring a system in a way which increases cohesion within its parts and reduces coupling between them. It is therefore important that the specification should be able to reflect this structuring.

1.3 Z and Object-Orientation

In this section we summarise the similarities and differences between the conventional way of structuring Z specifications and the way that the object-oriented approach would suggest. This leads to our objectives in bringing the techniques together.

The conventional method of building up a Z specification has a lot in common with the object-oriented approach. For example, one of the well known case studies in Z is the CAVIAR system, described in Hayes’ book[1]. CAVIAR is a visitor administration system, and comprises a number of subsystems. Each subsystem is concerned with one aspect of visitor administration: there are subsystems for hotel reservations, transport reservations, conference room bookings and so on. Each subsystem is an instantiation of a generic resource-user system. The specification proceeds by:

1. Identifying the basic sets with which the specification is concerned. These are things like visitors, meetings, hotel rooms and so on.