Inheritance In Object Oriented Z

Elspeth Cusack
British Telecom
St Vincent House
Ipswich IP1 1UX
United Kingdom

Abstract

The Z notation for the formal specification of software and systems is based on set-theory and first-order predicate calculus and is widely used and understood. Recent research has demonstrated that by extending the notation to include the idea of a class schema grouping together a state schema and the operation schemas referring to it, an object oriented specification style can be supported.

This paper introduces the concept of inheritance into object oriented Z in two ways. First of all, it defines derivation, an incremental inheritance technique for expressing new class schemas in terms of old ones. Secondly, it characterises subtyping inheritance as a technique for the hierarchical classification of objects. Thus derivation in Z is a basis for reusing existing specification modules. Subtyping inheritance is the basis for considering the substitutability of one class for another.

1 Introduction

There has been for some time considerable research interest in formalising the object oriented approach, using λ-calculus (for example [4, 6]) or the theory of abstract data types (for example [14]). Recently there have been various attempts to develop object-oriented styles of using pre-existing formal specification languages such as CSP [16, 7], LOTOS [21, 18, 8, 9] and Z [20, 5, 11, 12]. This latter work has been strongly motivated by the need for modelling and specification techniques capable of rigorously describing international standards for large scale open distributed systems [22, 23].

Objects, interpreted as locally-understandable communicating modules of specification, have a natural affinity with distributed systems. Object oriented specification has intuitive appeal, and offers the potential benefits of modularity, flexibility and reusability. (These attributes are as attractive to
communication system designers as they are to programmers.) Formal specification in the language of choice provides a fixed notation with implementation-independent semantics, and involves the development of a mathematical model of the system about which we can reason. Formal specification and object orientation seem in short to have complementary strengths, and therefore to be suitable for combination.

This paper contributes to the development of object oriented Z by introducing two inheritance techniques representing different ways of strengthening the 'is-a' relation. The techniques are derivation (section 6), an incremental inheritance technique for expressing new classes in terms of existing ones, and subtyping inheritance (sections 7, 8), a technique for the hierarchical classification of objects according to the substitutability of one class for another. The introduction of inheritance to object oriented Z is made possible by an enrichment of usual typing in Z (sections 4 and 5). Sections 2 and 3 set the scene with background information on object oriented Z and inheritance.

The Z notation has been developed over the past twelve years for the specification of systems and software, and is widely understood and used [1, 15]. Z is based on set-theory and first order predicate logic. Familiarity with the concepts and notation of Z is assumed in this paper, and can be easily acquired from Spivey's book [20].

A style of program development from specifications in languages such as Z is presented in [17].

2 Object oriented Z

A Z specification consists of modules called schemas, which may be linked by an informal commentary. Each schema consist of a declaration part and a predicate (which may be empty) over the declared variables. Schemas describe static aspects such as the states the system can occupy, and the invariant relationships (if any) that are maintained as the system state changes. For example:

\[ \text{Invoice} \]
\[ \text{id : IDset; state : \{unpaid, paid\}} \]

A binding of a Z schema is an object (an abstraction of an implementation entity) with components named by identifiers corresponding to the variables declared in the schema. Thus a schema determines a set of bindings. In the example, the identifiers \text{id} and \text{state} are used to read the value of the corresponding component of each binding of the schema \text{Invoice}.

Different schemas describe dynamic aspects, such as the operations that can take place, in terms of the relationships between the inputs and outputs and the resultant changes of state. Here is the schema which explains what happens when an invoice is paid: