FORMAL SPECIFICATION AND OBJECT-ORIENTED DESIGN

Mary BUCHANAN and Carol BRITTON

School of Information Science, Hatfield Polytechnic,
College Lane, Hatfield, Herts, AL10 9AB, UK2

In recent years, object-oriented design and formal specification languages have become increasingly important in the development of software systems. In this paper we use the formal specification languages OBJ1 and OBJ3 to investigate the extent to which they support object-oriented design in general and inheritance in particular.

Keywords: Formal Specification, Inheritance, OBJ

1. INTRODUCTION

The paper describes the results of applying object-oriented concepts to the formal specification of a system.

The purpose of the research out of which this work has arisen is to discover the potential of using an object-oriented approach throughout the development of a system. Underlying the object-oriented paradigm is the concept of classes of objects organised into hierarchies which reflect client and inheritance relationships [1]. We aim to determine the degree to which a formal specification can support such class structures. The ultimate purpose is to ascertain whether such class structured specifications would form a sound foundation for systems which are to be designed and implemented in an object-oriented manner. In addition, we wish to consider the viability of producing specifications which are easily maintainable and potentially reusable. In this paper we investigate the extent to which the formal language OBJ can be used in an object-oriented style.

2. THE CASE STUDY: AN APPOINTMENT SYSTEM FOR LECTURERS

The case study on which this exercise is based can be summarised as consisting of lecturers who have timetabled teaching commitments and other more randomly organised appointments. A system is required to manage lecturers' appointments and to provide information concerning both appointments and teaching commitments.

3. INITIAL ANALYSIS

We used and freely adapted the early stages of JSD to capture the data objects in the system in terms of both their attributes and their actions. Entity structure diagrams [2] were derived for these data objects. Subsequently, such diagrams were used to illustrate the overall functionality required of the Appointment System.

At this stage we had derived a diagrammatic specification which was object-based. However, the specification lacked formality and did not provide a model of inheritance. We next turned to the formal specification language OBJ in an attempt to address these problems.

4. THE FORMAL SPECIFICATION: OBJ1

OBJ1 [3] is a functional language which uses equational algebra to define abstract data types and the algorithms which manipulate them. Our intention was to use OBJ1 to define the classes of objects which are used to capture the common characteristics which objects of the class share. A class can be viewed as abstract in the sense that the data it describes is not instantiated and it can be defined by its operations alone. Since an abstract data type is a set of values of a type characterised by the operations which can be performed on the type, an abstract data type can be used to define a class of objects.

The JSD analysis had identified Diary, Lecturer and Timetable as the data objects required by the system. These were to be the principal units (called OBJs in OBJ) in the formal specification. In order to build these OBJs we needed to consider their attributes as identified by JSD. In order to maximise their potential reuse we decided to specify each attribute in a separate OBJ. A typical example is the OBJ Time which was used in both the Diary and the Timetable and which could easily be imported into other OBJ specifications.

In OBJ it is necessary to define any OBJs before they are referenced by others; they must be specified before any objects which use them. This imposed a hierarchical structure on our formal specification.

At this point we had a specification which combined the advantages of formality with a clear modular structure. This structure reflected client relationships in the problem domain, such as Diary's use of AppointmentList and Name, but it failed to model any inheritance relationships. In the Appointment System we wanted to specify not only a general Time type, but also a TimeDiary type which inherited from Time and enriched it to define the constraints which were specific to the Appointment System. We were able to define the Time OBJ without difficulty. However, although we were able to enrich Time in a TimeDiary OBJ, we could not define a type TimeDiary to be type compatible with the type Time.

Our inability to capture this inheritance adversely affected the OBJ specification in two ways. First, we lost the advantages of a conceptually simple mapping to an object-oriented design. Secondly, the specification itself suffered from a lack of clarity. Elsewhere in the specification, the failure to model inheritance resulted in a complexity of OBJ code which masked the simple relationships between data objects.

5. REFINING THE SPECIFICATION: OBJ3

OBJ3 [4] is based on order-sorted algebra which enables a user to declare that a type (sort in OBJ) is included in another type and so provides a formal logic for inheritance or 'is-a' hierarchies. In order to determine how well OBJ3 can model inheritance, we wrote a Time OBJ to correspond with our OBJ1 version and a TimeDiary OBJ such that Time was declared as a subtype of TimeDiary. As a consequence we would expect to be able to use a TimeDiary type wherever we use a Time type, but not vice versa. We declared TimeDiary values such as oneHr30Min in order to be able to distinguish these from Time values represented as, for example, oneHr.

An abbreviated version of test results, run on the OBJ3 interpreter, is shown in Table 1.

TABLE 1: OBJ3 TEST RESULTS

```
1      OBJ> reduce oneHr >Hr twoHr .

       reduce in TIMEDIARY : oneHr >Hr twoHr
       result Bool: false

2      OBJ> reduce in TIME : oneHr >Hr twoHr .

       reduce in TIME : oneHr >Hr twoHr
       result Bool: false

3      OBJ> reduce in TIME : isEndDay(sixHr) .

       No successful parse for the input:
       result Err: err!!(isEndDay ( sixHr ))

4      OBJ> reduce in
       TIMEDIARY : isEndDay(sixHr) .

       reduce in TIMEDIARY : isEndDay(sixHr)
       result Bool: false

5      OBJ> reduce isEndDay(oneHr30Min) .

       reduce in TIMEDIARY : isEndDay(oneHr30Min)
       result Bool: false
```

In OBJ3, expressions denoted by "reduce <Exp> ." are evaluated in the context of the last module entered into the system. Test 1 demonstrates that the >Hr operation, which was defined in the TIME module, can be used in the context of the TIMEDIARY module; in other words, TimeDiary has inherited the operation from Time. By using the expression "reduce in <Mod> : <Exp>", one can nominate a different module as the context of the evaluation. Thus Test 2 shows the >Hr operation working in the context of the TIME module.

The operation isEndDay is specified in TimeDiary such that it takes a Time and returns a boolean result (isEndDay : Time -> Bool). Test 3 shows that one cannot use isEndDay in the context of the TIME module because the operation is not defined in this module. However, test 4 shows the same test run successfully in the TIMEDIARY context. Test 5 illustrates that, provided we are in the TIMEDIARY context, a TimeDiary type (here oneHr30Min) can be used wherever a Time type has been specified.

We developed another version of TimeDiary in which we redefined the +Hr operation so that it behaved differently from the +Hr operation in Time. Testing showed that each version worked within the context in which it had been defined.
OBJ3 can successfully emulate the behaviour of inheritance. We can define new types as variations of old types and we can inherit all the operations of the old types; the old types themselves remain unchanged. We can add extra operations to the new type and we can redefine operations inherited from the old type.

6. CONCLUSIONS

6.1 Formality

The OBJ specification exhibited the advantages that come from using a formal notation. There were other advantages which arose from the fact that the specification was executable, however, a discussion of these is outside the scope of this paper.

6.2 Units of Modularity

The OBJ modules correspond well to the classes in an object-oriented system in that OBJ modules can be used both to define the classes of objects which model the data in the system and to define the classes needed to provide the overall functionality required of the system.

The hierarchical structure of the module dependency in an OBJ specification can be used to make explicit the client relationship between classes of objects. Inheritance can be specified clearly and simply in OBJ3, but not in OBJ1.

6.3 Reusability

The OBJ data modelling modules can be combined together in different ways to achieve different specifications. We could provide additional functionality for the existing Appointment System which could be achieved without alteration to any of the existing classes in the system. We could also use fundamental classes such as Time and Date to create completely new systems and we could reuse aggregates of modules such as the Diary module since an OBJ module, together with any modules which it imports, provides a reusable package.

REFERENCES

[1] Meyer, B., Object-Oriented Software Construction (Prentice Hall, London, 1988).

[2] Jackson, M., System Development (Prentice Hall, London, 1983).

[3] Walter, C.D., Gallimore, R.M., Coleman, D. and Stavridou, V., UMIST OBJ1 Manual Version 1.0 (Department of Computation, UMIST, Manchester,1986).

[4] Goguen, J. and Winkler, T. Introducing OBJ3 (SRI International, SRI-CSL-88-9,1988).