28

Can a flat notation be used to specify an OO system: using Z to describe RM-ODP constructs

D.R. Johnson
Department of Defense, R2SPO
9800 Savage Road
Ft. Meade, MD 20755-6000, USA
drj@tycho.ncsc.mil

H. Kilov
IBM T J Watson Research Center
30 Saw Mill River Road
Hawthorne, NY 10532, USA
kilov@watson.ibm.com

Abstract

We discuss the general advantages of formal specifications and the particular importance of the Z specification language and the fundamental concepts in the Reference Model for Open Distributed Processing, both of which are in the process of standardization by ISO. After discussing some of the difficulties involved, we offer suggestions on how Z can be used successfully for specifying and modeling open object-based systems.

Keywords

Reference Model for Open Distributed Processing (or RM-ODP), General Relationship Model (or GRM), Z, fundamental concepts

1. SPECIFICATIONS: RIGOROUS AND FORMAL

To build a system that satisfies users' needs, we have to understand these needs and precisely specify them. These specifications should be understandable to both the users (subject matter experts) and developers. In other words, they have to serve as a contract for building a system. In more technical terms, the semantics of the system's behavior has to be precise and explicit. Moreover, it also has to be abstract, i.e., should not contain unnecessary implementation details. Otherwise, the implementation code by itself — which precisely and explicitly specifies the system's behavior — would certainly solve the problem...

Definitions of behavioral semantics can be expressed in a "structured" natural language — to absorb the shock of their introduction! — , but, due to the inevitable ambiguity of a natural language, we should have a precise specification in a notation based on mathematics to rely upon in cases of doubt. Specifications in English are usually insufficient: they are incomplete (e.g., do not contain information "evident" to the subject matter experts), ambiguous, or both. Thus, a formal specification may (have to) be referred to in cases of doubt or misunderstanding; and "the very use of a formal notation enforces developers to think and express themselves precisely" (Mac an Airchinnigh et al., 1994). This need to express a specification formally often leads to non-trivial questions that expose the knowledge of subject matter experts (SMEs) — the essence of requirements elicitation! — in a way understandable to the developers. Therefore

E. Najm et al. (eds.), *Formal Methods for Open Object-based Distributed Systems*

the developers will not need to fill in the gaps often existing in informal specifications. Although a formal specification can be (almost) unambiguously translated into stylized English (and the result shown to the customer (ISO, 1995a; Kilov, Ross, 1994; Redberg, 1994)), the reverse is not true. If behavior is not understood and specified properly then its implementation cannot be checked for conformance, and the implementor often will have to create the specification (but all too often will not write it down explicitly!) because the one provided to him is too complex, ambiguous or incomplete. Moreover, in the absence of a complete and precise specification understanding the behavior of an existing system will be possible only by looking into its implementation (application code). This is often a quite formidable task.

As Nancy Leveson recently observed, "There seems to be an assumption that users can find errors in natural language specifications. In my experience with English requirements specifications of several hundred pages for complex systems, this is not possible. We have found that users can find errors much more easily in a formal specification that is readable by them than in a large natural language specification. So our goal should be to find formal specification languages that are readable by non-mathematicians." Indeed, a concept is not really understood until it is formally specified. Consider, e.g., the ever-present questions like "what is the difference between a subtyping and a composition?" — a correct answer provides precise, i.e., formal, specifications of both by explicitly formulating the invariants of these relationships (Kilov, Ross, 1994), and then, if need arises, applies these specifications to the particular application-specific construct. This approach, borrowed from programming (Dijkstra, 1976; Gries, 1981), has been successfully used in information modeling (Kilov, Ross, 1994) and is referred to in national and international standards (ISO, 1995b; ISO, 1995a).

This paper emphasizes semantics rather than syntax and problems rather than solutions (to quote D L Parnas, "This is in sharp contrast to other papers on 'formal methods', which provide precise specifications of syntax but give short shrift to the question of what information should be provided using that syntax."). In the spirit of (Parnas, 1995; Parnas, Madey, 1995), we do not pretend to provide a lot of completely original contributions. Rather, our goal is to help the open systems specifier to produce formal or at least rigorous specifications which will help to understand the system and which therefore will actually be used by all kinds of specification readers — from subject matter experts to designers to implementors.

2. THE IMPORTANCE OF Z AND ODP

A specification is written using some notation. It appears that Z (Nicholls, 1995) is one of the most appropriate formal specification languages: it is simple, elegant and general enough; it has been widely used on a variety of projects; it is in the process of being standardized by ISO; and its declarative approach is the appropriate one to specify requirements rather than solutions, i.e., to communicate intent rather than pseudo-code. Z provides a precise declarative specification of behavior, and the intent of an operation is precisely specified in its postcondition. (The context for operations is specified in the invariant.) The level of abstraction in a Z specification is not fixed: it may be adapted to the problem at hand. In fact, it may be helpful to have two or more specifications to deal with one problem. The first can be a high level specification stating requirements in terms most meaningful to the customer. Others would be less abstract and more oriented toward implementation. Generally, the connections between these several precise specifications can themselves be described precisely (e.g., using linking invariants (Hoare, 1994)). In this manner, it will be possible to check whether the implemented behavior confirms to the specified behavior.

Creating a good specification is far from trivial. It need not be done from scratch: appropriate existing concepts and constructs should be (discovered rather than reinvented and) reused. This need for reuse suggests that it would be desirable to consider the most general specifica-

tions as the first candidates for a reusable specification library. ("Abstractions are, of course, the ultimate in reusability!" (Mac an Airchinnigh et al., 1994).) To do that, we need at least rigorous specifications of this kind as a starting point. Fortunately, such rigorous specifications are standardized by ISO: the Reference Model for Open Distributed Processing (RM-ODP) (ISO, 1995b) and the General Relationship Model (ISO, 1995a) represent excellent examples.

The goal of open distributed processing (ODP) standardization is to provide and maintain the overall architectural framework for ODP for all business sectors. The RM-ODP provides a framework for standardization of ODP specifications, but permits a number of different approaches to their realization. The ODP standards describe arbitrary distributed processing systems. Many concepts in these documents are independent of distribution and are fully applicable for understanding and describing an architecture of any system. The underlying approach of these descriptions is object-oriented. Finally (but perhaps most importantly), these documents explicitly emphasize semantics and specify it in a rigorous manner.

No particular notations are recommended by ODP standards: the semantics of a reference model is independent of a notation used to represent this model. (In a good programming text concepts are presented independent of a particular notation used to express these concepts (Dijkstra, 1976); "Program into a language and not in it." (Gries, 1981).) The ODP fundamental concepts are rigorously defined in (ISO, 1995b) in "structured English", and the most important of them are interpreted in (ISO, 1995c) "in terms of the constructs of the different standardized formal description techniques", such as LOTOS, SDL, and Z. This multilingual approach to formal specifications is valuable in that different formal languages have different expressive strengths and weaknesses and can highlight different viewpoints.

3. THE BAD NEWS AND THE GOOD NEWS

It has been noticed by many that Z provides excellent facilities for writing concise, clear, and understandable specifications at different abstraction levels. Thus, a combination of the existing standardized rigorous specifications of concepts used in ODP (ISO, 1995b) and its formalization in Z would be an excellent basis for a reusable library for object-oriented specification development. However, the currently standardized Z (Nicholls, 1995) is not an object-oriented notation. In particular, Z often does not directly support abstraction (in particular, modularization) within a specification — one of the most important properties of an object-oriented specification (ISO, 1995b). Z provides for a "flat" specification, with some restricted structuring mechanisms (schemas and axiomatic definitions may be considered the most important ones). Thus, we "only" need to provide some guidelines for the specifiers...

We should be honest about the difficulties in such an endeavor. Within the broad Z community there is no clear consensus on how best to use Z in the Object Oriented (note the capital letters) way. The problem is not lack of proposals but lack of agreement. There are proposals to modify Z to accommodate OO principles and other proposals to handle issues in a certain way as a matter of convention, without changing Z to enforce those conventions. Additionally, different existing object models are based on different concepts, and this may lead to quite different specification approaches exemplified in different object-oriented extensions to Z, different "OO analysis" approaches, and so on. As an important example, the classical object model does, while the generalized object model does not, require a distinguished recipient for each operation (OODBTG, 1991; Kilov, Ross, 1994), leading to quite different specification (and implementation) approaches to collective behavior and relationships (ISO, 1995a). Within the OO community there is some appreciation of the possible benefits of formal specification languages but no large body of experience to provide guidance on how to use a language like Z.

While these difficulties are real, they should not lead us to paralysis. There is certainly enough general experience and clear thinking in both the Z and OO communities that we can

start working together to our mutual benefit. And problems due to different frameworks and existing knowledge base are almost non-existent: many languages and approaches have cultures that surround them, with Z and ODP being no exception, but active industrial usage of Z and ODP seems to be in their early stages.

There are a few pitfalls we should be aware of at the start. One is that, whether one is using a "natural" language like stylized English or a formal language like Z, it is very difficult to write a single clear precise definition of a very general concept. Clarity and precision **may** be at odds with generality. This is the same problem that confronts writers of dictionary definitions. Their usual solution is to have multiple definitions. The dictionary on this desk gives eight definitions for "composition", two of which are further subdivided. Even within a restricted subject area where a word has a technical meaning, it may be advantageous to use slightly different terms, each with a single precise meaning, rather than a general term with slightly different meanings in different contexts. On the other hand, the use of abstraction suggests that these definitions have something in common, and it's quite challenging to precisely specify exactly these commonalities, no more and no less. But it is even more difficult to provide a formal mathematical specification of such concepts as "composition", "action", "subtype", etc. [with reasonably clear semantics, but unclear signatures?] Obviously, the specifications in (ISO, 1995b) provide an excellent attempt to formulate rigorous specifications of these terms.

Simplicity and elegance are always desirable. Sometimes they are achieved due to conventions common to a particular (sub)culture. A danger with conventions is that a concept which is well understood in one culture (Z or ODP) may not be understood so well or may be understood differently in the other culture. There is a similar problem with technical terminology. The ODP concept of "type", for example, is not at all the same as the Z notion of "type" — these are the same names referring to quite different things. (Note that ODP defines (ISO, 1995b, Clause 12.1) a name as a term which, **in a given naming context**, refers to an entity!) Thus, there exist no "correct" notions of, e.g., "type" — each notion is valid within its context.

It is often the case that constructs which a knowledgeable person would recognize as being basically the same can be presented in various different ways in Z. The differences are sometimes a mark of different specifications tailored to serve different purposes. One document, for example, may be expressed in terminology familiar to managers and customers while another on the same topic may use terminology related to the machinery behind the interface ("different alphabets", discussed, e.g., in (Hoare, 1994)). Some differences, on the other hand, are more a reflection of the personal style and habits of the writers. It is important to note that the precision and clarity of the language usually makes it easy to see exactly how the notions of one specification document are related to the notions of another on the same topic.

4. OBJECT EXTENSIONS TO Z: OBJECTS OR MODULES?

Although the need to master the complexity of large specifications has been widely acknowledged, currently, for Z Notation, these mechanisms exist as optional informal conventions which are not always clearly described. This may lead to specifications quite difficult for human understanding: a reader may be lost in a specification of this kind. As encapsulation is not enforced with this approach, the specifications might become, during their "maintenance", quite unsafe. For example, the fact that new relationships among variables can be added anywhere (Alencar, Goguen, 1992) may lead to very serious problems in understanding and thus supporting semantic integrity, if these relationships are not appropriately structured. This important problem is not Z-specific: the RM-ODP permits specification changes because changes are inevitable. Acquiring and losing ODP object types — e.g. to model different viewpoints — is a very good example.

Many object extensions to Z have been invented to provide clear structuring constructs — on a higher level than traditional schemas — for a Z specification. However, most traditional object extensions to Z are based on legacy OO programming languages and thus are more oriented towards design than towards analysis (aka business specification development). Their underlying object model is classical: it is class- (and message-) based. Such a model makes formulating a business specification substantially more difficult because it requires the specifier to make unnecessary choices each time he describes a collection of interrelated things or an operation applied to this collection. The classical object model describes everything in terms of classes and messages between classes — whatever the term "class" means, and more often than not this meaning is quite different from the RM-ODP definition of a class —, and therefore any collection will have to be described in these terms if we use an object extension to Z based on such a model. This often leads to describing (and defining!) relationships in terms of attributes of interrelated things — an approach that makes the specification too complicated and, perhaps more importantly, needlessly obscures important aspects of specification semantics. It is well-known, however, that in business things are not isolated, and that therefore a specification should be able to describe collections of things in a natural manner (ISO, 1995a).

Obviously, the existing Z Notation does not have these problems: a relationship may well be defined as e.g. a Z relation the source and target of which need not be elementary. In addition, RM-ODP seems to acknowledge collective behavior and thus a generalized object model: an action is associated with **at least** one object and thus is not necessarily a message; a contract is defined as an agreement governing part of the collective behavior of a set of objects; enabled behavior is defined as behavior characterizing a set of objects; and so on. Thus, a "classical" object extension to Z may not be the best notation for specifying RM-ODP constructs!

It appears that the most important complexity taming construct to be added to basic Z is a parameterized module that describes a collection of related classes. A module groups together the state space schema (including the invariant), the initial state schema, and appropriate operation schemas; it also includes requirements on actual parameters that instantiate the module. This approach has been used, e.g., in OOZE (Alencar, Goguen, 1992): the external properties of a module are simpler than the sum of internal detail, leading to substantial simplification. As an example, consider the quite straightforward way to specify generic relationships such as dependency or composition (Kilov, Ross, 1994; ISO, 1995a) in a module-based extension of Object-Z sketched in (Kilov, 1993). Both the invariant and generic operations (like *Create a Dependent for a given Parent*) of such a relationship may be relatively easily specified in terms of a Z relation, without the need to determine which of the relationship participants (e.g. Parent or Dependent) should be defined first, and which of them should own the operations. This approach also permits distinguishing between potential and actual relationship participants. For example, it may be possible, by means of appropriately modeling actual object identities as a subset of the set of potential object identifiers, to specify the existence or non-existence (the postcondition for deletion) of an object instance.

But do we need to wait for OO extensions to Z to be widely accepted? It is possible to formulate guidelines and conventions for using Z Notation in specifying properties of an object-oriented system in general and the architectural semantics of an ODP system in particular. This is analogous to providing useful guidelines for representing "structured programming" concepts and constructs in such languages as Fortran (Gries, 1981).

5. Z AND LARGE SPECIFICATIONS

There are at least two ways of relating a Z specification and a set of ODP concepts. One approach is to take an existing Z specification and extract ODP concepts out of it. Another ap-

proach, used in this paper and proposed by ANSI ASC X3J21 Technical Committee, is to describe a way in which a given ODP specification or concept can be expressed in Z.

A typical Z specification describes a collection of interrelated objects, but the structuring units of a Z specification do not directly correspond to structuring units (objects) of an ODP specification. Z does not provide, by itself, such concepts as objects and encapsulation; neither does Z directly support modules the invariants of which have to be described in one structuring unit. Inferring which operations may affect a particular state schema could, at worst, require examining all operations; and new constraints on old data and operations could be introduced at arbitrary points of a specification. But clearly the writers of a specification determine how they choose to use the language. For their own benefit and that of the readers of the specification, they can certainly choose discipline over chaos. Conventions and practices can be stated and followed. Z has some well-known conventions involving schema names, the letters Δ and Ξ and variable decorations ', ?, and !. Similar — explicitly stated! — conventions, e.g., on schema names, variable names and their decorations can be used to remind the reader and writer of the structure and organization built into the specification.

As Z does not directly support encapsulation (essential for describing objects), an object can only be described by a collection of relevant fragments of a Z specification. These fragments should include: normalized state information (an object's invariant), a schema that describes the initialization of an object, and schemas that describe operations applied to one or more objects. The properties specified in the state schemas for the object represent the object's invariant and should be satisfied by any operation in which the object participates. An invariant may also be provided in an axiomatic description, so that an invariant of an object is not necessarily contained within one Z schema. Moreover, the object's invariant implies the pre- and postconditions of its operations in the operation schemas.

An invariant may refer to more than one object (and will therefore be a part of the definition of the appropriate collection of objects). In the same manner, an action can be performed by the cooperation of several objects (the specification of an action in Clause 8.3 of (ISO, 1995b) explicitly refers to "at least one object" with which an action is associated), and therefore a schema describing such an action needs to refer to several objects. Thus, the approach for specifying collective behavior provided by OOZE (Alencar, Goguen, 1992) and even by Base Z, is closer — in this aspect — to the need to write (and read) understandable business specifications: it does not require the specifier to supply an owner for each operation and for each collective state. For example, a composition can be described, in Z, as a relation between a composite and a set of components, together with appropriate invariants stating e.g. that the relation and its transitive closure is irreflexive; and each kind of composition can be described by means of appropriate predicates representing its invariant. In particular, the invariant defining a containment (Kilov, Ross, 1994; ISO, 1995a) — a hierarchical composition for which the existence of a component implies the existence of its composite — can be easily represented in (Base) Z. The concept of a distinguished owner of the invariant, or a distinguished owner of an operation is not needed for a business specification; and neither is it used in a Z specification developed using these guidelines.

Although undisciplined changes lead to bad specifications, changes by themselves are inevitable in open systems, and RM-ODP is quite explicit about that. In fact, the existence of changes differentiates open from closed systems (Wegner, 1995). Therefore a specification notation for open systems should permit changes, and Z seems to be appropriate for this purpose. In particular, it permits adding e.g. new constraints introducing new properties which will be conjoined with the global property. Obviously, this should be done in a disciplined manner — but such considerations are valid for any specification notation that permits changes.

How do we model an ODP type? In ODP, a type (e.g., of an object) is a predicate characterizing a collection (e.g., of objects). An object is of a particular ODP type if the predicate holds

for that object (ISO, 1995b, Clause 9.7). Thus, an ODP object may have several types. Some of these types may be dynamically acquired and lost, providing a natural way to specify business requirements (Harrison et al., 1996) without artificial restrictions imposed by legacy OO systems. For example, a person, or a business, or a charity can become a homeowner — i.e. acquire the "homeowner" type — and later cease to become a homeowner, and after that, again become a homeowner (of a different home). An approach for use with Z is to model an object type as a set, since in Z predicates are expressed as sets. Any object, interface, or action in Z satisfies all its (ODP) types. An ODP subtype, described using a particular predicate, can be modeled in Z as a subset whose members satisfy that particular predicate. A function applicable to a subtype has the appropriate subset as its domain.

How do we model the environment of an object — the part of the model which is not part of that object? The current approach in Part 4 of RM-ODP (ISO, 1995c) recommends doing so by means of only input and output; however, it seems that such an approach is inadequate because Z provides additional — and probably more appropriate — ways of doing so. Free variables are the Z construct that helps here. They are the ones that are not included in the declaration of the schema, but used (referred to) in the schema predicate. In particular, the object's invariant is described by Z constructs that do not involve input or output, but may well use free variables referring to other objects. In this manner, a relationship between objects — described by the collective state of these objects — may be clearly specified. The existence of such free variables in an operation schema implies that this schema represents an (ODP) interaction. Therefore, the part of the environment of the object which pertains to that object is described both in terms of its input and output and in terms of free variables for expressions in the schemas describing that object. By using input and output parameters of the appropriate operation schema defining the action, as well as free variables (from schemas (transitively) included in the delta-list of the operation schema), it is possible to state whether an action is an internal action or an interaction.

In transforming a specification into a more detailed (less abstract) specification, it is possible to state that every observation that satisfies the refinement should also satisfy its original specification (Hoare, 1994). Thus, observable behavior of a system will explicitly correspond to its specified behavior — a fact necessary in system specification and development (Workshop, 1995). If the variables of the specification and its refinement are the same then for all these variables the global property of the refinement implies the global property of its original specification. If, however, the alphabet of the specification is a proper subset of the alphabet of its refinement then existential quantification of such new variables introduced in the refinement should be included in the implication mentioned above (Hoare, 1994).

This paper, based on the existing approaches of (ISO, 1995b; ISO, 1995c) and on the authors' own contributions discussed at length in the ANSI ASC X3 Technical Committees on Formal Specification Languages and Open Distributed Processing, provides a possible approach for expressing ODP specifications in Z. Obviously, we don't try to state that the way presented here is the only possible one. Guidelines and conventions used in Base Z to formalize fundamental OO concepts rigorously described in (ISO, 1995b) are a good framework for inclusion of object-oriented mechanisms for handling large specifications in the next version(s) of the Z standard.

6. REFERENCES

Alencar A.J. and Goguen J.A. (1992) OOZE. In: *Object orientation in Z* (Workshops in Computing), ed. by S.Stepney, R.Barden and D.Cooper. Springer Verlag, 79-94.

Dijkstra, E.W. (1976) *A discipline of programming.* Prentice-Hall.

Gries, D. (1981) *The science of programming.* Springer Verlag.

Harrison, W., Kilov, H., Ossher, H., Simmonds, I. (1996) From dynamic supertypes to objects: A natural way to specify and develop systems. *IBM Systems Journal*, **35**, 2.

Hoare, C.A.R. (1994) *Mathematical models for computing science.* Oxford, UK, August 1994.

ISO (1995a) ISO/IEC JTC1/SC21, Information Technology - Open Systems Interconnection - Management Information Services - Structure of Management Information - Part 7: General Relationship Model. 10165-7.2, 1995.

ISO (1995b) ISO/IEC, Open Distributed Processing - Reference Model. Part 2: Foundations (IS 10746-2 / ITU-T Recommendation X.902, March 1995).

ISO (1995c) ISO/IEC, Open Distributed Processing - Reference Model. Part 4: Architectural Semantics.

Kilov, H. (1993) Information modeling and Object Z: Specifying generic reusable associations. In: *Proceedings of NGITS-93 (Next Generation Information Technology and Systems),* Haifa, Israel, June 28-30, 1993, ed. by O.Etzion and A.Segev, 182-191.

Kilov, H. and Ross, J. (1994) *Information Modeling: an Object-oriented Approach.* Prentice-Hall.

Mac an Airchinnigh, M., Belsnes, D. and O'Regan, G. (1994) Formal Methods & Service Specification. In: *Towards a Pan-European Telecommunication Service Infrastructure* (Lecture Notes in Computer Science, Vol. 851). Ed. by H.-J.Kugler, A.Mullery, N.Niebert, Springer Verlag, 563-572.

Nicholls, J. ed. (1995) *Z Notation, Version 1.2*, The University of Oxford, September 1995.

OODBTG (1991) Object Data Management Reference Model. (ANSI Accredited Standards Committee. X3, Information Processing Systems.) Document Number OODB 89-01R8. 17 September 1991. (Also in: *Computer Standards & Interfaces*, **15** (1993), 124-142.)

Parnas, D.L., Madey, J. (1995) Functional documents for computer systems. *Science of Computer Programming*, **25**, 1, 41-61.

Parnas, D.L. (1995). On ICSE's "Most influential" papers. *Software Engineering Notes*, **20**, 3, 29-32.

Redberg, D. (1994) Object-oriented information modeling techniques to specify a TMN system. In: *Proceedings of the OOPSLA'94 Workshop on Precise Behavioral Specifications in OO Information Modeling*, 101-108.

Wegner, P. (1995). *Models and paradigms of interaction.* ECOOP-95 tutorial notes.

Workshop (1995) Proceedings of OOPSLA'95 Workshop 21: Fourth Workshop on specification of behavioral semantics. (Semantic integration in complex systems: Collective behavior in business rules and software transactions, ed. by H.Kilov, K.Tyson, and B.Harvey).

7. BIOGRAPHIES

Haim Kilov has been involved with all stages of information management system specification, design, and development. His approach to information modeling widely used in industry has contributed clarity and understandability to enterprise and application modeling, leading to specifications that are demonstrably better than "traditional" ones. It has been described in *Information modeling: an object-oriented approach* (Prentice-Hall, 1994). Haim Kilov is an active contributor to several international standardization technical committees. He has been a speaker and a program committee member at numerous international conferences. His interests are in the areas of information modeling, business specifications, and formal methods.

In 1971 **D. Randolph Johnson** received his Ph.D. in mathematics from Yale University and joined the Mathematics Department at the University of Pittsburgh. Since 1978 he has been employed by the US Department of Defense working in such areas as speech communication, software system development, and formal methods. Since 1992 he has been contributing to the development of an ISO standard for Z as a member of the ISO and ANSI committees.