# An Approach to an RM-ODP Toolkit in Z

D. Randolph Johnson

R213, National Security Agency
9800 Savage Road STE 6529
Fort Meade MD 20755-6529 USA
Tel: (301) 688-0748
Fax: (301) 688-0255
Email: drj@tycho.ncsc.mil

Haim Kilov *

Merrill Lynch, Technology Strategy and Planning
World Financial Center, South Tower
New York, New York 10080-6105 USA
Tel: (212) 236-5816
Fax: (212) 236-9600
Email: haim_kilov@ml.com

**Abstract**

We discuss the usage of the (non-OO) Z specification language to represent some fundamental concepts of the ISO Reference Model for Open Distributed Processing (RM-ODP) and the ISO General Relationship Model (GRM). After discussing some of the difficulties involved, we offer suggestions on how Z can be used successfully for specifying and modeling open object-based distributed systems. Although fundamental specification concepts of RM-ODP and GRM are less well-known than "traditional" mathematics, we propose an RM-ODP toolkit somewhat analogous in its usage to the well-known mathematical toolkit in Z.

**Keywords:** Reference Model for Open Distributed Processing (RM-ODP), General Relationship Model (GRM), Z, fundamental concepts, relationships, multiple typing, toolkit

**Workshop Goals:** Learning; networking; discuss how to use model oriented formal specification languages, such as Z, in component-based software engineering; discuss how to make such languages more suitable for this purpose.

*This paper was written while Haim Kilov was at T. J. Watson Research Center, IBM.

## 1 Background

Randy Johnson is a mathematical logician by training and was first involved in formal methods part time from 1981 to 1984. From 1983 to 1990 he was doing large system software development. Since 1991 he has been involved full time in formal methods. He has been working on the international standard for Z as a member of the ISO committee SC22/WG19 since 1992 and has been a member of the ANSI committee NCITS/J21 for formal specification languages since its establishment.

Haim Kilov is a member of and active contributor to several international standardization committees. In particular, he contributed significantly to the development of GRM and RM-ODP. He is a member of NCITS/J21 and chairs the OMG semantics working group. He organized and co-chaired six OOPSLA workshops on behavioral semantics (he co-edited a book recently published by Kluwer based on the first four of these), as well as the ECOOP97 one, and has been an invited speaker and program committee member at numerous international conferences. He has a significant number of publications. His interests are in the areas of precise and formal specifications, information modeling, and business specifications including business patterns.

## 2 Specifications: Rigorous and Formal

To build a system that satisfies users needs, we must understand these needs and precisely specify them. These specifications should serve as a contract for building a system. To be enforceable, contracts have to be precise and explicit. They have to specify semantics – as opposed to just signatures – without relying on implicit default knowledge, or on "natural language" (for the reasons eloquently expressed in [Dij96] and elsewhere). This has been acknowledged in industry: in particular, current activities within the Object Management Group, including but not restricted to its semantics working group (of the Object Model subcommittee), are concerned with exactly these issues. All too often, the only reliable semantics of the system has been provided by the implementation code, which was not understandable by humans; the very serious problems of such an approach are only too well-known. It is also well-known (e.g. from Dijkstra's work in the mid-1960s) how to make semantics understandable: use abstraction and separate concerns.

Semantics can be defined in a structured "natural language" (examples include standards [ISO95a, ISO95c] – the subject of this paper), but, due to the inevitable ambiguity of natural language, a notation based on mathematics should be relied upon in cases of doubt. As the use and reuse of such a notation is the topic of our paper, some familiarity with the concepts of RM-ODP and with the Z specification language is assumed. Specifically, we will use object-oriented terminology as defined in [ISO95a, ISO95b, ISO, ISO95c] and try to explain any new terminology we introduce. We will also use names within their naming context (as specified in [ISO95a]) and especially distinguish object-oriented terminology from Z terminology whenever confusion is a danger, e.g. for "type".

The very use of a formal notation forces specifiers to think and express themselves precisely and explicitly. The ease and efficiency of reasoning depends upon the language used [Dij96]. Although a formal specification can be unambiguously translated into stylized English and the result shown to the customer [ISO95c, KR94, KMS96, Red96], the reverse is not true. If the business semantics is not understood and specified properly, then a system specification cannot be checked for conformance with its business specification. The implementor will have to create both the system and the business specification as he understands it, usually leading to the expression of business semantics as "business logic" in the application code. The Year 2000 problem is one example of the results.

### 2.1 RM-ODP vs. legacy (traditional OO)

The object-oriented "way of doing things" has been introduced, in part, to provide a more natural path from business to system specifications, and then to system design and code. Nevertheless, the concepts of the traditional (message-oriented) object model are quite far from concepts naturally and conveniently used in business (and often system) specifications. Specifically, the traditional object model imposes a mindset of using a single hierarchy of subclasses of isolated objects exchanging messages and typically underestimates relationships between objects. It also requires properties of collections (both collective state and collective behavior) to be expressed in terms of their refinement using (attributes of) individual objects. This approach leads to quotes like "The object-oriented way does add a certain amount of clutter to the specification and does mean that certain things are more awkward to express. This is because it is moving, even if only slightly, towards implementation." [Hal94b].

The shortcomings of the classical, traditional, object model have been recognized for some time and have been formulated in standardization documents [ANS91]. The generalized object model is free of these shortcomings, although it has far fewer available implementation mechanisms. However, a specification – be it a business or a system one – should not be based on available implementation mechanisms: this consideration clearly follows from separation of concerns and has been known in computing science for quite a while (from the work of E. W. Dijkstra [Dij65] and others).

Fortunately, there exist significant international standards based on the same ideas, specifically on the concept of abstraction. RM-ODP and GRM are important examples of these standards and will be considered in this paper. RM-ODP emphasizes common fundamental concepts encountered in any open distributed system, including distribution-independent concepts! It is used for descriptions of any system, not just software; and it includes both viewpoint-specific concepts, as well as – perhaps, more importantly – concepts common to all viewpoints - enterprise, information, computational, and so on. In doing so, RM-ODP stresses commonalities in specifications – from business to systems – and as such is ideally suited to serve as a basis for specifying the semantics of any system. Additionally, the semantics of some concepts which have been only "lightly touched" by RM-ODP, can be expressed using the same approach. This has been accomplished for relationship specifications in [ISO95c] where the emphasis is on concepts used to understand and specify relationships in a representation-independent manner.

For communication, the value of a specification structured in accordance with the user's conceptual model and expressed in his terminology cannot be over-emphasized. This is true even if the implementation is structured or described differently. Fortunately, the generality and power of the concepts in RM-ODP and GRM often provide suitable material for this user-oriented style.

### 2.2 Formalization of various OO approaches in Z

Anthony Hall [Hal90, Hal94b, Hal94a] and others have shown how many basic aspects of an object-oriented approach can be handled very well in Z. These include such issues as objects in isolation, object identity, object creation and deletion, and subtyping. Although Hall shows several ways of handling such issues, his presentation is based on examples. We will try to treat topics generically and to discuss some points previously treated very lightly or not at all.

In traditional OO, objects are isolated, and state and behaviour are specified in an implementation-oriented, or, to be more precise, in an implementation-restricted, way. By contrast, a more general object model defines operations "regardless of how this effect is achieved" [Hal94a]. The generalized object model, described in more detail in [ANS91, KR94] does not require invariants and operations to be "owned" by a single object; rather, it uses collective state for invariants, and collective behavior for operation specifications. Both RM-ODP and especially the General Relationship Model follow this more general approach. In this manner collections of objects and their operations can be described in a convenient and natural way.

As "traditional" Z does not always directly and explicitly correspond to an object model (for example, explicit information hiding in Z is not possible; but see below), various OO extensions to Z have been created for this purpose. These extensions usually supported the classical (message-oriented) object model, and therefore have difficulties in specifying collections of objects (collective state and collective behavior). An exception is OOZE [AG92], a development environment and language explicitly using the concept of a module. It is inspired by Z but has an algebraic semantics rather than a set theoretic model-based semantics. Standard Z itself does not have any problems in specifying collections of objects and their interactions, although it has certain other shortcomings.

[HW93] strongly suggests the need for parameterized modules, with structured parameters, in base Z ("a library can... be viewed as a super-schema"). It suggests that it is possible to achieve this goal, including specifying the internal structure of the parameters, by means of "simply adding a normal Z predicate equating the variable [of the library] to the value", after the library has been instantiated with appropriate sets for generic set parameters. Another idea mentioned in the paper, "having an explicit parameters section at the head of the library to specify the types of and constraints on the parameters", may be even better. Our present goal, however, is to see what can be done with standard Z as it is.

Unlike Hall [Hal90, Hal94b, Hal94a] and the chapters in *Object Orientation in Z* [SBC92], we try to give a general description of our approach, not just examples. This is in the same spirit as the well-known Z mathematical toolkit. While the notion of, say, injection is encountered in many specifications, the specifier does not describe this anew each time since it is already in the toolkit, available for reuse. In the same manner, the notion of such a generic relationship as a hierarchical composition, which is more complex, is encountered in many object-oriented specifications, and its specification should not be repeated each time. It should be defined once and made available for reuse. When appropriate, we will use examples to help in understanding the meaning and use of the notions we define. To be of value, the components in a toolkit must, of course, be widely useful and be precisely specified. RM-ODP Part 4.2 [ISO], the US Comments on this document [X3T96], and our paper in FMOODS'96 [JK97] describe some of these ideas in detail. We now want to expand upon this earlier work.

In addition to specifying the invariants of collections of objects, our toolkit will also include the specification of generic operations applied to these collections. For example, in defining a generic composition, we will specify not only its signature (above the line) and semantics (the invariant – below the line), but also such operations as "delete a component instance" (designated by its identifier in a given naming context) with several possible outcomes, such as "OK, composition still exists", "OK, but no composition exists anymore", "no such component instance", "not an identifier", and so on. Some of these operations have been specified semi-formally: on the one hand, Clause 8.1 of [ISO95c] specifies the operation invariants, as well as pre- and postconditions for the following relationship management operations applied to any relationship: ESTABLISH, TERMINATE, BIND, UNBIND, QUERY, and NOTIFY; and on the other hand, for example, Clause F.4 of [ISO95c] specifies, in substantially greater detail, the signature and pre- and postconditions for these operations applied to a generic Composition relationship. Naturally, a formal specification of these operations leads to their being understood more clearly, in the same manner as [ISO] leads to substantially better understanding of at least some Clauses of [ISO95a].

# 3 Our Toolkit

## 3.1 Objects in Isolation

We will not repeat the excellent work done by Anthony Hall (and others) in showing how isolated objects can be specified in Z. Instead, we will emphasize those properties of isolated objects which are less well-established in traditional OO literature, but considered to be of sufficient importance as to be included in [ISO95a]. In some cases, these properties are described in traditional OO models only partially; multiple naming and multiple typing are good examples. Although using these properties is clearly essential for a successful business specification, in many cases "OO analysis and design methods" have not used them because existing "OO implementation mechanisms" did not support them.

An interesting and very nice treatment of isolated objects in Z using a message-oriented object model is provided in [DBS96]. The authors clearly distinguish between input messages accepted by an object, input messages blocked by an (unreceptive) object, output messages accepted by the environment, output messages blocked by the (unreceptive) environment, and internal state transitions. Moreover, they consider and formally treat in detail the [ISO95a] notion of behavioral compatibility for isolated objects. As a result, we may observe that it is possible (and in fact not very difficult) to provide a clear formal description of the distinction between "visible" and "invisible" operations (resulting in state changes), thus supporting, to a certain extent, the OO concept of information hiding. Nevertheless, [DBS96] is quite restrictive because of its usage of the message-oriented object model, and in its plans to consider object composition using the same approach: it appears that such treatments quickly become too complicated. Therefore, a representation-independent approach to specifying collections of objects, such as the one in [ISO95c], seems to be warranted. We note also that, since a composite object may be decomposed at different abstraction levels (RM-ODP), a discussion of an isolated object may as well refer to an isolated component.

## 3.2 Names

Recall [ISO95a, Clause 12] that a name is defined only within a particular naming context. Some names are unambiguous in their contexts and are called identifiers. Thus, an entity (of which an object is a model) can have several names, and several identifiers, within a particular naming context. In order to designate an entity, its identifier should be made visible. As the names (and identifiers) of an entity may be created, changed, and deleted, and the naming contexts may also be created and deleted, we need some property of an entity which will be independent of names and naming contexts, but will distinguish an object from any other object, as required by [ISO95a, Clause 8.1]. Surrogate is such a distinguishing property of an object, and it is not necessarily made visible. Hall calls this property "object identity" and models it in Z by a schema component called "self". It is independent of any naming context: indeed, a property distinguishing between objects should not disappear when a particular naming context is deleted! Since the representation of a name as, for example, a character string is ambiguous without the naming context – a certain string of digits could easily be a name of an insurance policy in one context and a name of a savings account in another – the full representation as an ordered pair (name,context) is needed in general. In a very simple specification, such as one with only one naming context, this representation could also be simplified.

The pool for surrogates is obviously independent of any naming contexts. Thus, when an object (in fact, an object instance – we will clearly distinguish between object types, classes, and instances, and "object" will usually mean "object instance") is created, a new ("fresh") surrogate value is taken from the surrogate pool, and a new pair is added to the surrogate - object relation. (This relation is a partial bijection: any object has exactly one surrogate corresponding to that object.)

A fragment of toolkit concerned with name operations should include creation and deletion of a naming context, as well as creation, deletion, and changing of a name within a given naming context. These operations are quite straightforward. However, as it should be always possible to designate an object, it should always have at least one identifier (and therefore at least one naming context) – these are conjuncts of the invariant of an object. Whether a particular context may have only names, and not identifiers, is an interesting issue – it would seem the answer should be "yes" as long as there exists at least one identifier in any naming context. Imagine a context in which otherwise different things ("objects") become indistinguishable ("any software engineer can be replaced with any other software engineer with approximately the same experience"). Notice that these issues become explicit as a result of the need to create a formal specification (even without writing the specification itself).

### 3.3 Types

An object can have several types ([ISO95a, Clause 9.7] states that a type is a predicate characterizing a collection of instances), and thus several subtyping hierarchies. Moreover, a type can be attached to or detached from an object. This approach, although often absent in popular implementation mechanisms, is convenient and natural; not only has it been successfully used in specifications [ISO95a, KR94, KMS96], but it has been supported by such implementation mechanisms as subjects [HKOS96]. Out of multiple types of an object, the template type [ISO95a, Clauses 9.11 and 9.19] is the only essential one because a template of an object specifies the common features of a collection of objects in sufficient detail that an object can be instantiated using it. (For example, a person – template type – can also be a homeowner, an employee, a policyholder of an insurance policy, a Republican, a Manchester United supporter, a conference participant (for a particular conference), an author, and so on. Observe also that not all of these non-template types are subtypes of a person: for example, a homeowner or a policyholder is not necessarily a person.) Non-template types are used to separate specification concerns, for example, in specifying viewpoints. They can be attached to or detached from an instance of a template type; however, information (predicates) presented by the non-template types is not sufficient to instantiate an object.

An object state schema will include the set of its types and the set of its (name,context) pairs. The first of these set will always be non-empty, since it will always contain at least the template type of the object, essential for instantiating the object. The latter set is also non-empty since we want to be able to designate an object instance, and identifier usage seems to be the only clear way of doing so shown in RM-ODP. This is also obvious in practice.

The state of the system will include two, possibly three, functions, each having the given Z type [SURROGATE] as the Z type of its domain. The actual domain of each of these functions is the set of surrogates of existing objects. One of the functions maps each surrogate to the set of (name,context) pairs for the object. The second function maps each surrogate to the set of types of the object. If desired, we could also have a function in the system state mapping a surrogate to the set of identifiers of the object. Since the identifiers form a subset of the names, this third function is redundant but this may be useful nevertheless.

## 3.4 Invariant of an object

There are certain general aspects of the specification which are so basic that they are part of the invariant included in the state schema of all objects; surrogate' = surrogate is an example. All operations in which this object participates must preserve the surrogate of the object. Other requirements are slightly less basic and are part of the invariant for most, but not all, operations. For instance, (name',context') = (name,context), unless the operation changes names, including the change of an identifier. In fact, the set of (name,context) pairs has to remain the same, unless names are changes/added/deleted. Note that it is useful to have special name-related operations in the toolkit. They would be used, for example, to correct a Social Security Number which has been entered incorrectly into some record.

The invariant of an object (as well as the invariant that defines a particular collection of objects, see below) may be considered as a conjunct of the global property. This approach is correct, but hardly helpful for large specifications: a better structuring seems to be essential for human understanding. For example, the notion of sections in [Uni95] provides a good indication of an approach to come in standard Z. In the absence of a syntactically-supported construct, we can only rely on conventions (such as invariant of an operation described, eg, in [ISO95c]) that help us to consider only the relevant fragments of a large specification. This may lead to redundancies (such as explicit inclusion of the relevant fragments of the object invariant in operation schemas) which are very helpful for human understanding.

Note that conjuncts of the template type are checked as a fragment of the precondition for creation or initialization of an object. Clearly the template type must be known. Each input value provided as a property value must satisfy the invariant of that value's type (ie domain – see [Kil89]). In addition, these values collectively must satisfy the invariant of the object. An object satisfying its template type obviously satisfies every conjunct of this type. Each of these conjuncts may be considered a supertype of the template type, e.g. – for a person – being a legal entity, having a name, etc. But what of optional types for which no information is known at initialization?

### 3.5 Attaching and detaching a type

Attaching a type is handled through an operation which adds a new predicate to the set of predicates in the object's state schema. The promoted operation on the system state schema modifies the corresponding value of the function mapping each surrogate to the set of the object's types. Detaching a type is handled similarly by deleting a type from the set of types which is both a component of the object state schema and a value of the correspondong function in the system state schema. However, the template type may not be deleted and must continue to be satisfied.

This introduces the notion of a context (rather close to the Z global property) which is essential for real life specification. Many components in such a specification can be understood, used and reused only within a context. A significant difference between our toolkit and the standard Z mathematical toolkit is the greater importance of the context for the specification and understanding of objects.

To achieve the full generality of type treatment in RM-ODP, we would like to allow objects acquiring and losing types not just to change appropriate property values, but also to acquire and lose properties – attributes with a particular value. Unfortunately, Z does not allow us to use the rather straightforward approach to specifying this in general that we used for names and types. For them we could easily use a set in Z since the elements were all of a consistent Z type: the Cartesian product of NAME and NAMINGCONTEXT for (name,context) pairs and the power set of the class of objects for types (predicates) of objects in the class. The properties which might conceivably belong to a given object are numerous and varied, in general, and could not reasonably be specified as any single Z type. It is possible in Z to effectively add new components to the state schema of an object by defining a new schema which includes a reference to the old schema along with the new components. In general, there is no corresponding ploy for deleting components, although hiding and quantification can some times achieve the desired effect. The idea of assigning a dummy nil value seems more to suggest that the true value is unknown or has not yet been assigned than to indicate that the component in question should not have been there in the first place.

The Z free type construction can be used to build up a Z type which represents a union of several different Z types. As a general approach, especially for *business* specifications, this seems inadequate. Even for a specific business specification in Z, this approach would require a remarkable degree of clairvoyance to foresee all the properties which might ever belong to the objects specified. Actually, only the Z types of the properties need to be divined at the beginning, not the exact values. A discussion about whether we need to foresee all possible properties is going on (electronically and otherwise) in some OMG groups in which the semantics of "roles" is being considered.

The free type construction is a considerably more plausible approach for a *system* specification. With early knowledge of the approximate size and complexity of the system being specified, one could perhaps construct a sufficiently rich Z free type to provide good representations for all of the system objects and their properties. This would likely be based on integers and strings and a few iterations of Cartesian product and power set. The state schema of an object could then be specified to include a set of elements of this free type. To represent the addition and deletion of properties of system objects, elements would be added to or deleted from this set. The only clairvoyance needed is that required to construct a Z free type sufficiently rich to reasonably represent the general nature of properties of objects in the system specification. The exact values would be specified later, for example, as operations are defined. As in the treatment of names and types, there would be the corresponding system state function mapping each surrogate to the corresponding component of the object state, in this case the set of its property representations. Thus, for each new (previously unknown) property type specified in a business specification, a mapping should be produced to (previously known) property types of a system specification.

[ISO95a] does not prohibit an instance from having more than one template type. This may be important, in particular, for specifying viewpoints. These template types should be mutually consistent, so that the conjunction of their invariants should not be false. Moreover, it seems to be possible to change the template type for an instance. These issues have not been explicitly treated in [ISO95a], but their treatment becomes essential as soon as the concepts described semi-formally are to be formalized. This is a very important (and relatively well-known) advantage of formalisms! Satisfaction of the object type is not sufficient: most objects participate in relationships and therefore must satisfy the invariant of these relationships.

### 3.6 Non-isolated Objects and Generic Relationships

Despite the (over-)emphasis on isolated objects, their classes and subclasses in traditional, legacy, OO modeling, objects do not exist and cannot even be understood in isolation. This is acknowledged in [ISO95a] and, especially, in [ISO95c]. It is also quite well-known that the ideas of collective state and behavior are not new – consider the remark in [Hal60]: "...the familiar axiomatic approach to elementary geometry [...] does not offer a definition of points and lines; instead it describes what it is that one can do with these objects." (Thus, we do not offer a definition of an object nor a definition of an entity.)

Following these considerations, we may observe that in most cases an operation, in particular, creation or deletion of an object, refers to other, related, objects. In other words, the pre- and postconditions of such an operation refer to properties of other objects. The same is true with respect to the invariant of this operation.

It seems to be relatively straightforward to define collections of objects in this manner, and Z provides excellent ways for doing so. However, in order to provide a reusable library – like a mathematical toolkit – for generic relationships (like hierarchical ordered composition), we will need to use non-elementary parameters, in the same manner as the mathematical toolkit does. In particular, we will need to explicitly refer to "structured" formal parameters. That is, firstly, we must specify "below the line" of a Z schema the properties of these parameters of interest to the user of our toolkit and thus be able to check whether the corresponding actual parameters satisfy these properties. Secondly, we must be able to refer, in the toolkit, both to these parameters and to their components. In other words, we want to specify generic invariants of constructs (such as relationships) of our toolkit. Consider the generic composition relationship, also known as "aggregation", as an example.

### 3.7 Composite objects

In order to express such an invariant for a generic composition relationship, we will have to recall that the specification of a generic composition refers to a composite type and a finite non-empty set of component types, and to a composite instance and a possibly empty set of component instances of each of these component types. (More detailed information is provided for various subtypes of the generic composition, see [ISO95c, KR94].) In some cases, particularly those in which the different components play quite different roles in the composite object, it may be clearer to consider an ordered tuple of components rather than a set ("ordered composition" is a subtype of a composition). We will also state that, for each instance of a composition relationship, the set of types of the composite instance is not equal to the set of types of any of its component instances.

Specifying these properties in Z is rather straightforward. To begin with, we will use the same approach to specifying the set of known instances of a particular composite object as we used to specify the set of known instances of isolated objects: these sets are represented, through the objects' surrogates, by the domains of appropriate functions. As mentioned earlier, creating a new instance of an isolated object adds a new pair to the "surrogate-object" function for a new surrogate value; deleting an instance of an object deletes the corresponding pair. Similarly, in the simple case, creating a new composition instance adds a new pair to the "composite instance-sets of its component instances" relation; and deleting an instance of a composition deletes the corresponding pair. In the more complex situation, we modify the Z relation representing the relationship between the composite object and its components by adding, changing, or deleting elements of the component set.

When a new composite object is to be created, some extra care must be taken. A number of conditions must be checked and each of the possible errors handled in some appropriate way. First, each of the component object instances must exist and satisfy its individual invariant. Secondly, the would generally be an additional precondition concerning the collective state of the whole set of component objects. Only when all of these conditions are satisfied could the composite object be created with its own invariant.

It is substantially more complicated to specify in Z that there exist two kinds of properties of the composite instance: those independent of any components, and those determined by properties of these components and therefore having values changeable as a result of adding/deleting a component, or changing a component's properties. A composite instance should have at least one property of each of these kinds. As a simple example, consider a document, such as the proceedings of a conference, as a composition of chapters: the title and author of the document are properties of the first kind, while the number of pages and the abstract – properties of the second kind (observe that there is no automated way to calculate the value of the abstract!).

This approach is very close to the one used in the mathematical toolkit: indeed, both (formal parameter) sets and their elements are referred to in the specification of injection; and it is quite possible to imagine "structured" elements (eg pairs of an injection!) used either as actual parameters, or even as formal parameters to define an injection for which both the source and target are injections.

## 4 Conclusion

This paper, based on the existing approaches of [ISO95a, ISO] and on the authors own contributions which became US positions and have been incorporated in [ISO] (and, earlier, in [ISO95c]), provides a possible approach for creating a toolkit to express generic specifications in Z. Obviously, we don't try to state that the way presented here is the only possible one. A Z toolkit like the one outlined here may be of substantial importance to specifiers who wish to understand, specify, and (re)use important standardized generic concepts encountered in all applications. This is analogous to the mathematical toolkit in [Uni95].

**Acknowledgments:** We would like to thank the anonymous referees for their constructive comments and helpful suggestions on an earlier version of this paper.

## References


[AG92] Antonio J. Alencar and Joseph A. Goguen. Ooze. In Susan Stepney, Rosalind Barden, and David Cooper, editors, *Object Orientation in Z*, Workshops in Computing, pages 79–94. Springer-Verlag, 1992.

[ANS91] ANSI Accredited Standards Committee. X3, Information Processing Systems. *Object Data Management Reference Model*, 1991. Document Number OODB 89-01R8, also in Computer Standards & Interfaces, Vol. 15 (1993), pp. 124-142.

[DBS96] R. Duke, C. Bailes, and G. Smith. A blocking model for reactive objects. *Formal Aspects of Computing*, 8(3):347–368, 1996.

[Dij65] Edsger W. Dijkstra. Programming condidered as a human activity. In *Proceedings of the IFIP Congress 65*, pages 213–217. Macmillan and Co., 1965.

[Dij96] Edsger W. Dijkstra. A first exploration of effective reasoning. University of Texas at Austin, EWD 1239, 1996.

[Hal60] Paul R. Halmos. *Naive set theory.* D. Van Nostrand Company, 1960.

[Hal90] Anthony Hall. Using z as a specification calculus for object-oriented systems. In D. Bjorner and H. Langmaack, editors, *Proceedings of VDM 90*, number 428 in Lecture Notes in Computer Science, pages 290–318. Springer-Verlag, 1990.

[Hal94a] Anthony Hall. Specifying and interpreting class hierarchies in z. In Jonathan P. Bowen and J. Anthony Hall, editors, *Z User Workshop, Cambridge 1994*, Workshops in Computing, pages 120–138. Springer-Verlag, 1994.

[Hal94b] Anthony Hall. *ZOO Course Notes*, 1994.

[HKOS96] W. Harrison, H. Kilov, H. Ossher, and I. Simmonds. From dynamic supertypes to objects: A natural way to specify and develop systems. *IBM Systems Journal*, 35(2):244–256, June 1996.

[HW93] I. Hayes and L. Wildman. Towards libraries in z. In Jonathan P. Bowen and John E. Nicholls, editors, *Z User Workshop*, pages 37–51. Springer-Verlag, 1993.

[ISO] ISO/IEC. *Open Distributed Processing - Reference Model. Part 4: Architectural Semantics.*

[ISO95a] ISO/IEC. *Open Distributed Processing - Reference Model. Part 2: Foundations*, 1995. IS 10746-2 / ITU-T Recommendation X.902.

[ISO95b] ISO/IEC. *Open Distributed Processing - Reference Model. Part 3: Architecture*, 1995. IS 10746-3 / ITU-T Recommendation X.903.

[ISO95c] ISO/IEC JTC1/SC21, Information Technology. *Open Systems Interconnection - Management Information Services - Structure of Management Information - Part 7: General Relationship Model*, 1995. CD ISO/IEC 10165-7.2.

[JK97] D. R. Johnson and H. Kilov. Can a flat notation be used to specify an oo system: using z to describe rm-odp constructs. In Elie Najm and Jean-Bernard Stefani, editors, *Formal Methods for Open Object-based Distributed Systems*, pages 391–398. Chapman & Hall, 1997.

[Kil89] H. Kilov. Domains and semantic integrity. *Computer Standards & Interfaces*, 9:143–147, 1989.

[KMS96] H. Kilov, H. Mogill, and I. Simmonds. Invariants in the trenches. In H. Kilov and W. Harvey, editors, *Object-oriented Behavioral Specifications*, pages 77–100. Kluwer Academic Publishers, 1996.

[KR94] H. Kilov and J. Ross. *Information Modeling: an Object-oriented Approach.* Prentice-Hall, 1994.

[Red96] D. Redberg. The search for the linking invariant: behavioral modeling vs. modeling behavior. In H. Kilov and W. Harvey, editors, *Object-oriented Behavioral Specifications*, pages 241–264. Kluwer Academic Publishers, 1996.

[SBC92] Susan Stepney, Rosalind Barden, and David Cooper, editors. *Object Orientation in Z*, Workshops in Computing. Springer-Verlag, 1992.

[Uni95] The University of Oxford. *Z Notation, Version 1.2*, September 1995.

[X3T96] ANSI X3T3. *Recommendation for vote on DIS 10746-4.2 "Information technology - Basic reference model of Open Distributed Processing - Part 4: Architectural semantics"*, December 1996. Document X3T3-96/131.

## 5 Biography

**D. Randolph Johnson** received his Ph.D. in mathematical logic from Yale University in 1971 and joined the Mathematics Department at the University of Pittsburgh. Since 1978 he has been employed by the US National Security Agency working in such areas as speech communication, software system development, and formal methods. Since 1992 he has been contributing to the development of the ISO standard for Z as a member of the ISO and ANSI committees. Since 1991 he has been working on the application of formal methods to NSA systems, particularly component-based software.

**Haim Kilov** has been involved in all stages of information management system specification, design, and development. His approach to information modeling, widely used in financial, insurance, telecommunications, document management, and other areas, has contributed demonstrable clarity and understandability to specifying (components of) businesses and systems. It has been described in a book published by Prentice-Hall in 1994. Haim Kilov is using and extending his approach to specifications in customer engagements, and does research and consulting in the areas of business specifications and information modeling. He was affiliated with Bellcore and IBM and currently is affiliated with Merrill Lynch.