A Review of Object-oriented Approaches in Formal Methods

ANTONIO RUIZ-DELGADO*, DAVID PITT† AND COLIN SMYTHE‡

Centre for Satellite Engineering Research, University of Surrey, UK
† Department of Mathematics and Computing Science, University of Surrey, UK
‡ Department of Computing Science, University of Sheffield, UK
**Present address: Mercury Communications, Lakeside House, Cain Road, Bracknell, Berkshire RG12 1XL, UK*
E-mail: a.delgado@mcl.co.uk

This paper presents a survey of recent approaches in the application of the object-oriented paradigm to formal specification. The complexity of current information systems demands the use of a higher degree of formalism in the development process. Formal languages such as Z, VDM and Lotos have been used extensively in academic environments and research projects; however, their utilization in the 'real world' is still relatively small. The use of object-oriented concepts has now been suggested as a good solution to the lack of expressiveness that characterizes most of these languages. Several approaches addressing this issue have appeared in the literature recently, including extensions to most existing languages. In this paper we review some of these techniques and discuss problems and issues relevant to the combination of formal methods and object orientation.

Received February 16, 1995, revised November 8, 1995

1. INTRODUCTION

One of the recognized strengths of the object-oriented paradigm is that a common core of basic ideas can support modelling at all levels of abstraction, from conceptual description of real world entities to implementation of software applications in computer languages. In its general applicability lies much of its strength and there are few areas of computer science that have not been influenced in one way or another by the object-oriented paradigm.

The latest addition to the list of disciplines 'touched' by the object approach is what has become to be known in the literature as *formal methods*. By formal methods we mean the set of activities—specification, reasoning, refinement—that add mathematical rigour to the development, analysis and operation of computer-based systems. In many respects, formal methods suffer from the same kind of problems that have always characterized the object-oriented paradigm. Many people talk about them, but few know what they are and even less actually use them. Just as happened with object-orientation, formal methods are often presented by devotees as 'the' solution to the increasing complexity of current information systems, but the reality shows that the majority of the available methods are still only used in academic environments.

Both object orientation and formal methods have evolved separately, and it is only recently that some researchers have started to investigate the benefits of their possible integration. Indeed, as pointed out [1], many aspects of the object-oriented paradigm are similar to those typically found in formal methods. The description of class behaviour in an implementation-independent fashion, the use of inheritance to indicate subtype relations or the reliance on well understood mathematical abstractions such as sets, sequences and functions are mechanisms that formal methods practitioners will be most familiar with. Yet, despite these similarities there has been little work on their possible integration in the past. In 1991 both OOPSLA [1] and ECOOP [2], the world leading conferences on object orientation, held discussion panels addressing the issue. These events somehow represented the recognition by the research communities that convergence was necessary.

This paper reviews the work done by various groups in this new area of object-oriented formal specification. We are not so much interested in comparing the capabilities offered by different languages or assessing their degree of 'object-orientedness'. A collection of case studies from different object-orientated extensions can be found in Lano and Haughton [3]. The application domain of the methods surveyed is diverse and so are the requirements imposed on the language, which makes any comparison task difficult. Our purpose is to show that object-oriented notions are useful in a formal development environment.

The rest of the paper is structured as follows. Section 2 discusses some general issues in the combination of formal methods and object orientation. The value added by the paradigm in a specification context is summarized in eight points. An informal survey of the state-of-the-art in object-oriented formal specification is then presented in Section 3. The idea is to give the reader a global view, covering the most important approaches reported in the literature over the last 3–4 years. Short descriptions are given for each approach, highlighting their distinctive points. Finally, some concluding remarks are presented in Section 4. Familiarity with formal methods and object orientation is assumed throughout the paper.

2. A SYNERGETIC COMBINATION

The cross-breeding of formal methods and object orientation may take place coming from either of both ends. Object-orientation people need formal techniques in order to support their methodologies with a sound formal basis and formal methods people use object-oriented engineering concepts to make their mathematical models easier to handle.

2.1. The object-orientation perspective

From an object-oriented point of view a number of benefits are to be gained with the adoption of formal methods:

- The description of languages and models using a formal notation way enhances considerably our own understanding of them. Better development methods are produced if formal techniques are used by their designers.
- There are plenty of complex dependencies between object-oriented notions which can be studied if our object model has a formal definition.
- In order to promote the reuse of software components advocated by object orientation, we have to describe unambiguously what those components do (Interface Specification). This will facilitate the development of libraries at both the conceptual and design level.
- Notations with formal semantics are necessary for the development of better CASE tools.
- The use of formal techniques will ultimately make possible the automatic verification of the software produced with object-oriented methods.

For examples of how formal notations can be introduced into current object-oriented analysis and design methods, see Wilson [4] or di Giovanni and Iachini [5].

There are also a number of approaches combining object-oriented programming languages with formal techniques. Cheon and Leavens have developed interface specification languages for Smalltalk [6] and C++ [7], using Larch, an algebraic specification language. Wills [8, 9] combines Smalltalk with VDM in *Fresco,* an environment for the specification and implementation of object-oriented software components. SPECS-C++ [10] is a model-based executable specification language for C++ classes, which also follows the VDM style.

2.2. The formal methods perspective

For the formal methods community the rationale of the combination is obviously different. Their intention is to bring powerful and well-studied software engineering mechanisms into their languages, in order to make them more 'user-friendly'. The slow adoption of formal methods in the industry has been mainly caused by two reasons. First, potential formal methods users need a good mathematical background to understand not only the underlying models (logic, set theory, algebra, etc.) but also in many cases the syntax of a formal language. Second, most languages lack adequate structuring constructs to model complex problems. They all seem to work fine on a small scale but when it comes to deal with complex systems they prove totally impractical.

The use of concepts borrowed from the object-oriented paradigm has now been widely recognized as a possible solution to the problems mentioned above. Many formal specification languages have now been either extended or re-interpreted within an object-oriented framework. There are also other languages that have been designed following the paradigm from the beginning. They all try to benefit from the inherent simplicity and the excellent structuring capabilities of object-orientation in a fully formal fashion (Figure 1).

The following eight points summarize the added value provided by the object-oriented paradigm in a formal methods environment.

1. *Object-oriented concepts can make formal specification languages more attractive and easy to use.* The distinction between objects and classes advocated by the paradigm allows the specifier to distinguish clearly between the actual entities in the system structure (objects) and the templates where common features of similar objects are defined (classes). A more concise specification is achieved by factoring out commonalties. Similarly, the natural separation of abstraction levels induced by the distinction between internal behaviour of an entity and external communication through well defined interfaces (encapsulation) produces cleaner specifications, easier to refine and to manipulate.
2. *Object orientation fills the gap between structural (static) and behavioural (dynamic) specifications.* Two different aspects of a system can be subject to specification. We may be interested in describing how the system is structured, its components and their

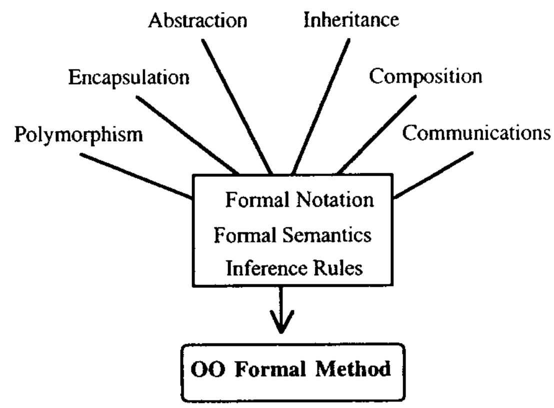

FIGURE 1. Incorporation of object-oriented features into formal methods.

interrelations. Or we may want to describe its operational behaviour, as seen by an external user. Traditionally, the specification of both aspects had to be carried out separately, often using different formalisms. The language Lotos, for instance, uses process algebra and ACT-ONE, respectively. This is not the case with object-oriented languages as one of the key notions of the paradigm is the integration within class descriptions of data and behaviour. In fact, object orientation can provide the framework for the balanced integration of various specification styles and problem solving mechanisms: data abstraction, semantic classification, state transitions, declarative specification of behaviour, message passing, etc.

3. *Concurrency can be seen as a logical consequence of the inherent distributed nature of the object model.* The specification of concurrency has always been an area of great concern within the formal methods community. Systems where a number of different activities are being carried out at the same time are difficult to handle. The object-oriented paradigm perceives the world as a collection of loosely coupled agents executing activities in parallel and working in cooperation. This view makes objects the natural unit for concurrency control.
4. *Reuse of specifications can be achieved using inheritance.* Much has been talked about the importance of reuse at all levels in the development process. The utilization of inheritance as a 'code sharing' mechanism, which is so useful in programming, can also provide the basis for specification reuse.
5. *Refinement relations can be established through class specialization.* A basic class specification can be augmented with more and more details, producing descriptions of the same entity at different levels of abstraction (top-down specification style). Starting from an initial solution satisfying some basic requirements, further properties can be imposed without violating those that have already been established (Rules for determining which form of inheritance leads to refinement and laws of refinement preservation are explicitly dealt with in Lano and Haughton [11] and other references [12, 13].) This form of inheritance is sometimes called *semantic* inheritance, as opposed to the purely *syntactic* mechanism described above.
6. *Object composition provides for bottom-up specification styles.* The concept of composition is also known in the object-oriented literature as *aggregation*. This structuring technique corresponds to a 'part-of' relationship, which allows us to define the inner structure of a complex object in terms of smaller objects that work in cooperation provide the desired functionality.
7. *Object orientation not only simplifies the actual writing of specifications but also can help in the formal reasoning stages.* The inherent modularity of the object-oriented approach can be exploited to facilitate the construction of mathematical proofs. It is natural to think that what holds for a class must also hold for every subclass (reuse of proofs!), but this has to be carefully studied and formalized in conjunction with the inheritance mechanisms supported by the language. The same can be said about composition of objects and its implications on the proof system. What properties can be inferred from the properties of the components? It all depends on the type of compositionality and its formal definition. The question becomes particularly difficult when concurrency between components is involved. (Jones [14] shows how the object-orientated paradigm provides a way of controlling interference and defining granularity that results in fewer and more tractable proof obligations.) Experience in the use of formal methods shows that the specification style determines considerably the amount of effort needed in the formal reasoning phase. Many specifiers structure their descriptions with a proof-oriented mind so that reasoning becomes easier. At later stages the specification can be refined to reflect more the real structure of the final implementation. The election of the adequate objects and classes is fundamental in this respect as they will constitute the basic units in the proof model. Static properties will usually involve only one class, whereas dynamic properties require reasoning about the combined behaviour of several classes.
8. *An object-oriented formal method can be integrated smoothly in the whole development cycle.* Many organizations have already adopted some of the current (informal) object-oriented analysis and design methodologies and have therefore become familiar with the paradigm. The introduction of a higher degree of formality would be much easier if specification languages were based on the same principles.

3. OVERVIEW OF OBJECT-ORIENTED FORMAL SPECIFICATION LANGUAGES

For our purposes, object-oriented specification languages are those which provide at least the three basic object-oriented features (i.e. objects, classes and inheritance) and are based on a semantic model that allows formal manipulation of the specification. The underlying model is usually a well known mathematical framework, such as algebra, first-order logic, temporal logic, set theory, etc. In many cases variations or mixtures of different formalisms are used. A summary of approaches and their formal basis is shown in Table 1.

Note that we have only considered languages whose primary purpose is formal specification. Certain object-oriented programming languages like Eiffel [15] and POOL [16] claim to offer specification capabilities, but since this is not their main objective we have not included them. Equally, popular object-oriented design

TABLE 1. Summary of object-oriented specification languages and their formal underlying model

Name	*Formal Model*
TROLL	Logical framework called *Object Specification Logic*
CSML	Algebraic semantics. Development method based on Jackson System Development
EAM	Entity-Relationships model and Abstract Machines
DisCo	*Joint Actions* and Statecharts
COLD	Operational semantics
MONDEL	Operational semantics. Formal verification method based on Coloured Petri Nets
Object-Z	History-based formal semantics, integrated in the set-theoretic model of Z
OOST	Set theory and *Relational Neutral Systems* ('Rest stays unchanged' semantics)
OOZE	Algebraic semantics derived from the underlying OBJ system
Z++	Z-based semantics for objects and operations. Algebraic for classes and inheritance
MooZ	Z (transformational semantics)
LOOPN	Based on Coloured Petri Nets
CO-OPN	Timed Petri Nets and *Distributed Transition Systems*
VDM++	VDM (transformational semantics)
SDL92	SDL (transformational semantics)

methodologies such as Booch [17] and Rumbaugh *et al.* [18] cannot be strictly considered formal methods because they only provide informal notations.

We have classified object-oriented specification approaches in two main groups. The first one is formed by new languages that have been designed from inception following the paradigm. The second comprises approaches that are more or less based on well-established specification techniques.

3.1. New languages

Within the group of new languages we will concentrate first on a family of new techniques for the object-oriented specification of a particular kind of software systems, i.e. *Information Systems* (IS). The history of object orientation in IS development is quite long, there are plenty of (informal) requirement analysis techniques based on the object paradigm. The need for a higher degree of formality has now been widely recognized and a number of new specification languages have appeared.

Describing a complex dynamic IS may requires the modelling of some aspect of the real-world by storing information about entities, their relationships and the activities that change these relationships. The emphasis is on formally modelling a real-world phenomena and its environment. Since this is an area of research that has grown out of database analysis and design, there is a strong influence of abstract data type (ADT) theories. The ADT theory views objects as values. Every single object in the system is characterized by a unique identifier drawn from the carrier specified in the object class schema.

CSML—Conceptual Model Specification Language [19]—and **TROLL** [20–22] are good examples of current research in the area of formal description of IS. CSML has been developed at Vrije Universiteit Amsterdam. It is property-oriented (as opposed to model based) and allows the description of object classes using structural and behavioural specifications. Reasoning about both aspects is done separately, using standard equational logic. The language is supported by a development method [23], based on the Jackson System Development. The language **TROLL** is very similar in scope and modelling power. However, It provides, a better integration of static and dynamic aspects into class specifications. It also allows the specification of temporal behaviour using built-in predicates for enabledness and occurrence of events, and temporal connectives. The concepts underlying the language are based on the work done in the IS-CORE (Information Systems—COrrectness and REusability) project, sponsored by the European Commission. **EAM** [24] is another language for IS formal specification. It couples ERC+ (an object-based Entity-Relationship model) with the Abstract Machines approach. Systems are described in terms of elementary machines, also called classes, which augment ERC+ entity and relationships types with invariants and behaviour modelling

The remaining approaches in this section can be broadly termed as languages for specification and design of distributed and/or reactive computer systems. The abstraction level of the descriptions generated with these languages is generally lower than the conceptual models developed with IS specification languages. Here, the intention is to describe the structure and behaviour required from a concrete, implementable system.

The group headed by Kurki-Suonio at Tampere University of Technology in Finland has been involved for some time in the development of **DisCo**—Distributed Co-operation [25, 26]—a language for the specification of concurrent and reactive systems. Apart from being object oriented, the approach is also characterized by its authors as *action oriented* [27]. What makes this technique different from others is that instead of describing the behaviour of individual objects, the specification focus on the *joint actions* that the objects are to accomplish in cooperation. The semantic model is state-oriented and permits the direct application of temporal logic for the specification of safety properties [28].

The language **COLD**—Common Object-oriented Language for Design [29]—and its related methodology, SPRINT [30], originated at Philips Research Laboratories in The Netherlands. They are the result of an effort to introduce formalism and rigour into current object-oriented design methods. The language offers good support for the specification of dynamic behaviour by allowing the use of different styles: axiomatic, pre-/post-conditions, algorithmic. However, it lacks some structuring features normally expected from an object-oriented language. This is due to the fact that COLD has been adapted to the object paradigm at a later stage.

The language **MONDEL** [31], on the other hand, offers capabilities that make it truly formal and object-oriented. MONDEL is the product of a joint research project led by von Bochmann at BNR and the Computer Research Institute of Montreal in Canada. Although its intended areas of application are distributed systems and communication protocols [32], the approach is general enough to be used in many other areas. The style of the specifications is essentially algorithmic, but there is also provision for declarative assertions. The language is executable and has formally defined semantics. A development methodology based on the Entity-Relationship model and a formal verification technique [33] using Coloured Petri Nets are also proposed in the MONDEL project.

The area of real-time systems specification is the main target of the language **ENVISAGER**, developed by the GTE Laboratories and Arizona State University. Real-time systems are those which have to respond to external input stimuli within a finite and specified time. Adequate expression of timing constraints is therefore necessary for the specification of such systems. ENVISAGER [34] is an operational language and uses a variation of temporal logic for description of time varying properties of systems. Objects react to events and communicate through asynchronous message passing. Timing constraints specify the maximum, minimum or absolute time between two events of interest.

3.2. Incorporation of object orientation into existing languages

The second group of languages reviewed is formed by object-oriented dialects of well known formal description techniques. The language Z [35] has received a lot of attention (see Stepney *et al.* [36] for a survey), but other techniques such as Lotos [37], Petri Nets [38] and VDM [39] have also been 'affected' by the object-oriented paradigm. Three different approaches are possible:

- Re-interpretation of the language by giving a set of guidelines that allow us to specify the system in an object-oriented fashion. The advantage of this approach is that proof systems and tool support available for the original languages can still be used. However, on the other hand, the extent to which they follow the object-oriented paradigm may be very limited.
- Extension of its syntax with additional constructs. The formal semantics of the new constructs are normally mapped to the semantics of the non object-oriented version (transformational semantics).
- Full transformation into an object-oriented language. This involves the definition of a new language based on its precursor, but not necessarily compatible with it. Syntax and semantics have to be redefined.

We will first look at object-oriented re-interpretations in the next subsection and continue with a review of proper extensions and modifications.

3.2.1. Object-oriented interpretations

The interpretation of formal languages in an object-oriented manner has been an active area of work in the formal methods group led by Cusack, at BT Labs, in the UK. This group developed a set theoretic model that incorporated the basic structural ideas of the object-oriented paradigm [40, 41]. In this basic model classes, inheritance and polymorphism are expressed in terms of partial specifications (class templates), instantiation rules, incremental modification of class templates and a preorder on the set of class templates which 'maps into' the set-theoretic inclusion hierarchy of classes. They claim that by identifying these four concepts in a target specification language, it is possible to interpret the language according to the paradigm. The model has subsequently been used to provide object-oriented interpretations of CSP [42], Lotos [43] and Z [44, 45]. This work was later improved and formalized in the *ISO Reference Model for Open Distributed Processing (ODP)* [46].

Another approach to the object-oriented specification in Lotos is described in Mayr [47]. The behaviour of an object is specified as a Lotos process and gates are used to model the interfaces. Communicating objects can be specified as a parallel composition of processes. This contrasts with Black [48] who sets out an empirical formalism for a state-base specification entity called an *object* and then seeks its interpretation in Lotos. The difficult task of describing inheritance, a key concept in object-orientation, using Lotos constructs seems to be the main problem to overcome [49].

The style proposed by Hall [50] consists of some conventions for writing an object-oriented specification using standard Z. Modelling of object states, use of object identities, expression of the state of the system in terms of the objects it contains and definition of operations in terms of single objects are some of the conventions proposed. Whysall and McDermid [51] describe another approach to specifying objects in Z which is particularly appropriate if the specifications are to be used for subsequent refinement and proof. Objects are described separately by so-called *export* and *body* specifications. The export specifications are algebraic in style and describe the overall behaviour of the object independent of the internal details. These export specifications can thus be used to reason about the behaviour of the object. The body specifications are model-oriented specifications of the constituents parts of each object, in particular their state and methods. These body specification are used as the basis for subsequent refinement.

3.2.1. *Object-oriented versions*

Out of all the object-oriented versions of Z, **Object-Z** [52, 53] is probably the most complete. It originated at the University of Queensland, in Australia, and now is being used in several projects elsewhere. In Object-Z, state and operation schemas are encapsulated into *classes*. The formal model is based upon the idea of a *class history* [54]; a history is a sequential record of the operations, together with the underlying operation states, undergone by an object. A class is subsequently modelled by the set of all possible histories that an object of that class can undergo. By a simple extension of the Z type framework, it is possible to treat classes as types, and hence consider the declaration of variables of a class type.

The language **OOST**—Object-oriented Set-Theoretic [55, 56]—is another specification language similar to Z in that it is based on ZF set theory. OOST takes full account of the fundamental issues of concurrency, synchronization, communication and interference. A very important role is played by the special rule of *historical inference*, whereby only the *minimal* effect (change of state) need actually be specified for each possible event. The *Relational Neutral Systems* [57] gives a semantic model for the analysis of potential behaviour of OOST classes instances. This model, based on relationships between class events and state, provides a formal framework for the reasoning of overlapping occurrences of events.

In contrast to Object-Z and OOST, which keep the set-theoretic semantic model, **OOZE**—Object-oriented Z Environment [58]—adopts a semantics based upon order sorted, hidden sorted algebra. They argue that reasoning about complex set-theoretic specifications is much more difficult than algebraic based ones. The model-based features of Z are maintained but the underlying model is different. This allows them to use the property proof facilities provided by their OBJ3 system [59]. Other Z versions are **Z++** [60] and **MooZ** [61]. The semantic model of Z++ [62] is a combination of algebra, for classes and inheritance, and Z, for methods and objects. Special emphasis is placed on providing proof rules for inheritance as a way of expressing refinement.

Petri Nets are probably one of the most widely used specification techniques. Two object extensions have been found in the literature, **LOOPN** (Language for Object-oriented Petri Nets) [63] and **CO-OPN** (Concurrent Object-oriented Petri Nets) [64]. The former extends the capabilities of Coloured Petri Nets with some object-oriented features. The main innovation is the definition of *token types* and *modules* as classes, with inheritance and polymorphism facilities. The language has been used for specifying and simulating communication protocols. CO-OPN is a much more elaborate approach. Its inventors have combined a modular algebraic specification language to describe data types, and algebraic nets to specify object concurrent behaviour. The semantics of the model [65] is defined using the notion of, which gives a formal framework for specification refinement.

With respect to VDM, the only extension we are aware of is **VDM++** [66]. This approach claims to provide the usual object-oriented modularity, as well as synchronization and parallelism. Its semantics are defined by mapping VDM++ constructs to the standard VDM syntax. There are still no formal rules to deal with concurrency and object constructs.

Finally, SDL [67], the language used by the ITU (International Telecommunications Union) for the formal specification of telecommunication standards, has now been extended to encompass object-oriented notions. The new version, called **SDL92** [68], tries to keep changes to a minimum. The modifications introduced concentrate mainly on the provision of a generalization and specialization mechanism for both block and process constructs. A much clearer distinction is also made between type definitions and their instantiation.

4. FINAL REMARKS

The review presented in the previous section certainly gives us an idea of the amount of research that is currently being done on the combination of formal methods and object-orientation. However, are object-oriented formal languages any better that conventional ones? It is not easy to answer this question since the majority of the approaches reviewed are still very much under development and there is not much experience in their practical use. There is, however, little doubt among formal methods researchers and practitioners that object-orientation adds a new dimension to their models, bringing them closer to 'real world'. At this point, we should not forget that a trade-off exists between the number of features provided by a language and the ability to manipulate mathematically the specifications produced with it. The adoption of the object paradigm brings powerful abstraction and structuring mechanisms but at the same time creates some additional problems. Classes, inheritance, composition and other usual constructs need to be formally defined and integrated within the semantic model of the language if we want to use them in a truly formal methodology. This represents a major challenge for the language designers.

The popularity of the object-oriented paradigm itself can be also a source of potential problems. The number of interpretations and the ever increasing and sometimes contradictory terminology makes it difficult to integrate or even compare different approaches. Attempts to introduce formalism into the paradigm have often been hindered by this anarchy of terms and concepts. Although standardization communities and other international initiatives are working on the provision of agreed formal definitions, the diversity of views and models is still an ongoing problem.

ACKNOWLEDGEMENTS

The financial support provided to the first author by the European Commission through the research programme *Human Capital and Mobility* is acknowledged.

REFERENCES

[1] de Champeaux, D., America, P., Coleman, D., Duke, R., Lea, D. and Leavens, G. (1991) Formal techniques for OO software development. In *Proc. Object-Oriented Programming, Systems and Languages Conf. (OOPSLA'91)*, pp. 166–170. ACM Press, New York.

[2] Hogg, J. (1992) Object-Oriented Formal Methods (Report on ECOOP'91 Workshop W3). *OOPS Messenger*.

[3] Lano, K. and Haughton, H. (1993) *Object Oriented Specification Case Studies (Object Oriented Series)*. Prentice-Hall, Englewood Cliffs, NJ.

[4] Wilson, J. (1993) Formal methods and object oriented analysis. *Br. Telecom Technol. J. (Special Issue on Object Oriented Technology and its Applications)*, **11**, 18–31.

[5] diGiovanni, R. and Iachini, P. L. (1990) HOOD and Z for the development of complex systems. In Bjorner, D., Hoare, C. and Langmaack, H. (eds), *VDM'90: VDM and Z: Formal Methods in Software Development, LNCS 428*, pp. 262–289. Springer-Verlag, Berlin.

[6] Cheon, Y. and Leavens, G. T. (1994) The Larch/Smalltalk interface specification languages. *ACM Trans. Software Eng. Methodol.*, **3**, 221–253.

[7] Cheon, Y. and Leavens, G. T. (1994) A quick overview of Larch/C++. *J. Object-Oriented Program.*, **76**, 39–49.

[8] Wills, A. (1991) Capsules and types in Fresco: Smalltalk meets VDM. In *Proc. ECOOP'91, Eur. Conf. on Object-Oriented Programming, LNCS 512*, pp. 59–76. Springer-Verlag, Berlin.

[9] Wills, A. (1992) *Formal Specification of Object-Oriented Programs*, Ph.D. thesis, University of Manchester.

[10] Wahls, T., Baker, A. L, and Leavens, G. T. (1994) *The Direct Execution of SPECS-C++: A Model-Based Specification Language for C++ Classes*, Technical Report 94-02b, Department of Computer Science, Iowa State University.

[11] Lano, K. and Haughton, H. (1992) Reasoning and refinement in object-oriented specification languages. In *Proc. ECOOP'92, Eur. Conf. on Object Oriented Programming, LNCS 615*. Springer-Verlag, Berlin.

[12] Whysall, P. and McDermid, J. (1991) Object-Oriented Specification and Refinement. In Morris, J. and Shaw, R. (eds), *Proc. 4th Refinement Workshop, Workshops in Computing*, pp. 151–184. Springer-Verlag, Berlin.

[13] Bailes, C. and Duke, R. (1991) The ecology of class refinement. In Morris, J. and Shaw, R. (ed.), *Proc. 4th Refinement Workshop, Workshops in Computing*, pp. 185–196. Springer-Verlag, Berlin.

[14] Jones, C. (1993) Reasoning about interference in an object-based design method. In *FME'93: Industrial-Strength Formal Methods, LNCS 670*, pp. 1–18. Springer-Verlag, Berlin.

[15] Meyer, B. (1988) *Object Oriented Software Construction*. Prentice-Hall, Englewood Cliffs, NJ.

[16] America, P. (1987) POOL–T: a parallel object-oriented language. In Yoneza, A. and Tokoro, M. (ed.), *Object-Oriented Concurrent Programming*, pp. 199–220. MIT Press, Cambridge, MA.

[17] Booch, G. (1991) *Object Oriented Design with Applications*, Benjamin-Cummings, London.

[18] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. (1991) *Object Oriented Modeling and Design*. Prentice-Hall, Englewood Cliffs, NJ.

[19] Wieringa, R. (1990) Equational Specification of Dynamic Objects. In Meersman, R., Kent, W. and Khosla, S. (eds), *Object-Oriented Databases: Analysis, Design, and Construction (DS-4)*, pp. 415–438. North-Holland, Amsterdam.

[20] Jungclaus, R., Saake, G., Hartmann, T. and Sernadas, C. (1991) *Object Oriented Specification of Information Systems: The TROLL Language*, Technical Report 91–04, Technische Universität Braunschweig, Germany.

[21] Saake, G., Jungclaus, R. and Ehrich, H.-D. (1992) Object-oriented specification and stepwise refinement. In Meer, D., Heymer, V. and Roth, R. (eds), *Proc. IFIP TC6 Int. Workshop on Open Distributed Processing (IFIP Trans. C1)*, pp. 99–121. North-Holland, Amsterdam.

[22] Hartmann, T. and Jungclaus, R. (1991) Abstract description of distributed object systems. In Tokoro, M., Nierstrasz, O. and Wegner, P. (eds), *Proc. ECOOP'91 Workshop on Object-Based Concurrent Computing, LNCS 512*, pp. 227–244. Springer-Verlag, Berlin.

[23] Wieringa, R. J. (1991) Steps towards a method for the formal modeling of dynamic objects. *Data and Knowledge Eng.*, **6**, 509–540.

[24] Auddino, A. (1994) Formal modelling of objects and subsystems in an information systems framework. In *Proc. 4th Euro-Japanese Seminar on Information Modelling and Knowledge Bases*. IOS Press, Amsterdam.

[25] Järvinen, H.-M. and Kurki-Suonio, R. (1991) DisCo specification language: marriage of actions and objects. In *Proc. 11th Int. Conf. on Distributed Computer System*, pp. 142–151. IEEE Computer Society Press, New York.

[26] Järvinen, H.-M., Kurki-Suonio, R., Sakkinen, M. and Systä, K. (1989) Object-oriented specification of reactive systems. In *Proc. 12th Int. Conf. on Software Engineering*, pp. 63–71. IEEE Computer Society Press, New York.

[27] Kurki-Suonio, R. B. R. (1988) Distributed cooperation with action systems. *ACM Trans. Programming Languages Syst.*, **10**, 513–554.

[28] Järvinen, H.-M. and Kurki-Suonio, R. (1990) *The DisCo Language and Temporal Logic of Actions*, Technical Report 11, Tampere University of Technology, Software Systems Laboratory.

[29] van der Linden, F. (1993) *An Object-Oriented Approach to SPRINT*, Technical Report RWR-508-re-92413, Philips Research Laboratories, The Netherlands.

[30] van der Linden, F. (1993) *Object-Oriented Specification in COLD*, obtained from the author.

[31] von Bochmann, G., Barbeau, M., Lecomte, L., Williams, N., Erradi, M. and Mondain-Monval, P. (1991) *Mondel: An Object-oriented Specification Language*, Technical Report CRIM-91/10-03, CRIM/BNR/Université de Montréal.

[32] von Bochmann, G., Poirier, S. and Mondain-Monval, P. (1992) Object-oriented design for distributed systems: the OSI directory example. In *Proc. IFIP Int. Conf. on Upper Layer Protocols, Architectures and Applications*.

[33] Barbeau, M. and von Bochmann, G. (1991) *Formal*

Verification of Object-Oriented Specifications Using Colored Petri Nets, obtained from the authors.

[34] Diaz-Gonzalez, J. and Urban, J. (1991) Language aspects of ENVISAGER: an object-oriented environment for the specification of real-time systems. *Comp. Languages*, **16**, 19–37.

[35] Spivey, J. (1989) *The Z Notation: A Reference Manual, International Series in Computer Science*. Prentice-Hall, Englewood Cliffs, NJ.

[36] Stepney, S., Barden, R. and Cooper, D. (eds) (1992) *Object Orientation in Z, Workshops in Computing*. Springer-Verlag, Berlin.

[37] Bolognesi, T. and Brinksma, E. (1987) Introduction to the ISO specification language LOTOS. *Computer Networks ISDN Syst.*, **14**, 25–29.

[38] Peterson, J. (1981) *Petri Net Theory and the Modelling of Systems*. Prentice-Hall, Englewood Cliffs, NJ.

[39] Jones, C. (1986) *Systematic Software Development Using VDM*. Prentice-Hall, Englewood Cliffs, NJ.

[40] Cusack, E. (1988) Fundamental aspects of object oriented specification. *Br. Telecom Technol. J.*, **6**, 76–81.

[41] Cusack, E. (1991) Refinement, conformance and inheritance. *Formal Aspects Comput.*, **3**, 129–141.

[42] Cusack, E. (1990) Formal object oriented specification of distributed systems. In *Proc. Workshop on Specification and Verification of Concurrent Systems*, pp. 71–81. Springer-Verlag, Berlin.

[43] Cusack, E., Rudkin, S. and Smith, C. (1990) An object-oriented interpretation of LOTOS. In Vuong, S. (ed.), *Formal Description Techniques II, FORTE'89*, pp. 211–226. North-Holland, Amsterdam.

[44] Cusack, E. (1992) Object oriented modelling in Z for open distributed systems. In de Meer, J. (ed.), *Proc. Int. Workshop on ODP*. North-Holland, Amsterdam.

[45] Rudkin, S. (1992) Modeling information objects in Z. In de Meer, J. (ed.), *Proc. Int. Workshop on ODP*. North-Holland, Amsterdam.

[46] ISO/CCITT (1993) ISO/IEC JTC1 SC21 N7524 (CD 10746-2.2) RM-ODP Part 2: Descriptive Model.

[47] Mayr, T. (1989) Specification of object-oriented systems in LOTOS. In Turner, K. (ed.), *Formal Description Techniques, FORTE'88*, pp. 107–119. North-Holland, Amsterdam.

[48] Black, S. (1989) *Objects and LOTOS*, Technical Report, Hewlett-Packard Laboratories, Bristol.

[49] Rudkin, S. (1992) Inheritance in LOTOS. In Park, K. and Rose, G. (eds), *Formal Description Techniques IV, FORTE'91*, pp. 409–424. North-Holland, Amsterdam.

[50] Hall, J. (1990) Using Z as a specification calculus for object-oriented systems. In Bjorner, D. Hoare C. and Langmaack H. (ed.), *VDM'90: VDM and Z: Formal Methods in Software Development, LNCS 428*, pp. 290–318 Springer-Verlag, Berlin.

[51] Whysall, P. and McDermid, J. (1991) An Approach to object-oriented specification using Z. In Nicholls, J. (ed.), *Z User Workshop: Proc. 4th Ann. Z User Meeting, Workshops in Computing*, pp. 193–215. Springer-Verlag, Berlin.

[52] Carrington, D. A., Duke, D., Duke, R., King, P., Rose, G. and Smith, G. (1990) Object-Z: an object oriented extension to Z. In Vuong, S. (ed.), *Formal Description Techniques II, FORTE'89*, pp. 281–296. North-Holland, Amsterdam.

[53] Duke, R., King, P., Rose, G. and Smith, G. (1991) The Object-Z specification language. In Korson, T., Vaishnavi, V. and Meyer B. (eds), *Technology of Object-Oriented Languages and Systems: TOOLS 5*, pp. 465–483. Prentice-Hall, Englewood Cliffs, NJ.

[54] Duke, D. and Duke, R. (1990) Towards a semantics for Object-Z. In Bjorner, D., Hoare, C. and Langmaack, H. (eds), *VDM'90: VDM and Z: Formal Methods in Software Development, LNCS 428*, pp. 242–262. Springer-Verlag, Berlin.

[55] Schuman, S. and Pitt, D. (1987) Object-oriented subsystem specification. In Meertens, L. (ed.), *Program Specification and Transformation*, pp. 313–341. North-Holland, Amsterdam.

[56] Schuman, S., Pitt, D. and Byers, P. (1990) *Object-Oriented Process Specification*, Technical Report CS-90-01, Department of Mathematical and Computing Sciences, University of Surrey.

[57] Pitt, D. and Byers, P. (1991) *The Rest Stays Unchanged (Concurrency and State-Based Specification)*, Technical Report CS-91-07, Department of Mathematical and Computing Sciences, University of Surrey.

[58] Alencar, A. and Goguen, J. (1991) OOZE: an object-oriented Z environment. In America, P. (ed.), *Proc. ECOOP'91, Eur. Conf. on Object-Oriented Programming, LNCS 512*, pp. 88–95. Springer-Verlag, Berlin.

[59] Goguen, J., Meseguer, J., Winkler, T., Futatsugi, K. and Jouannaud, J.-P. (1992) *Introducing OBJ3*. Technical Report SRI-CSL-92-03. Computing Science Laboratory, SRI International, USA.

[60] Lano, K. (1991) Z++: an object-oriented extension to Z. In Nicholls, J. (ed.), *Z User Workshop: Proc. 4th Annu. Z User Meeting, Workshops in Computing*, pp. 151–172. Springer-Verlag, Berlin

[61] Meira, S. and Cavalcanti, A. (1991) Modular object-oriented Z specifications. In Nicholls, J. (ed.), *Z User Workshop: Proc. 4th Annu. Z User Meeting, Workshops in Computing*, pp. 173–192. Springer-Verlag, Berlin.

[62] Lano, K. and Haughton, H. (1992) An Algebraic Semantics for the Specification Language Z++. In *Proc. Algebraic Methodology and Software Technology Conf. (AMAST'91)*, Springer-Verlag, Berlin.

[63] Lakos, C. and Keen, C. (1991) Modelling layered protocols in LOOPN. In *Proc. 4th Int. Workshop on Petri Nets and Performance Models*, pp. 106–115. IEEE Computer Society, New York.

[64] Buchs, D. and Guelfi, N. (1992) Distributed system specification using CO-OPN. In *Proc. 3rd Workshop on Future Trends of Distributed Computing Systems*. pp. 26–33. IEEE Computer Society, New York

[65] Buchs, D. and Guelfi, N. (1991) CO-OPN: a concurrent object oriented Petri net approach. In *Proc. 12th Int. Conf. on Theory and Applications of Petri Nets*, Gjern, Denmark.

[66] Dürr, E. and van Katwijk, J. (1992) VDM++: a formal specification language for object-oriented designs. In *Proc. ComEuro 1992*, pp. 214–219. IEEE Computer Society Press, New York.

[67] Saracco, R. and Tilanus, P. (1987) CCITT SDL: overview of the language and its applications. *Computer Networks ISDN Syst.*, **13**, 65–74.

[68] CCITT (1991) *Tutorial on Object-Oriented SDL*. Study group X (Maintenance of SDL) Contribution D.74.