A Relational Framework for the Integration of Specifications

Eerke Boiten and John Derrick
University of Kent at Canterbury
Canterbury Kent, CT2 7NF, UK
E.A.Boiten@ukc.ac.uk, J.Derrick@ukc.ac.uk

2003

Abstract

We describe a framework for viewpoint specification using formal specification languages. In order to establish consistency and to further develop specifications, specifications need to be integrated ("unified").

This integration is not defined in terms of their semantics, but more abstractly in terms of, so-called, development relations, which represent acceptable "developments" (e.g. refinements) of each of the viewpoint specifications. The framework is motivated by its instantiations with a number of specification languages (e.g., LOTOS and Z) and different development relations.

Keywords

Viewpoint specification, partial specification, consistency, formal methods, Z, LOTOS.

1 Introduction

The use of viewpoints is often advocated as a solution for the problems of scale arising from the development of large systems. However, the approach comes with its own problems. By assuming a formal specification context, we can clearly identify and rigorously analyse these problems, and give a framework for their solution.

In this section, we first describe the wider software engineering context of viewpoint specification, and then describe the main problems informally. Section 2 lists the requirements for a framework for their solution. A number of central design decisions for the framework are discussed in Section 3. A simplified version of the framework is presented in full in Section 4, with a number of its instantiations. Section 5 sketches the full framework, and Section 6 concludes by discussing related work.

1.1 Context

For projects involving the specification and development of large systems, the *structuring* of their descriptions is crucial to the project's success. Traditionally, systems were decomposed according to functionality – modern approaches favour, in addition to this, decompositions according to "aspects" or "viewpoints" (Finkelstein et al., 1992). These may cover non-functional aspects like security, safety, distribution, timing, etc. Typically, appropriate specialised languages would be used for each of those aspects. Alternatively, these viewpoints may be views of the system's functionality from different participants (for example, a borrower's and a library manager's view of the action of borrowing a book in a library system). We use the term *partial specification* (Zave and Jackson, 1991) to include both those possibilities.

A prominent method with features of partial specification is the Unified Modeling Language UML (Rumbaugh et al., 1999) which allows systems to be described using diagrams and notations of various kinds. For example, the effects and ordering of operations can be (partially) deduced from information provided in statecharts, in OCL annotations, and from object interaction diagrams, all of which can feature in a UML specification. However, none of these takes precedence or is assumed to fully characterise the behaviour of the system being specified. For a further discussion of these issues in UML, see (Derrick et al., 2002).

The main inspiration for our work in this area, however, has been the Open Distributed Processing (ODP) standard (ISO/IEC/ITU-T, 1995-98; Putman, 2000). This defines a viewpoints framework for the description and development of distributed systems, using a fixed collection of five viewpoints with predefined scopes. For example, the *enterprise* viewpoint is concerned with communities, rôles and policies; the *computational* viewpoint describes the system as a collection of distributed objects with their interfaces. A case study in using ODP in the domain of air traffic control is the ECHO study presented in (Eurocontrol, 1997), and analysed in (Derrick and Boiten, 2002; Taylor et al., 2002). An overview of our approach, using a communication protocol case study, is given in (Boiten et al., 2000).

These viewpoint methods are aimed at large projects, which will be undertaken by teams of people. For that reason, they should also account for independent *development* of the partial specifications. It should be clear that this independence is limited by the degree of coherence and overlap between the partial specifications, as described below.

1.2 Issues

The problems raised by a partial specification approach revolve around a single fundamental issue. Namely, the tension caused by, on the one hand, the partial specifications needing to be "independent", and on the other hand, the partial specifications all describing aspects of a *single* envisaged system. In particular, as soon as *multiple* partial specifications constrain the *same* entity in the system,

the possibility arises that the constraints imposed are actually contradictory, i.e., *inconsistent*. The value of aspect-oriented methods is said to lie in the relative orthogonality of the partial specifications. However, there would not be much of a software development problem if the partial specifications did not ultimately interfere. If they did not, we could develop the functional specification first, and be assured that it would always be possible to superimpose the security, real-time, etc. requirements on the final product afterwards. Clearly this is not a realistic expectation, so consistency is going to be a serious issue in any partial specification approach.

Two main classes of consistency can be distinguished. First there is what we call *structural* consistency: if a viewpoint refers to something defined elsewhere, that definition should indeed exist in another viewpoint. (For example, when a method call on a particular object occurs in an interaction diagram, this method is indeed defined on the object's class.) This consistency problem is largely solved; checking for this is already supported by any reasonable CASE tool.

The more challenging type of consistency is usually called *behavioural* consistency, but we might also call it *constraint consistency*. This is defined informally as: the constraints placed by the partial specifications on an entity of the system are jointly satisfiable. Such an "entity" might be an object, a data item, an operation, or a behaviour. For example, in UML, constraint consistency on a behaviour might be violated when a statechart allows action a only to occur after action b, whereas an object interaction diagram puts b before a.

Note that we have defined constraint consistency in terms of entities of the system, rather than in terms of elements of the partial specifications. This is a crucial distinction to make, as we can hardly expect the views of the same element from different viewpoints to be identical. If it is indeed given the same name in two viewpoints, its representation (its *type*) may still be different. Consider, for example, a service in a distributed system which is implemented transparently by any of a number of servers. From the client's viewpoint, the service may well be provided by a single virtual server. An abstract view of the server side may consider a *set* of servers – a more concrete view, which takes load balancing into account, may have an even more detailed representation of the collection of servers, e.g., a queue. Nevertheless, all of these views represent the same entity in the eventual system. Thus, such a difference in representation is *not* a structural inconsistency. Another apparent structural inconsistency is when a single operation in one viewpoint corresponds to a sequence of operations in another, or when an entity has one name in one viewpoint, and a different name in another.

All of these relations between elements of the various partial specifications need to be documented – as the "glue" that holds the partial specifications together. Constraint consistency (and even structural consistency) is to be considered modulo these relations. Using ODP terminology, we call such relations the *correspondences* between the partial specifications. An elementary implicit level of correspondence is provided by the use of identical names in partial specifications, as used in the ECHO study, but this is rarely sufficient.

Correspondences also need to cross language boundaries, as partial specifications need not use the same specification language. In practice, correspondences may be implemented by reference to a common model of the system's domain knowledge, comparable to a data dictionary. Such a common model may well be an informal one. For example, in the ECHO case study a base for the correspondences between the viewpoints is found in the developers' knowledge of the elements of air traffic control software. A notable disadvantage of using a common model is that all viewpoint developers need to be involved in its definition and development, even when particular notions only concern a small number of viewpoints. As a consequence, this puts the intended advantages of a de-centralised development at risk.

A related problem caused by the desire for de-centralisation is the tension between consistency checking and independent development. If a number of viewpoints have been found to be consistent, and subsequently independently developed further, consistency will *not* be guaranteed. (This will be even more obvious in the formal setting presented later.) As a consequence, consistency is only guaranteed immediately after a successful consistency check. If consistency is continuously required, a check needs to be made after every change to every viewpoint – which is clearly undesirable. Thus, a compromise needs to be made. One possible approach is the use of inconsistency-tolerant methods (Nuseibeh, 1996; Miarka et al., 2002), but we will not consider this further in this paper.

2 Objectives

We will in this paper develop a framework that aims to support the use of partial specification in a practical and rigorous way, taking into account the problems outlined above. The framework being rigorous implies that we consider only formalised notations[1], as it would be hard to pin down a notion of consistency otherwise.

The framework requires the following features.

- It should allow for the definition of correspondence relations between partial specifications, potentially in different specification languages.
- Consistency needs to be checked between any number of partial specifications. As a witness of a successful consistency check, an integration of the partial specifications involved (a "*unification*") needs to be produced. This could be used as a basis for implementation of the system, and also to define consistency between n specifications incrementally, in terms of consistency between pairs of specifications (Bowman et al., 1999). An unsuccessful consistency check should provide *feedback*, exhibiting the sources of inconsistency in terms understandable to the specifiers. Thus,

[1] This does not wholly exclude UML – parts of it have been formalised (Evans et al., 1999). However, we argue that a notion of refinement is essential for partial specification, and for UML this has not been fully explored yet (Paech and Rumpe, 1994).

it should mainly use the vocabularies of the partial specifications and of the correspondences between them.

- The framework should support independent evolution of the partial specifications.

3 Approaches to the Framework

The requirement that the specification notations be *formalised* implies that they have a formal semantics of some sort. This semantics might be viewed as a basis for consistency checking, but this may not lead to a practical approach. First, the semantic domains of the different notations used are likely to be different as well. This might be resolved by embedding all different semantics in some unifying framework, but this makes the second problem more prominent: the result of "intersecting the semantics" of two partial specifications is not likely to be in terms understandable to the specifiers. The same goes for the feedback of failed inconsistency. For example, when the common semantics is predicate logic as in (Zave and Jackson, 1993), conjunction is used to combine specifications, and inconsistency appears as "false".

For this reason, we aim at methods for integrating specifications which are more *syntactic* in nature.

The most intuitive definition of consistency between partial specifications is nevertheless in terms of a particular common "semantics". Every partial specification constrains the eventual implementation of the system – to put it differently, for every partial specification there is a set of acceptable implementations. Consistency is then joint implementability of the partial specifications: the sets of acceptable implementations should have a non-empty intersection. However, this definition is neither practical nor abstract. It is impractical, because the sets of acceptable implementations in any given programming language are likely to be huge or even infinite, and due to many programs having the same semantics, intersections of such sets will still be huge. Also, such intersections being empty will not provide useful feedback in case of inconsistency.

Abstraction, however, is the key to improving on this definition. Most specification languages do not have a direct implementation relation, but they do have notions of "refinement" or "development". Such refinement relations are normally characterised by the fact that a refined specification cannot be distinguished from its original – which amounts to its implementations forming a subset. For some specification languages, multiple refinement relations exist, reflecting varieties in the ways in which specifications may be "distinguished". (For example, refinement relations in process algebra (Hoare, 1985; Milner, 1989) correspond to varying notions of testing (Henessy, 1988).) In a partial specification approach, one can imagine specifications in the same language being interpreted differently according to the particular rôle of the viewpoint at hand – it may describe permissible behaviour, required behaviour, etc. Examples of the various rôles of process algebra specifications for ODP viewpoints,

and how their consistency is checked, may be found in (Steen et al., 1999). From working out examples of viewpoint specification and consistency in Z (Boiten et al., 1999), we also found that, depending on the interpretation of viewpoints, different and non-standard refinement relations are needed (Derrick and Boiten, 2001).

All in all, it is clear that it will actually not be enough to consider just the viewpoint specifications and their correspondences. We will also need to take into account the permissible developments of each viewpoint, as indicated by its associated "*development relation*".

4 Simple Framework

In order to present the central notions, we first present a somewhat simplified version of our framework. The assumptions made in this version are:

- all specifications are written in the same specification language $\mathcal{L}$;
- the correspondence relation is characterised by the use of identical names only.

The first assumption is not particularly strong, as it would be possible to consider an embedding of all specification languages into a single "universal" one.

For brevity and clarity, we will call the combination of a partial specification *with* its associated development relation a *pspec*, and a set of such pspecs a *pspecset*.

Definition 1 (Pspec and pspecset) A *pspec* is a tuple $(spec, dev)$ such that $spec \in \mathcal{L}$ and dev is a member of $\mathcal{L} \leftrightarrow \mathcal{L}$, the set of all relations on $\mathcal{L}$. A *pspecset* is a set of pspecs. □

A pspecset is consistent if all of its pspecs have a shared common development.

Definition 2 (Consistency) A pspecset $\{(spec_i, dev_i)\}_{i \in I}$ is *consistent* iff

$$\exists\, s : \mathcal{L} \bullet \forall\, i : I \bullet (spec_i, s) \in dev_i$$

In that case, such an s is called a *unification* of the pspecset. □

A least unification (most general unification) of two pspecsets is defined as a "least upper bound" of their unifications:

Definition 3 (Least unification) The pspec $(spec, dev)$ is a *least unification* of pspecs $(spec_1, dev_1)$ and $(spec_2, dev_2)$ iff

$$\begin{aligned} &dev = dev_1 \cap dev_2 \land (spec_1, spec) \in dev_1 \land (spec_2, spec) \in dev_2 \land \\ &\forall\, spec' : \mathcal{L} \bullet ((spec_1, spec') \in dev_1 \land (spec_2, spec') \in dev_2) \\ &\qquad \Rightarrow (spec, spec') \in dev \end{aligned}$$

□

Depending on the language and collection of development relations used, least unifications may or may not exist.

The stepwise development of a pspecset (modelling stepwise system development) is defined as a transition system, as follows.

Definition 4 (Stepwise development; Termination)
Stepwise development is a transition system on pspecsets, containing the following transitions $\longrightarrow$ for every pspecset D:

1. If $(spec, dev) \in D \wedge (spec, spec') \in dev$ then $D \longrightarrow D'$ where
$$D' = (D - \{(spec, dev)\}) \cup (spec', dev)$$
2. If $(spec_1, dev_1) \in D$ and $(spec_2, dev_2) \in D$ and $(spec, dev)$ is a least unification of $(spec_1, dev_1)$ and $(spec_2, dev_2)$ then $D \longrightarrow D'$ where
$$D' = (D - \{(spec_1, dev_1), (spec_2, dev_2)\}) \cup (spec, dev)$$

A pspecset D is *terminating* if

$$\exists\, spec : \mathcal{L};\ dev : \mathcal{L} \leftrightarrow \mathcal{L} \bullet D \longrightarrow^* \{(spec, dev)\}$$

where $\longrightarrow^*$ is the reflexive and transitive closure of $\longrightarrow$. □

The transition system $\longrightarrow$ represents arbitrary steps in which partial specifications are developed or unified. A development objective terminates if *at least one* sequence of such steps reduces it to a single specification. One would assume that in that case, the pspecset is consistent. This is indeed true under (mostly sensible) restrictions on the development relations used, see (Bowman et al., 1999; Bowman et al., 2002) for a detailed analysis.

Theorem 1 *When least unifications exist for all consistent pairs of pspecs, then consistency of a pspecset implies termination. When, in addition, all development relations involved are preorders, then termination also implies consistency.* □

Note that in this model, implicitly we are still considering an alternative "semantics" of a partial specification: its semantics may be viewed as the set of all its images under the development relation. The restrictions on development relations in Theorem 1 amount to these sets being upward closed and having minima.

A final element of the framework is *feedback*. This is provided through so-called *projections*. A common strategy for debugging programs is to reduce a buggy program step by step until one is left with a "smallest" program still exhibiting the same bug. The lack of irrelevant detail surrounding it then makes it easier to analyse and remove the bug. Projections have the same rôle for partial specifications: they reduce inconsistent specifications (in analogous ways) leaving us with "smaller" but still inconsistent viewpoints. In order for projections to produce meaningful results, they need to be monotonic with respect to the development relations involved.

Definition 5 (Admissable projection) A *projection* is a partial function from $\mathcal{L}$ to $\mathcal{L}$. A projection f is *admissable* for a pspecset $\{(spec_i, dev_i)\}_{i \in I}$ iff f is applicable to all specifications and their possible developments:

$$\forall\, i : I \bullet spec_i \in \mathrm{dom}\, f$$
$$\forall\, i : I;\ s : \mathcal{L} \bullet (spec_i, s) \in dev_i \Rightarrow s \in \mathrm{dom}\, f$$

and, in addition, f is monotonic with respect to all development relations dev_i in D:

$$\forall\, i : I;\ s, s' : \mathcal{L} \bullet (s, s') \in dev_i \Rightarrow (f(s), f(s')) \in dev_i$$

The application of a projection f to a pspecset D is denoted by $f(D)$, and defined by elementwise application of f to all specifications in D; if any of these are outside the domain of f, then $f(D)$ is not defined. □

Admissability of a projection fully depends on the development relations used. In the context of stepwise development as defined above, it is sufficient to check that the projection is admissable for the initial pspecset. This is because monotonicity with respect to two relations implies monotonicity with respect to their intersection.

The composition of two admissable projections is itself an admissable projection. Thus we can define a pre-order on admissable projections by

$$f \leq_D g \quad \equiv \quad \exists\, h \bullet h(f(D)) = g(D)$$

for f and g admissable projections on a given D, and h admissable on $f(D)$.

Admissable projections f necessarily preserve consistency – if s is a unification of D, then $f(s)$ is a unification of $f(D)$. However, they do not necessarily preserve *in*consistency, as they may actually remove the conflicting requirements. The most useful feedback from an inconsistent pspecset is its image under a maximal inconsistency preserving projection.

Definition 6 (Maximal inconsistency preserving) The projection f is *maximal inconsistency preserving* for an inconsistent pspecset D iff f is admissable on D, $f(D)$ is inconsistent, and of all projections satisfying those properties, f is maximal with respect to the ordering $\leq_D$. □

Maximal projections need not exist, and they are likely not to be unique (consider bijections h in the definition of $\leq_D$).

This framework has been instantiated for the process algebra LOTOS (Bolognesi and Brinksma, 1988) with a number of development relations in (Steen et al., 1999). It has also been instantiated, extended with correspondence relations, for the Z notation (Spivey, 1992) with its standard refinement relation, see (Boiten et al., 1999). The Z instantiation of consistency checking operates on a schema-by-schema basis, so projections (from the whole specification to single schemas) implicitly play a large rôle. Further useful projections would hide state components which do not contribute to the inconsistency. The relationships between various non-standard development relations for Z are explored in (Derrick and Boiten, 2001).

5 Full Framework

The full framework contains a number of additional elements, and some of its definitions are generalisations of the corresponding ones in the simple framework. This paper only sketches some of these, and the issues arising from them, which are still subject of further research.

The first issue is heterogeneity. Disregarding correspondences for the moment, the definition of consistency remains as in Definition 2, except that the unification of all pspecs can no longer be taken from the single language $\mathcal{L}$. The notation that it *does* belong to can be deduced from the ranges of the development relations involved: it should contain the intersection of all those. Thus, there is an issue of *type-correctness* for development relations in a pspecset.

The need for the development relations to have a common co-domain carries the risk of reintroducing the problems indicated earlier for a "common semantics" approach. A solution to this lies in considering additional structure in the development relations. For example, if the common co-domain of the development relations is an implementation notation $\mathcal{I}$, one might consider only development relations of the form $dev_i \fatsemi imp_i$ where dev_i is a relation on a specification notation $\mathcal{L}_i$, and imp_i is a function from $\mathcal{L}_i$ to $\mathcal{I}$, representing the transliterations of a subclass of "implementable" specifications into their implementations. For Z, this subclass could be all operations where all after-state components and outputs are defined by equations relating them to before-state components and inputs – these naturally correspond to an assignment-based imperative implementation. For a process algebra, these could be deterministic "regular" processes (i.e., whose definition has the pattern of a deterministic regular grammar), which correspond to deterministic automata.

In a specialised setup like that, unifications between specifications in the same language can be defined at the level of abstract development relations as before. Also, translations between specification notations can be defined and verified without reference to a common semantics. Consider a translation $trans_{i,j}$ from $\mathcal{L}_i$ to $\mathcal{L}_j$. If

$$\begin{aligned} trans_{i,j} \fatsemi dev_j &\subseteq dev_i \fatsemi trans_{i,j} \\ trans_{i,j} \fatsemi imp_j &\subseteq dev_i \fatsemi imp_i \end{aligned}$$

and dev_i is transitive, then it is a sound development step to replace the pspec $(spec_i, dev_i \fatsemi imp_i)$ by its translation $(trans_{i,j}(spec_i), dev_j \fatsemi imp_j)$. A translation from LOTOS to Object-Z is given in (Derrick et al., 1999). The first condition above, compatibility of development relations, is investigated for process algebras and Z-like languages in (Derrick et al., 1996; Bowman and Derrick, 1999).

Another challenge is posed by the introduction of *correspondence relations* between the viewpoints. The single-language model with correspondence relations for Z is described in detail in (Boiten et al., 1999). From that, it is clear that having correspondences between more than two viewpoints induces a requirement to ensure consistency *between* those correspondences. In addition, a

data refinement step in one viewpoint may lead to a modification in its correspondence with another viewpoint, as it might have been referring to a state component that has been removed by data refinement.

A potential solution for this problem has its base in category theory, inspired by the approaches to composition of specifications of Specware (Srinivas and Jüllig, 1995) and (Fiadeiro et al., 1997). Every relation can be factored as the composition of the inverse of a function and a function (a "span"). In such a decomposition of a correspondence between two viewpoint specifications, the "intermediate" specification contains essentially representations of the elements in the viewpoints that need to be related. In a development step, the intermediate can be left unchanged, modifying only its images in the viewpoints, thereby establishing a new correspondence relation. Unification is then a pushout of the span diagram; correspondences can be combined as pullbacks.

6 Related Work

The use of Z for partial specification, and consequences for consistency checking were first investigated by Ainsworth et al. (1994), their approach was based on refinement. P. Zave and M. Jackson (1991; 1993; 1996) investigated the use of combinations of Z and other notations for partial specification, using predicate logic as a common semantics. Both these approaches lack the notion of a correspondence relation. A different approach was taken by D. Jackson (1995), using a correspondence relation as the mechanism for composing Z specifications, however not necessarily establishing refinements.

A relational approach to combining specifications is described by (Boudriga et al., 1992; Frappier et al., 1995). Große-Rhode (2001) defines "transformation systems" as a common semantic domain. Due to its expressiveness and some built-in redundancy, it appears a promising basis for combining specifications and integrating methods. Many other approaches to this are described in the proceedings of the Integrated Formal Methods conference series, e.g., (Butler et al., 2002).

Acknowledgements

Our research in this area was supported by EPSRC through the grants "Cross Viewpoint Consistency in ODP", "OpenViews" and "A Constructive Framework for Partial Specification". We thank our colleagues in these projects: Marius Bujorianu, Peter Linington, Howard Bowman, Maarten Steen, and Chris Taylor, for their contributions.

References

Ainsworth, M., Cruickshank, A. H., Wallis, P. J. L., and Groves, L. J., 1994, "Viewpoint specification and Z," *Information and Software Technology*, Vol. 36(1),

pp. 43–51.

Boiten, E. A., Derrick, J., Bowman, H., and Steen, M. W. A., 1999, "Constructive consistency checking for partial specification in Z," *Science of Computer Programming*, Vol. 35(1), pp. 29–75.

Boiten, E. A., Bowman, H., Derrick, J., Linington, P. F., and Steen, M. W. A., "Viewpoint consistency in ODP," *Computer Networks*, Vol. 34(3), pp. 503–537.

Bolognesi, T., and Brinksma, E., 1988, "Introduction to the ISO Specification Language LOTOS," *Computer Networks and ISDN Systems*, Vol. 14(1), pp. 25–59.

Boudriga, N., Elloumi, F., and Mili, A., 1992, "On the lattice of specifications: Applications to a specification methodology," *Formal Aspects of Computing*, Vol. 4, pp. 544–571.

Bowman, H., Boiten, E. A., Derrick, J., and Steen, M. W. A., 1999, "Strategies for consistency checking based on unification," *Science of Computer Programming*, Vol. 33, pp. 261–298.

Bowman, H., Steen, M. W. A., Boiten, E. A., and Derrick, J., 2002, "A formal framework for viewpoint consistency," *Formal Methods in System Design*, Vol. 21, pp. 111-166.

Bowman, H., and Derrick, J., 1999, "A junction between state based and behavioural specification," Invited Paper in Fantechi, A., Ciancarini, P., and Gorrieri, R., (editors), *Formal Methods for Open Object-based Distributed Systems*, Kluwer.

Butler, M., Petre, L., and Sere, K. (editors), 2002, "Integrated Formal Methods, Third International Conference", Lecture Notes in Computer Science 2335, Springer-Verlag.

Derrick, J., Akehurst, D. H., and Boiten, E. A., 2002@ "A framework for UML consistency", in Kuzniarz, L., Reggio, G., Sourrouille, J. L., and Huzar, Z. (editors), *¡¡UML¿¿ 2002 Workshop on Consistency Problems in UML-based Software Development*, pages 30-45.

Derrick, J., and Boiten, E. A., 2001, "Refinement in Z and Object-Z: Foundations and Advanced Applications," FACIT series, Springer-Verlag.

Derrick, J., and Boiten, E. A., 2002, "Applying ODP to an air traffic management system: viewpoints and correspondences," submitted for publication.

Derrick, J., Boiten, E. A., Bowman, H., and Steen, M., 1999, "Viewpoints and consistency: translating LOTOS to Object-Z," *Computer Standards and Interfaces*, Vol. 21, pp. 251–272.

Derrick, J., Bowman, H., Boiten, E.A., and Steen, M., 1996, "Comparing LOTOS and Z refinement relations," *FORTE/PSTV'96*, pp. 501–516, Chapman & Hall.

European Organisation for the Safety of Air Navigation, 1997, "ECHO final report2," 1.0 edition.

Evans, A. S., France, R. B., Lano, K. C., and Rumpe, B., 1999 "Metamodelling semantics of UML," in Kilov, H., (editor), *Behavioural Specifications for Businesses and Systems*, chapter 4, Kluwer.

Fiadeiro, J., Lopes, A., and Maibaum, T., 1997, "Synthesising Interconnections," in Bird, R., and Meertens, L., (editors), *Algorithmic Languages and*

Calculi, pp. 240–264, Chapman & Hall.

Finkelstein, A., Kramer, J., Nuseibeh, B., Finkelstein, L., and Goedicke, M., 1992, "Viewpoints: a framework for integrating multiple perspectives in system development," *International Journal on Software Engineering and Knowledge Engineering, Special issue on Trends and Research Directions in Software Engineering Environments*, Vol. 2(1), pp. 31–58.

Frappier, M., Mili, A., and Desharnais, J., 1995, "Program construction by parts," in (Möller, 1995), pp. 257–281.

Große-Rhode, M., 2001, "Semantic integration of heterogeneous formal specifications via transformation systems", Habilitation thesis, Technical University of Berlin, Technical Report 2001/13.

Henessy, M., 1988, "Algebraic Theory of Processes," MIT Press.

Hoare, C. A. R., 1985, "Communicating Sequential Processes," Prentice Hall.

ISO/IEC/ITU-T, 1995-98, "ISO/IEC 10746 — ITU-T Recommendation X.901-X.904, Open Distributed Processing - Reference Model".

Jackson, D., 1995, "Structuring Z specifications with views," *ACM Transactions on Software Engineering and Methodology*, Vol. 4(4), pp. 365–389.

Miarka, R., Derrick, J., and Boiten, E. A., 2002, "Handling inconsistencies in Z using quasi-classical logic," in Bert, D., Bowen, J. P., Henson, M. C., and Robinson, K., (editors), *ZB 2002*, Lecture Notes in Computer Science 2272, pp. 204–225, Springer-Verlag.

Milner, R., 1989, "Communication and Concurrency," Prentice-Hall.

Möller, B., (editor), 1995, "Mathematics of Program Construction, 3rd International Conference, Kloster Irsee, Germany," Lecture Notes in Computer Science 947, Springer-Verlag.

Nuseibeh, B., 1996, "Towards a framework for managing inconsistency between multiple views," in Finkelstein, A., and Spanoudakis, G., (editors), *SIGSOFT '96 International Workshop on Multiple Perspectives in Software Development (Viewpoints '96)*, pp. 184–186, ACM.

Paech, B., and Rumpe, B., 1994, "A new concept of refinement used for behaviour modelling with automata," in Naftalin, M., Denvir, T., and Bertran, M., (editors), *FME'94: Industrial Benefit of Formal Methods*, Lecture Notes in Computer Science 873, pp. 154–164, Springer-Verlag.

Rumbaugh, J., Jacobson, I., and Booch, G., 1999, "The Unified Modeling Language: Reference Manual," Object Technology Series, Addison-Wesley.

Putman, J., 2000, "Architecting with RM-ODP," Prentice Hall.

Spivey, J.M., 1992, "The Z notation: A reference manual," 2nd edition, Prentice Hall.

Srinivas, Y. V., and Jüllig, R., 1995, "Specware: Formal support for composing software," in (Möller, 1995), pp. 399–422.

Steen, M. W. A., Derrick, J., Boiten, E. A., and Bowman, H., 1999, "Consistency of partial process specifications," in Haeberer, A., (editor), *AMAST'98*, Springer-Verlag.

Taylor, C. N., Boiten, E. A., and Derrick, J., 2002, "Interpreting ODP viewpoint specification: Observations from a case study," in *FMOODS 2002*.

Zave, P., and Jackson, M., 1991, "Techniques for partial specification and specification of switching systems," in Nicholls, J. E., (editor), *Sixth Annual Z User Workshop*, pp. 205–219, Springer-Verlag.

Zave, P., and Jackson, M., 1993, "Conjunction as composition," *ACM Transactions on Software Engineering and Methodology*, Vol. 2(4), pp. 379–411.

Zave, P., and Jackson, M., 1996, "Where do operations come from? A multiparadigm specification technique," *IEEE Transactions on Software Engineering*, Vol. 22(7), pp. 508–528.