The VDM+B project: Objectives and Progress

J.C. Bicarregui[1], Th. Dimitrakos[1]*, K. Lano[2], T. Maibaum[2], B.M. Matthews[1], B. Ritchie[1]

[1] CLRC Rutherford Appleton Laboratory, Oxfordshire, OX11 OQX, UK

[2] Dept. of Computing, Imperial College, London SW7 2BZ, UK.

Abstract

The VDM+B project is developing the underpinnings for an integration of VDM and B enabling their co-use within one formal development. In this paper, we describe the objectives for the project, the approach being undertaken and the current status of the work.

1 Introduction

VDM[2] and B[1] are among the few formal methods currently in use by industry and supported by commercial tools. Both are model-oriented methods for the development of sequential systems based on first order calculi and set theory. Both have a set of proof rules defined for formal verification and validation. Both have a formal semantics: for B this is defined in terms of weakest preconditions, for VDM it is denotational. As yet, neither set of proof rules has been verified with respect to the semantics.

In earlier studies [10, 6, 12] we have noted that VDM and B have different focuses: VDM is primarily concerned with high level design and data refinement, whereas B in practice is most suited to low level design, algorithm refinement and the generation of code. We have undertaken application experiments and development scenarios to determine the feasibility and potential benefits of heterogeneous development: using VDM, supported by the VDM-toolbox[24], for early lifecycle specification and validation activities; and B, supported by the B-Toolkit[4], for later design and verification tasks. The VDM+B project is consolidating this work by developing the formal underpinnings for a combined method employing VDM and B in heterogeneous development.

In this paper, we review the context and motivations for the VDM+B project, discuss the major issues for establishing the formal foundation of the integration, and review current progress and plans for their resolution.

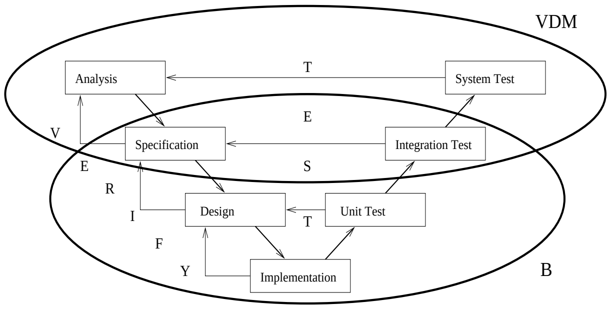

Figure 1: The lifecycle identified for heterogeneous development using VDM and B

2 Context

VDM and B are two of the most industrially used formal methods. Both have been used for a variety of applications and are supported by commercial toolkits. VDM's origins lie in the definition of programming language semantics in the 1970s, but it has for many years been used in systems specification and development generally[25]. A development of VDM and Z, Jean-Raymond Abrial originated B whilst at the Programming Research Group at Oxford University in the early 1980s.

Although VDM and B have the same expressive power in theory, a comparison undertaken during the B User Trials project[1] observed [10] that VDM encourages a style of specification where implicit invariants and explicit frames are employed with postconditions to describe operations as abstractly as possible whereas the representation of operations with explicit invariants and implicit frames employed in B encour-

* Theodosis Dimitrakos (theo@inf.rl.ac.uk) is the correspondence author

[1] **The B User Trials project** (1992-1995) [IED4/1/2182] was a collaborative project between RAL, Lloyds Register of Shipping, Program Validation Limited and the Royal Military College of Science and played a major part in bringing the B-Toolkit up to industrial quality. A summary of the project is in [7].

ages overspecification and the introduction of implementation bias reducing possible non-determinism. This difference arises from the different focus of the two methods and has led to the development of different functionality in the supported forms of the methods. The first project to bring VDM and B together to exploit their different strengths was the MaFMeth[2]. which assessed a methodology covering the whole life cycle combining the use of VDM for early development with B for refinement and code generation, the development lifecycle depicted in Figure 1. The project demonstrated the commercial viability of the use of formal methods by collecting quantitative evidence of the benefits in terms of both fewer faults made in development and their earlier detection. The translation between notations was however conducted informally and the results show that this translation was error prone, not only because of its manual nature but also because of the lack of clarity in the correspondence between the notations. However, it *was* found that animation, test case generation and proof are all cost-effective ways to find faults in formal texts [6]. The Spectrum project[3] was a feasibility study into the commercial viability of integrating the VDM-Toolbox and B-Toolkit. The evaluation was being undertaken from three perspectives: the industrial benefit of using the combined tool, the technical feasibility of the combination of the two tools and the commercial case for the development of a combined tool. The project developed heuristic methods for systematic transformation between specifications in VDM and B which could form the basis of machine support[9]. However, within the feasiblity study there was little scope for the research necessary to unify the two methodologies in terms of their underlying semantics and proof rules.

3 Issues

In recognition of the pragmatic nature of the earlier approaches to heterogeneous development using VDM and B, the **VDM+B**[4] project aims to establish a formal foundation of heterogeneous development in VDM and B. In this section we discuss some issues relevant to this objective.

The core elements of the two languages are very similar. Both expression languages are based on sets, sequences, tuples and relations. Both define abstract machines in terms of state, invariant and operations. Both have explicit preconditions for operations. However, VDM defines state transitions via relational postconditions, whereas B uses generalised substitutions. Both languages have formal semantics. For B, this is given by Abrial [1] in terms of weakest preconditions. For VDM, it is denotational [2]. A translation between these two forms is given in [1]; however, for our purposes this needs to be extended to cover a wider class of expressions.

An obvious point of concern is the foundational differences in the languages. VDM is based on the 3-valued Logic of Partial Functions (LPF) whereas B is based on classical First Order Predicate Calculus. Work on developing proof support for VDM [3] has shown that in a framework with dependent types, such as PVS [31], most specifications which employ partial functions for their expressivity can be directly translated to functions which are total over a subdomain. The remaining uses of partiality represent a particular form of lazy concurrent disjunction which is built into LPF but not available in B.

Although the two notations are founded on a different logic, the proof rules in the B-Toolkit do have a flavour more akin to those of VDM where typing hypotheses are used as guards to the expression construct introduction and elimination rules.

Both languages have a comprehensive set of proof rules defined and supported. For B, they are defined in [1] and built into the B-Toolkit. For VDM, they were developed in the Mural system[11] and published in [8].

In the absence of a standard form for proofs that would enable proofs developed in one system to be checked with another, it is important for the certification of formal developments to be able to "second source" the theorem proving capability. This will allow proof support to be developed in a number of systems and contribute to the certification of theorem proving capability for use in safety critical systems. Current support for the languages has not been certified in this way.

A further area of difference is higher level modular structuring. Several approaches to modularisastion exist for VDM. The VDM standard language, VDM-SL, has no structuring mechanism although a form of modularisation is given as an informative annex. The IFAD VDM Toolbox supports a simple form of modules and VDM++ [27] has an object-oriented notion of structuring based on classes. On the other hand, the ability to incrementally present a specification is central to B where implementations can be

[2] **The MaFMeth project** (1994-1995) [EC ESSI 1061] between Bull SA, B-Core UK Ltd, and RAL. A summary of the key results of the project is in [6].

[3] **The Spectrum project** (1997) [EC ESPRIT 23173] between RAL, GEC Marconi Avionics, Dassault Electronique, Space Science Italia, CEA, IFAD and B-Core UK Ltd [9].

[4] **The integration of two industrially relevant formal methods. (VDM+B)** (1998-2001) [EPSRC GR/L68452 and GR/L68445] between RAL and Imperial College London

constructed in a structured way by composing implementations of separate components [29].

Thus transformations between structured specifications in the two formalisms should, in some sense, preserve the locality of information. For example, in moving from a single module of VDM where the structure is based around a hierarchical definition of record types, we would hope to achieve a B specification which used machines to mirror the structure of the records. The danger is that in "coding up" such a complex refinement into the translation we put at risk the soundness of the translation. One possible approach [29] is for the translation to result in two levels of B specification and a refinement between them. In this way the translation is kept simple, whilst the complexity of the refinement is localised within the one formalism and hence more amenable to verification.

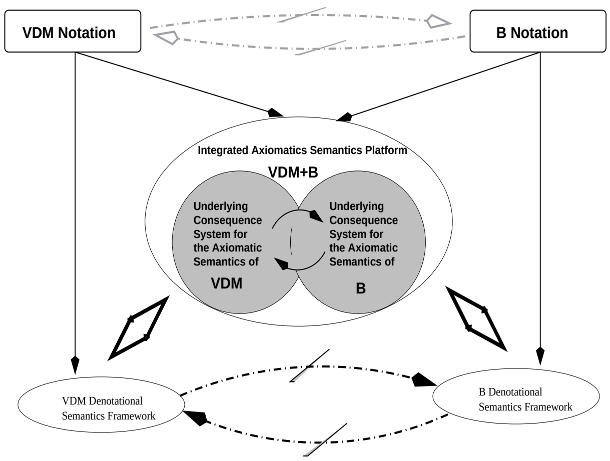

Figure 2: A pictorial overiview of our approach to the integration of support for VDM and B.

4 Progress to date

To date, attention has been focused on underpining the co-use of VDM and B by the application of a general framework for integrating heterogeneous logics. Macroscopically, our approach to the integration of support for VDM and B, or indeed various other formal notations, can be divided into three interdependent steps: (1) Specifying the intended interrelations at the syntactical level of the formal notation. (2) Establishing compatible interrelations between the axiomatic (logical) semantics and between the denotational semantics of the formal notations. (3) Integrating the consequence systems that accommodate the axiomatic (logical) semantics of the interrelated formal notations into one compound consequence system. This integration should *locally conserve* each entailment, and keep proof and denotational semantics distinct and local to each component.

Investigations for stage 1, designing (partial) translations between the VDM and B notations, have been conducted in the Spectrum and MaFMeth projects. (See [12, 29, 6].) In the current project, we have focussed on developing sound logical and mathematical foundations for steps 2 and 3, providing a "unifying" framework for formally presenting the logical (axiomatic) semantics of formal notations, and a method to synthesise the integrated logical (axiomatic) semantics. The main idea of the latter is, on the one hand, to produce a common logical framework where each formal notation can be faithfully interpreted and, on the other hand, to reuse the proof calculi and denotational semantics of each formal notation and hence avoid (re)building those for the integrated system. As far as proof support is concerned, in particular, we do not intend to replace the purpose-built proof support of the existing tools with a more complex (and probably less efficient) general purpose theorem prover that could be used by all components. Our intention *is* to build a compound logical system which can accommodate the existing axiomatic semantics for all the component notations, and hence support their interoperability, while keeping proof (and denotational semantics) *distributed* and *localised* to each component. Of course, the mathematically sound and pragmatically meaningful integration of the components' entailments also depends on the correct interrelation of the components' (denotational) semantics and affects the interoperability of the associated proof mechanisms.

In our study of the problem of integrating heterogeneous formal notations with emphasis on the integration of the axiomatic (logical) semantics, we have blended together concepts and methods from formal logic, categorical algebra and institution theory. We use Meseguer's *General Logics* [30] as a "unifying" presentation of the *logical* (axiomatic) and the *denotational* semantics of a formal notation, related via a *soundness* condition. We also use from [14] the concept of a *non-plain mapping of Logics* (which is a slight adaptation of Meseguer's "(simple) map of logics" in [30]) as the basic correctness preserving means of relating *Logics*.[5]

The semantics of a formal notation, such as VDM or B, is expressed in a logic. First of all, the grammar

[5] A *map of Logics* encodes a *correctness preserving* covariant interpretation of the entailment component of the source *Logic* into the target *Logic* coupled with a contravariant mapping at the model level. A *non-plain map of Logics* relativises these interpretations to the area bounded by a family of *structural axioms*.

of the logic is used to specify the built-in types and operations of the formal notation and to interpret the user defined types and operations. Secondly, the entailment (logical consequence) of the logic provides the basis for describing the axiomatic semantics of the specifications and for proving theorems entailed by the specifications with respect to this axiomatic semantics. Finally, the models of the logic provide the structures on top of which the denotational semantics of the formal notation are defined. The interrelation of formal notations is grounded on the interrelation of the underlying logics describing their semantics. In particular, the logical (axiomatic) semantics of a VDM specificaton is provided by an LPF-theory which is a consistent extension of the join (colimit) of the LPF-theories describing the basic types, type constructions and operations of the specification. Analogously the axiomatic semantics of a B specification are provided by a theory in classical logic which is a consistent extension of the join (colimit) of the theories describing the abstract machine structure and the types and operations of the B specification. In general, the logical semantics of VDM specifications are provided by the colimit objects (theories) of finite diagrams in the finitely cocomplete category of LPF theory presentations while the logical semantics of B specfciations are provided by the colimit objects of finite diagrams in the finitely cocomplete category of first order specifications.

One significant difference of (*non-plain*) *maps of logics* compared to other approaches to relating logical consequence systems or satisfaction systems, is that a *map of logics* correlates each signature Σ in the source logic with a *theory presentation* $\mathbf{f}(\Sigma) = \langle \mathbf{f}^{\langle s \rangle}(\Sigma), \mathbf{f}^{\langle a \rangle}(\Sigma) \rangle$ in the target logic – instead of just a signature in the target logic, as is common in the literature[6]. In many applications of formal logic in information systems engineering, plain morphisms which map signatures to signatures are not flexible enough. It is often necessary to map built-in elements of one logic into explicitly specified (sometimes precisely defined) elements of another logic. The (non-plain) *maps of logic* overcome such problems by supporting the interpretation of theorems over a signature Σ of the source logic into theorems over the corresponding signature $\mathbf{f}^{\langle s \rangle}(\Sigma)$ in the target logic with a set of *structural axioms* $\mathbf{f}^{\langle a \rangle}(\Sigma)$ in the language of $\mathbf{f}^{\langle s \rangle}(\Sigma)$. These structural axioms assist in interpreting some features of the source consequence that can be specified in the target logic but not simulated directly by the target consequence.

Intuitively, this form of correctness preservation states that the translation of a theorem proved in the source logic $\mathcal{A}$ results in a lemma in the target logic $\mathcal{B}$, and can be therefore used for proving theorems in $\mathcal{B}$, only when some explicitly specified structural conditions hold. In fact, this weaker version of correctness preservation again appears to be an instance of a more general *conditional correctness preservation property*, another instance of which was observed by Fiadeiro and Maibaum in [21] and [19] while building calculi to support concurrent and object-oriented system specifications. This analogy is further emphasised in [13] where we show why, where and when the structural axioms that support interrelating a collection of logics are "internalised" into *locality* axioms which support formal reasoning *inside* the integrated logic. (See also Appendix **D.** of [14] for a more detailed technical analysis.)

Using the above described framework, we have modelled in [13] the interpretation of LPF into classical (infinitary) logic introduced by Jones and Middelburg in [26] to provide an indicative example of an interesting *non-plain mapping* of *Logic*s. We have also studied a general method for integrating a collection of interrelated *Entailment Systems*, the logical consequence oriented component of *Logic*s. This method is presented in [13]. It is related to the "flattening" of indexed categories analysed by Tarlecki, Goguen and Burstall in [37] and extends a method that was first introduced in [14] using the Grothendieck construction [23, 5] in an essential way.

The basic advantages of this method for the correctness preserving integration of the logical semantics of inter-dependent specification formalisms include the following: a compound consequence system is produced which can accommodate the axiomatic semantics of each component formalism; remote *interentailment* reasoning along non-plain mappings is transformed into internal reasoning along language translations; the grammar and the entailment underpinning the axiomatic semantics of each component formalism are locally conserved in the result of the integration; hence, (i) the existing proof support for each component formalism can be reused; (ii) the denotational semantics for each component formalism are not affected by the integration and they can be reused; (iii) theorems proved in one formalism $\mathcal{X}$ can be used as lemmata in a proof conducted in a compatible signature of an interrelated formalism $\mathcal{Y}$, provided that the corresponding locality (structural) axioms are satisfied. In particular, to keep the proofs local and distributed to each component while facilitating the interoperability of the proof support in the integrated system, seems to be an advantageous alternative to redesigning a more complex, and probably less efficient, proof calculus for the compound system. A detailed description of this method is given in [13].

[6] See [28] for a comparison of various mappings between Institutions, and further references.

Another interesting outcome of this research has been to provide a sufficiently clear basis for explaining why, where and when the *structural axioms*, that may assist in interpreting built-in elements of a "source" logic into explicitly specified artefacts over a "target" logic, give rise to *locality axioms* that assist in logical reasoning along language translations inside the integrated *Entailment System*. We are currently analysing the pragmatic impact of these results and investigating how this can be applied to assist the integration of tool support for interrelated formal notations. We focus on how we intend to apply this method to facilitate the co-use of existing support for the VDM and B formal methods over an integrated axiomatic semantics.

5 Conclusion

This project has been investigating the formal underpining of the co-use of interrelated heterogeneous specification formalisms. We have blended together concepts and methods from specification theory, formal logic and categorical algebra in order to: introduce an abstract mathematical framework for uniformly presenting the axiomatic and denotational semantics of specification formalisms, and interrelating such presentations by means of correctness preserving mappings; provide a general method for integrating the heterogeneous consequence systems that accommodate the axiomatic semantics of interrelated specification formalisms; and to explain why and when the integrated consequence system may need the additional support of structural axioms in order to support multilogical reasoning.

It is important to stress that our intention has *not* been to blend selected features of different formalisms into a new single formalism, but to develop a unifying framework for *integrating* heterogeneous formalisms into one compound formalism with respect to the structure of the component entailments and the (explicitly specified) interpretations between them.

Future work in this project will continue this development and also address the other issues outlined in section 3.

With respect to proof support, we believe that the integrated axiomatic semantics should enable the interpretations of VDM and B to be expressed as seperate modules within one supporting system. The intermodular translations and the associated locality axioms would then be realised by a third interfacing module.

With respect to structuring, we plan to interpret a structured specification as a diagram in the category of theory presentations over the logic that is used for the axiomatic semantics. Different structuring mechanisms give rise to distinctly shaped diagrams of theory presentations with different types of theorem preserving morphisms as arcs. Under some generally weak assumptions, theory presentation diagrams can also be treated as formal objects in a (functor) category, which are related to each other by diagram morphisms built from *compatible families of theory presentation morphisms*[35] equipped with a notion of *parallel composition* which facilitates the orthogonally modular horizontal and vertical refinement and structuring of complex specifications.[7] Somewhat similar diagrammatic specifications can also be used for describing the structured theories of Maude [17]. In fact, Durán and Meseguer have recently proposed in [18] a method to construct an *Institution* $\mathbf{S}(\mathcal{I})$ on top of an *Institution* $\mathcal{I}$, such that structured $\mathcal{I}$-theories are treated as ordinary theories of $\mathbf{S}(\mathcal{I})$. We intend to investigate the applicability of this framework in our work. Also the use of parametric theory presentation diagrams, originally proposed by Dimitrakos in [15, 14] and more recenlty extended by Dimitrakos and Maibaum in [16], supports the incremental construction of complex parameter specifications from simpler modules. It allows the explicit specification of special relations between the possibly interconnected, but yet individually distinct, constituent specification modules of the parameter and the body specification. The ability to induce an instantiation from a compatible family of parallel morphisms reduces the instantiation effort: the instantiation of a complex parameter is decomposed to a family of simpler parallel morphisms from the constituents of the parameter. It also provides the basis for synthesising the instantiation of a complex parameter from the instantiations of its constituent modules, and it facilitates the compatible instantiation of nested parameterisations. In recognition of the fact that this is a more problematic aspect of both formalisms, we hope that this approach may contribute to the development of elegant and useful structuring mechanisms for both formalisms.

In recent years there is an increasing recognition of the fact that no single formalism is likely to be best suited to all formal development tasks, just as no single programming language is ideal for all applications [36, 32, 33]. Our overarching aim in this project is to integrate useful formalisms without necessitating the abandonment of existing methods and tools.

[7] Diagrammatic presentations and diagram morphisms based on compatible families of parallel theory interpretations have been successfully applied in state-of-the art formal software development environments such as SPECWARE [34] built at Kestrel Institute, California.

References

[1] J.-R. Abrial, The B-Book : Assigning Programs to Meanings, Camb. Univ. Press, 1996.

[2] Andrews et al. Information Technology - Programming Languages - Vienna Development Method-Specification Language. Part 1: Base Language. ISO DIS 13817-1, 1995

[3] S. Agerholm, Translating Specifications in VDM-SL into PVS, 9th International Conference in Higher Order Logic Theorem and Its Applications, LNCS 1125, Springer Verlag, September 1996.

[4] B-Core (UK) Ltd. The B-Toolkit. Welcome page URL <http://www.b-core.com>, 1996.

[5] M. Barr and C. Wells. *Category Theory for Computer Science*, volume 5 of *Series in Computer Science*. Prentice Hall International, 1990.

[6] J.C.Bicarregui, J.Dick, B.Matthews, E.Woods, Making the most of formal specification through Animation, Testing and Proof. Sci. of Comp. Prog. Elsevier Science. Feb. 1997.

[7] J.C. Bicarregui *et al.*, Formal Methods Into Practice: case studies in the application of the B Method. I.E.E. Transactions on Software Engineering, 1997.

[8] J.C. Bicarregui, J.S. Fitzgerald, P.A. Lindsay, R. Moore and B. Ritchie, Proof in VDM – A Practitioner's Guide, FACIT, Springer-Verlag, ISBN 3-540-19813-X, 1994.

[9] J.C. Bicarregui, B.M. Matthews, B. Ritchie, and Sten Agerholm. Investigating the integration of two formal methods. In *Proceedings of the 3rd ERCIM Workshop on Fomral Methods for Industrial Critical Systems*. May 1998. To appear in Formal Aspects of Computing 1999.

[10] J.C. Bicarregui and B. Ritchie, Invariants, Frames and Postconditions: a comparison of the VDM and B notations, IEEE Trans. on Software Engineering, Vol.21, No.2, pp.79-89, 1995.

[11] J.C. Bicarregui and B. Ritchie. Reasoning about VDM Developments using the VDM Support Tool in Mural. Proceeding of VDM 91, Prehn and Toetenel (Eds), LNCS 552, Springer-Verlag.

[12] J.C. Bicarregui, B.M. Matthews, B. Ritchie, and S. Agerholm", Investigating the integration of two formal methods, Proceedings of the 3rd ERCIM Workshop on Formal Methods for Industrial Critical Systems. To appear in Formal Aspects of Computing 1999.

[13] J.C. Bicarregui, Th. Dimitrakos, B.M. Matthews, T.S.E. Maibaum, K. Lano, B. Ritchie. The VDM+B project. To appear at the FM'99 VDM workshop. Toulouse, France. September 1999.

[14] Theodosis Dimitrakos. *Formal support for specification design and implementation.* PhD thesis, Imperial College, March 1998.

[15] Theodosis Dimitrakos. Parameterising (algebraic) specifications on diagrams. In *Automated Software Engineering–ASE'98, 13th IEEE International Conference*, 1998.

[16] Theodosis Dimitrakos and Tom Maibaum. Parameterisation Cocones (Extending parametricity to cater for increasingly complex, modular designs). Submitted for publication. (Spring 1999)

[17] Francisco Durán. A Reflective Module Algebra with Aplicatons to the Maude Language. A PhD Thesis, University of Málaga, 1999.

[18] Francisco Durán and José Meseguer. Structured Theories and Institutions. Computer Science Laboratory, SRI International. Submitted for publication. (Spring 1999)

[19] J. Fiadeiro and T. Maibaum. Temporal Theories as Modularisation Units for Concurrent System Specification. *Formal Aspects of Computing*, 4(3):239–272, 1992.

[20] J.L. Fiadeiro and T. Maibaum. Categorcal Semantics of Parallel Porgrams. *Science of Computer Programming*, pages 111–138, 1997.

[21] J.L. Fiadeiro and T.S.E. Maibaum. Generalising interpretations between Theories in the Context of (π-)institutions. In S. Gay G. Burn and M. Ryan, editors, *Theory and Formal Methods*, pages 126–147. Springer-Verlag, 1993.

[22] Nissim Francez and Ira R. Forman. Superimposition for interacting processes. In J. C. M. Baeten and J. W. Klop, editors, *CONCUR '90: Theories of Concurrency: Unification and Extension*, volume 458 of *Lecture Notes in Computer Science*, pages 230–245, Amsterdam, The Netherlands, 27–30August 1990. Springer-Verlag.

[23] A. Grothendieck. *Catégories fibrées et descente, Exposé VI in Revetements Etales et Groupe Fondamental (SGA1).* Lecture Notes in Mathematics 224. Springer, Berlin, 1971.

[24] IFAD (Dk). The VDM-SL toolbox. Welcome page URL <http://www.ifad.dk>, 1996.

[25] C.B.Jones, Systematic Software Specification Using VDM (2nd Edition), Prentice Hall, 1990.

[26] C.B. Jones and C.A. Middelburg. A typed logic of partial functions reconstructed classically. *Acta Informatica*, 31:399–430, 1994.

[27] K. Lano, S. Goldsack, Integrated Formal and Object-oriented Methods: The VDM++ Approach', 2nd Methods Integration Workshop, Leeds Metropolitan University, April 1996;

[28] A. Martini, U. Wolter, A Single Perspective on Arrows between Institutions . Seventh International Conference on *Algebraic Methodology and Software Technology* - AMAST'98. Springer LNCS 1548, pp. 486-501 (1999).

[29] B. Matthews, B. Ritchie, and J. Bicarregui, 'Synthesising structure from flat specifications', Proc. of the 2nd International B Conference, Montpellier, France, April 22-24, 1998.

[30] Jose Meseguer. General logics. In H.D. Ebbinghaus, editor, *Logic Colloquium'87*, pages 275–329, 1989.

[31] S. Owre et al. PVS: Combining Specification, Proof Checking, and Model Checking, Computer-Aided Verification, CAV '96, Rajeev et al (Eds) Springer-Verlag LNCS 1102, 1996.

[32] R.F. Paige A Meta-Method for formal method integration. In Proc. FME'97, eds Fitzgerald, Jones, and Lucas, Springer-Verlag LNCS 1313, 1997.

[33] Proceedings of the 2nd Methods Integration Workshop, Leeds, EWIC series, Springer-Verlag, 1996.

[34] Y.V. Srinivas and R. Jullig. SPECWARE^{TM}: Formal suport for composing software. In *Mathematics of Program Construction*, July 1995. (KES.U.94.5).

[35] Y.V. Srinivas and R. Jullig. Diagrams for Software Synthesis. Kestrel Institute, Paolo Alto, California, 1993. Appeared in the *Proceedings of* KBSE'93.

[36] A. Tarlecki. Towards heterogeneous specifications. In *Frontiers of Combining Systems FroCoS'98*, Applied Logic Series. Kluwer Academic Publishers, October 1998. To appear.

[37] A. Tarlecki, J. Goguen, and R. Burstall. Tools for semantics of computation: indexed categories. *Theoretical Computer Science*, 91:239–264, 1991.