The VDM+B project: Objectives and Progress

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

[1] CLRC, Rutherford Appleton Laboratory, UK
{jcb,theo,bmm,br}@inf.rl.ac.uk,
[2] Dept. of Computing, Imperial College, London,UK
{tsem,kcl}@doc.ic.ac.uk

Abstract. The VDM+B project is developing the formal underpinnings for an integration of VDM and B enabling their co-use within one 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 [12, 7, 11] 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 IFAD VDM Tools [25], 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.

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.

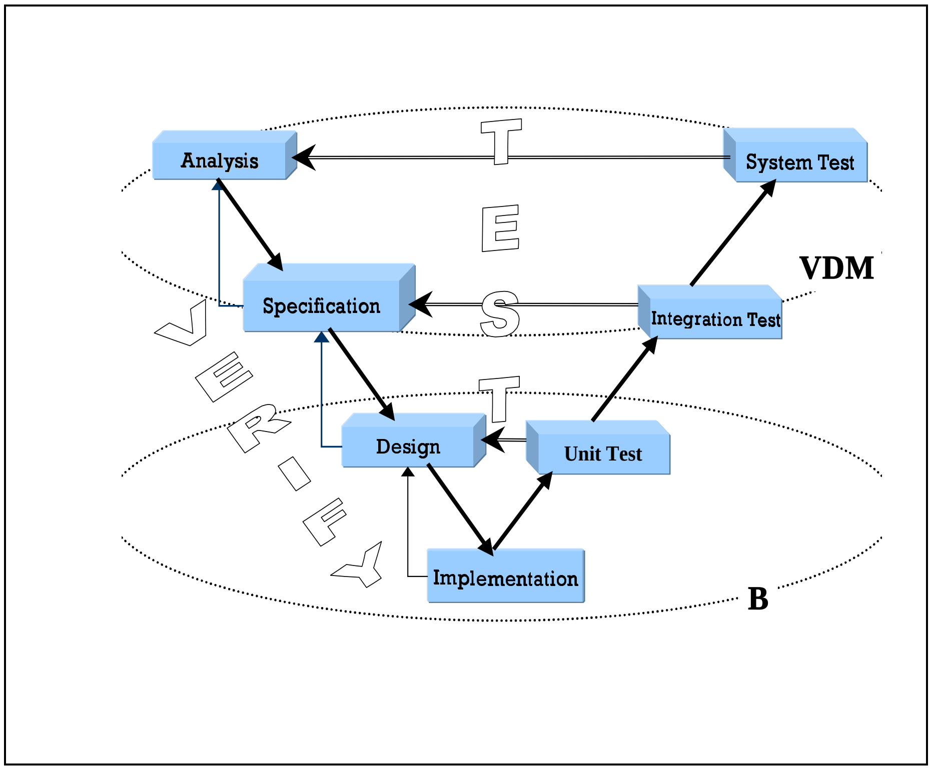

Figure 1. The lifecycle identified for heterogeneous development using VDM and B

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[26]. Areas to which VDM has recently been applied include railway interlocking systems, ammunition control systems, semantics of data flow diagrams, message authentication algorithms, relational database systems and medical information systems. A directory of VDM usage examples is available[42].

A development of VDM and Z, Jean-Raymond Abrial originated **B** whilst at the Programming Research Group at Oxford University in the early eighties. It was further developed by British Petroleum Research and DIGILOG and supported forms are now commercially available from B Core Ltd. and Steria Technologies de l'Information. Examples of its use include the development of communication security protocols, subway speed control mechanisms, railways signalling, executable database programs and data processing systems. A directory of B is maintained at [5].

Although VDM and B have the same expressive power in theory, a comparison undertaken during the **B User Trials** project[1] observed [12] 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 encourages overspecification and the introduction of implementation bias reducing possible non-determinism. In both notations, the invariant is useful for quickly conveying an understanding of the reachable values of the state. However the use of invariants in operation definitions differs. In B, postconditions (in the form of generalised substitutions) have to be written so as to ensure the maintenance of the invariant. In VDM the state invariant is effectively part of the state typing information, and as such is assumed to be maintained in addition to the postcondition.

VDM's implicit maintenance of the invariant led to the choice discussed in [12] of how much of the information in the invariant is repeated in a postcondition. There is often some tension between the most concise form that relied on properties of the invariant for its correctness, and a longer, but more explicit form, that included some redundant information. This choice can be seen as an opportunity to prove the stronger forms from the weaker. Which form is chosen may make a significant difference to the complexity of the proofs: the form that most clearly conveys the information may not be the form that will be most usable in proofs. Indeed, the stronger form is more likely to be helpful when the specification is being proved to be a reification of another, and the weaker form when it is itself being reified.

In the B notation, on the other hand, one writes operations so as to imply the preservation of the invariant. This can encourage a tendency to describe *how* the invariant is maintained, which may lead to less abstract specifications. The greater programmatic feel of the B notation is reinforced by the use of generalised substitutions, as opposed to VDM's relational post-conditions. Although the two forms have the same expressive power, in some cases (as for example in *slave* in the $b2$ machine described in [12]) we found it convenient to give greater algorithmic detail in the B version. This would appear to imply that the B notation is more useful for the development of algorithms. Indeed, the process of operation decomposition has been given greater attention in the B methodology than for VDM. By contrast, perhaps VDM's relational postconditions give a greater facility for non-algorithmic specifications of complex operations.

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. Table 1 shows the complementary nature of the features currently provided by each toolkit.

[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 [8].

The MaFMeth project[2] was the first project to bring VDM and B together to exploit their different strengths. This was an ESSI "application experiment" assessing 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 [7].

The Spectrum project[3] was a feasibility study into the commercial viability of integrating the IFAD VDM tools 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 [32, 11]. However, within the feasibility study there was little scope for the research necessary to unify the two methodologies in terms of their underlying semantics and proof rules.

Task	IFAD VDM Tools	B-Toolkit
Requirements capture	×	×
Visualisation	×	×
Abstract Specification	√	×
Type checking	√	∼
Prototype code generation	√	∼
Test coverage	√	×
Animation/Execution	√	√
Modularity	∼	√
Refinement	×	√
Proof	×	√
Final Code generation	×	√
Design documentation	×	√
Version Cntl/Config Mgmt	×	∼

Key
√ = good support
∼ = some support
× = no support

Table 1. The complementary functionality of the VDM and B Toolkits.

[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 [7].

[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.

3 Issues

In recognition of the pragmatic nature of the earlier approaches to heterogeneous development using VDM and B, the **VDM+B**[4] project has been established. The goal of this project is establish a formal foundation of heterogeneous development in VDM and B. However, there are significant differences in approach have been taken in the design of VDM and B. In this section we discuss some of these issues.

3.1 Core elements and foundational differences

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] as weakest preconditions. For VDM, it is denotational [2, 29]. 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 [34], 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.

3.2 Proof support

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[13] and published in [10].

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

[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

critical systems. Current support for the languages has not been certified in this way.

3.3 Modular structuring

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 Tools supports a simple form of modules and VDM++ [28] 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 constructed in a structured way by composing implementations of separate components.

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 risk the soundness of the translation. One possible approach [32] 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.

4 The VDM+B project: Progress to date

The VDM+B project is developing the formal underpinnings for an integration of VDM and B enabling their co-use within one development. To date, in the first year of the VDM+B project, 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 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 [11, 32, 7].) In the current project, we have focussed on developing sound

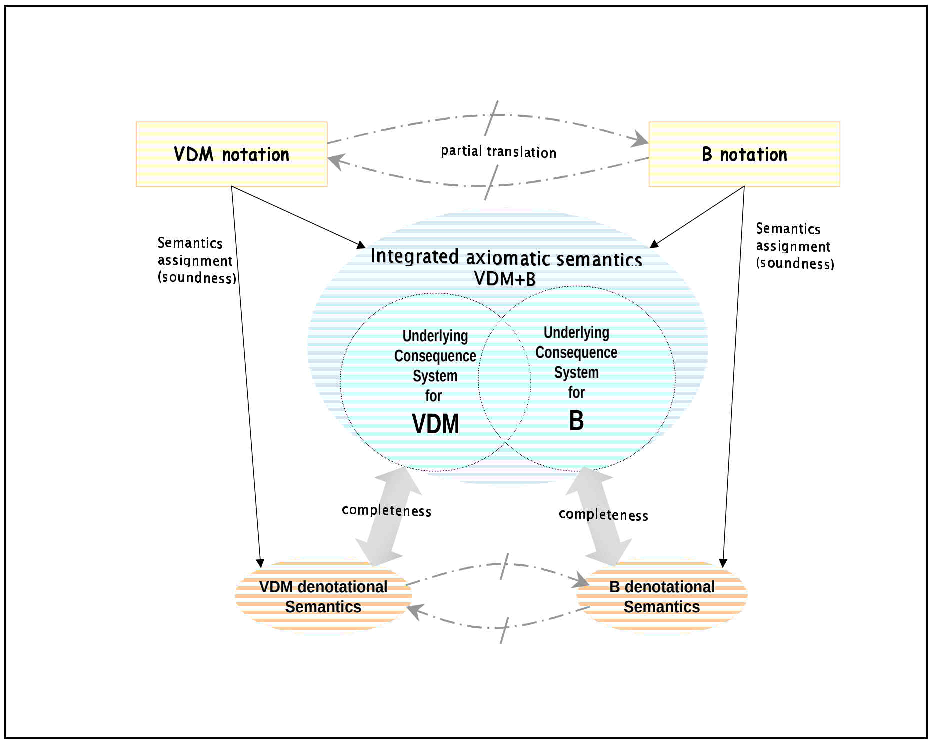
VDM notation
partial translation
B notation
Semantics assignment (soundness)
Integrated axiomatic semantics VDM+B
Semantics assignment (soundness)
Underlying Consequence System for VDM
Underlying Consequence System for B
completeness
completeness
VDM denotational Semantics
B denotational Semantics

Figure 2. A pictorial overview of our approach to the integration of VDM and B.

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 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.That is, to allow existing proof structures to be used whilst identifying the *structural* conditions which ensure the sound interpretation of theorems from one formalism into the other. 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* [33] 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 [33]) as the basic correctness preserving means of relating *Logic*s.[5]

Using this framework, we have modelled the interpretation of LPF into classical (infinitary) logic introduced by Jones and Middelburg in [27] to provide an indicative example of an interesting *non-plain mapping* of *Logic*s in [17]. 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 [17], it is related to the "flattening" of indexed categories analysed by Tarlecki, Goguen and Burstall in [41] and extends a method that was first introduced in [14] using the Grothendieck construction [24, 6] in an essential way.

One 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. The reader is referred to [17] for a more detailed presentation and to [14] for a technical analysis of the employed formal framework.

4.1 The rôle of general logics

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

[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* localises these interpretations in the area bounded by a family of *structural axioms*.

The interrelation of formal notations is grounded on the interrelation of the underlying logics describing their semantics. 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 specifications are provided by the colimit objects of finite diagrams in the finitely cocomplete category of first order specifications.

4.2 The rôle of non-plain maps of logics

The basic idea behind a *map of logics* is to interpret the syntax while preserving the entailment and keeping the satisfaction invariant. The interpretation of the syntax follows essentially the same pattern as within a single formalism. First, map the category of source signatures to the category of target signatures and, second, encode the operation of the source grammar functor into the target in order to obtain a mapping of sentences over the source into accordingly structured sentences over the target. The preservation of the entailment and the invariance of the satisfaction, on the other hand, may be viewed as complementary instances of the general *correctness preservation property* for the case of mappings between logics. Two, perhaps more familiar, complementary instances of the same general correctness property are the theorem preservation property of theory interpretations and the model invariance along signature translations in the same logic. The preservation of the entailment via *Entailment System* morphisms provides a *relevant correctness* criterion for translating theories from one entailment system into another while the invariance of the satisfaction provides a *relevant correctness* criterion for relating sentences in one logic with models in another.

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.

The interpretation of LPF into classical logic introduced by Jones and Middelburg in [27], for example, can be modeled by means of a non-plain map of

[6] See [30, 31] for a comparison of various mappings between Institutions, and further references.

logics. In this case, the signature transformation and the corresponding structural axioms take the following form:

f maps each LPF signature $\Sigma = \langle S, F, P\rangle$ to a classical signature $\mathbf{f}^{\langle s\rangle}(\Sigma) = \langle \Phi, \Pi\rangle$ consisting of

1. A set Φ of function symbols consisting of
   (a) three constants: t,f and $\uparrow$;
   (b) an n-ary function symbol f_f for each n-ary function symbol $f \in F$ of the LPF-signature Σ;
   (c) an n-ary function symbol f_p for each n-ary predicate symbol $p \in P$ of the LPF-signature Σ.
2. A set Π of predicate symbols consisting of
   (a) Two unary predicates U and B;
   (b) A unary predicate symbol τ_s for each type symbol $s \in S$ of the LPF-signatures Σ.

f assigns to each LPF-signature $\Sigma = \langle S, P, F\rangle$ a set$\mathbf{f}^{\langle a\rangle}(\Sigma)$ of *structural axioms* on the classical logic signature $\mathbf{f}^{\langle s\rangle}(\Sigma)$ which is defined as follows:

$$\mathbf{f}^{\langle a\rangle}(\Sigma) = \mathbf{Dom}\ \cup\ \mathbf{Truth} \cup\ \mathbf{Type}\Sigma\ \cup\ \mathbf{Func}\Sigma\ \cup\ \mathbf{pred}\Sigma$$

where

Dom $= \{\ \mathsf{U}(\uparrow) \wedge \exists y \cdot \mathsf{U}(y) \wedge \neg(y =\uparrow)\ \};$
Truth $= \{\ \neg(\mathsf{t} = \mathsf{f}) \wedge \neg(\mathsf{t} =\uparrow) \wedge \neg(\mathsf{f} =\uparrow),\ \forall b \cdot \mathsf{B}(b) \leftrightarrow (b = \mathsf{t} \vee b = \mathsf{f} \vee b =\uparrow)\ \};$
Type$\Sigma = \{\ \forall y \cdot s(y) \rightarrow \mathsf{U}(y) \wedge \neg(y =\uparrow)$: for every type symbol $s \in S$ of Σ };
Func$\Sigma = \{\ \forall y_1 \ldots y_n \cdot \mathsf{U}(y_1) \wedge \ldots \wedge \mathsf{U}(y_n) \rightarrow \mathsf{U}(\mathbf{f}^{\langle s\rangle}(f)(y_1, \ldots, y_n)$: for every n-ary function $f \in F$ of Σ };
Pred$\Sigma = \{\ \forall y_1 \ldots y_n \cdot \mathsf{U}(y_1) \wedge \ldots \wedge \mathsf{U}(y_n) \rightarrow \mathsf{B}(\mathbf{f}^{\langle s\rangle}(p)(y_1, \ldots, y_n)$: for every n-ary predicate symbol $p \in P$ of Σ };

The structural axioms in **Dom** and **Truth** assert that the domain of values contains *at least one* value *different* to the special element ($\uparrow$) that is used for interpreting non-denoting terms, and that the domain of truth values contains *exactly two distinct* truth values *in addition* to the the special element ($\uparrow$) that is used for interpreting non-denoting formulae. These structural axioms do not depend on the structure of Σ. The structural axioms in **Type**Σ, **Func**Σ and **Pred**Σ, on the other hand, depend on the internal structure of Σ. These axioms state that the type interpretations concerned do not contain the special value $\uparrow$, that all the functions concerned are inside the domain U and that all predicates concerned are range over the truth value domain B. There are also formulae stating that free variables always denote. See [17] for further details on this example.

Intuitively, the above mentioned 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 [22] and [20] while building calculi to support concurrent and object-oriented system specifications. This analogy is further emphasised in [17] 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.)

4.3 A formal foundation for the integration

In [17], we present a method to integrate any small graph (diagram) $\mathbf{D}_{\mathcal{E}}$ consisting of consequence systems (as nodes) and non-plain maps between them (as arrows) into a *weakly structural* consequence system $\mathcal{E}$. Weakly structural consequence systems are distinguished by the fact that the family of entailment relations presenting logical consequence is only conditionally stable under translation. This essentially means that, in the context of a proof, the preservation of correctness by common language modifications, such as addition of new symbols, renaming or identification of existing symbols, etc., relies on the satisfaction of a set of axiomatic conditions (*locality axioms*) in the modified language

The main idea of the integration is to "internalise" inter-entailment reasoning along non-plain maps of $\mathbf{D}_{\mathcal{E}}$ into logical reasoning along signature translations supported by *locality* axioms in a way that the entailments are locally conserved inside the integrated system. That is, the theorems proved at each component are the same before and after the "internalisation". In other words, the semantics of the component consequence systems are not affected by the integration. Each signature Σ in a node X of $\mathbf{D}_{\mathcal{E}}$ becomes a signature "Σ labeled by X" $\langle \mathsf{X}, \Sigma \rangle$ in $\mathcal{E}$. The grammar and the entailment of $\mathcal{E}$ is synthesised from the grammars in the nodes of $\mathcal{E}$ so that the language of $\langle \mathsf{X}, \Sigma \rangle$ is given by the language of Σ and the entailment on Σ inside the consequence system X. The signature morphisms inside $\mathcal{E}$ are the signature morphisms inside each node of $\mathbf{D}_{\mathcal{E}}$ together with some "new" morphisms induce by the integration. The latter can be considered as tuples $\langle \mathsf{f}, y \rangle$ for each arc $\mathsf{f}{:}\mathsf{X} \longrightarrow \mathsf{Y}$ of $\mathbf{D}_{\mathcal{E}}$ and each signature morphism $y{:}\mathsf{f}(\Sigma) \longrightarrow \Sigma'$ with (Σ) being the Y-image of a signature Σ in X via f. The composition of signature morphisms in $\mathcal{E}$ reduces to a combination of the composition operations inside each node of $\mathbf{D}_{\mathcal{E}}$ and the composition of non-plain maps. Furthermore, the logical consequence in $\mathcal{E}$ is *weakly structural*; that is, the structural axioms of the non-plain maps that correspond to the arcs of $\mathbf{D}_{\mathcal{E}}$ become *locality axioms* along signature morphisms in $\mathcal{E}$.

If this method is applied to the classical interpretation of LPF [27] mentioned in section 4.2 then the following signature translations and locality axioms are induced by the integration:

1. if $i{:}\mathsf{A} \longrightarrow \mathsf{B}$ is a theory interpretation between LPF theories then

$$\langle LPF, i \rangle : \langle LPF, \mathsf{A} \rangle \longrightarrow \langle LPF, \mathsf{B} \rangle$$

   is a theory interpretation in the integrated consequence system;

2. if $i{:}\mathsf{A}{\longrightarrow}\mathsf{B}$ is a theory interpretation in classical logic then
$$\langle CL, i\rangle : \langle CL, \mathsf{A}\rangle \longrightarrow \langle CL, \mathsf{B}\rangle$$
is a theory interpretation in the integrated consequence system;
3. if $\mathsf{A} = \langle \Sigma_A, \mathbf{A}\rangle$ is an LPF theory, $\mathsf{B} = \langle \Sigma_B, \mathbf{B}\rangle$ is theory in classical logic and $\sigma{:}\mathbf{f}^{\langle s\rangle}(\Sigma_A){\longrightarrow}\Sigma_B$ is a translation from the classical logic signature $\mathbf{f}^{\langle s\rangle}(\Sigma_A)$ to the classical logic signature Σ_B, where $\mathbf{f}^{\langle s\rangle}(\Sigma_A)$ is the classical image of the LPF signature Σ_A given in section 4.2 based on [27], then $i = \sigma\mathsf{o}\mathbf{f}^{\langle s\rangle}_{\Sigma_A}$ is a theory interpretation in the integrated consequence system iff
   (a) the transformations of the LPF Σ_A-sentences in $\mathbf{A}$ into classical logic Σ_B-sentences provided by i, are entailed from $\mathbf{B}$ in classical logic, and
   (b) the σ-translations of the classical logic sentences in $\mathbf{f}^{\langle a\rangle}(\Sigma_A)$ are entailed in classical logic by $\mathbf{B}$.

See [17] for a more detailed analysis.

In the case of B and VDM this has the effect that, given a translation from VDM into B, the (translations of the) theorems in (the LPF-interpretation of) a VDM specification $S1$ can be used as lemmata in an appropriately rich B specification $S2$ only when certain conditions (described by the *locality axioms*) are satisfied in (the first order classical theory interpreting) $S2$.

The basic advantages of this method for the correctness preserving integration the logical semantics of inter-dependent specification formalisms include the following:

1. a compound consequence system is produced which can accommodate the axiomatic semantics of each component formalism;
2. remote *interentailment* reasoning along non-plain mappings is transformed into internal reasoning along language translations;
3. the grammar and the entailment underpining the axiomatic semantics of each component formalism are locally conserved in the result of the integration; hence
   (a) the existing proof support for each component formalism can be reused;
   (b) the denotational semantics for each component formalism are not affected by the integration and they can be reused;
   (c) 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.

A detailed description of this method is given a separate technical report [17].

4.4 Issues (re)addressed in the context of the VDM+B project

The VDM+B project has been investigating the formal underpining of the co-use of interrelated heterogeneous specification formalisms with emphasis on the logics underlying the VDM and B formal methods. The VDM+B project will continue this development and also address the other issues outlined in section 3.

Core elements and foundational differences So far, in the first year of the VDM+B project, we have identified a general logical framework which we consider to provide a useful foundation for the formal underpining of the co-use of the VDM and B methods in a formal development. In this framework one can uniformly present the axiomatic and denotational semantics of VDM and B, and interrelate them by means of correctness preserving mappings. We have also provided a general method for integrating the heterogeneous consequence systems which accommodate the axiomatic semantics of VDM and B and identified the need of structural axioms in order to support multilogical reasoning in the integrated consequence system.

It is important to stress that so far our intention has *not* been to blend selected features of VDM and B into a new single formalism, but to develop a unifying framework which can accommodate the co-use of VDM and B *with respect to* the structure of the component formalisms and the (explicitly specified) interpretations between them.

Proof Support With respect to proof support (subsection 3.2), we recognise that in practice it may be hard to to reuse or transform proofs (derivations) across different formalisms. Consequently, we have taken care so that the integration method we are developing will allow existing proof structures to be used whilst identifying the locality conditions which ensure the sound interpretation of theorems from one formalism into the other. Hence, 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. We believe that the integrated axiomatic semantics should enable the interpretations of VDM and B to be expressed as separate modules within one supporting system. The intermodular translations and the associated locality axioms would then be realised by a third interfacing module. This will allow the co-use of existing or new purpose build provers for VDM and B.

Modular structuring With respect to structuring (section 3.3), 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 relate to each other by diagram morphisms built from *compatible families of theory presentation morphisms*[38] 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

[7] Diagrammatic presentations and diagram morphisms based on compatible families of parallel theory interpretations morphisms have been successfully applied in state-

describing the structured theories of Maude [18]. Based on Institution-theory, Durán and Meseguer recently proposed in [19] a method to construct a model theory $\mathbf{S}(\mathcal{E})$ on top of a model theory $\mathcal{E}$, such that (hierarchically) structured $\mathcal{I}$-theories treated as ordinary theories of $\mathcal{I}$. We plan to investigate the applicability of this method in our work. Also the use of parametric theory presentation diagrams proposed by Dimitrakos in [15, 16] supports the incremental construction of complex parameter specifications from simpler modules. It also 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 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.

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:

1. introduce an abstract mathematical framework for
   (a) uniformly presenting the axiomatic and denotational semantics of specification formalisms, and
   (b) interrelating such presentations by means of correctness preserving mappings;
2. provide a general method for integrating the heterogeneous consequence systems that accommodate the axiomatic semantics of interrelated specification formalisms; and to
3. 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, so far, has *not* been to blend selected features of VDM and B into a new single formalism, but to develop a unifying framework for co-using VDM and B with respect to the axiomatic semantics of each component and their interrelation with respect to correctness preserving interpretations between them, which need to be provided explicitly.

of-the art formal software development environments such as the SPECWARE [37] built at Kestrel Institute, California.

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 [40, 35, 36]. Our overarching aim in this project is to integrate useful formalisms without necessitating the abandonment of existing methods and tools.

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 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. The B method (virtual library page) `http://www.comlab.ox.ac.uk/archive/formal-methods/b.html`.
6. M. Barr and C. Wells. *Category Theory for Computer Science*, volume 5 of *Series in Computer Science*. Prentice Hall International, 1990.
7. 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.
8. 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.
9. J.C. Bicarregui, J. Dick, E. Woods, Quantitative Analysis of an Application of Formal Methods, Proc. FME'96, 3rd Int. Symp. of Formal Methods Europe, LNCS 1051, Springer-Verlag.
10. 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.
11. 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. Also appeared in Formal Aspects of Computing, 1999.
12. 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.

13. 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.

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*, pages 221–224, 1998. An extended revised version available on request at *theo@inf.rl.ac.uk*.

16. Theodosis Dimitrakos. On specifications parameterised by diagrams. Information Systems Engineering, CLRC, Rutherford Appleton Laboratory. Submitted for publication. An abstract also to appear at the Workshop on Algebraic Development Techniques, France 1999. Copies available at <*http://www.itd.clrc.ac.uk/Person/T.Dimitrakos*>.

17. Th. Dimitrakos, J.C. Bicarregui and T.S.E. Maibaum. Integrating Heterogeneous Formalisms: Framework and Application. Technical Report, Rutherford Appleton Laboratory, February 1999. Copies available at <*http://www.itd.clrc.ac.uk/Person/T.Dimitrakos*>.

18. Francisco Durán. A Reflective Module Algebra with Appliction to the Maude Language. PhD thesis. University of Málaga, 1999.

19. Francisco Durán and José Meseguer. Structured Theories and Institutions. Computer Science Laboratory. SRI Inernational. Submitted for publication.

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

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

22. 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.

23. 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.

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

25. IFAD (DK). The IFAD VDM-SL Tools. Welcome page URL <http://www.ifad.dk>, 1996.

26. C.B. Jones, Systematic Software Specification Using VDM (2nd Edition), Prentice Hall, 1990. Out of print. Copies available via `ftp` from

<ftp://ftp.cs.man.ac.uk/pub/cbj/ssdvdm.ps.gz>.

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

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

29. P.G. Larsen, B.S. Hansen, 'Semantics for Underdetermined Expressions', Formal Aspects of Computing, Vol 7. 1995.

30. A. Martini, U. Wolter, A Systematic Study of Mappings between Institutions. Proceedings of the *12th Workshop on Algebraic Development Techniques*, Springer LNCS 1376, pp. 300-315 (1998).

31. 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).

32. 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.

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

34. 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.

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

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

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

38. Y.V. Srinivas and R. Jullig. Diagrams for Software Synthesis. Kestrel Institute, Paolo Alto, California, 1993. Also appear in *Proceedings of the* KBSE'93.

39. Y.V. Srinivas. Refinement of Parameterized Algebraic Specifications. IFIP TC2 Working Conference on Algorithmic Languages and Calculi. Chapman & Hall 1997.

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

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

42. The VDM examples repository: `http://www.ifad.dk/examples/examples.html`. See also FME Industrial Applications Database at `http://www.fme-nl.org`.