

Towards a UML Profile for UML-B

Colin Snook, Ian Oliver and Michael Butler
{cfs,mjb}@ecs.soton.ac.uk, ian.oliver@nokia.com

Declarative Systems and Software Engineering Group
Technical Report DSSE-TR-2003-5
October 2003

<http://www.dsse.ecs.soton.ac.uk/techreports>

School of Electronics and Computer Science
University of Southampton
Southampton SO17 1BJ, United Kingdom

Towards a UML Profile for UML-B¹

Colin Snook^{*}, Ian Oliver^{**} & Michael Butler^{*}

^{*} University of Southampton, Southampton, England
{cfs,mjb}@ecs.soton.ac.uk

^{**} Nokia Research Centre, Helsinki, Finland
ian.oliver@nokia.com

Abstract. The UML is a popular modelling notation that has a natural appeal to hardware and software engineers and is adaptable through extension mechanisms. Formal (mathematical) modelling languages, on the other hand, are seen as difficult and costly to use and have achieved only limited use despite the benefits that they offer. In previous work, we have proposed an integration of UML and the formal notation, B and provided an automatic translator that produces a B specification. The integrated modelling notation, UML-B, inherits from both UML and B but primarily, is a specialisation of the UML. To achieve this integration we have specialised UML modelling elements via stereotypes, added tagged values to represent B modelling features and imposed constraints to ensure that UML-B models are translated into usable B. Here we describe ongoing work to define UML-B as a profile in accordance with the UML extensibility mechanisms.

1 Introduction

The UML [11] is successful as a visual modelling notation for the design and communication of object-oriented systems. Formal (mathematical) modelling notations provide verification and validation benefits. By providing a formal interpretation of UML modelling elements and adding behavioural specifications in an integrated format translation to a formal notation is possible. This enables the consistency of a model, internally and between successive refinements, to be formally verified by proof. Also, model checking tools provide an effective, faster and cheaper verification mechanism than proof while still being more rigorous than traditional verification techniques such as reviewing. Animation provides a means of validating specifications prior to implementation. These verification and validation processes are not available in the UML even if annotated constraints are added in the UML constraint language, OCL [16]. Hence the main motivation is to provide a translation from the UML to a recognised formal notation that has good tool support to enable rigorous verification and validation of UML models. However, there is no such formal notation that follows the object-oriented paradigm. Therefore translation from unrestricted UML models is problematic. To overcome this we define a specialisation and enhancement of the UML that is amenable to translation into the chosen formal notation.

¹ This work was funded by the EU Research Project, PUSSEE (IST-2000-30103). [10]

1.1 The B Language and Toolkit

The B language [1] is a formal specification notation that has strong decomposition mechanisms and good tool support. The primary aim of decomposition in B is to obtain compositionality of proof. There are two commercial tools for B, Atelier B [3] and the B Toolkit [2]. B is designed to support formally verified development from specification through to implementation. To do this it provides tool support for generating and proving proof obligations at each refinement stage. There are also tools available for model checking and animation of B models [7]. To make large-scale development feasible, B provides structuring mechanisms to decompose a project into ‘modules’. Each module consists of ‘components’ (machines, refinements or implementations). Each module includes an abstract machine and possibly, refinements until an implementation is reached. The implementation may then import abstract machines of other modules in a hierarchical module structure.

B was originally intended for specifying software systems where a procedural style is adopted. We refer to this style of B as ‘procedural B’. A new style of modelling in B has recently been developed for systems modelling where the necessary conditions for events to occur and the consequent state changes are specified. We call this style of B ‘event B’ [4]. Event B has some similarities of interpretation with B action systems [16].

1.2 U2B translator and UML-B profile

The U2B translation [12],[13],[14] translates class diagrams with attached state charts into a B. The Class diagram defines the structure of B components and their variables. Further textual information in the specifications of classes and operations defines constraints and operation semantics. This textual information is expressed in a form of the B notation that adopts an object oriented dot style for referencing class instances. State Machines may be attached to classes and used to define the effect of operations on a state variable. Refinements between classes can be represented and a large system can be decomposed using a hierarchy of UML packages. Different styles of model are catered for including a conventional B development or an event or action systems approach.

However, B is not an object oriented modelling language and consequently some UML features are difficult to translate. B contains restrictions in order to achieve its primary goal of decomposition into provable modules. In order to achieve a successful translation, restrictions must be placed on the UML models that can be translated and in many cases UML modelling elements are assigned special meanings. The UML contains extensibility mechanisms for defining specialisations of its notation for use in particular modelling types. The UML profile [9] is the main mechanism for doing this. A UML profile may include the following steps:

1. Identification of the UML subset relevant to the profile.
2. Definition of specialisations using the extension mechanisms of UML (stereotyping and tagged values).
3. Imposition of syntactical rules that restrict the models that can be created. (Well-formedness rules)

4. Definition of a particular semantics.

This paper describes a UML profile, called UML-B, that embodies these restrictions and semantics.² The semantics of UML-B are provided by examining the semantics of the equivalent B model produced by the U2B translator.

UML-B is a class of models. It is a specialisation of the class of models that use the UML notation. It also inherits from the class of models that use the B notation. The U2B translator reads a UML-B model and creates a B model.

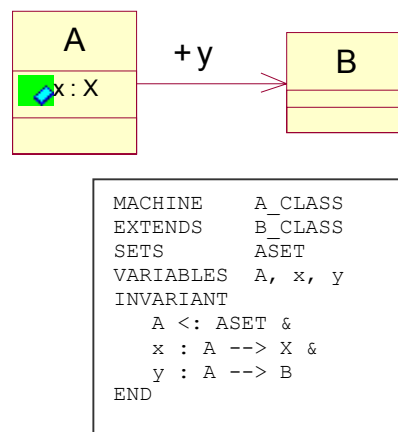


Fig. 1. A simple class diagram and associated B machine

1.3 Overview of U2B Translator

The U2B translator bears some similarities with other authors proposals (e.g. [8]) for translating UML into B. The main difference in our approach is that we specialise the UML in order to ensure that the resulting B is amenable to verification.

Structure and Static Properties. A B component is created for each class and includes modelling of class instances. Attributes and (unidirectional) associations are translated into variables whose type is a function from the current instances to the attribute type (as defined in the Class diagram) or the associated class instances. Attribute types may be any valid B expression that defines a set. A simple example of a class with an association and an attribute and its corresponding B translation is shown in Fig. 1. (A separate machine will be generated for class *B*).

Class cardinality may be fixed (set to a particular value), mutable (set to a range such as 0..n) or singleton (set to 1). For variable instance classes, instance modelling is by a variable (as shown in Fig. 1) and a create operation is automatically provided. For fixed instance classes, instances are modelled by a constant set of integers that are

² Due to space restrictions only partial details of the profile are given in some sections. A full version of the paper can be found at <http://www.ecs.soton.ac.uk/~cfs/>

used to index the instances. If class cardinality is singleton, the U2B translator creates a machine without instance modelling.

In UML, model elements may have associated textual specification fields. For classes, one of these is a ‘documentation’ field. Any valid B clause can be added in the documentation field. For example, we use this method to specify invariants. Each clause must be headed by its B clause name.

Dynamic Behaviour. The dynamic behaviour is specified either in a text field or in a statechart attached to the class. An operation may be specified completely by (pre-condition and semantics) text fields, completely by statechart transitions, or by a simultaneous composition of both.

Textual Behaviour Specification. Text fields are used to specify pre-conditions, guards and semantics for the operation. As there is no specific text field for a class invariant we use a clause in the documentation text of the class' specification. UML does not impose a particular notation for constraints. Since we wish to translate to B we use B notation but support the object-oriented modelling conventions of implicit self-referencing and use of the dot notation for explicit instance references.. For example, if class *A* has an association, *as*, to class *B*, we might write *as.at*, where *at* is an attribute of class *B*. This would be translated to *at(as(thisA))*.

Statechart Behavioural Specification. If a statechart model is attached to a class U2B combines the behaviour it describes with any textual semantics of the operations. The name of the statechart model provides an attribute whose type is the set of states Transitions associate operations with state changes. The transition's event must be the operation name. Additional guards and actions may be attached to transitions. The translator finds all transitions associated with an operation and compiles a SELECT substitution of the following form

```
SELECT statevar=sourcestate1 & sourcestate1_guards
THEN statevar:=targetstate1 || targetstate1_actions
WHEN statevar=sourcestate2 & sourcestate2_guards
THEN statevar:=targetstate2 || targetstate2_actions
ELSE skip END ||
<operation body from semantics window>
```

2 UML Subset Relevant to UML-B.

Since, in UML-B models, full behavioural information is attached to the model elements themselves, the scenario-based notations of UML (collaboration diagrams) are not used. Currently UML-B does not use the use case notation either, though this may be considered in the future. This section identifies the subset of the UML notation upon which UML-B is defined. This is done, first by identifying the relevant packages and excluded classes from the UML metamodel, and secondly by listing the UML model elements. In the latter, we take the opportunity of specifying some constraints on ownership relations between model elements.

2.1 Identified Base Subset of the UML Metamodel

The UML-B profile retains the following from the UML Metamodel

Foundation::Core:: *Excluding:*
 Method (from Backbone),
 AssociationClass (from Relationships),
 Permission and Usage (from Dependencies),
 Interface, Node, Component, ElementResidence and Artifact (from Classifiers),
 Foundation::ExtensionMechanisms
 Foundation::DataTypes (*for use in the metamodel*)
 BehaviouralElements::CommonBehaviour *Excluding:*
 Signal, Exception and Reception (i.e. all of signals)
 CreateAction, ReturnAction, SendAction TerminateAction, UninterpretAction and DestroyAction (from Actions)
 SubSystemInstance, ComponentInstance, NodeInstance (from Instances)
 BehaviouralElements::StateMachines *Excluding:*
 SynchState, SubState, CompositeState,FinalState and SubMachineState (from Main)
 SignalEvent, TimeEvent,ChangeEvent (from Events)
 ModelManagement *Excluding:*
 Subsystem

2.3 Ownership and Type of UML Subset Model Elements

Table 1 lists the retained UML model elements and specifies UML-B constraints on the ownership of one model elements by an other.. Where the UML feature is a text field, type information is also given. The indentation indicates ownership of features by other UML features. The number range in the left hand column of the table, headed #, gives the permitted cardinality of the ownership of that feature. For example there must be one and only one package at the top level that owns everything else. This package may have one stereotype. If it does, then the value of that stereotype must be one of the strings listed under stereotypes a.

Table 1. Ownership cardinality and field type constraints on the use of UML modelling elements in UML-B. (# = cardinality constraints of ownership)

#	UML feature	field type
1..1	package	
1..1	stereotype	procedural system event system action system
0..n	package	
1..1	stereotype	module
1..n	class	
1..1	stereotype	machine refinement implementation
1..1	name	identifier
1..1	cardinality	number range

#	UML feature	field type
0..n	parameter	
1..1	name	identifier
1..1	type	type set
0..n	attribute	
1..1	name	identifier
1..1	type	type set
0..1	initial value	value ::set
0..n	operation	
0..1	documentation	comment
1..1	name	identifier
0..1	return type	identifier list
0..n	parameter	
1..1	name	identifier
1..1	type	type set
0..1	pre-condition	UML-B predicate
0..1	semantics	UML-B substitution
0..1	documentation	UML-B clause list
0..1	state model	
1..1	name	identifier
0..1	initial state	
1..n	state	
1..1	name	identifier
0..n	transition	
1..1	name	identifier
0..1	guard	UML-B predicate
0..1	action	UML-B substitution
1..1	target	<state> <decision>
0..n	decision	
1..n	transition	
0..n	association	
1..1	client multiplicity	number range
1..1	supplier	<class>
1..1	supplier rolename	identifier
1..1	supplier multiplicity	number range
0..n	dependency	
1..1	stereotype	sees includes imports refines
1..1	name	identifier
1..1	supplier	<class>

3 UML-B Tags

In this section we describe how UML-B specialises standard UML model elements. This is done using the extensibility features (stereotypes and tagged values) of UML. We assume the unification suggested by Fontoura, Pree and Rumpe in their UML-F

profile for Framework Architectures []. In this unification stereotypes are equivalent to boolean tagged values. The existing stereotype notation, <<stereotypeName>>, may be used but is deemed equivalent to a tagged value: {stereotypeName=true}. This enables us to talk about *tags*, which are *tagged values*, some of which may be *boolean* a.k.a. *stereotypes*. In many case, although a tag has been defined in the UML-B profile, the specialisation is derived implicitly from the context of the model element. Where this is the case the conditions for implication are described via footnotes.

3.1 Model Types

The UML-B model can use one of several modelling styles. The intended modelling style is indicated by a tag on the top-level package (*Logical Package*). The translation to B operates differently depending on the modelling style. The available modelling styles are shown in table 2 (note that these tags are only valid when applied to the top level package in a model).

Table 2. UML-B tags for model types

Tag Name	Applies to	Type	Description
procedural system	Logical Package	Boolean	The model type is a conventional model where operations are interpreted like procedures.
event system	Logical Package	Boolean	The model represents an event system where operations are interpreted as events
action system	Logical Package	Boolean	The model represents an action system where operations are interpreted as actions

3.2 Model Architecture

Packages are used to represent B modules. Each package (module) contains one or more classes. Classes represent abstract machines, refinements or implementations. (See the B definitions of these modules and component types for an understanding of their meaning). The components may have dependency relationships between them that are interpreted in various ways. The tags that are used to distinguish these modelling specialisations are shown in table 3.

Table 3. UML-B tags for model structuring and interrelationships

Tag Name	Applies to	Type	Description
module	Package	Boolean	the package represents a B module.
interface	Class	Boolean	The class represents an abstract machine component.

Tag Name	Applies to	Type	Description
machine	Class	Boolean	The class represents an abstract machine component.
refinement	Class	Boolean	The class represents a refinement component.
implementation	Class	Boolean	The class represents an implementation component
includes	Dependency	Boolean	the supplier component is included in the client (equivalent to machine inclusion)
imports	Dependency	Boolean	the supplier component is imported in the client (equivalent to machine importation)
sees	Dependency	Boolean	the supplier component is seen in the client (equivalent to machine sees)
refines	Dependency	Boolean	the client component refines the supplier component (equivalent to refinement in B). (For notation the UML realises arrow is used).
parameters	Dependency	list of identifiers(,)	a list of actual parameters when instantiating a parameterised component (via includes or imports)

3.3 Additional Information in UML-B

The tags in this section provide a mechanism for specifying additional modelling elements that are not catered for by specialisations of UML modelling elements. Most of these tags provide a mechanism for adding textual B clauses to the UML modelling element representing a UML-B component. In these cases, the tag name is the corresponding clause name in B from which the purpose of the tag value can be deduced. (Only the most commonly used clauses are shown here). Characters in brackets in the type column are the separator to be used when the type is a list.

A class cardinality tag is used to specify variations in class instance modelling. If a number range is specified, variable instance modelling is provided including a create operation. However, in many modelling situations a fixed number of objects exist. If the class cardinality is set to a fixed number (i.e. the upper and lower extent of the range are equal, then a constant set of instances is modelled. Since all instances exist from the beginning, there is an implicit initialisation phase where they are assigned initial values. It is also useful to model an implicit single instance. If the cardinality is set to one, no instance modelling is provided simplifying the model.

A tag is provided in order to identify the list of operation return parameters. B infers the type of return parameters from the operation body. Hence the UML operation

return type is unused. However, instead, B needs to know which identifiers represent the definition of the returned values.

Table 4. UML-B tags for instance cardinality and additional B fields

Tag Name	Applies to	Type	Description
CARDINALITY	Class	number range	the cardinality of the set of current class instances is constrained to equal a value in this number range. (n or * can be used to indicate the lack of a limit at either end of the range.
CONSTRAINTS	Class	predicate	a predicate that expresses a constraint on the parameters of the component
SETS	Class	identifier list (;)	a list of deferred or enumerated sets
CONSTANTS	Class	identifier list (,)	a list of concrete constants
PROPERTIES	Class	predicate	a predicate that expresses a constraint on the constants of the component
DEFINITIONS	Class	definition list (;)	a list of definitions
VARIABLES	Class	identifier list (,)	a list of variables
INVARIANT	Class	predicate	a predicate that expresses a constraint on the variables of the component
results	Operation	identifier list (,)	list of identifiers that indicate values returned by the operation body.

4 UML Model Elements Used Without Tags

This section describes how standard UML model elements are used in UML-B.

Class Name - The class name is used as the basis of the component name. The class name also represents a set of instances which may be referred to in the behavioural specifications of other classes.

Class Parameters - If a parameterised class is used, the class parameters are interpreted as constants or sets when they are instantiated. The class parameter name is a placeholder for the actual constant or set. The parameter type may be used to specify a set that will be used to constrain the possible values of the parameter.

Class Attributes - Class attributes are variables of the component. The attribute type is a set that constrains the possible values of the variable in an invariant. The attribute may be given an initial value that, for variable instance classes, is the value assigned to the variable in the 'create' operation, and for fixed instance and singular classes, is the value given to the variable at initialisation.

Class Associations - Class associations are variables of the component. The supplier role name is the name of the variable and the type of the variable is a relationship between the instances of the client and supplier classes. The multiplicities of the supplier and client roles in the association represent an additional constraint invariant on the values of the association variable.

Class Operations - Class operations describe possible changes in the state of the class. In an event or action system they represent a description of events or actions that might happen within the system. In this case, apart from describing the change to the state variables of the class, they may also constrain (via guards) the state under which they do and do not occur. Note that an event does not occur unless there is an available choice action where all the guards are true, including those of call operations if applicable. In a procedural system, the operation must not control when it is available and hence there is always an implicit do nothing transition interpreted from state charts if no state transition is unguarded.

Operation Parameters - The operation parameter name is a placeholder for the actual value passed when the operation is invoked. (For event and action systems the parameter represents variations in a family of similar events or actions). The parameter type may be used to specify a set that will be used to constrain the possible values of the parameter.

Operation Pre-conditions - For procedural systems the operation pre-condition describes constraints on the parameters and class variable state in which the operation actions will occur. If the operation is invoked outside of its pre-condition anything can occur. For event or action systems, the pre-condition represents a guard that defines when the operation does and does not occur.

Operation Semantics - The operation semantics describe changes to the state of class variables that occur when the operation is called or when the event/action occurs. (The operation semantics occur in parallel to any state changes defined by a state model).

Class State Model - The class state model also describes changes to the state variables that occur when the operation is called or when the event/action occurs. The state model name represents an attribute of the class whose value determines the current state as described in the state model.

State Model States - The state model states are a set of values that the state model attribute can take.

State Model Transitions - State model transitions represent possible changes to the state model attribute. The new value of the state model attribute is the target state but the new value can only be assigned when the current value equals the starting state. The state change can only occur when the relevant operation is called or when the event/action occurs. The relevant operation or event/action is given by the transitions event attribute. The transition may also specify additional guards and actions.

State Model Decision Pseudostates - Decision pseudostates may be used to split a single transition into a choice of alternative transitions and to merge several transi-

tions into a single transition. This is for convenience since the choice/merge can be represented without decision pseudostates as separate transitions. However, the technique has been found useful when refining state models.

5 Summary of UML-B Mapping on to UML

This section summarises the specialisation of UML model elements in UML-B by listing the UML-B model element and giving the UML model elements and/or tags from which it is derived.

Table 5. Summary of mapping from UML-B model elements to standard UML model elements

#	UML-B model element (items in italics have no corresponding feature in B)	UML model element and/or UML-B profile tag
1..1	event B model	Logical View package <<event system>>
1..1	action B model	Logical View package <<action system>>
1..1	procedural B model	Logical View package <<procedural system>>
0..n	B module	package <<module>>
1..n	machine component	class <<machine>>
0..n	refinement component	class <<refinement>>
1..1	implementation component	class <<implementation>>
1..1	component name	class name
1..1	component instance modelling variable or constant	class name
1..1	component instance modelling constraint or invariant and initialisation	class {CARDINALITY}
0..n	component parameter	class parameter
1..1	parameter name	parameter name
1..1	parameter type constraint	parameter type
0..n	component variable	class attribute
1..1	variable name	attribute name
1..1	variable type invariant	attribute type
0..1	variable initial value	attribute initial value
0..n	component operation	class operation
0..1	operation documentation	operation documentation
1..1	operation name	operation name
0..1	operation return identifiers	operation {results}

#	UML-B model element (items in <i>italics</i> have no corresponding feature in B)	UML model element and/or UML-B profile tag
0..n	operation parameter	operation parameter
1..1	parameter name	parameter name
1..1	parameter type precondition	parameter type
0..1	operation precondition	operation precondition operation semantics
0..1	operation body	class state model (see below)
0..1	component state variable	class state model
1..1	state variable name	state model name
1..1	state variable type set and invariant	state model set of states
1..n	type set element	state name
0..1	state variable initial state state variable (affects interpretation of transitions)	state model target of transition from initial state
0..n	state variable changes in operation body (see operation body above)	state model decision pseudostate
0..n	(locate relevant operation)	state model transition
1..1	state variable additional guard	transition name
0..1	state variable additional action	transition guard
0..1	state variable new value	transition action
1..1	value	transition target
0..n	component mapping to instances of another component	class association
1..1	mapping extended other component	association supplier association supplier role name
1..1	mapping variable name	association client role
1..1	mapping variable invariant (range part)	multiplicity
1..1	mapping variable invariant (domain part)	association supplier multiplicity
0..n	component includes other component	class dependency <<includes>>
1..1	includes other component	dependency supplier

#	UML-B model element (items in italics have no corresponding feature in B)	UML model element and/or UML-B profile tag
0..1	includes rename includes actual parameters	dependency name dependency {parameters}
0..n	component imports	class dependency <<imports>>
1..1	imports other component	dependency supplier
0..1	imports rename imports actual parameters	dependency name dependency {parameters}
0..n	component sees	class dependency <<sees>>
1..1	sees other component	dependency supplier
0..1	component refines	class dependency <<refines>>
1..1	refined component	dependency supplier
0..1	component textual B clauses	class documentation

6 Well-Formedness Rules

This section lists rules that must be followed when constructing UML-B models.

1. Identifiers must be unique throughout a model. An identifier is a string of two or more characters from a..z, A..Z, 0..9, _ The first character must be from a..z, A..Z. Due to instance modelling naming conventions, class name identifiers must not contain characters from a..z. An identifier list is a comma separated list of identifiers.
2. Number Ranges must be an expression of the form min..max giving the set of numbers which are permitted. Min or max can be left indeterminate by setting to either n or *. For example, 0..n = any number, 1..n = any number but at least one, 1..1 = exactly one, 5..5 = exactly five
3. Type Sets must be an expression resulting in a set that is visible to the class. For example, any valid B base type (e.g. NAT), the name of another class, an expression involving other attributes/associations of the class (e.g. union, function), a set (POW, FIN etc) of any of these, a sequence (seq, iseq etc) of any of these.
4. Attribute initial values are optional but if given must either be a value from the type set or “...” followed immediately by a B expression giving a subset of the type set. If the latter form is used the variable will be non-deterministically assigned one of the values from the subset.
5. All Relationships must be binary
6. Only association relationships can loop back to the same class
7. Only sees and imports relationships can be between classes in different packages
8. Associations must be navigable in (only) one direction.
9. Each package must own exactly one root machine (a root machine is a class that is not the supplier of any intra-package association or dependency and not the client) of any refines relationship.

10. The structure of intra-package relationships (excluding sees relationships) between classes must be a tree.
11. Multiple associations between the same pair of classes must be in the same direction
12. UML-B predicates and substitutions are equivalent to their B counterparts except that they use the dot notation for instance referencing.
13. UML-B substitutions must not contain simultaneous calls to operations of another machine (whether via the same or different associations).
14. UML-B substitutions must not contain calls to operations within that machine (even if the class has an association to itself).
15. A class cannot be the client of a refines relationship and the supplier of an association.
16. Implementations cannot be the client or supplier of an association
17. The chain of refines relationships within a package must be a linear sequence from an implementation or refinement (the most concrete specification) to a root machine (the most abstract specification).
18. An implementation class must be the client of a refines relationship and must not be the supplier of a refines relationship
19. If a class has parameters the only relationships it can be the supplier for, are includes relationships.
20. Includes relationships to a parameterised class instantiate the supplier class and must have the following format for the relationship name: `rename(actual1,actual2...)`, where `rename` is an optional new name for the instantiated class (to distinguish it and its namespace from other instantiations) and the list of actual parameters matches the formal parameters of the instantiated class (and are of valid type).
21. Between each pair of states that has a connecting sequence of transitions (via decision points if sequence is more than one transition), there must exist exactly one transition that is named and that name matches an operation of the class.

7 Future Work

The current translation produces a B component for each class in the UML model. Due to the restrictions B places on references between components this imposes significant restrictions on the relationships between classes. To enable the use of more typical class relationship structures we are now developing translation strategies that produce a single B component for an entire UML package. Further restrictions then prevent a mapping from class methods to B operations. Therefore instead of taking an operational approach to defining class dynamics we define behaviour as externally available actions in terms of the pre and post state of class features. This is translated into B operations having write access to all class features in the package.

8 Conclusions

The UML-B profile defines a specialisation of UML models with additional tags that enable the translation of UML-B models into the B notation. The B form allows rigorous verification and validation of UML-B models. While it is necessary to place restrictions on the kind of UML models that can be translated, the essential benefits of the UML are retained, especially if the changes envisaged in current work are successful.

References

1. Abrial, J.R.: The B Book - Assigning programs to meanings. Cambridge University Press. (1996)
2. B-Core: B-Toolkit User's Manual, Release 3.2. B-Core(UK) Ltd, Oxford (UK). (1996)
3. ClearSy: AtelierB User Manual, V3. ClearSy System Engineering, Aix-en-Provence (F).
4. ClearSy: Event B Reference Manual v1. ClearSy, (2001).
5. Fontoura, M., Pree, W. & Rumpe, B.: UML-F profile for Framework architectures. Addison Wesley (2002)
6. LeComte, T.: Abstract System Modelling. Draft <http://www.keesda.com/pussee/> (2003)
7. Leuschel, M. & Butler, M.: ProB: A Model Checker for B (to be submitted to FME2003) <http://www.ecs.soton.ac.uk/~mal/systems/prob.html>
8. Meyer, E. & Souquières, J.: A systematic approach to transform OMT diagrams to a B specification. In J. Wing, J. Woodcock, J.Davies, editors, World Congress on Formal Methods in the Development of Computing Systems, FM'99, Vol. I, LNCS, Vol.1708, Springer-Verlag, pp.875-895. (1999)
9. OMG Manual: <http://www.omg.org>
10. PUSSEE project web site: <http://www.keesda.com/pussee/>
11. Rumbaugh, J., Jacobson, I. & Booch, G.: The Unified Modelling Language Reference Manual. Addison-Wesley. (1998)
12. Snook, C. and Butler, M.: Verifying dynamic properties of UML models by translation to the B language and toolkit. In G.Reggio, A.Knapp, B.Rumpe, B.Selic, R.Wieringa, editors, Dynamic Behaviour in UML Models: Semantic Questions, UML 2000 workshop proceedings, pp.99-105. (2000)
13. Snook, C. and Butler, M.: Using a graphical design tool for formal specification. In G.Kadoda, editor, Proceedings of the 13th Annual Workshop of the Psychology of Programming Interest Group, pp. 311-321. (2001)
14. Snook, C. and Butler, M.: U2B - UML to B translation tool and Manual V4.4 available at <http://www.ecs.soton.ac.uk/~cfs/U2Bdownloads/U2Bdownloads.htm>
15. Warmer, J. and Kleppe, A.: The Object Constraint Language: precise modeling with UML. Addison-Wesley. (1999)
16. M. Walden and K. Sere.: Reasoning About Action Systems Using the B-Method. Formal Methods in Systems Design 13(5-35). Kluwer Academic Publishers. (1998)