Using BridgePoint 5.1 for Executable UML

We had three primary goals in constructing the models in *Executable UML*:

- First the reader must be able to execute the models. This means the reader must be able to download a tool (BP) and execute the models in the book, as directly as possible.
- Second, the examples in the book must be compliant with UML 1.4 and the UML Action Semantics.
- The third goal is pedagogical. The material has to be easy to understand on first reading. Details of exactly how the models are built in practice can be delayed.

Consequently, there are some minor differences between *Executable UML* and BridgePoint 5.1 that we described in this document. These differences fall into four main areas:

- Use of identifiers
- Use of referential attributes
- Action language shorthands
- Minor notational reconciliation

Use of Identifiers

We recommend you think about how each instance of a class is identifed, because it promotes better understanding of exactly what each abstraction is. Once you have worked out exactly what an abstraction is, it may be identified by a naturally occurring combination of attributes, in which case, you should make those attributes an identifier of the class in your model. If not, *Executable UML* does not require a contrived identifier on every class. Many model compilers, however, do require every class to have an identifier, so it is safer to include a contrived identifier anyway.

BridgePoint 5.1 and the Model Verifier work correctly without required identifiers. However, the default settings for the Audits will give warnings about classes without identifiers. Change these settings to eliminate the audit warnings. Note that referential attributes make links easier to visualize in the Model Verifier.

Use of Refererential Attributes

We recommend you think about how each association is formalized in the model. Generally, UML (as distinct from Executable UML) allows and even encourages the addition of referential attributes, though most usually as an implementation decision. Many model compilers, however, do require every association to be formalized, so it is safer to include a the referntial attribute anyway.

BridgePoint 5.1 and the Model Verifier work correctly without referential attributes. However, the default settings for the Audits will give warnings about relationships not formalized with referential attributes. Change these settings to eliminate the audit warnings.

Association Classes

Because Executable UML does not require contrived identifiers and referential attributes, association classes need not be constructed unless the association has additional attributes or a lifecycle that would be formalized as the state machine of the association class.

BridgePoint 5.1 requires association classes for all many-to-many associations, regardless of whether the association has additional attributes or a separate state machine. BP will not create a many-to-many association unless the association already has an association class.

Constrained Association Loops

Because Executable UML cannot rely on contrived identifiers and referential attributes, Executable UML expresses constrained association loops as constraints. We recommend you think very carefully about constrained association loops, and express them as written constraints in the relation descriptions. Few model compilers, if any, make use of the complex metamodel that supports constrained association loops, but it is wise to check, just in case some optimization is being made.

Action Language Shorthands

Executable UML uses BridgePoint's Object Action Language (OAL). However, there are some differences between the action language in the book and that supported by BridgePoint 5.1.

Class Abbreviation and Event Label Conventions

The book always uses a class abbreviation equal to the class name itself, while OAL uses class abbreviations (keyletters) and event labels (keyletter + number) rather than class names and event names. For the following examples, assume two classes: Dog (keyletter D) and DogOwner (keyletter DO).

1. The BP create object statement uses the class abbreviation, not the class name:

Book statement	BP 5 equivalent
create object instance myDog of Dog;	create object instance myDog of D;

2. BP navigation expressions use the class abbreviation, not the class name:

Book statement	BP 5 equivalent
select one owner related by myDog->DogOwner[R1]	select one owner related by myDog->DO[R1]

3. BP generate statements use the class abbreviation in addition to the class name

Book statement	BP 5 equivalent
generate D1:bark to myDog;	generate D1:bark to myDog;]

Note that BP 5.1 limits the class keyletter to 15 characters.

Generalization Hierarchy Action Language Shorthands

The action language used in *Executable UML* has some additional shorthands. Some of which come about as a consequence of treating an object as a single entity conforming to several classes, and other are mere shorthands designed to make the models smaller on the page. There is no requirement to use these shorthands.

1. There is no single statement in BP to create an object of a subclass and superclass. Instead, create each object separately and relate them

Book statement	BP 5 equivalent
create object instance shipClerk of ShippingClerk;	create object instance shipClerk of ShippingClerk; create object instance whClerk of WarehouseClerk; relate shipClerk to whClerk across R27;

2. BP supports compound generalization by creating each object separately and relating them.

Book statement	BP 5 equivalent
create object instance newBook of (BookProduct, SpecialOrderProduct);	create object instance newBook of BookProduct; create object instance newProduct of Product; create object instance newStkProd of StockedProduct; relate newBook to newProduct across R11; relate newProduct to newStkProd across R12;

3. BP supports reclassificationby deleting the old subclass and creating the new one.

Book statement	BP 5 equivalent
reclassify theClerk as OffDutyClerk;	select one whClerk related by theClerk->WarehouseClerk[R27]; unrelate whClerk from theShippingClerk; delete object instance whClerk create object instance newOffDutyClerk of OffDutyClerk; relate whClerk to newOffDutyClerk across R27;

This same sequence is required when reclassifying objects with state machines. The final state of the old subclass must unrelate the subclass from the superclass and send a creation signal to create the new subclass. The initial state of the new subclass must look up the superclass using an identifier and relate itself to the superclass. See the Clerk examples in the online bookstore case study for examples.

Association Class Action Language Shorthands

The following shorthands are used in association classes.

1. In BP linkobjects and the link are created separately. The link object must be created first and then the link is made “using” the link object. Linking without the link object does not create the link object by default, it is an error.

Book statement	BP 5 equivalent
relate newBook to newAuthor across R2 creating newAuthorship;	create object instance newAuthorship of Authorship; relate newBook to newAuthor across R2 using newAuthorship;

2. BP selects the link object that relates two association end objects by selecting with identifier and referential attribute values.

Book statement	BP 5 equivalent
select theAuthorship that relates theBook to theAuthor across R2;	select any theAuthorship from instances of Authorship where selected.authorID == theAuthor.authorID and selected.bookID == theBook.bookID;

This of course requires identifiers on the two association end classes (Book and Author) and referential attributes in the association class (Authorship).

Minor Notational Differences

This section outlines differences between the diagrams shown in *Executable UML* and those produced by BridgePoint 5.1

Class Diagrams

1. Derived attributes are tagged {M} in BP, and with a slash ('/') in the book, so as to be closer to UML.

2. Multiplicity strings always written in their fullest form in the book:
   * in BP is always written 0..* in the book to avoid confusion
   BP 0,1 is always written 0..1

3. Signals received by a class do not appear in the bottom compartment of the class box.

Statecharts

1. In BP, signals are labeled with the class abbreviation plus a number. These signal labels are required in Object Action Language (OAL) statements along with the signal name. The book suppresses this additonal information for brevity.

2. Two or more creation events are shown with distinct initial pseudostates.

Collaboration Diagrams

1. When there are events in both directions between two classes, BP draws two separate lines and sets of arrows, while the book draws a single line.

2. BP does not produce annotated (numbered) collaboration diagrams or sequence diagrams.