# Background on issues BACM-84 and BACM-83

The BACM specification has two distinct tool targets: UML-based tools and RDF/logic based tools. These targets have historically had different semantic foundations. The BACM specification attempts to define a metamodel that is consistent with both targets by including SMOF as a required feature.

The MOF Support for Semantic Structures – SMOF is an OMG specification that alters MOF in order to support modeling RDF/logic systems. Among other things, it eliminates the MOF requirement that an instance has a single metaclassifier that is fixed when the instance is created, and replaces it with instances that can have multiple metaclassifiers and a MOF method that allows instance metaclassifiers to be added and deleted. This is useful in business architecture because it allows modelers to create concepts that are simultaneously classified by two or more metaclassifiers in the BACM metamodel. For example, a type of assembly robot may be a Performer with respect to a CapabilityBehavior in one point of view, while simultaneously being a BusinessObject that is the subject of an asset management Capability in another point of view. For this reason

However, there are metaclassifiers that are intrinsically disjoint, and no instance may have both as metaclassifiers at the same time. For example, metaclassification of an instance as both an Outcome and a BusinessObject does not make sense in any perspective as these are fundamentally distinct concepts. The SMOF specification deals with this by adding two new types of constraint to the MOF language: disjoint and equivalent. The disjoint constraint defines a group of metaclassifiers such that no instance may have more than one member of this group as a metaclassifier at the same time. There is an interesting philosophical discussion to be had about the meaning of changing an instance that is an Outcome to being a BusinessObject, but that is only tangentially relevant to the practice of business architecture.

Consistency is a notion of logic systems meaning that according to the proof theory stipulated, the logic system contains no contradictions. Proof theories vary in computational complexity from polynomial time and space to being practically unsolvable and most current computational proof theories fall into the polynomial category but are less powerful in terms of inconsistencies they can detect. OWL 2 is based on such a proof theory with the opaque title “SROIQ Description Logic”. UML-based tools have lacked computational consistency checkers that would help UML modelers detect logic mistakes such as assuming that instances are disjoint when they in fact have a common metaclassifier. With the help of SMOF, a BACM-based business architecture model can be translated into OWL 2 and its consistency verified by a consistency checker.

Both OWL 2 and UML distinguish classes from individuals (OWL 2) and classifiers from objects (UML). In UML, an object is associated with its classifier and an object’s classifier may be discovered by a MOF method implemented by the object. In OWL 2, an individual is associated with its classifiers either by a direct assertion or an implied assertion created by a consistency checker (reasoner). An OWL 2 class and a UML classifier are intensional definitions of sets of OWL individuals and UML objects, respectively. An intensional definition is a group of logic propositions all of which must be true of an individual for the individual to be a member of the class (respectively, an object instance of the classifier). This group of logic propositions includes association end quantifiers in UML and ObjectProperty and DatatypeProperty statements in OWL 2. UML modeling tools create objects from the metaclassifiers in a way that insures that some of the logic propositions are true and others can be checked by the tools. OWL 2 systems allow the modeler to create individuals and then use the consistency checker to determine the classification of these individuals.

A business architecture model is created from the BACM metamodel by specializing the concepts in the BACM metamodel. For example, a “Market Management” capability (from the Business Architecture Guild Financial Services Reference Model) specializes the concept of a Capability (as the ability to manage a business object for the benefit of the business) to be an ability to manage a market for the benefit of a business. Note that “market” is not actually an object (UML) or an individual (OWL 2). It is a classifier. The “New York Stock Exchange” term would identify an object (individual) that is classified by the class “market”. Nearly all of the business concepts that would appear in a business architecture model are classifiers, not individuals.

In a UML modeling tool, one can create both UML classes and UML objects (instances of a UML class) based on a metamodel that is itself specified in UML. To make this work, the tools treat instances of the UML metamodel classifiers as UML classes, and instances of UML classes as UML objects. UML modeling tools must treat the BACM metamodel in the same way; the BACM metamodel effectively specializes the MOF metamodel as altered by SMOF and is itself specialized by the creation of business architecture models. For some UML-based tools, this may be easiest to accomplish by using profiles and stereotypes to represent the specialization of UML classes and associations to the BACM metamodel concepts. The use of profiles and stereotypes faces challenges in dealing with multiple metaclassification allowed by SMOF, and such tools may be forced to implement common multiple metaclassifications (e.g. Performer/BusinessObject) in their profiles. Other UML-based tools will be able to directly specialize the BACM classifiers to create business architecture models.

The RDF/logic based tools will accomplish the same objective by treating the BACM metamodel as an unmodifiable upper ontology that can be specialized to create a business architecture ontology. The modeler will be able to create and delete generalizations as needed to define multiple metaclassifications.

The BACM metamodel as currently specified does not define objects or individuals. This is likely to be a future extension of the BACM metamodel so that it can represent specific things in a business. But, existing UML-based and RDF/logic based tools will already allow the definition of objects (individuals) and assert their classifications. This ability must be used with caution by modelers; it is difficult to make any sense of what an object (individual) of type ValueStreamStage would represent. ValueStreamStage is an abstraction. On the other hand, an object (individual) of type Process would represent an execution of that process. Consequently, the extension of the BACM specification to introduce objects (individuals) is likely to specify constraints on the creation of such individuals, much like the effect of the “isAbstract” property of UML classes.

Business architecture practice defines when the semantic notions of generalization, composition and aggregation can be used. For example, neither Capabilities nor ValueStreamStages may be specialized. Capabilities and ValueStreamStages may be composed, but not aggregated. The BACM specification uses the special association names “generalizes”, “owns” and “aggregates” to designate syntactically when the use of these concepts is allowed. It will be challenging for UML-based tools to implement these restrictions as the UML metamodel does not restrict their use. For some tools, a rule checker process may be the best mechanism to help modelers avoid making these mistakes. This same problem exists for RDF/logic based tools as the underlying metamodel does not restrict where generalization, composition and aggregation may be used. For RDF-based models, a set of predefined SPARQL queries can detect any violations. Note that issue BACM-83 has been raised because a preliminary examination of the BACM metamodel has revealed inconsistent and incomplete use of these special associations, and it is possible that they will be deleted from the metamodel in favor of rules stipulating when such associations can be used in a business architecture model.