Int J Software Informatics, Volume 6, Issue 2 (2012), pp. 111–124
International Journal of Software and Informatics, ISSN 1673-7288

E-mail: ijsi@iscas.ac.cn
http://www.ijsi.org
Tel: +86-10-62661040

Towards an Interoperable Metamodel Suite: Size Assessment as One Input

Brian Henderson-Sellers[1], Muhammad Atif. Qureshi[1], and Cesar Gonzalez-Perez[2]

[1] (Faculty of Engineering and Information Technology, University of Technology, Sydney, Broadway, NSW, Australia)

[2] (Institute of Heritage Sciences (Incipit), Spanish National Research Council (CSIC), Santiago de Compostela, Spain)

Abstract In recent years, many metamodels have been introduced in the software engineering literature and standards. These metamodels vary in their focus across, for example, process, product, organizational and measurement aspects of software development and have typically been developed independently of each other with shared concepts being only accidental. There is thus an increasing concern in the standards communities that possible conflicts of structure and semantics between these various metamodels will hinder their widespread adoption. The complexity of these metamodels has also increased significantly and is another barrier in their appreciation. This complexity is compounded when more than one metamodel is used in the lifecycle of a software project. Therefore there is a need to have interoperable metamodels. As a first step towards engendering interoperability and/or possible mergers between metamodels, we examine the size and complexity of various metamodels. To do this, we have used the Rossi and Brinkkemper metrics-based approach to evaluate the size and complexity of several standard metamodels including UML 2.3, BPMN 2.0, ODM, SMM and OSM. The size and complexity of these metamodels is also compared with the previous version of UML, BPMN and Activity diagrams. The comparatively large sizes of BPMN 2.0 and UML 2.3 suggest that future integration with these metamodels might be more difficult than with the other metamodels under study (especially ODM, SSM and OSM).

Key words: BPMN; UML; ODM; SMM; OSM; metamodel; size; complexity; interoperability

Henderson-Sellers B, Qureshi MA, Gonzalez-Perez C. Towards an interoperable metamodel suite: Size assessment as one input. *Int J Software Informatics*, **Vol.6, No.2 (2012): 111–124. http://www.ijsi.org/1673-7288/6/i116.htm**

1 Introduction

Over the last few years, several metamodels have been introduced in the software engineering literature and through standards organizations. These metamodels vary in their focus across, for example, process, product, organizational and measurement aspects of software development. These metamodels have typically been developed

Corresponding author: Brian Henderson-Sellers, Email: brian@it.uts.edu.au
Received 2010-12-19; Revised 2011-05-24; Accepted 2011-09-26; Published online 2012-02-16.

independently of each other within the standards organizations with shared concepts being only accidental. There is thus an increasing concern in the standards communities that possible conflicts of structure and semantics between these various metamodels as well as their increasing size and complexity will hinder their widespread adoption. As a first step towards engendering interoperability and/or possible mergers between metamodels, we have undertaken a preliminary assessment of their size and complexity. To do this, we have used the Rossi and Brinkkemper[1] metrics-based approach to evaluate the size of several standard metamodels. This measurement of size and complexity will help us to understand the likely issues to be considered when merging metamodels or creating bridges between them to ensure interoperability.

Over the last 15 years, class-based metamodelling has been used as a basic underpinning representational approach. Following work on CDIF[2, 3], suggestions were made in 1994[4, 5] that this would be an appropriate means to facilitate the then-required merger of the large number of extant object-oriented modelling notations. In time this led, under the auspices of the Object Management Group (OMG), to the creation of a number of metamodels, not only for UML[6] but also SPEM (Software Process Engineering Metamodel)[7], SMM (Software Metrics Metamodel)[8], ODM (Ontology Definition Metamodel)[9], OSM (Organization Structure Metamodel)[10] and BPMN (Business Process Modelling Notation)[11].

Although these metamodels were endorsed by a single organization (the OMG), nevertheless, as noted above, they have essentially been developed in "silos" of independent concept sets. It has become increasingly clear that for a more widespread adoption of metamodels in the wider software engineering community, a greater degree of interoperability needs to be sought[12]. This need is further endorsed by the rise of industry interest in model driven software engineering (MDSE)[13]. A recent example is the change in OMG's (Object Management Group) OMA (Object Management Architecture) to MDA (Model Driven Architecture)[14] as well as the various conferences on the topic.

Our overall research topic focusses on how to ensure that these (and all future) software engineering metamodels are interoperable – first within a single standards organization (here the Object Management Group) and later across multiple such organizations. As a first step towards such a consistent metamodel suite, we need to assess what degree of commonality (syntactic and semantic) exists in the underlying conceptual models; whether a straightforward merger of conceptual elements in the metamodels is possible (and desirable) in order to expand and align the scopes of, say, a pair of metamodels; whether discrepancies and differences in conceptual models need a bridging structure, say as a mapping between sets; or whether the underlying conceptual models are so very different that any interoperation is virtually impossible, thus suggesting that a combined but newly created and more comprehensive metamodel be constructed across the two domains in question.

This paper is a first step towards the long-term goal of acquiring such a quantitative understanding of an interoperable metamodel suite. In order to evaluate the impact of size and complexity on the interoperability of any pair of metamodels, an, initial assessment of these factors is required for the selected metamodels. Different aspects of size can be considered for evaluation such as the number of classes, relationship and properties as well as the number of constraints. As an initial step, we have

used the metrics provided by Rossi and Brinkkemper[1], originally devised for evaluating and comparing the underpinning concepts of object-oriented methodologies. Although these are essentially size metrics, it can be argued that there is a reasonable correlation for most systems between size and complexity[15] such that an increase in the size of a metamodel is likely to increase the complexity and hence increase the cognitive load for the modeller, thus making it harder to learn and follow[16].

Although out of scope for the research reported in this present paper, it is important to note that although a metamodel depicts the abstract syntax of its specific domain of focus, it may also define some semantics – this is especially important when we consider interoperability. In other cases, the metamodel is purely abstract syntax and the semantics has to be supplied in some other way. In either case, in order to ensure an unambiguous semantics, various techniques have been investigated including ontologies e.g.[17] and logic-based languages such as VDM[18]. The combination of abstract syntax, semantics and possibly a concrete syntax (a.k.a. notation)[1)] is known as a (modelling) language since it defines the grammar and the rules of "sentence construction" as an analogy to natural languages. Whether the modelling languages built from the above metamodels can have general applicability or whether domain specific languages provide a more efficient and effective answer e.g. Refs. [19-21] is still open for debate. To collect data to answer such a research question leads to the recognition that formal measures of the metamodels are a prerequisite.

It should also be noted that these OMG metamodels are not static in nature. Over the years, new versions have been introduced that, typically, extend the scope of the focus domain. For instance, the scope of UML moved from object technology to include component technology and, later, to include execution semantics for code generation capabilities. This leads not only to an increase in size but also, typically, to an increased complexity of the metamodel involved. This complexity is compounded when more than one metamodel needs to be used during the life cycle of a software development project. It is not easy for different key players of software development, such as method engineers, methodologists and analysts, to either comprehend or control this compounded complexity. We therefore propose that there is a need to formulate a way in which these metamodels can be used in an interoperable fashion. This interoperability may indeed reduce their joint complexity, hence making them easy to understand and use. Whether interoperability should result in simply the building of conceptual mappings between pairs of metamodels or total integration (merger) will be the next step in our research project. As well as analyzing the latter, we also propose to include an investigation of the applicability of Situational Metamodel Engineering (SMME)[22] by which problem-specific metamodels can be "carved out" of a more comprehensive, merged metamodel.

As a first step towards this goal, in this paper we have used the Rossi and Brinkkemper approach[11] to evaluate the size and aspects of the complexity of the BPMN 2.0 metamodel, OSM[10], ODM[9], SMM[8], SPEM[7] and UML 2.3[23]. We have analyzed and compared their size and complexity with each other and also with the values published by other authors for BPMN (V1.0 and 1.2), UML 1.4.1 and UML Activity diagram. Although Version 1.4.1 of UML is not the latest version, it is the

[1)] The literature is undecided as to whether concrete syntax should or should not be included in the definition of a modelling language

version that is used in the cited papers and an almost identical version, but labelled UML 1.4.2 was adopted as the ISO International Standard 19501. We have observed that the complexity of the current BPMN and UML metamodels is higher than the complexity of other metamodels presented in this paper. Secondly, the complexity of these two metamodels has increased with the release of each new version. Thirdly, the complexity values of OSM and SMM are significantly lower – suggesting that the effort required to reconcile these metamodels at the conceptual level may be much lower for these last two OMG standards – a topic worthy of a separate research investigation.

The rest of this paper is structured as follows: the next section gives a brief overview of the metamodels selected for evaluation followed by a discussion in Section 3 of the metrics-based evaluation approach we have used. The results of our investigation into the relative complexities of the metamodel suite are presented in Section 4 followed by an identification of related work in this area (Section 5). We conclude in Section 6 including some recommended future research directions.

2 Metamodels

In this section, we give a brief overview of the selected metamodels. We have restricted our choice to metamodels endorsed by the Object Management Group because (1) they are fairly well known but also, and probably more importantly, (2) one might conjecture that models emanating from a single organization would have more inter-coherence than models from different organizations. We thus initially exclude other metamodels from organizations such as EIA e.g. Ref. [24], ISO e.g. Ref. [25] and WfMC e.g. Ref. [26]. Furthermore, although we are ultimately concerned with improving the quality and cohesion of modelling languages, here we exclude notational issues (a.k.a. concrete syntax) as might be characterized by, for example, the notations proposed for use for UML[27] and for ISO/IEC 24744[26] as well as semantic issues because we propose that the size of a metamodel can be calculated without its concrete syntax and semantics.

The models to be considered are those encapsulated in the Business Process Modelling Notation[11], the Software Metrics Metamodel[8], the Ontology Definition Metamodel[9], a recent draft of the Organization Structure Metamodel[10] and the current versions of SPEM[7] and UML[58].

2.1 BPMN

BPMN (Business Process Modelling Notation) is the result of standardization efforts started in 2001 by BPMI (Business Process Management Initiative) and was then transferred to the OMG in 2006. BPMN is designed to support modelling of end-to-end business processes. The latest specifications of BPMN are found in its version 2.0 beta[11]. There are three basic types of models within BPMN: Processes, Choreographies and Collaborations. Processes include both private and public processes. Private processes can be either executable or non executable. Process interactions are modelled by collaborations and/or Choreographies.

There are five basic categories of modelling elements used in any BPMN model: Flow Objects, Data, Connecting Objects, Swimlanes and Artifacts. Flow Objects are the main elements to define the behaviour of a Business Process. There are three types of flow objects: Events, Activities and Gateways. Data are represented as Data

Objects, Data Inputs, Data Outputs, Data Stores and Properties. Flow objects are connected to each other through Connecting Objects. There are four different types of connecting objects: Sequence Flow, Message Flow, Association and Data Association. Pools and Lanes are used to group modelling elements. Events and Gateways also have extended sets of elements.

2.2 SMM

SMM (Software Metrics Metamodel)[8] is a metamodel for representing measurement information related to software, its operation and its design. SMM contains meta-model classes and associations to model Measurements, Measures and Observations.

Measures are evaluation processes that assign comparable numeric or symbolic values to entities. The SMM associates a Measure to each measurement. Measurement results are values from ordered sets. Contextual information such as who, where and when is related by Observation.

SMM comprises 38 classes among which are 10 core classes including Measure, Measurement and Observation. Excluding the Timestamp class and the Date class, all core classes are extended from the SMM_Element class.

2.3 ODM

ODM (Ontology Definition Metamodel) facilitates the modelling of knowledge about real world concepts that are independent of an application[9]. Ontology as a discipline has its roots in Philosophy. The ODM includes four normative metamodels: RDF (Resource Description Framework), OWL (Web Ontology Language), CL (Common Logic) and TM (Topic Maps).

Three additional UML profiles are available for ODM for RDF, OWL and TM. These profiles enable the use of the UML notation for ontology modelling and the generation of corresponding ontology descriptions. In this paper we have used these four metamodels but excluded the UML Profiles.

2.4 OSM

OSM (Organization Structure Metamodel)[10] is an effort from the OMG to provide support for modelling of organizational structures in modern enterprises. Modern organizations typically have a large variety of job titles and relationships within that organization such as groups, teams, communities etc. and enterprise relationships with other organizations. These types of relationships do not fit easily into traditional hierarchical structures.

OSM has become a point of integration for many other metamodels like BMM (Business Motivation Model), SBVR (Semantics of Business Vocabulary and Rules) and BPMN etc. The current specifications for OSM are found in the 3rd initial submission in response to the RFP issued by the Object Management Group. This specification is not yet fully mature but it does define an initial metamodel for organizational structures worthy of discussion here. This metamodel contains 11 classes, among which Participant is the main generalized class with specialization classes of Person, OrgRole and OrgRelationship.

2.5 SPEM

SPEM (Software & Systems Process Engineering Metamodel)[2)][7] is a metamodel for representing elements of a process model. It focusses on conceptual descriptions of work that is undertaken, things that are produced or consumed during that work and the people and tools involved in their production. It also supports the sequencing of work.

SPEM packages the metalevel classes into either Method content or Managed content, the former being a way of defining "the core elements of every method such as Roles, Tasks, and Work Product Definitions" whilst the latter is said to define "the fundamental concepts for managing textual descriptions for process and method content elements".

2.6 UML

UML (Unified Modeling Language)[58] facilitates the visualization and documentation of models of software systems. It also aims to support models of business systems. UML include constructs to support both structural and behavioural aspects of systems. The current version of UML (2.3) includes 6 structural and 7 behavioral and interaction diagrams.

In this paper we have used the UML superstructure specifications given by OMG [58]. It is the superstructure that is the main constituent part of UML as far as the user is concerned – the infrastructure is focussed on tool developers and metamodellers whilst UML profiles provide support for domain-specific modelling and are therefore both out of scope for this size and complexity assessment.

3 Size and Complexity Metrics

To measure the size and complexity of the metamodels, we have used the metrics proposed by Rossi and Brinkkemper[1]. The rationale for choosing this approach is that it has already been used to evaluate process models[29] and BPMN[30].

Rossi and Brinkkemper[1] have presented a set of 17 metrics, based on the metamodel's vocabulary (i.e. concepts and properties). Specifically, these are the aggregate and average measurement of object types, their attributes/properties, their relationships with each other and roles. Formally, a model M of a technique T is defined as M = {O, P, R, X, r, p}, where:

• O is a finite set of object types. An object is defined as a "thing" that exists independently[1]. For example, class, association and classifier are different object types in the software and systems process engineering metamodel[7].

• P is a finite set of property types or attributes that are the characteristics associated with the object types.

• R is a finite set of relationship types. A relationship is an association between two or more object types.

• X is a finite set of role types, a role being the name of the connection between an object type and its association.

• The variables r and p represent mappings from role types to relationship types and objects types; and from non-property types to property types, respectively –

[2)] The acronym comes from the first version: Software Process Engineering Metamodel

although neither is used for the metrics calculations.

In this paper we apply these metrics to evaluate the size and complexity of the metamodels (mentioned in Section 2) in order to provide a direct comparison with each other and the values published using previous versions of BPMN[31] and with UML activity diagram[30]. We have used four metrics from Rossi and Brinkkemper[1], three of them to measure the size of different object types, property types and relationship types in these metamodels and the fourth to measure the overall complexity of the metamodel. These metrics are elaborated below.

Definition 1. $n(O_T)$: number of individual object types in a technique. In the case of BPMN 2.0, we find that, for example, $n(O_{BPMN2.0}) = 159$. BPMN 2.0 has more than 80 class diagrams showing these object types and their relationship with each other. We have excluded some of these diagrams (e.g. diagram interchange and execution semantics) in our evaluation because they were not present in other process modelling notations like the UML activity diagram.

Table 1 Size and complexity measures

Metamodel	Size measures			
	$n(O_T)$	$n(R_T)$	$n(P_T)$	Total Complexity
UML (V 1.4.1) Activity Diagram	8	5	6	11.18
UML Full (V1.4.1)	57	53	72	106.00
UML Superstructure (V 2.3)	288	23	154	327.40
BPMN 1.0	22	10	85	88.30
BPMN 1.2	90	6	143	169.07
BPMN 2.0	159	17	294	334.70
ODM	100	27	21	105.69
OSM	11	6	13	18.05
SMM	38	6	41	56.22
SPEM	101	16	56	116.59

Definition 2. $n(P_T)$: is the measure of the number of different property types. In BPMN 2.0, for example, we have calculated that $n(P_{BPMN2.0}) = 294$. These property types are reported in BPMN 2.0 specifications in more than 85 different tables. An important point to understand is that an attribute with the same name in two different objects is considered separately for every object. For example, "complexBehaviorDefinition" and "conditionalEvent-Definition" both have a "condition" that will be treated separately as "complexBehaviorDefinition condition" and "conditional-EventDefinition condition".

Definition 3. $n(R_T)$: is the measure of the different relationship types in a method. In BPMN 2.0 specifications, for example, we have identified $n(R_{BPMN2.0}) = 17$. The reason for this large number of relationship types is that each relationship type with different cardinalities is treated differently. For example "Composition 1 to 1" and "Composition 1 to *" are treated as two relationship types, as suggested by Rossi and Brinkkemper[1]. While this could be argued to overcount the number of relationship types, we retain it for consistency with earlier work (see also later discussion in Section 4.2).

Although offered as complexity metrics in Ref. [1], these three definitions are more realistically measures of size. However, some measure of complexity is given by:

Definition 4. $C(M_T)$:$\sqrt{n(O_T)^2 + n(R_T)^2 + n(P_T)^2}$. The total complexity of BPMN is calculated from this equation as $C(M_{BPMN2.0}) = 334.70$ and the total complexity of UML as 327.4. These values alone do not give the complete picture unless they are compared with the complexity values for other methods/techniques/metamodels. Indeed, it is even arguable if this is a true complexity metric. However, in the spirit of maintaining continuity and consistency with the Rossi and Brinkkemper metrics set, we will defer further technical evaluation of its validity to further work. In the following section, we provide an analysis and assessment of some of those other metamodels.

4 Metamodel Analysis

4.1 Results of metric counts

Table 1 gives an overview of the size and complexity values not only of BPMN2.0, UML 2.3, SMM, ODM, OSM and SPEM but also values from the literature for BPMN1.0 from Ref. [32], BPMN1.2 from Ref. [30], UML activity diagrams from Ref. [33] and the full UML from Ref. [29]. The two studies of UML both appear to be analyzing Version 1.4.1 of UML, although this is not stated explicitly in the papers. As a further update, we include, in this table, values from UML Version 1.1[27]. BPMN 2.0 is apparently the most complex of these metamodels although UML 2.3 is almost equivalent in complexity.

Table 1 show that the complexities of BPMN and UML have increased progressively for every new version of BPMN and UML. BPMN 1.0 is the least complex among them and then gradually BPMN 1.2 and BPMN 2.0 increase in complexity. All measures (object types, relationship types and property types) in BPMN have increased in almost the same proportion in its new versions as compared to the previous version. The only exception is the "number of relationship types" in BPMN 1.2 which has been reduced to 6 from its previous version's 10.

Another observation is that prior to UML 2.3, BPMN 2.0 was the most complex as compared to its own preceding versions and earlier versions of UML. As noted above, UML 2.3 has a complexity (327.4) close to BPMN 2.0 (334.7). Indeed, since the complexity values of UML 2.3 presented here do not include its profiles and infrastructure (as discussed above), we can conjecture that for those using the full UML, there will be additional complexity present. This suggests that overall UML 2.3 is the most complex of the metamodels studied here. Furthermore, the values in Table 1 confirm the increasing complexity with time/version number.

All versions of BPMN including version 1.0 are far more complex than UML Activity Diagrams, which have a similar scope to BPMN and are often used as an alternative way of modelling processes. The size and complexity of these metamodels create difficulties for normal users to utilize them without training, as was highlighted in the case of BPMN[34].

Expressiveness of a metamodel is another important consideration. Complex metamodels may be more difficult to learn but more effective to apply by experienced users[1], and should lead to smaller models[35]. Expressiveness of any metamodel does

not guarantee its wide application in the real world e.g. the theoretical size and complexity of BPMN differs considerably from its practical complexity. Very few constructs are used in practice in comparison with the much larger, full set of BPMN constructs[36].

The evolution of BPMN and UML has been our initial, prime focus since the scopes of the two metamodels clearly overlap; hence, integration is an obvious first target. Our second consideration is possible integration with one or other of the remaining metamodels (ODM, OSM, SMM, SPEM). Of these three metamodels, ODM and SPEM are of comparative complexity; whereas OSM and SMM have much less complexity. This tentatively suggests that, while integration of OSM and SMM may be easier, it may be more efficacious to first consider integration of ODM or SPEM.

4.2 Evaluating the validity of metrics suite

As explained in Section 3, we have used the metrics in Ref. [1] to measure the size and complexity of the various metamodels analyzed in this paper. This was done for compatibility purposes, since the works of Refs. [29, 30, 32, 33] have all used this approach in the past.

However, we have found two major issues while applying these metrics. The first relates to how relationships are counted; the second refers to the way in which size and complexity are related. This section explains these issues in detail.

According to OPRR[37], $n(R_T)$ refers to the number of relationship types. By looking at Fig. 1 and Fig. 2 of the said work, and especially at their table 1, it seems that the concept of what a relationship type is, as far as OPRR is concerned, has a much finer granularity than what UML or most practitioners would nowadays consider a relationship type to be. For example, 1-to-1 associations and 1-to-many associations are two different relationship types in OPRR, while UML and most current practitioners of conceptual modelling would probably agree that they are the same relationship type, having only a cardinality attribute that allows association instances (i.e. links in UML parlance) to exist in different amounts. Similarly, OPRR distinguishes between qualified and non-qualified, optional and non-optional, and generalization and specialization. Such a fine granularity of relationship types may be based on data modelling practices of the 1990s, such as Barker[38] and SSADM[39], which used "crow's foot" notation that visually distinguished between different cardinalities. Nowadays, and as exemplified by UML, the tendency is towards a simpler categorization scheme of relationship types. We believe that the fact that OPRR considers every combination of cardinality, optionality and qualification as a different type of relationship creates a bias that misrepresents the size (and complexity) of a metamodel, greatly overestimating the real number of relationship types that it comprises. For example, the list of 15 different relationship types that are represented in Fig. 1 of Ref. [1] can be easily reduced to 3 (association, aggregation and generalization) by using a "UML-friendly" approach. This suggests that the use of OPRR can easily result in an overestimation of the relationship type size of a metamodel by up to 500%.

Another area of concern is that, according to Ref. [1], complexity is calculated (their definition 12, p. 217) as the square root of the sum of the squares of the different counts computed on the method (this is captured in our Definition 4 (Section 3)).

In particular, these counts are the number of object types, relationship types and property types. This establishes a direct coupling between size and complexity, by which an increment in size always implies an increment in complexity, and vice versa. However, intuition tells us that complexity is not a mere function of the number of elements in a model, but also of how they are organized and how carefully they have been crafted. It is also easy to prove that models with very different numbers of object, relationship and property types can yield exactly the same complexity values using Rossi and Brinkkemper's metrics. For example, a model with 100 object types arranged in a linear fashion and connected by 99 associations, and with no properties, has a complexity value of 141. A model with 40 classes arranged in a 4 x 10 matrix where each class is connected by associations to all its neighbours contains approximately 135 associations; if this model has no properties, its complexity value is also 141. However, the first model is conceptually much easier to comprehend than the second, since it is comprised of a simple chain of classes, while the second model is a tightly coupled mesh of classes with interlinking connections. This means that the complexity metric of Rossi and Brinkkemper should be used with appropriate caution, and perhaps considered as being more representative of overall size rather than of actual cognitive complexity.

5 Related Work

Different approaches have been used to evaluate the complexity of software engineering metamodels and methodologies. One is based on data collected through different methods (e.g. surveys, laboratory experiments, case studies) and is generally known as empirical evaluation e.g. Ref. [40]. The second type of evaluation is based on a features comparison where typically a checklist of method features is used to evaluate a certain method e.g. Ref. [41]. Another approach is based on ontological analysis of a method[42] that uses mapping between models of the real world and models of the method to check the expressiveness of the modelling method. Ontological completeness and ontological clarity are two major measures in these approaches[43]. Using similar ideas relating to concept deficit and concept overload, Kargl et al.[44] introduce a metric for the explicitness of a metamodel, which evaluates the concepts in the metamodel (viewed as an abstract syntax) in respect of the concepts expressible in the concrete syntax (its associated notation). Lastly, metamodel-based metrics[21,45,46] are used to evaluate the complexity of a method based on the structural properties of that method. Ontology-based evaluation of a method examines how well a method represents the real world and determines how difficult the method is to use in the real world, while a metamodel-based evaluation of a method determines how difficult that method is to learn. The former is concerned with usage while the latter is concerned with learning[30].

For the various versions of UML, a suite of standard OO design metrics was used in Ref. [47] to depict changes from UML version 1.1 up to version 2.0. The increase in size (in terms of number of metalevel classes) was shown to be monotonic increasing over these versions – an observation confirmed in Table 1 here.

In the case of BPMN, a variety of these different approaches has been used to evaluate the complexity and quality of its models. Some of these studies are based on empirical data and some are based on metrics. In the proceeding paragraphs, we

have briefly discussed some of these studies as reported in the literature. Bodart et al.[40] presented a theory about the usage of optional attributes in conceptual models (e.g. ER Diagrams) and conducted some experiments to test their predictions. Their study does not include BPMN.

Wahl and Sindre[34] evaluated BPMN[31] using a semiotic quality framework[48] that is based on linguistic as well as semiotic concepts. They have discussed different aspects of quality for conceptual models and conceptual modelling languages and have evaluated BPMN against those aspects; however, their work omits an evaluation of BPMN at the metamodel level.

Aguilar et al.[49] have evaluated BPMN models based on a set of experiments using FMESP (Framework for the Modelling and Evaluation of Software Processes)[50]. Their evaluation is also limited to the models of BPMN and does not cover the metamodelling aspect. Muehlen and Recker[36] have assessed the practical complexity of BPMN using different mathematical and statistical techniques on 120 BPMN diagrams collected from different sources. BPMN constructs were divided into core and extended sets and the most frequent set of BPMN elements was identified. The complexity of BPMN models was measured by calculating the average number of semantically different constructs used in a model.

Chinosi and Trombetta[51] proposed BPeX (Business Process eXtensions) for validating BPMN diagrams through the analysis of weak points in the BPMN 1.1 metamodel. Their analysis is not quantitative, using terms such as "weak hierarchical structure" or "single relationship type, generalization" and does not suggest any metrics that might determine the degree of weakness of the hierarchical structure of the BPMN metamodel.

Recker et al.[52] have evaluated BPMN both theoretically and practically. They have used the Bunge ontology[53,54] to identify shortcomings of BPMN and then conducted a series of interviews with BPMN users to validate their propositions. Two important findings of their study were "deficit constructs" and "excess constructs" in BPMN. Their study also lacks any discussion about the metamodel of BPMN or its evaluation.

Dijkman et al.[55] have proposed some semantics for mapping BPMN models to Petri nets to statistically check the semantic correctness of models. Their mapping covers only "flow elements" of BPMN models and does not provide any mechanism to map other constructs of BPMN to Petri nets.

Recker et al.[56] and Indulska et al.[30] evaluated the complexity of the BPMN metamodel using metrics proposed by Rossi and Brinkkemper[1] and compared the complexity of metamodels of BPMN and UML. They pointed out that BPMN has a high level of complexity as compared to UML. As noted earlier, our extension to version 2.0 of BPMN is largely based on their study.

6 Conclusions and Future Work

Previous studies of the complexity of BPMN and UML have shown values increasing with time as newer versions of each metamodel standard have been published (see also Ref. [46]). With metamodel integration in mind we have re-assessed the most recent versions of BPMN and UML in particular, finding them both to have increased in complexity over time. We have then extended the analysis by using the Rossi and

Brinkkemper[1] approach to include metrics counts of four other OMG metamodel standards (ODM, OSM, SMM and SPEM). Thus, although conceptually one might anticipate a ready integration of UML and BPMN (because of the overlap in domain of discourse), the numbers in Table I suggest that, instead, integration of OSM and SMM or between one of these (OSM, SMM) and BPMN might be an easier target[58]. This will be an additional element of our future research plan for creating a suite of integrated metamodels. Finally, we undertook an analysis of the research methodology itself viz. the Rossi and Brinkkemper metrics suite. We have identified two threats to validity: (1) the likely inflation of the count of relationship types and (2) concern that the equation for total complexity is not truly representative of complexity since a single "complexity value" can be achieved with entity configurations that ostensibly have very different true complexity.

These topics will be the foci of future papers resulting from this research project. Future work already planned includes the evaluation of the semantics of each metamodel and the conceptual mappings across metamodels; approaches to merging metamodels using experiences gained in merging ontologies e.g. Ref. [57]; and the application of SMME[22] to take the merged metamodel and "carve" out situationally specific metamodels on a case-by-case basis. It should also be worthwhile to evaluate the potential of better complexity metrics (e.g. metrics discussed in Refs. [46, 59, 60]) that may help us to increase the value of this study beyond the use of a 'standard' metrics set such as those of Rossi and Brinkkemper, potentially replacing their complexity metric (Definition 4 above) with one or more true complexity metrics. Finally, a deeper understanding of the roles of size and complexity in the specific context of metamodel integration might allow one to highlight optimal areas for metamodel integration as well as identifying likely semantic incompatibilities.

Acknowledgement

The first author wishes to acknowledge support from the Australian Research Council. We also thank the anonymous reviewers for their insightful comments, which helped us to improve the manuscript. This is contribution number 10/09 of the Centre for Object Technology Applications and Research within the Centre for Human-Centred Technology Design at the University of Technology, Sydney.

References

[1] Rossi M, Brinkkemper S. Complexity metrics for systems development methods and techniques. Information Systems, 1996, 21(2): 209–227.
[2] CDIF—CASE Data Interchange Format—Overview. Interim Standard, EIA/IS-106, 1994.
[3] ISO/IEC 15474. CDIF Framework. 1998.
[4] Henderson-Sellers B. COMMA: an architecture for method interoperability. Object Analysis and Design, 1994, (1): 25–28.
[5] Monarchi D, Booch G, Henderson-Sellers B, Jacobson I, Mellor S, Rumbaugh J, Wirfs-Brock R. Methodology standards: help or hindrance? Ninth Annual OOPSLA Conference. ACM SIGPLAN, 1994, 29(10): 223–228.
[6] UML Summary (2 of 10), v1.1, Document number ad/97-08-03. 1997.
[7] Software & Systems Process Engineering Meta-Model Specification, OMG document no formal/ 2008-04-01. 2008.
[8] Architecture-Driven Modernization (ADM): Software Metrics Meta-Model (SMM) FTF – Beta 1. OMG document no ptc/2009-03-03. 2009.

[9] Ontology Definition Metamodel Version 1.0, OMG document no formal/2009-05-01. 2009.
[10] Organization Structure Metamodel (OSM) 3rd initial submission. OMG document no bmi/09-08-02. 2009.
[11] Business Process Model and Notation (BPMN) FTF Beta 1 for Version 2.0, OMG document no dtc/2009-08-14. 2009.
[12] OMG, large volume of emails in online discussion group during 2009.
[13] Bezivin J, Gerbe O. Towards a precise definition of the OMG/MDA framework. The 16th IEEE International Conference on Automated Software Engineering. Los Alamitos: IEEE Computer Society, 2001. 273–280.
[14] Model Driven Architecture (MDA). OMG document no ormsc/2001-07-01. 2001.
[15] Kan SH. Metrics and Models in Software Quality Engineering, Addison-Wesley, 2003. 528.
[16] Cant S, Jeffery DR, Henderson-Sellers B. A conceptual model of cognitive complexity of elements of the programming process. Inf. Software Technol., 1995, 37(7): 351–362.
[17] Gonzalez-Perez C, Henderson-Sellers B. An ontology for software development methodologies and endeavours. In: Calero C, Ruiz F, Piattini M, eds. Chapter 4 in: Ontologies in Software Engineering and Software Technology. Berlin: Springer-Verlag, 2006. 123–152.
[18] Bjørner D, Jones CB. The Vienna Development Method: The Meta-Language. LNCS 61. 1978.
[19] Greenfield J, Short K. Software Factories: Assembling Applications using Patterns, Model, Frameworks and Tools. Chichester: J. Wiley and Sons. 2004.
[20] Kelly S, Tolvanen JP. Domain-Specific Modeling: Enabling Full Code Generation. Chichester: John Wiley &Sons, 2008.
[21] Kelly S, Pohjonen R. Worst practices for domain-specific modeling. IEEE Software, 2009, 26(4): 22–29.
[22] Hug C, Front A, Rieu D, Henderson-Sellers B. A method to build information systems engineering process metamodels. Systems and Software, 2009, 82(10): 1730–1742.
[23] Unified Modeling Language: Superstructure, Version 2.3. OMG document no formal/2010-05-05. 2010.
[24] Flatscher RG. Metamodeling in EIA/CDIF – meta-metamodel and metamodels. ACM Trans. Modeling and Computer Simulation, 2002, 12(4): 322–342.
[25] ISO/IEC 24744. Software Engineering – Metamodel for Development Methodologies, Geneva: ISO. 2007.
[26] Xiao Z, Chang H, Wen S, Yi Y, Inoue A. An extended meta-model for workflow resource model. In: Lang J, Lin F, Wang J, eds. KSEM 2006. LNCS 4092, Berlin: Springer-Verlag. 2006. 525–534.
[27] Unified Modeling Language: Superstructure, Version 2.1.1. OMG document no formal/2007-02-05. 2007.
[28] ISO/IEC 24744. Software Engineering – Metamodel for Development Methodologies Addendum: Notation, Geneva: ISO. 2010.
[29] Siau K, Cao Q. Unified modelling language: a complexity analysis. Journal of Database Management, 2001, 12(1): 26–34.
[30] Indulska M, Muehlen M, Recker J. Measuring method complexity: the case of the Business Process Modelling Notation. BPMcentre.org. 2009.
[31] Business Process Modelling Notation Specifications, v1.0. http://www.omg.org/spec/BPMN/1.1/. 2001.
[32] Kretschmer P. JWT metamodel compared to BPMN metamodel. Distributed Systems Lab, University of Augsburg, 2007.
[33] Siau K, Erickson J, Lee L. Theoretical vs. practical complexity: the case of UML. J. Database Management, 2005, 16(3): 40–57.
[34] Wahl T, Sindre G. An analytical evaluation of BPMN using a semiotic quality framework. Proc. CAiSE'05 Workshop, 2005. 533–544.
[35] Mohagheghi P, Aagedal J. Evaluating quality in model-driven engineering. International Workshop on Modeling in Software Engineering (MISE'2007). Los Alamitos: IEEE Computer Society Press, 2007. 6.
[36] Muehlen MZ, Recker J. How much language is enough? Theoretical and practical use of the

Business Process Modeling Notation. Proc. of 20th International Conference on Advanced Information Systems Engineering. LNCS 5074, Berlin: Springer-Verlag, 2008. 465–479.
[37] Smolander K. OPRR: a model for modelling systems development methods in next generation CASE tools. Amsterdam: IOS Press, 1991. 224–239.
[38] Barker R. CASE Method: Entity Relationship Modelling. Addison-Wesley Professional, 1990.
[39] Goodland M, Slater C. SSADM Version 4. McGraw-Hill, 1995.
[40] Bodart F, Patel A, Sim M, Weber R. Should optional properties be used in conceptual modelling? A theory and three empirical tests. Information Systems Research, 2001, 12(4): 384–405.
[41] Yadav SB, Bravoco RR, Chatfield AT, Rajkumar TM. Comparison of analysis techniques for information requirement determination. Comm. ACM, 1988, 31(9): 1090–1097.
[42] Wand Y, Weber R. On the ontological expressiveness of information systems analysis and design grammars. Journal of Information Systems, 1993, 3(4): 217–237.
[43] Weber R. Ontological Foundations of Information Systems. Melbourne: Coopers & Lybrand and the Accounting Association of Australia and New Zealand, 1997.
[44] Kargl H,Strommer M, Wimmer M. Measuring the explicitness of modeling concepts in metamodels. Proc. ACMS/IEEE 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS/UML 2006), Workshop on model size metrics. Genova, Italy. 2006. (http://www.modelcvx.org/publications/workshop.html).
[45] Vanderfeesten I, Reijers H, Mendling J, van der Aalst W, Cardoso J. On a quest for good process models: the cross-connectivity metric. Advanced Information Systems Engineering, 2008. 480–494.
[46] Ma H, Shao W, Zhang L, Ma Z, Jiang Y. Applying OO metrics to assess UML meta-models. UML2004. LNCS 3273, 2004, In: Baar T, et al., eds. Berlin: Springer-Verlag, 2004. 12–26.
[47] Gemino A, Wand Y. Evaluating modeling techniques based on models of learning. Comm. ACM, 2003, 46(10): 79–84.
[48] Lindland OI, Sindre G, Sølvberg A. Understanding quality in conceptual modeling. IEEE Software, 1994, 11(2): 42–49.
[49] Aguilar ER, Ruiz F, Garcia F, Piattini M. Evaluation measures for business process models. In: Procs. ACM Symposium on Applied Computing, New York: ACM, 2006. 1567–1568.
[50] Garcia F, Piattini M, Ruiz F, Canfora G, Visaggio CA. FMESP: framework for the modeling and evaluation of software processes. J. Syst. Archit, 2006, 52(11): 627–639.
[51] Chinosi M, Trombetta A. Modeling and validating BPMN diagrams. Procs. IEEE Conference on Commerce and Enterprise Computing. Los Alamitos: IEEE Computer Society Press, 2009. 353–360.
[52] Recker JC, Indulska M, Rosemann M, Green P. How good is BPMN really? Insights from theory and practice. In: Ljungberg J, Anderson M, eds. Proc. 14th European Conference on Information Systems, Gotteborg: IT University of Gotteborg, 2006. 1–12.
[53] Bunge M. Treatise on Basic Philosophy: Vol. 3: Ontology I: The Furniture of the World, Boston: Reidel, 1977.
[54] Bunge M. Treatise on Basic Philosophy: Vol. 4: Ontology II: A World of Systems, Boston: Reidel, 1979.
[55] Dijkman RM, Dumas M, Ouyang C. Semantics and analysis of business process models in BPMN. Information and Software Technology, 2008, 50(12): 1281–1294.
[56] Recker JC, zur Muehlen M, Siau K, Erickson J, Indulska M. Measuring method complexity: UML versus BPMN. Proc. of 15th Americas Conference on Information Systems, San Francisco, Atlanta: Association for Information Systems, 2009.
[57] Sowa J. Building, sharing and merging ontologies. available from http://www.jfsowa.com/ontology/ontoshar.htm, 2001.
[58] Qureshi MA, Henderson-Sellers B. Merging software engineering metamodels: using an ontological merging technique. (paper in preparation)
[59] Henderson-Sellers B. Object-Oriented Metrics. Measures of Complexity. 1996, (NJ): Prentice Hall. 234.
[60] Mohagheghi P, Dehlen V, Neple T. Definitions and approaches to model quality in model-based software development-a review of literature. Inf. Software Technol., 2009, 51(12): 1649–1669.