Structure Metrics for XML Schema

Joost Visser*

Departamento de Informática
Universidade do Minho
Braga, Portugal
Joost.Visser@di.uminho.pt
http://www.di.uminho.pt/~joost.visser

Abstract. XML schemas are software artifacts claiming an increasingly central role in software construction projects. Schemas are used as interface definitions, data models, protocol specifications, and more. Standards bodies are employing schemas for standards definition and dissemination. Using code generators that accept schemas as input, software components are generated for data interchange and persistence.
With increased reliance on schemas comes the necessity of properly embedding these artifacts in the software engineering process. In particular, schema metrics must be developed to enable quantification of schema size, complexity, quality, and other properties, instrumental to retaining control over the software processes in which they are involved.
In this paper, we propose a suite of metrics for the XML Schema language that measure structural properties. The metrics are mostly adaptations of existing metrics for other software artifacts, such as programs and grammars. Apart from definitions of the metrics, we report on application of these metrics to a series of open source schemas, using our XsdMetz tool. We suggest how the measurement results may be used to assess potential risks in schemas.
Keywords: XML, XSD, Schemas, Software metrics, Document processing, Software quality, Software risk assessment, Dependency graph, Tree impurity, Coupling, Coherence, Component analysis.

1 Introduction

XML Schema was the first separate schema language for XML to achieve Recommendation status by the World Wide Web Consortium (W3C) [8]. It is one of several schema languages proposed to supersede Document Type Definitions (DTDs) for the specification of structure, content, and semantics of XML documents. The main components defined/declared in an XML Schema are *elements*, *attributes*, simple and complex *types*.

XML Schemas are software artifacts that are claiming an increasingly central role in software construction projects. Schemas have come into use as interface

* Supported by the Fundação para a Ciência e a Tecnologia, Portugal, under grant number SFRH/BPD/11609/2002.

definitions, as data models, as protocol specifications, and more. Standards bodies are employing schemas for standards definition and dissemination. Using code generators that accept schemas as input, software components are generated for data interchange and persistence.

With increased reliance on schemas comes the necessity of properly embedding them in the software engineering process. This involves both tool support (e.g. for schema editing, conversion, visualization) and methodology (e.g. schema design patterns and schema style guides), areas in which progress is currently being made at high speed.

An area in which investigation has started only recently is schema *metrics*. In software engineering in general, metrics play a role in monitoring and controlling the software process, or in specifying and improving software quality aspects such as performance or reliability. To wit, the 200-page *Guide to the Software Engineering Body Of Knowledge - SWEBOK* [1] contains about 500 references to the topic of metrics[1]. Over time, an extensive array of software engineering metrics have been defined and applied [4, 3]. Metrics for XML schemas are needed for quantification of schema size, complexity, quality, and other properties, instrumental to control the processes in which they are involved.

The first definition of a suite of metrics for the XML Schema language has been provided by Lämmel *et al.* [6]. Their metrics range from simple counters of various types of schema nodes to more involved metrics such as McCabe, depth, and breadth. Table 1 shows an overview. All these metrics can be seen

Table 1. Overview of XML Schema metrics defined by Lämmel *et al.* [6].

XML-agnostic schema size	
File size	KB or LOC
XSD-agnostic schema size	
Number of all XML nodes	#NODE
Number of all XML nodes for annotation	#ANN
XSD-aware counts	
Number of global, local, or all element declarations	$\#EL_g$,$\#EL_l$,#EL
Number of global, local, or all complex-type definitions	$\#CT_g$,$\#CT_l$,#CT
Number of global, local, or all simple-type definitions	$\#ST_g$,$\#ST_l$,#ST
Number of global, local, or all model-group definitions	$\#MG_g$,$\#MG_l$,#MG
Number of global, local, or all attribute-group definitions	$\#AG_g$,$\#AG_l$,#AG
Number of global, local, or all attribute declarations	$\#AT_g$,$\#AT_l$,#AT
$\#EL_g + \ldots + \#AT_g$	#GLOBAL
McCabe complexity for XSD	
McCabe cyclometric complexity	MCC
Depth and breadth of content models	
Code-oriented and instance-oriented breadth	
Code-oriented and instance-oriented depth	

[1] With search keys like *quantification*, *metric*, *measure*, *statistic*, and their derivations.

as *size* metrics; the property measured by each of them is schema size, albeit measured in many different ways. Even the McCabe metric, originally developed with the intention to measure (program) complexity, is well-known to be strongly and significantly correlated with size, and its use as complexity metric has been criticised [4].

In this paper we propose a number of more advanced schema metrics that may be used to measure other properties than size. The proposed metrics are adaptations of existing metrics for other software artifacts, such as programs and grammars. All metrics are defined over graph representations of schema structure, and can hence be categorized as structure metrics. Apart from definitions of the metrics, we report on application of these metrics to a series of open source schemas, measured using our XsdMetz tool. Also, we discuss how the measurement results may be used to assess potential risks in schemas. Rigorous empirical validation of the metrics as measures for external schema properties falls outside of the scope of this paper.

In Section 2 we define graph representations of schema structure. These graph representations are the basis of the structural metrics which we define in Section 3. In Section 4 we comment on the implementation of the metrics suite in our XsdMetz tool, and we discuss its application to a series of important XML schemas. Section 5 discusses related work, while Section 6 provides concluding remarks and a reflection on future work.

2 Graph representations of schema structure

The various components of an XML schema (such as elements, types, groups, and attributes) can be dependent on each other in the sense that the definition or declaration of one component may mention other components. Whenever such a dependency exists, we say that the mentioned component is an *immediate successor* of the defined/declared component. Based on this immediate successor relation, a graph representation of the schema structure is obtained, where nodes are components, and edges are successor relations. Figure 1 shows a small schema and its associated *successor graph* (SG).

When two nodes in a directed graph can be reached one from the other and *vice versa*, these nodes are called *strongly connected*. In the successor graph of Figure 1, this is the case for the nodes representing the global complex type `EmployeeType` and its local element `Emp`. A set of strongly connected nodes is termed a *strongly connected component*. From any directed graph that potentially contains such cycles, we may derive a directed acyclic graph (DAG) that contains the strongly connected components as nodes, and that contains edges between two components whenever at least one edge exists between a node in one component and a node in the other. For our example, the derived *strongly connected components graph* (CG) is shown in Figure 2. We can regard strongly connected components as a specific notion of *module* for dependency graphs in general, and by extension, for XML schemas in particular.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Emp" type="EmployeeType" />
  <xs:complexType name="EmployeeType">
    <xs:sequence>
      <xs:element name="Emp" type="EmployeeType" />
    </xs:sequence>
    <xs:attribute name="EmployeeID" type="xs:ID" />
    <xs:attribute name="FirstName" type="xs:string"/>
    <xs:attribute name="LastName" type="xs:string"/>
  </xs:complexType>
</xs:schema>
```

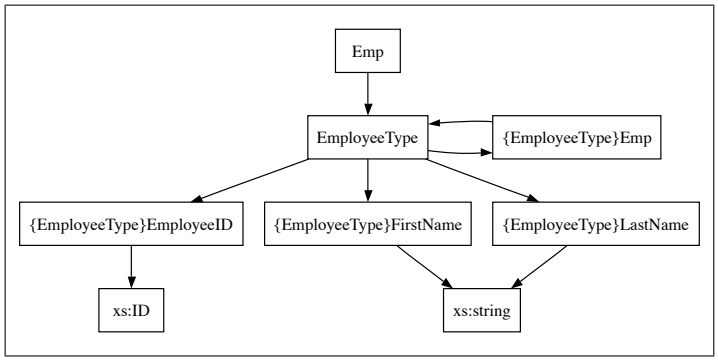

Fig. 1. A simple example schema, adapted from the online *.NET Framework Developer's Guide* (http://msdn.microsoft.com/library/), for representing employee hierarchies. The graph depicts the corresponding successor relation. Local names such as FirstName are qualified with their surrounding global, in this case {EmployeeType}.

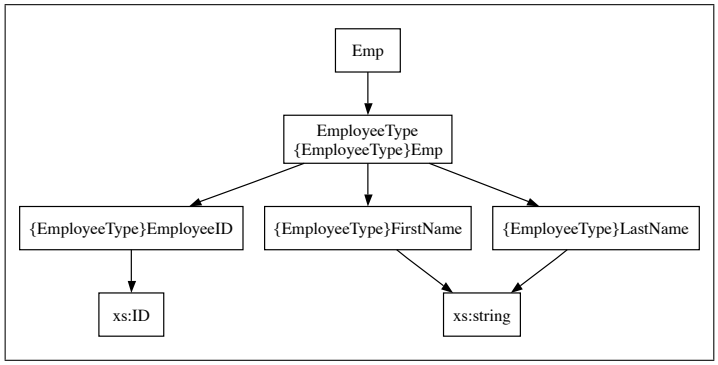

Fig. 2. Graph of strongly connected components derived from the successor graph of Figure 1. The global complex type `EmployeeType` and its local element `Emp`, which are mutually dependent, are wrapped into a single component (module).

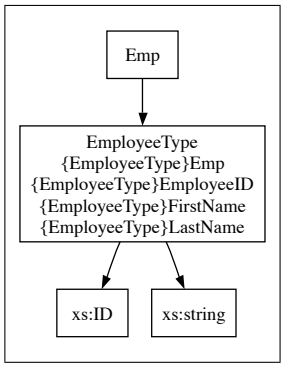

Fig. 3. Graph of global declarations/definitions, derived from the successor graph of Figure 1. The various local components of `EmployeeType` are wrapped into a single component together with that global type itself.

Other, perhaps more intuitive notions of *module* can be employed for XML schemas. For example, namespaces, schema documents (i.e. `xsd` files), or global declarations/definitions can each be employed to group nodes of a successor graph into modules. Similar as in the case of modules as strongly-connected components, each of theses module notions can be used to obtain a derived dependency graph. In this paper, we will take into consideration the dependency graph derived using the notion of modules as global declarations/definitions. An example of such a derived graph of globals is shown in Figure 3.

These three types of dependency graphs are the basis of the structure metrics that we define in the upcoming section.

3 Definition of metrics

We will provide definitions for a range of structure metrics. For each metric, we will show computed values for the example schema of Figure 1.

3.1 Tree impurity

The tree impurity metric indicates to what extent a given graph deviates from a tree structure with the same number of nodes. Fenton *et al.* [4] define tree impurity for connected undirected graphs without self-edges as $\frac{2(e-n+1)}{(n-1)(n-2)} \cdot 100\%$, where n is the number of nodes, and e is the number of edges. A tree impurity of 0% means that the graph is a tree and a tree impurity of 100% means that it is a fully connected graph.

Tree impurity can also be applied to directed graphs that may contain self-edges, simply by disregarding the direction and the self-edges. In this way, tree

impurity is applicable both to successor graphs and to strongly connected components graphs. In particular, for the graphs of Figure 1, 2, and 3, we obtain the values of $TI_{SG} = 4.76\%$, $TI_{CG} = 6.67\%$, and $TI_{GG} = 0\%$, respectively.

3.2 Fan-in, fan-out, and instability

A pair of classic metrics are fan-in and fan-out. The fan-in of a node in a directed graph is the number of its incoming edges. Conversely, the fan-out is the number of outgoing edges of the node. Both metrics are directly applicable to the nodes of each type of dependency graph. For the graphs as a whole, the averages and maximums of these metrics can be relevant. For the graphs of Figure 1 and 2, we have the following values:

SG	avg	max
fan-in	1.125	2
fan-out	1.125	4

CG	avg	max
fan-in	1.0	2
fan-out	1.0	3

GG	avg	max
fan-in	0.75	1
fan-out	0.75	2

The maximum fan-in and fan-out, in particular, can be useful to spot unusual nodes, which subsequently may be inspected to identify the cause of the abnormality, and a possible cause of action to correct the situation.

Based on fan-in and fan-out, a measure called *instability* can be defined as the fan-out fraction of total fan, i.e. as: $\frac{fan\text{-}out}{fan\text{-}in+fan\text{-}out} \cdot 100\%$. For our example graphs, we have the following basic statistics regarding node instability:

SG	avg
instability	45.8

CG	avg
instability	46.4

GG	avg
instability	41.7

The instability metric ranges between 0% (no outgoing edges) and 100% (only outgoing edges). Low instability of a node indicates that it is dependent on few other nodes, while many nodes are dependent on it. Thus, low instability corresponds to a situation where changes to the node will affect relatively many other nodes, and would hence be costly or difficult. In other words, instability may be interpreted as resistance to change.

3.3 Efferent and afferent coupling, and again instability

Coupling is a notion similar to fan, but taking modules into account, where any group of nodes may be viewed as a module. The number of edges from nodes outside the module to nodes inside the module is called *afferent coupling* (Ca). Conversely, the number of edges from nodes inside the module to nodes outside is called *efferent coupling* (Ce). Their sum is simply *coupling*. As in the case of fan-in and fan-out, an instability metric can be defined based on afferent and efferent coupling, as: $\frac{Ce}{Ce+Ca} \cdot 100\%$.

For our example schema, the coupling metrics corresponding to strongly connected components and to global declarations/definitions present the following basic statistics:

CG	avg	max
afferent coupling	1.0	2
efferent coupling	1.0	3
internal edges	0.29	2
instability	46.43	

GG	avg	max
afferent coupling	1.0	2
efferent coupling	1.0	3
internal edges	1.25	5
instability	43.75	

Coupling is often mentioned in one breath with *coherence*, which we discuss next.

3.4 Coherence

Whereas coupling assesses the connections of a module with nodes external to it, the *coherence* of a module concerns the degree to which its internal nodes are connected with each other. As a general coherence metric we propose the ratio of internal edges of a module versus all edges that start and/or end in a node inside the module. Thus, $Ch = \frac{Ci}{Ci+Ca+Ce} \cdot 100\%$, where Ci is the number of edges between nodes inside the module, i.e. the internal edge count. In the limit case of singleton modules, we set $Ch = 100\%$, rather than 0%, expressing that singletons are fully coherent.

Note that this measure of coherence takes external edges into account, not only internal edges. Assuming a stable node count for a module, its coherence increases both with the addition of internal edges and with the removal of external edges. In other words, coherence is compromised both by lack of connections between internal nodes, and by too many connections to the outside, i.e. by breaches of encapsulation.

As an alternative measure of coherence, one may use tree impurity, as defined before, on the level of modules. For our example graphs, we can derive the following basic statistics:

CG	min	avg	max
internal edges	0	0.29	2
coherence	33.3	90.5	100
tree impurity	0	85.7	100

GG	min	avg	max
internal edges	0	1.25	5
coherence	55.6	88.9	100
tree impurity	0	75.0	100

Both Ch and TI range between 0% and 100%, and both increase when internal edges are added. But, whereas Ch also depends on the number of connections to external nodes, TI of a module is independent from the external edge count.

3.5 Normalized count of modules

For each notion of module, we can define a normalized count of modules by expressing the module count as a ratio of *potential* module count, which is the number of nodes in the underlying dependency graph. Thus, $NCM = \frac{\#M}{\#N} \cdot 100\%$. Thus, the more nodes get grouped into modules, the lower the normalized count of modules. A value of 100% indicates that no grouping has occurred, i.e. each node sits in a separate module (full fragmentation). A value approaching 0% indicates that all nodes are grouped together in a very small number of modules (monoliths).

Table 2. Quick reference of structure metrics.

Metric			Measured per ...
TI	tree impurity	%	... graph (SG, CG, or GG), or module (subgraph) of SG.
Fi	fan-in	$\mathbb{N}$	... node (SG), or module (node in CG or GG).
Fo	fan-out	$\mathbb{N}$	... *idem.*
If	instability (based on fan)	%	... *idem.*
Ca	afferent coupling	$\mathbb{N}$	... module (node in CG or GG), by counting edges in SG.
Ce	efferent coupling	$\mathbb{N}$	... *idem.*
Ic	instability (based on coupling)	%	... *idem.*
Ci	internal edges	$\mathbb{N}$	... *idem.*
Ch	coherence	%	... *idem.*
NCM	normalized count of modules	%	... derived graph (CG or GG), in relation to SG.
NM	count of nodes per module	$\mathbb{N}$	... derived graph (CG or GG).

The interpretation of the normalized count of modules depends on the underlying notion of module. In case of the notion of modules as strongly connected components, (mutual) recursion gives rise to non-singleton modules. Correspondingly, *NCM* is a measure of *recursiveness*, where low *NCM* indicates a high degree of mutual recursiveness. In the case of the notion of modules as global definitions/declarations, local elements give rise to non-singleton modules. Correspondingly, *NCM* is a measure of *encapsulation*, where low *NCM* indicates a high degree of encapsulation.

For the running example, we have the following basic statistics: $NCM_{CG} = 87.5\%$ and $NCM_{GG} = 50.0\%$. Note that in the case of XML Schema, mutual recursiveness must always involve global components. Thus, increased mutual dependencies reduce the opportunity for encapsulation, and *vice versa*.

Apart from comparing the total number of modules with the total number of nodes, it may be interesting to count nodes per module (*NM*). For our running example, we have the following extremes: $NM^{max}_{CG} = 2$ and $NM^{max}_{GG} = 5$.

For quick reference, Table 2 lists all metrics defined in this section together with an indication of their scale and of the entities on which they are measured.

4 Data collection

We have implemented support for the XML Schema metrics defined in the preceding section in a prototype tool, called XsdMetz. The tool was implemented in the functional programming language Haskell, using purely functional graph representations and algorithms. The tool shares most of its code with other metrics extraction tools, in particular with the SdfMetz tool which calculates metrics

Table 3. Schemas that were sampled with the XsdMetz tool.

File name	File size	Provider	Purpose
ABCD_2.06	141K	TDWG, CODATA	Access to Biological Collection Data
AWSECommerceService	91K	Amazon	Web service
MARC	6K	Library of Congress	Bibliographic data
PressureVessel	151K	Codeware Inc.	Data on pressure vessels and heat exchangers
WS-ServiceGroup	4K	IBM and others	Web services
XMLSchema	84K	W3C	Schema for XML Schema
diva	86K	Uppsala University	Digital Scientific Archive
ebaySvc	1.9M	eBay	Web services
uddi_v3	39K	OASIS Consortium	Universal Description, Discovery and Integration

from SDF grammar representations [2]. Apart from generating metric reports in several format, the XsdMetz tool exports successor graphs, strongly connected component graphs, and global definition/declaration graphs (see Section 2) in the *dot* format of GraphViz [5], which can be used to render the graphs.

We have applied XsdMetz to a series of freely available XML Schema specifications, listed in Table 3. The scope and character of this paper do not allow the measurement data to be presented here in full. Also, these measurements constitute merely a first step towards empirical validation of the metrics. Still, we will briefly discuss some observations to illustrate potential use of the metrics.

Table 4 shows the values we measured for the normalized count of modules. For almost all schemas, the normalized count of modules for the strong components graph is close to 100%, indicating very low degrees of *recursiveness*. Only the schema for XML Schema itself appears to contain more recursion, with a significantly lower value of 82.2%. This conclusion about recursiveness is cor-

Table 4. Measurement values for normalized counts of modules and number of non-singletong modules for entire graphs, and maximum number of nodes per module.

	NCM_{CG}	NCM_{GG}	NM_{CG}^{max}	NM_{GG}^{max}
ABCD	100	14.6	1	111
AWSE	99.6	14.0	3	236
MARC	100	55.6	1	6
Pres	99.7	9.67	5	132
WSSG	100	62.5	1	4
XMLS	82.2	52.4	38	12
diva	95.0	12.0	8	118
eBay	99.8	31.5	6	89
uddi	100	77.6	1	5

	KB	TI-SG	TI-CG	TI-GG
ABCD	141	0.19	0.19	3.9
AWSE	91	0.18	0.18	2.5
MARC	6	2.0	2.0	6.4
Pres	151	0.13	0.13	4.3
WSSG	4	1.2	1.2	1.1
XMLS	84	0.56	0.58	2.0
diva	86	0.41	0.45	14
eBay	1870	4.3e-2	4.3e-2	0.28
uddi	39	0.38	0.38	0.60

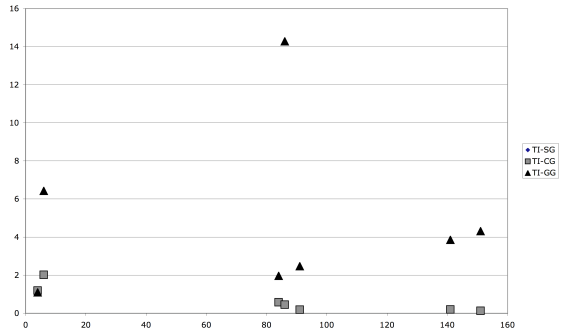

Fig. 4. Measurement values for tree impurity of entire graphs. The chart omits the values for the eBay schema, because its relatively large file size would dwarf the others. The TI-SG series is actually hidden by the TI-CG series, which presents extremely similar values.

roborated by the maximum numbers of nodes per module for strong component graphs (NM_{CG}^{max}), also shown in Table 4. None of the schemas have more than 3 groups of mutually dependent nodes, and the schema for XML Schema has an exceptionally large strong component, containing 38 nodes. The normalized count of modules for the global declarations/definitions graph presents much lower percentages, indicating a fair degree of *encapsulation*. The maximum number of nodes in such declarations/definitions (NM_{GG}^{max}) indicates that the largest module can be found in the Amazon schema, which encapsulates 236 nodes in a single module.

Figure 4 shows the measurement values for tree impurity for each of the dependency graphs. For the successor graph and the strongly-connected components graph, the numbers are very similar, and very low for all sampled schemas. The strong similarity between the tree impurity of SG and CG suggests that the shapes of these two graphs are almost identical, which implies that recursion is uncommon. This corroborates our observations regarding the NCM_{CG} metric. For all schemas, the tree impurity of the global declarations/definitions graph is much higher, with the diva schema as uncontested outlier ($TI_{GG} = 14$). The numbers for SG suggest that schemas in general present a strongly tree-shaped structure, with very little internal reuse. The higher values for GG with respect to SG indicate that most schemas perform extensive encapsulations (using local declarations/definitions), but that dependence on the same nodes is not the criterion for grouping nodes together (otherwise, TI would go down). The degree of encapsulation in the diva schema is exceptional, indicating an uncommon specification style.

The remaining metrics (fan, coupling, coherence, instability), are not applied on complete graphs, but per node or per module only. They are useful for categorizing modules and nodes, and for finding outliers. For instance, in the Amazon schema, values well over 200 are reached for fan-out and for efferent coupling for

both CG and GG. These extremes are traceable to a single element declaration `ItemAttributes` that contains a long list of local elements, ranging from `Actor` to `WirelessMicrophoneFrequency`.

5 Related work

5.1 XML Schema metrics

The first definition of a suite of metrics for the XML Schema language has been provided by Lämmel *et al.* [6]. Their metrics range from simple counters of various types of schema particles to basic complexity metrics such as McCabe, depth, and breadth. The structure metrics proposed by us are not intended to compete with, but rather to complement this suite of metrics. Since structure belongs to the essence of XML schemas, structure metrics are likely to be valueable.

5.2 Grammar metrics

Schema metrics are similar to grammar metrics, in the sense that both schemas and grammars can be regarded as specifying first-order algebraic datatypes, or term algebras. The first definition of a suite of grammar metrics was provided by Power *et al.* [7]. This work was adopted and extended by Alves *et al.* [2] to the feature-rich grammar notation SDF. Our suite of XML Schema metrics has partly evolved out of those SDF grammar metrics. In particular, the tree impurity and normalized count of modules pre-existed this paper as grammar metrics, while the fan, coupling, instability, and coherence metrics were introduced here. As mentioned, the XsdMetz and SdfMetz tools share code libraries for graph representation, transformation, and metrication, and future releases of SdfMetz are likely to include grammar variants of most schema metrics.

5.3 Software metrics

Extensive literature exists on the subject of software metrics (for overviews, see e.g. [4, 3]). The fan and coupling metrics are well-known in their specific application to program modules. We gave a generic graph-based formulation using an abstract notion of modules as groups of nodes. No standard measures of cohesion exist, in spite of its intuitive appeal. We gave two alternative definitions of cohesion of a module: the ratio of internal dependencies versus all dependencies in which the module is involved (Ch), and the tree-impurity of the internal dependency graph. A general definition of tree impurity existed before [4]. We adapted it to directed graphs and applied it on a per-module basis.

6 Concluding remarks

6.1 Contributions

We have defined a suite of eleven structure metrics for the XML Schema language, based on three different graph representations of schema structure: the

successor graph, and the strongly connected component graph and global declaration/definition graph that can be derived from it. We have implemented the metrics suite in a prototype tool, and applied it to freely available schemas ranging up to nearly two megabyte in ascii file size. We have cursorily described the intuitions behind the metrics and their potential use in schema assessment.

6.2 Future work

The suite of metrics presented here is not claimed to be complete in any sense. Many more general software metrics may be investigated to be applicable to XML Schemas.

Before the suite of metrics proposed in this paper can reliably be applied for schema assessment, the proposed metrics must be *validated*. Such validation has been explicitly kept out of the scope of the present paper. In particular, predictive value of the metrics for *external* schema properties must be established. Also, the correlations between the different metrics must be charted out.

Though we have specifically targeted XML Schemas to be measured with the proposed structure metrics, the formulation of the metrics is sufficiently general to attempt their application to other software artifacts. This can be other XML schema notations, grammars, protocols, models, data type definitions, APIs, or any artifact that allows dependency graphs to be extracted and for which appropriate modularization notions exist. In particular, we intend to direct our attention to formats used intermixed with XML Schema, such as SOAP, WSDL, XSL, and further XML formats for specifying application-specific information.

References

1. A. Abran, J.W. Moore, P. Bourque, and R. Dupuis (*eds.*). Guide to the Software Engineering Body of Knowledge - SWEBOK, Version 2004. IEEE Computer Society, www.swebok.org.
2. T. Alves and J. Visser. Metrication of SDF grammars. Technical Report DI-PURe-05.05.01, Universidade do Minho, May 2005.
3. R. Dumke. Software metrics - a subdivided bibliography. http://irb.cs.uni-magdeburg.de/sw-eng/us/bibliography/bib_main.shtml.
4. N. Fenton and S.L. Pfleeger. *Software metrics: a rigorous and practical approach*. PWS Publishing Co., Boston, MA, USA, 1997. 2nd edition, revised printing.
5. E. Koutsofios. Drawing graphs with *dot*. Technical report, AT&T Bell Laboratories, Murray Hill, NJ, USA, November 1996.
6. R. Lämmel, S. Kitsis, and D. Remy. Analysis of XML schema usage. In *Conference Proceedings XML 2005*, November 2005.
7. J.F. Power and B.A. Malloy. A metrics suite for grammar-based software. In *Journal of Software Maintenance and Evolution*, volume 16, pages 405–426. Wiley, November 2004.
8. H.S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML Schema part 1: Structures. W3C Recommendation, May 2001.