

Formalizing metrics for COTS

Miguel Goulão, Fernando Brito e Abreu

Department of Informatics, Faculty of Sciences and Technology, New University of Lisbon

2825-114 Monte de Caparica, Portugal

miguel.goulao@di.fct.unl.pt, fba@di.fct.unl.pt

Abstract

This paper describes a technique for formalizing metrics for COTS-based architectures. This technique is built upon the UML 2.0 metamodel and uses OCL as a metrics definition language.

As a proof of concept, an example based upon a set of reusability metrics for fine-grained JavaBeans components is presented.

1. Motivation

The quality of COTS-based architectures depends, to a large extent, on the quality of the COTS it reuses [13]. COTS reuse affects existing processes and creates the need for new ones, such as COTS evaluation [5].

It is important to assess the complexity of integrating COTS components that may be acquired from different providers. Metrics on the reusability and adaptability of COTS components are interesting, both from a provider's and a user's point of view. Neither code metrics, nor OO design metrics are adequate to COTS assessment. Code metrics collection is not feasible, since neither the source code nor the internal design model are usually available to an independent COTS evaluator.

Metrics should be defined in a non-ambiguous way, allowing independent validation efforts to evaluate their usefulness. Metrics research and adoption has suffered from several recurrent difficulties, such as: (i) Ambiguity in definition - may occur when metrics are defined using informal definitions; this leads to misleading interpretations and, therefore, different implementations of metrics collection tools may produce different metrics values on the same software artifacts; (ii) Inadequate specifying formalism - may occur when metrics are defined with a formalism that requires a strong mathematical background for understanding it, often not held by practitioners; (iii) Insufficient validation - in order to earn credibility and be deemed as useful by the software industry, metrics should undergo validation,

preferably performed by different evaluators rather than just by their proponents; this independent cross validation is seldom performed due to the difficulties faced in experiment replication.

The research described in this paper aims to deal with these problems by providing practitioners with techniques and tools that allow the formal definition of metrics without sacrificing their understandability. We use a widespread modeling notation (UML 2.0 class diagrams) combined with a formal language (OCL) to specify the metrics. The Object Constraint Language (OCL) allows the formal definition of metrics by navigating through the information contained in the UML meta-model in a non-ambiguous and accessible way to practitioners at ease with UML. Moreover, the metrics' specification itself is executable, making the replication of metrics collection efforts fairly straightforward.

This paper is organized as follows: in the next section related work is presented and the shortcomings of earlier quality models and metrics proposals are briefly discussed. In section 3, some relevant features of the UML2.0 metamodel and the usage of OCL to define metrics upon it are presented. Section 4 contains a formalization of a metrics set, using our approach. In section 5, conclusions are presented and future directions of this work are outlined.

2. Related work

Table 1 contains a non-exhaustive list of proposals towards the quantitative evaluation of components. Here, we can identify the typical problems mentioned in the earlier section: ambiguity in definition, inadequate formalism and insufficient validation. It should be noted that none of these proposals are sufficiently validated and most of them are not validated at all and that all of them rely, at least to some extent on informally defined information. Even the mathematically defined metrics have some elements of their formulae defined informally and require proprietary tools to collect them. While some of the approaches rely on hierarchical quality models (QM), others are based on unstructured sets

Where	QM/C	Met. Def.	Validation	Problems
[13]	QM	-	-	dv
[3]	QM	Informal	-	dfv
[12]	C	-	-	dv
[8]	C	-	-	dv
[6]	C	Informal	Descr.Stats.	dfv
[4]	C	Math	Descr.Stats.	dfv
[16]	QM	Math	Regr.Model	dfv

Table 1. Summary of contributions for CBD metrics

of quality characteristics (C).

The formalization technique presented in this paper can help overcome these weaknesses, namely by: (i) providing a formal specification to metrics that were only informally defined, as well as rewriting formally defined ones, in such a way that these new formal specifications are executable; (ii) tying the formal specification of metrics to a metamodel, thus ensuring that there are no ambiguities on the context under which the metrics are defined and computed; (iii) packaging metrics specifications in a format that can be used by other practitioners to validate metrics sets.

The application of our formalization technique requires the availability of a COTS model expressed with UML, perhaps reversed-engineered from some other format. This technique is an evolution from the one proposed earlier in [1] [2], in the realm of object-oriented design metrics. We are now extending it to address component evaluation, using the UML 2.0 metamodel enhanced expressiveness for representing software components.

3. Formalization technique

3.1. UML 2.0 Metamodel

Our formalization technique takes as input component infrastructures defined upon the UML 2.0 metamodel specification. The latter is currently being finalized by OMG and includes both the UML 2.0 infrastructure [15] and superstructure [14] specifications. The new metamodel includes improved support for specifying component-based architectures. As shown in [9], the ability to represent software components with their ports, provided and required interfaces and improved support for hierarchical decomposition provides UML 2.0 with structural abstractions with an expressiveness comparable to the one offered by Architecture Description Languages such as Acme [7]. These improvements make the UML 2.0 a much more suitable alternative to specifying software components than its previous versions, thus making it a sound baseline for our formalization

technique.

We choose to use UML rather than specific implementation languages such as Java as the source for our metrics collection activities to allow for a source code language independent metrics collection framework. In the particular case of Java Beans, we use a UML model which was reverse engineered from Jar archives. A similar approach can be followed to obtain component specifications from other sources, as long as UML tools support their reverse engineering.

3.2. Using OCL to define metrics

OCL 2.0 [10] is a formal language for specifying constraints and is part of the UML 2.0 standard. In order to specify such constraints, OCL offers a rich navigability through instances of the UML metamodel, allowing the collection of information about the navigated model. We found such expressiveness to be quite suitable to formally define metrics upon metamodel instances [1][2].

4. Formalizing a suite of reusability metrics

4.1. The suite of Washisaki *et al.*

In [16], Washisaki defined a metrics suite for assessing Java Beans components' reusability, discussed their usefulness and applicability, and presented a validation experiment based on a sample of over one hundred fine grained beans available at <http://www.jars.com/>. These metrics will be shortly presented in sections 4.2 through 4.5. Their formalization with OCL is presented in section 4.6, while the metrics collection environment is briefly described in section 4.7. It is out of the scope of this paper to critically discuss potential flaws or improvements on this metrics set, as our main goal is to illustrate the formalization technique.

4.2. Rate of Component Observability (RCO)

RCO represents the ratio of fields in the component's facade class that are readable properties. The facade class is the one that provides the component interface to other interacting components. While a very low observability makes it difficult for users to understand the component's behavior, a very high value may lead to difficulties in finding specific properties among the available ones.

$$RCO(c) = \begin{cases} \frac{P_r(c)}{A(c)}, & A(c) > 0 \\ 0, & otherwise \end{cases}$$

where:

$P_r(c)$: number of readable properties in c .

$A(c)$: number of fields in c 's facade class.

4.3. Rate of component customizability (RCC)

RCC represents the ratio of fields in the component's facade class that are writable properties. It is a measure of component's costumizability. A low value can imply a poor adaptability of the component, while a very high one indicates a break in the encapsulation of the component.

$$RCC(c) = \begin{cases} \frac{P_w(c)}{A(c)}, & A(c) > 0 \\ 1, & otherwise \end{cases}$$

where:

$P_w(c)$: number of writable properties in c .

4.4. Self-Completeness of Component's Return Value (SCCr)

This metric represents the ratio of business methods which return no value. It is a degree of the component's self-completeness and independence. The higher the value is, the higher the component portability.

$$SCC_r(c) = \begin{cases} \frac{B_v(c)}{B(c)}, & B(c) > 0 \\ 1, & otherwise \end{cases}$$

where:

$B_v(c)$: business methods without return value in c .

$B(c)$: business methods in c .

4.5. Self-Completeness of Component's Parameter (SCCp)

This metric represents the ratio of business methods with parameters, thus measuring the self-completeness of the information dealt by the component. A low value indicates a low dependency of the component on the exterior.

$$SCC_p(c) = \begin{cases} \frac{B_p(c)}{B(c)}, & B(c) > 0 \\ 1, & otherwise \end{cases}$$

where:

$B_p(c)$: Business methods with parameters in c .

4.6. OCL formalization

In this section, we present the metrics definitions formalization with OCL. All the presented functions are defined in the scope of the UML 2.0 meta-class Component. We start by defining Washizaki's metrics and then present a set of auxiliary functions used in the metrics definitions.

Component

```
-- Washizaki's metrics first...
RCC(): Real = if self.A() = 0
0
```

```
else
self.Pr()/self.A()
RCC(): Real = if self.A() = 0
0
else
self.Pw()/self.A()
SCCr(): Real = if self.B() = 0
1
else
self.Bv()/self.B()
SCCp(): Real = if self.B() = 0
1
else
self.Bp()/self.B()

-- Auxiliar functions
Pr(): Integer =
self.ownedOperation->select(o: Operation|
o.stereotype.name = 'getter')->size()
Pw(): Integer =
self.ownedOperation->select(o: Operation|
o.stereotype.name = 'setter')->size()
A(): Integer =
self.ownedAttribute->size()
Co(): Integer =
self.ownedOperation->select(o: Operation|
o.stereotype.name = 'constructor')->size()
Bv(): Integer =
self.ownedOperation->select(o: Operation|
not (o.stereotype.name = 'constructor') &&
not (o.stereotype.name = 'getter') &&
not (o.stereotype.name = 'setter'))->
select(b: Operation|
b.type = void)->size()
Bp(): Integer =
self.ownedOperation->select(o: Operation|
not (o.stereotype.name = 'constructor') &&
not (o.stereotype.name = 'getter') &&
not (o.stereotype.name = 'setter'))->
select(b: Operation|
b.ownedParameter->size() > 0)->size()
B(): Integer =
self.ownedOperation->size()
- self.Pr() - self.Pw() - self.Co()
```

An additional metric named EMI (Existence of Meta-Information) was included in Washizaki's set, but we discarded it here since its definition was java-dependent (existence of a BeanInfo class).

4.7. Metrics collection

Since the metrics are formalized as OCL expressions based upon the UML 2.0 metamodel, we require three abilities from our experimental environment: (i) the ability to generate an abstract representation of UML2.0-compliant

meta-objects out of a given components' architecture; (ii) the ability to instantiate the UML2.0 meta-classes with those meta-objects; (iii) the ability to parse and execute OCL expressions upon the metamodel populated with the meta-objects.

We have developed a prototype environment with these characteristics, by combining (i) a UML design tool (USE [11]) that allows loading a metamodel, populating it and evaluating OCL expressions upon it with (ii) an XMI front-end that generates all the required data from UML component infrastructures specifications in XMI. Please note that the reverse engineer tool used was able to classify the operations with the stereotypes 'constructor', 'getter' and 'setter'.

5. Conclusions and future work

The approach presented in this paper plays a central role in the undergoing project of establishing a framework for the quantitative evaluation of CBD software. Indeed, being able to automatically compute formally expressed metrics on component infrastructures is an enabling condition to pursue the following threads in our research: (i) propose a formal definition for CBD metrics defined by other researchers; (ii) define our own metrics for CBD; (iii) actively contribute to cross-validation of CBD metrics, both by conducting validation experiments for the formalized metrics and by sharing with the research and industrial communities a working environment upon which such validation experiments can be carried out.

The metrics formalized in this paper are centered in fine-grained components, but the approach is flexible enough to be applied at different levels of granularity and with different concerns. In particular, we are interested in exploring metrics for component assemblies, to assess the hypothesis that rather than evaluating isolated components, we should focus our attention in evaluating component assemblies, to capture the effect of the different components interaction in the overall quality of the component assembly.

With the finalization of the UML 2.0 standard, tools will begin to support both its metamodel and OCL expressions evaluation. By using OCL to express the metrics, we are laying the ground for an effective integration between metrics collection and development tools used by practitioners in their daily activities. Moreover, OCL expressions can also be defined to support heuristics-based quality wizards in design tools, in an elegant fashion that is tool independent, as long as the tool supports the standard UML metamodel and OCL evaluation.

The proposed approach is independent from the metamodel. As long as a metamodel can be defined, and one is able to populate it with the appropriate model elements, metrics upon it can be computed following a similar technique to the one presented in this paper.

References

- [1] F. B. Abreu. Using OCL to formalize object oriented metrics definitions. *INESC, Grupo de Engenharia de Software, Technical Report ES007/2001, <http://ctp.di.fct.unl.pt/QUASAR/>*, May 2001.
- [2] A. L. Baroni and F. B. Abreu. Formalizing Object-Oriented Design Metrics upon the UML Meta-Model. *Proceedings of the Brazilian Symposium on Software Engineering, Gramado - RS, Brazil*, 2002.
- [3] M. Bertoa and A. Vallecillo. Quality Attributes for COTS Components. *Proceedings of the 6th International Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE'2002), Malaga, Spain*, 2002.
- [4] M. A. S. Boxall and S. Araban. Interface Metrics for Reusability Analysis of Components. *Proceedings of the Australian Software Engineering Conference (ASWEC'2004), Melbourne, Australia*, 2004.
- [5] L. Brownsword, T. Oberndorf, and C. Sledge. Developing New Processes for COTS-Based Systems. *IEEE Software*, pages 48–55, 2000.
- [6] R. Dumke and A. Schmietendorf. Possibilities of the Description and Evaluation of Software Components. *ACM SIGSOFT Software Engineering Notes*, 5, 2000.
- [7] D. Garlan, R. T. Monroe, and D. Wile. Acme: Architectural Description of Component-Based Systems. *Foundations of Component Based Systems, G. T. Leavens and M. Sitaraman, Eds.: Cambridge University Press*, pages 47–68, 2000.
- [8] N. S. Gill and P. S. Grover. Component-Based Measurement: Few Useful Guidelines. *ACM SIGSOFT Software Engineering Notes*, 28, 2001.
- [9] M. Goulao and F. B. Abreu. Bridging the gap between Acme and UML for CBD. *Proceedings of the Specification and Verification of Component-Based Systems Workshop (SAVCBS'2003) at the ESEC/FSE'2003, Helsinki, Finland*, 2003.
- [10] OMG. Unified Modeling Language: OCL version 2.0. *OMG ptc/03-08-08, <http://www.omg.org/>*, 2003.
- [11] M. Richters. A UML-based Specification Environment. *<http://www.db.informatik.uni-bremen.de/projects/USE:> University of Bremen*, 2001.
- [12] S. Sedigh-Ali, A. Ghafoor, and R. A. Paul. Software Engineering Metrics for COTS-Based Systems. *IEEE Computer*, 2001.
- [13] R. Simao and A. Belchior. Quality Characteristics for Software Components: Hierarchy and Quality Guides. *Component-Based Software Quality: Methods and Techniques*, 2003.
- [14] U2-Partners. 2nd revised submission to OMG RFP ad/00-09-02: Unified Modeling Language: Superstructure - version 2.0. *OMG, <http://www.omg.org/>*, 2003.
- [15] U2-Partners. 3rd revised submission to OMG RFP ad/00-09-01: Unified Modeling Language: Infrastructure - version 2.0. *OMG, <http://www.omg.org/>*, 2003.
- [16] H. Washizaki, H. Yamamoto, and Y. Fukazawa. A Metrics Suite for Measuring Reusability of Software Components. *Proceedings of the Metrics'2003*, 2003.