Independent Validation of a Component Metrics Suite

Miguel Goulão, Fernando Brito e Abreu

Faculty of Sciences and Technology, Lisbon New University, Portugal

{miguel.goulao|fba}@di.fct.unl.pt

Abstract

This paper describes an independent validation study for a suite of reusability metrics for component based design (CBD). We use an approach to metrics definition and collection that is different from the one originally proposed by the original authors. The metrics under validation were proposed using a semi-formal notation, namely a combination of mathematical formulae with natural language descriptions for elementary parts of those formulae. They were then computed using proprietary tools. In contrast, we will present a formalization for the metrics suite that combines the UML 2.0 metamodel with OCL. By using this technique, our contribution provides: (i) a formal, portable and executable definition of the metrics set that can be used by other researchers and practitioners to perform independent validations of the metrics suite; (ii) a prototype working environment to perform such independent validation experiments, both with this and other metrics sets.

1. Motivation

Software quality modeling involves not only the definition of adequate taxonomies of quality attributes, but also the establishment of methods to assess those attributes. This assessment may be performed both in a qualitative and in a quantitative way. In this paper, we are concerned with the latter. The quantitative assessment of quality attributes requires the usage of software metrics. Quality models can then be built based on a combination of such metrics, using general purpose statistical techniques.

Statistical models require validation before they can be adopted by a large community. This validation should cover:

- **Internal consistency** – The model is specified through a set of mathematical statements, so, its' validity may be checked for mathematical correctness. A set of inputs are collected from the system represented by the model, as well as any assumptions about the system elements. The model allows computing a set of outputs representing the predicted behaviors of the systems being modeled. In an internally consistent model, the outputs are valid if the inputs are valid.

- **External consistency** – A model has external consistency if information collected from it is not contradicted by other valid information. This relates to the applicability of the model, as it focuses on the extent to which the assumptions made in the model apply beyond the sample from which the model was delivered.

The credibility of CBD quality models and related metrics suites depends not only on the soundness of their proposal, but also on the extent to which they are validated, not only by their own proponents, but also through third-party efforts. The current state of practice in what concerns CBD quality models and metrics is still far from reaching this level of maturity. Some quality models and metrics have been proposed [1, 2], but they lack proper validation in order to be widely accepted by the CBD community.

Although more validation studies are required, several difficulties hamper this task, such as the lack of available data for conducting case studies, problems with the interpretation of the models and metrics specifications and lack of supporting tools for data collection.

In this paper, we undergo a CBD metrics suite validation effort. While the original validation used semi-formally defined metrics and proprietary tools to collect them we will use a formal definition for such metrics and an experimental environment that combines widely used tools with standard languages for representing the metrics specification, heuristics based on those metrics and the experimental data. This approach creates the conditions so that further experiments with this metrics suite can be performed upon different samples of components. Furthermore, the techniques and tools used in this paper are generic and can be used in the specification and validation of other metrics sets, such as the one proposed by [3].

Our approach combines UML 2.0 class diagrams with OCL functions to specify both metrics and heuristics. With the upcoming adoption of the new UML standard [4, 5] it is likely that the major tool vendors will provide direct support to the new metamodel, as well as OCL [6], which is also part of the standard.

This paper is organized as follows. In section 2, we present some related work on independent validation of software metrics. In section 3, we present the formal definition of the metrics set used in this experiment as well as a brief discussion on its main features. Section 4 contains the formal definitions of a set of heuristics that helps interpreting the metrics. In section 5, a metrics collection experiment is described and discussed. The used metrics set is then analyzed for its limitations. Some limitations of the formalization technique applicability are also identified. The experimental setting is discussed in section 6. Conclusions and further work are outlined in section 7.

2. Related work

The idea of independent validation of quality models and metrics is not new. It was borrowed from other sciences, where it is common to cross-check model validity by conducting independent experiments. For instance, in the pharmaceutical industry, new drugs have to undergo independent testing before they are approved to be made available to the community.

In the realm of experimental software engineering, there are several examples of this sort of independent scrutiny, concerning software metrics suites. In [7], Basili checked the adequacy of Chidamber and Kemerer's metrics suite [8] to predict class fault-proneness. The same metrics suite has been assessed as a maintainability predictor in [9]. Metrics such as McCabe's cyclomatic complexity [10] have been extensively used both in academic and industrial settings and integrated in several commercial development tools.

It is more often the case where metrics are proposed but insufficiently validated. Examples can be found concerning metrics in general and CBD-related ones in particular (e.g. [3, 11, 12]). Other proposals, such as the one by Washizaki [13] go further along the way of validating the proposed models. The latter contrasts with the previous ones in that its validation relies on a sounder statistical analysis, rather than on anecdotal examples or "gut-feeling" hints on descriptive statistics collected from small samples. As it happens with most metrics proposals for CBD, to the best of our knowledge it still lacks independent validation. Among other reasons, the novelty of these metrics is one of the motives why they still have not been independently validated. We expect our work to contribute in mitigating this problem.

The formalization approach presented in this paper is an evolution of our previous work concerning the formalization of metrics for OO design. The approach of using OCL to perform such formalization was introduced in [14], using the GOODLY OO design language [15] as a base metamodel. With the growing adoption of UML by the software industry and academia, the need to make metrics available to the common practitioner has led us to developing the FLAME metrics extraction library in OCL, based upon the UML metamodel, upon which several metrics suites were formalized [16-19]. With the upcoming UML 2.0 standard, and its extended expressiveness for specifying component based architectures, as shown in [20], we have decided to evolve our approach so that it would support the new UML standard and benefit from it in the quantitative evaluation of CBD. The work presented in this paper differs from our previous work in metrics formalization in the following ways:

- we are now using a new metamodel as a basis for our experiments;
- the focus of the metrics being analyzed as shifted from OO design to CBD;
- the tool support is becoming more and more independent from proprietary formats, with the adoption of XMI as the input format for our metrics collection tools.

3. Washizaki's metrics set formalization

3.1. Metrics set description

Washizaki proposed a metrics set of 5 metrics for JavaBeans component reusability assessment in [13]. For each of the metrics, the authors presented:

- their intent;
- their definitions combining a mathematical and an informal formulation;
- a confidence interval [Lower Limit; Upper Limit] for each metric; if its value is outside this interval, the component is regarded as potential source of future problems, badly designed, or prone to exhibit bad behavior;
- interpretation heuristics based on such confidence intervals.

The quality characteristic, factors and criteria that lead to the development of each of the metrics are presented in Figure 1.

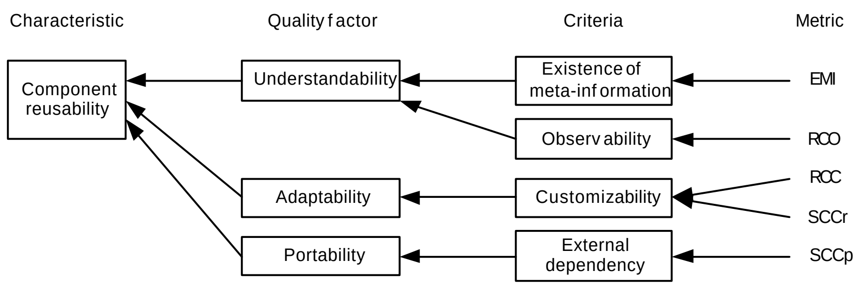

Figure 1 – Washizaki's quality model for reusability (adapted from [13])

In order to validate his approach, Washizaki performed a case study where a sample of 125 JavaBeans from *jars.com* [21] was used, along with a prototype tool to extract the metrics from *jar* files. At *jars.com*, components are rated for quality with an 8 levels scoring system that employs expert opinions on the components. The rationale is that this rating can be used for component reusability because the criteria publicized by *jars.com* (presentation, functionality and originality) are related to component reusability. This rating was used as a dependent variable in his validation experiment. His confidence intervals for high quality components, used in the definition of heuristics, were computed by selecting only the components at the top 2 quality levels and then computing the typical values for each metric in those components, with a confidence coefficient of 95%.

3.2. Metrics formalization

An enabling step for applying our technique is the formalization of the original metrics definitions to OCL, using the UML 2.0 metamodel as a basis upon which we represent the components. The discussion on the formalization was presented in [22]. Only 4 out of the 5 metrics in Washizaki's set were formalized. The remaining metric (Existence of Meta-Information – EMI) cannot be conveniently expressed upon the UML 2.0 metamodel because it is implementation language dependant. Here, we present the metrics formalization for the purpose of self-containment of the paper. The functions with the comment `(Washizaki)` are Washizaki's metrics, while the ones with the `(Auxiliar)` comment, are auxiliary functions.

```
Component
-- (Washizaki) Rate of Component Observability
RCO(): Real = if self.A() = 0 then
                 0.0
              else
                 self.Pr()/self.A()
              endif
```

```
-- (Washizaki) Rate of Component Customizability
RCC(): Real = if self.A() = 0 then
                 0.0
              else
                 self.Pw()/self.A()
              endif

-- (Washizaki) Self-Completeness of Component's return value
SCCr(): Real = if self.B() = 0 then
                  1.0
               else
                  self.Bv()/self.B()
               endif

-- (Washizaki) Self-Completeness of Components Parameter
SCCp(): Real = if self.B() = 0 then
                  1.0
               else
                  self.Bp()/self.B()
               endif

-- (auxiliar) Readable Properties
Pr(): Integer = self.ownedOperation->select(o: Operation|
                                            o.stereotype = 'getter')->size()

-- (auxiliar) Writable Properties
Pw(): Integer = self.ownedOperation->select(o: Operation|
                                            o.stereotype = 'setter')->size()

-- (auxiliar) Properties in the component
A(): Integer = self.ownedAttribute->size()

-- (auxiliar) Total number of constructors in the component
Co(): Integer = self.ownedOperation->select(o: Operation|
                                            o.stereotype = 'constructor')->size()

-- (auxiliar) Business methods with no return value
Bv(): Integer = self.ownedOperation->select(o: Operation|
   (not (o.stereotype = 'constructor'))and
   (not (o.stereotype = 'getter')) and
   (not (o.stereotype = 'setter')))->select(b: Operation|
                                            b.RETURN_TYPE() = 'void')->size()
```

4. *Formal definition of heuristics*

Along with his metrics set, Washizaki also proposed a set of heuristics, to aid in the metrics interpretation. We can further explore this formalization effort by providing the OCL specifications of such heuristics.

```
Class
AboveRange (limit: Real, value: Real): Boolean = value > limit

BelowRange (limit: Real, value: Real): Boolean = value < limit

OutOfRange (lowerLimit: Real, upperLimit: Real, value: Real): Boolean =
   (self.BelowRange (lowerLimit, value)) or (self.AboveRange (upperLimit, value))

Component
WarningRCO(lowerThreshold: Real, upperThreshold: Real): Boolean =
          self.OutOfRange (lowerThreshold, upperThreshold, self.RCO())
```

```
WarningRCC(lowerThreshold: Real, upperThreshold: Real): Boolean =
           self.OutOfRange (lowerThreshold, upperThreshold, self.RCC())

WarningSCCr(lowerThreshold: Real): Boolean =
           self.BelowRange(threshold, self.SCCr())

WarningSCCp(lowerThreshold: Real, upperThreshold: Real): Boolean =
   self.OutOfRange (lowerThreshold, upperThreshold, self.SCCp())

DesignWarning(RCO_LL:  Real, RCO_UL:  Real,
              RCC_LL:  Real, RCC_UL:  Real,
              SCCp_LL: Real,
              SCCr_LL: Real, SCCp_UL: Real): Boolean =
                (self.WarningRCO(RCO_LL, RCO_UL))
             or (self.WarningRCC(RCC_LL, RCC_UL))
             or (self.WarningSCCr(SCCr_LL))
             or (self.WarningSCCp(SCCp_LL, SCCp_UL))
```

We chose to define the first three predicates at the `Class` level, rather than at the `Component` one. In the UML 2.0 metamodel, `Component` is a subclass of `Class`, so the predicates can be used with components. This allows us to reuse those predicates in the definition of heuristics based on metrics for OO design. Three of the heuristics based on Washizaki's metrics (`WarningRCO`, `WarningRCC` and `WarningSCCp`) behave as a band-pass filter, in the sense a potential problem warning should be issued if the metrics value is either lower than the lower threshold, or higher than the upper one, for the `RCO`, `RCC` and `SCCp` metrics. The `WarningSCCr` heuristic only establishes a minimum threshold for the value of the `SCCr` metric. If the metric value is below the threshold, this should be interpreted as an indication of a potential problem regarding the quality characteristic being assessed by that particular metric (portability). Finally the `DesignWarning` heuristic is defined as a simple combination of the previous ones. The arguments of the `DesignWarning` predicate allow calibrating each heuristic, as more data gets collected. Table 1 summarizes the heuristics thresholds information provided by Washizaki. For each metric, we present its acronym, the average value found in the metrics sample, the lower and upper limits for high confidence and the number of components in the sample which fulfill this For the sake of completeness, we include also the thresholds for `EMI`. All the metrics in this suite are defined as ratios and their maximum possible value is 1,00. For this motive, the predicate of `SCCr` does not require an Upper Limit.

Metric	**Average**	**Lower Limit**	**Upper Limit**	**# components**
RCO	0,40	0,17	0,42	36
RCC	0,35	0,17	0,34	35
SCCr	0,85	0,61	1,00	108
SCCp	0,74	0,42	0,77	28
EMI	0,84	0,50	1,00	105

Table 1 – Washizaki's heuristics thresholds

Checking if a given component violates any of these heuristics can then be achieved by evaluating the following expression:

```
DesignWarning (0.17, 0.42, 0.17, 0.34, 0.61, 0.42, 0.77)
```

5. *Discussion*

5.1. A metrics collection experiment

In order to test our formalization technique we conducted the following experiment: we collected Washizaki's metrics upon a public domain component library. The *FukaBeans* component library [23] was developed according to the JavaBeans component system [24] by Washizaki's research team. Each component is distributed as a separate *jar* file.

Table 2 contains the metrics values for each of the components in the library. In our analysis, we used reverse engineered models obtained from those *jar* files. The bold values represent data points where the heuristics proposed by Washizaki are not followed.

Although all the average values for the metrics are well inside the quality intervals suggested by Washizaki's experiments, only two of the components (GameBean and GraphBean) comply with all the quality heuristics. Seven out of twelve components fail three out of four heuristics. It may seem surprising that the components developed by the metrics set proponents fail to meet the structural quality standards proposed by their own authors. It is worth noticing that while Washizaki's model was calibrated with commercial JavaBeans components, this sample contains JavaBeans developed with academic purposes, with usually less than 10 methods. A possible interpretation for the apparent lack of reusability, with respect to Washizaki's quality model, is that, for such relatively small interfaces, the model is vulnerable: since all the metrics are defined as ratios, the small number of elements used in their computation leads to a high standard deviation of metrics values.

JavaBean	RCO	RCC	SCCr	SCCp
CellBean	**0,037**	**0,111**	0,909	**0,818**
FileUtil	**1,000**	**0,667**	1,000	**1,000**
FilterBean	0,267	**0,133**	0,933	**0,200**
FukaCalendarBean	**0,444**	**0,444**	0,857	0,571
FukaGraphBean	**1,000**	**1,000**	1,000	0,733
FukaStopWatchBean	**0,667**	**0,667**	1,000	**0,200**
FukaTextBean	**0,000**	**0,000**	1,000	**1,000**
GameBean	0,250	0,250	1,000	0,556
GraphBean	0,182	0,273	1,000	0,714

StatementBean	**0,667**	**0,667**	**0,500**	0,500
DocumentBean2	**0,000**	**0,000**	1,000	**1,000**
WordBean2	**0,000**	**0,000**	1,000	**1,000**
Mean	*0,376*	*0,351*	*0,933*	*0,691*
Standard deviation	*0,376*	*0,333*	*0,145*	*0,294*

Table 2 - Washizaki's metrics for the FukaBeans Library

5.2. Definition of the metrics set

Several issues can be raised regarding the definition of this metrics set.

The original metrics definition is ambiguous in what concerns inheritance. It is unclear how inherited features (methods and attributes) should be accounted for. Our formalization only uses the directly defined features. While for this particular sample of components this is not a problematic issue, it is possible to define hierarchies of object-oriented components where this option would have an influence on the metrics values.

Another possible concern relates to the complexity associated with parameter types in the evaluation of the complexity of method interfaces. The metrics just count the number of parameters, thus being blind to parameter type repetition and parameter type complexity. It can be argued that none of them should be irrelevant, here. Consider two methods with N arguments. The first one has all the arguments of the same type, while the second has N different types of arguments, so, this may be a source of extra complexity that is not captured by these metrics. The parameters complexity may also have its weight in the perceived complexity by the component user, but, again, this is not reflected in this metrics set.

5.3. Applicability of the metrics set

Washizaki’s metrics set was designed to assess reusability of fine grained components (JavaBeans) through the analysis of their interface complexity. This limits somewhat the scope of model elements being analyzed. UML architectural components have a much richer expressiveness than the one used in these metrics, which leave out important model elements such as the provided and required interfaces, as well as the events the component may produce or consume. Furthermore, it does not address extra-functional aspects of the components.

5.4. Formalization technique limitations

Our formalization technique relies on the expressiveness of the underlying metamodel. As long as a concept can be consistently and unambiguously represented by the meta-objects of such metamodel, it should be possible to explore those meta-objects with appropriate OCL expressions and extract relevant information from them. Therefore, the formalization limitations may arise from two sources:

- Lack of expressiveness of the meta-model – the standard UML 2.0 metamodel may have to be extended with profiles to conveniently express some extra-functional properties
- Lack of expressiveness of OCL – OCL may lack the syntactic sugar to extract some information

In this paper, we briefly discussed one limitation that combines both issues, concerning the automatic detection of BeanInfo classes. Such detection would imply analyzing the classes' names and detecting JavaBeans naming conventions and these are not conveniently captured by the meta-model, or detectable with OCL expressions.

6. Experimental setting

In this section we describe a portable tool environment we are assembling to apply the formalization technique described in this paper to real-world examples. Since the metrics are formalized as OCL expressions based upon the UML 2.0 metamodel, we require three abilities from such an environment:

- the ability to generate an abstract representation of UML2.0-compliant meta-objects out of a given components' architecture;
- the ability to instantiate the UML2.0 meta-classes with those meta-objects;
- the ability to parse and execute OCL expressions upon the metamodel populated with the meta-objects.

Figure 2 presents an overview of our experimental environment. The UML metamodel specification was obtained from [25] in XMI format. The component assembly specifications we want to evaluate were also obtained in XMI format, produced by a UML design tool used for reverse-engineering the *jar* files and then converting the reverse-engineered models to XMI. OCL metrics specifications and heuristics specifications are also fed into the OCL expressions evaluator. The evaluation of metrics expressions and heuristics expressions are the outputs of this process.

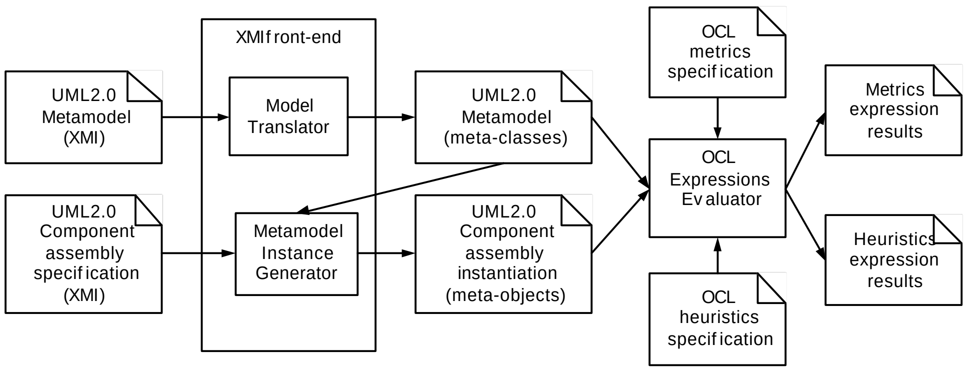
XMI front-end
UML2.0 Metamodel (XMI)
Model Translator
UML2.0 Metamodel (meta-classes)
OCL metrics specification
Metrics expression results
UML2.0 Component assembly specification (XMI)
Metamodel Instance Generator
UML2.0 Component assembly instantiation (meta-objects)
OCL Expressions Evaluator
Heuristics expression results
OCL heuristics specification

Figure 2 - Overview of metrics evaluation support

Until a new generation of case tools, with UML 2.0 compliant dictionaries and supporting OCL expressions evaluation is not available, we are using the USE tool as our OCL expressions evaluator. Both the USE tool and the XMI front-end will be further described in this section.

5.1. USE tool

The USE tool [26] was developed by Mark Richters at the University of Bremen to allow expressing constraints on UML class diagrams with OCL. This allows populating the specified system and evaluating the value of the OCL expressions. We now analyze with some detail the benefits and shortcomings of this tool.

5.1.1. Benefits of the USE tool

The USE tool includes a model loader. The OCL evaluator is used not only for checking the model state against the model's constraints, but also to query detailed information about the model's state. Both features are key elements for the metrics formalization: the former, by allowing to load the UML metamodel, so that we can populate it with model elements that correspond to the component infrastructure we want to measure; the latter, because each metric's specification is defined as a query upon the loaded model's state.

5.1.2. Drawbacks of the USE tool

Despite of its virtues, USE has a few shortcomings. The first one is that it only supports a subset of the UML, leaving out some model elements that would be quite useful to our intents, such as packages. Although this limitation can be circumvented by using qualified identifiers rather than simple ones, it is nevertheless an extra source of complexity for our metrics definition. A second shortcoming of the USE tool is that it uses non-standard import formats for UML models and objects. In our first formalization attempts, we developed an adaptor between a commercial UML modeling tool and USE [16, 18]. The drawback of this approach was being tied to a specific proprietary tool. The increasing adoption of XMI as a standard interchange format by tool producers, lead us to the requirement of developing an XMI front-end. Another drawback of the USE tool is the lack of an incremental loading facility that would allow the separation of metrics definition from heuristics and from the model itself.

5.2. XMI front-end

In order to generate USE specifications from models specified with several UML design tools, we have developed a XMI to USE adapter that takes as input an XMI specification and outputs the two following results:

- a USE specification equivalent to the UML class diagram expressed in XMI; we use this feature to generate a UML 2.0 metamodel in the USE input format (UML 2.0 metamodel meta-classes, in Figure 2);
- a USE command file with the instructions to generate the set of meta-objects representing the component assembly (UML 2.0 component assembly instantiation, in Figure 2).

With these 2 features we are then able to input the UML 2.0 metamodel in the USE tool and then load it with any given meta-objects representing a specific component assembly.

In order to illustrate the metrics collection process, we now provide a small example. Consider the JavaBean component Chart, in Figure 3. Although quite simple, this bean allows us to show how the metrics are computed. It contains two attributes and five operations. Out of those five operations, one is a constructor, one is a getter, one is a setter, and the remaining two are business operations. The setter operation is the only one with an argument. The comment label presented on the left shows the values of Washizaki's metrics, when computed for this JavaBean.

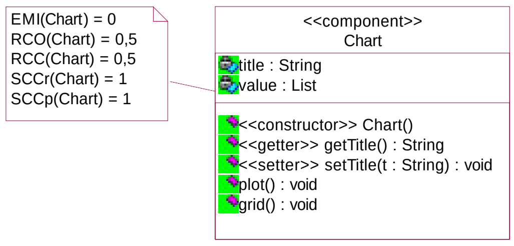

Figure 3 – Chart JavaBean class diagram

The metrics collection relies on a relatively small subset of the UML metamodel. A portion of it is depicted in Figure 4. From this metamodel extract, we can see how components are linked to their operations (ownedOperations) and properties (ownedAttributes).

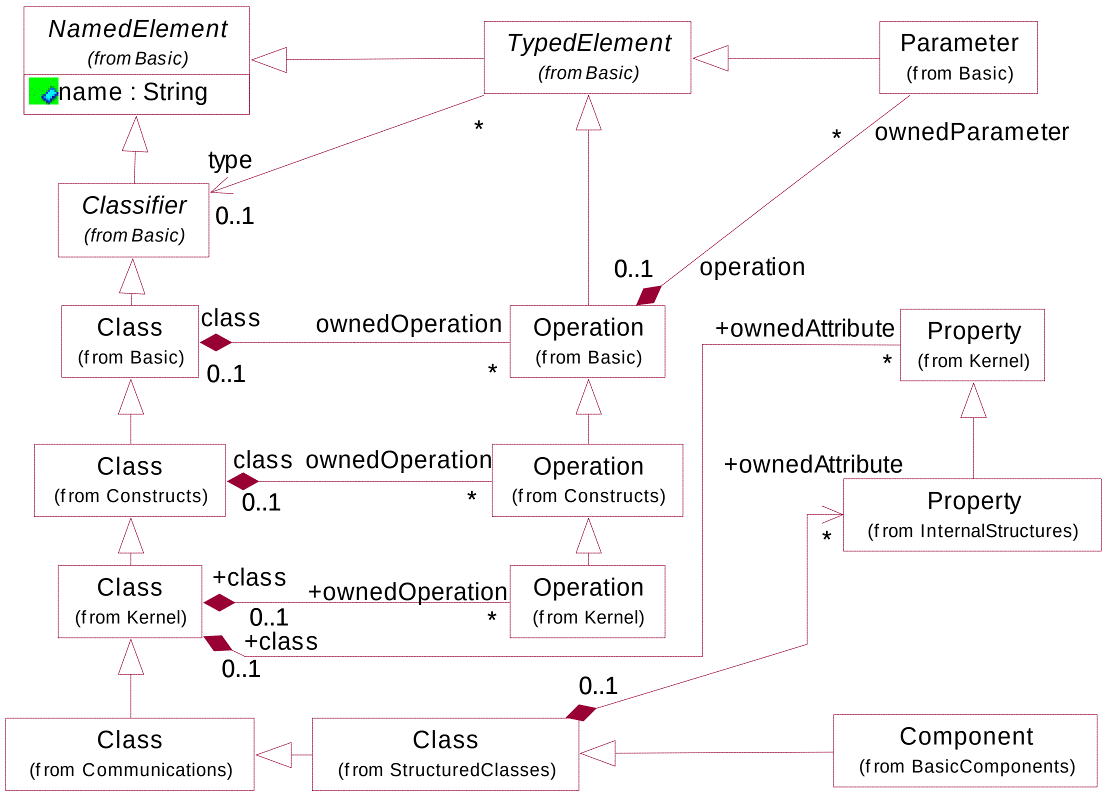

Figure 4 – UML 2.0 metamodel extract

To collect the metrics, we need to populate the metamodel with the necessary meta-objects. Again, we just show a few of them for illustration purposes, in Figure 5. In this extract, we show only the Chart component and its owned operations and attributes. The computation of the metrics is performed by navigating through this populated metamodel. For instance, to compute the number of properties, we simply compute the size of the collection of elements linked to `c1` by the `ownedAttribute` association.

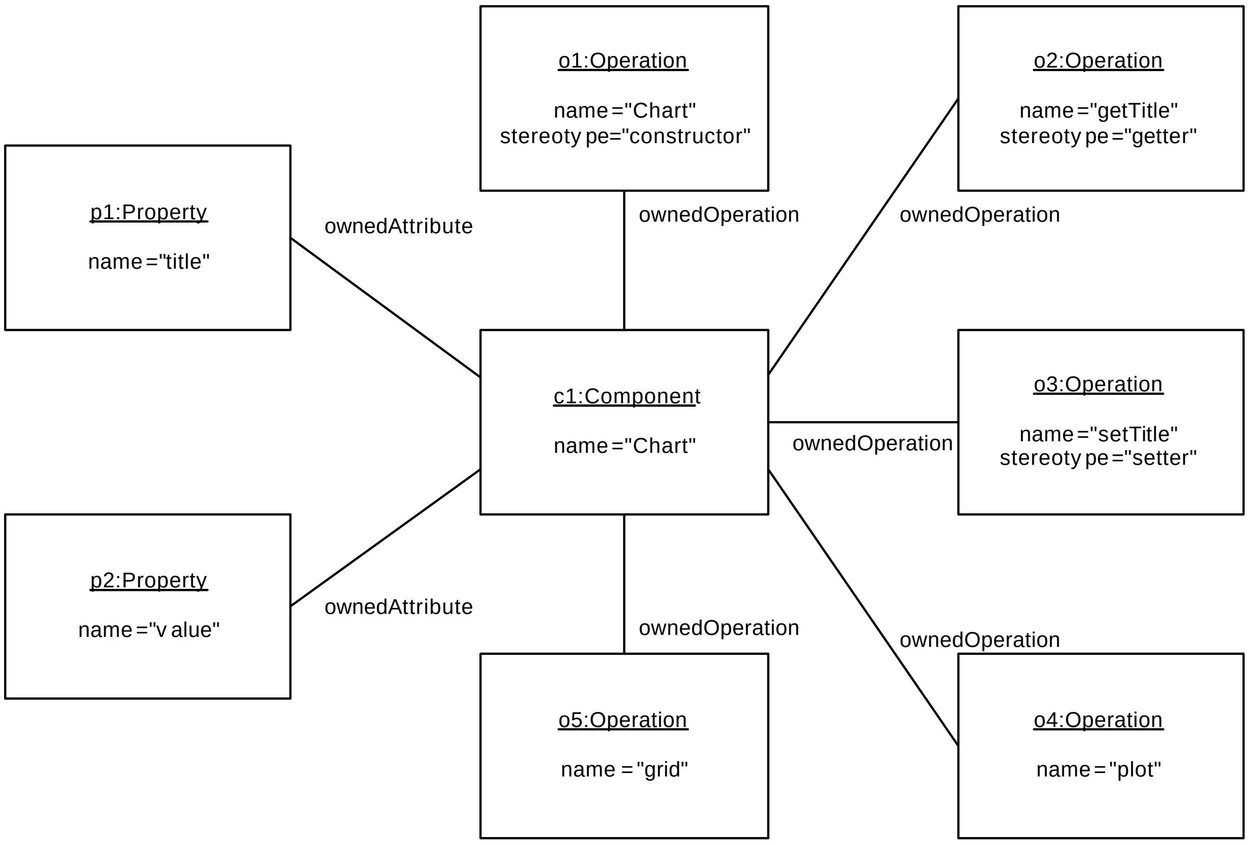

Figure 5 – Meta-objects diagram extract

7. Conclusions and future work

Independent validation of metrics is an essential step if they are to be adopted by a broader audience. The current CBD metrics state of the art is dominated by proposals that are insufficiently validated not only by their own authors, but also by third party experiments. This *status quo* results from several factors, ranging from the relative novelty of most CBD metrics proposals to the "traditional" difficulty of getting appropriate samples to validate the proposed metrics. Other common difficulties with a large number of metrics specifications are the ambiguity in metrics definition and the usage of inadequate specifying formalisms. Both hamper independent validation of metrics.

The work presented in this paper contributes to mitigate some of these problems. It provides a formal, portable and executable way of specifying metrics for CBD, using standard notations such as UML 2.0 class diagrams, OCL, and XMI, as well as a prototype environment upon which metrics collection can be performed. The emphasis on standard technologies aims to bring together the academic and practitioners communities, by providing a simple, but powerful mechanism to integrate metrics collection and heuristics-based help for CBD with the practitioners' normal development environments. We believe this level of integration is a key factor to foster independent validation of metrics proposals.

Although the metrics formalized in this paper are centered in fine-grained components, 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. The final objective is to capture the effect of the different components interaction in the overall quality of the component assembly.

The approach is generic in the sense that, from a conceptual point of view, it is independent from the underlying metamodel. In a parallel effort, we are exploring the definition of metrics for the Common Warehouse Metamodel.

References

[1] M. Bertoa and A. Vallecillo, "Quality Attributes for COTS Components", 6th International Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE'2002), Málaga, Spain, 2002.

[2] N. S. Gill and P. S. Grover, "Component-Based Measurement: Few Useful Guidelines", ACM SIGSOFT Software Engineering Notes, vol. 28, 2003.

[3] A. v. d. Hoek, E. Dincel, and N. Medvidovic, "Using Service Utilization Metrics to Assess and Improve Product Line Architectures", Ninth International Software Metrics Symposium (Metrics'03), Sydney, Australia, 2003.

[4] U2-Partners, "3rd revised submission to OMG RFP ad/00-09-01: Unified Modeling Language: Infrastructure - version 2.0", U2-Partners January 2003.

[5] OMG, "Unified Modeling Language: Superstructure - Version 2.0 - Final Adopted Specification", Object Management Group Inc. ptc/03-08-02, 2003 2003.

[6] OMG, "Unified Modeling Language: OCL (version 2.0)", Object Management Group Inc. ptc/03-08-08, August 2003.

[7] V. Basili, L. Briand, and W. L. Melo, "A Validation of Object-Oriented Design Metrics as Quality Indicators", IEEE Transactions on Software Engineering, vol. 22, pp. 751-760, 1996.
[8] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering, vol. 20, pp. 476-493, 1994.
[9] W. Li and S. Henry, "Object-Oriented Metrics that Predict Maintainability", Journal of Systems and Software, vol. 23, pp. 111-122, 1993.
[10] T. McCabe, "A Complexity Measure", IEEE Transactions on Software Engineering, vol. 2, pp. 308-320, 1976.
[11] R. Dumke and A. Schmietendorf, "Possibilities of the Description and Evaluation of Software Components", Metrics News, vol. 5, 2000.
[12] M. A. S. Boxall and S. Araban, "Interface Metrics for Reusability Analysis of Components", Australian Software Engineering Conference (ASWEC'2004), Melbourne, Australia, 2004.
[13] H. Washizaki, H. Yamamoto, and Y. Fukazawa, "A Metrics Suite for Measuring Reusability of Software Components", Metrics'2003, 2003.
[14] F. B. Abreu, "Using OCL to formalize object oriented metrics definitions", INESC, Software Engineering Group ES007/2001, May (versão 0.9) 2001.
[15] F. B. Abreu, L. M. Ochoa, and M. A. Goulão, "The GOODLY Design Language for MOOD2 Metrics Collection", ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering, Lisboa, Portugal, 1999.
[16] A. L. Baroni, S. Braz, and F. B. Abreu, "Using OCL to Formalize Object-Oriented Design Metrics Definitions", QUAOOSE'2002, Malaga, Spain, 2002.
[17] A. L. Baroni, "Formal Definition of Object-Oriented Design Metrics": Vrije Universiteit Brussel - Belgium, in collaboration with Ecole des Mines de Nantes - France and Universidade Nova de Lisboa - Portugal, 2002.
[18] A. L. Baroni and F. B. Abreu, "Formalizing Object-Oriented Design Metrics upon the UML Meta-Model", Brazilian Symposium on Software Engineering, Gramado - RS, Brazil, 2002.
[19] A. L. Baroni and F. B. Abreu, "A Formal Library for Aiding Metrics Extraction", International Workshop on Object-Oriented Re-Engineering at ECOOP'2003, Darmstadt, Germany, 2003.
[20] M. Goulão and F. B. Abreu, "Bridging the gap between Acme and UML for CBD", Specification and Verification of Component-Based Systems (SAVCBS'2003), at the ESEC/FSE'2003, Helsinki, Finland, 2003.

[21] JARS, "http://www.jars.com/".

[22] M. Goulão and F. B. Abreu, "Formalizing Metrics for COTS", MPEC'2004, Edimburgh, 2004.

[23] Y. Fukazawa, H. Washizaki, H. Yamamoto, T. Adachi, Y. Sakai, K. Satoh, and D. Hoshi, "FukaBeans: JavaBeans Components Library, http://www.fuka.info.waseda.ac.jp/Project/CBSE/fukabeans/".

[24] G. Hamilton, "JavaBeans (version 1.01-A)", Sun Microsystems, API Specification August 1997.

[25] N. Consortium, "Neptune Consortium UML 2.0 resources page - http://neptune.irit.fr/".

[26] M. Richters, "A UML-based Specification Environment". http://www.db.informatik.uni-bremen.de/projects/USE: University of Bremen, 2001.