Using OCL to Formalize Object-Oriented Design Metrics Definitions

Aline Lúcia Baroni $^{1, 2}$ Sofia Braz $^{1, 3}$ Fernando Brito e Abreu $^{1, 2, 4}$

FCT / Universidade Nova de Lisboa - Portugal ^{1}
École des Mines de Nantes - France ^{2}
Universidade da Beira Interior - Portugal ^{3}
INESC - Portugal ^{4}
{aline, braz, fba}@di.fct.unl.pt

Abstract

This paper describes the formalization effort of different sets of object-oriented metrics definitions using the Object Constraint Language (OCL), a part of the Unified Modeling Language (UML) standard. The formalization is based upon the UML meta-model. This approach allows unambiguous metrics definition, which in turn helps increasing tool support for Object-Oriented metrics. Also, it is possible to establish comparisons among the formalized sets of metrics.

1. Introduction

The lack of formalization has been long felt in the object-oriented software modeling area [Meyer1985, Wand1989]. In the first known book on the subject of metrics for the Object-Oriented paradigm all metrics are defined in natural language [Lorenz1994]. Additionally, many authors have used a combination of set theory and simple algebra to express their metrics [Chidamber1994, Abreu1995, Abreu1996a, Henderson-Sellers1996a].

In spite of the existence of many metric sets, problems often arise from the formality degree used to define them. When metrics are informally expressed, using natural language, people using metrics can interpret them in several ways. Two distinct teams can obtain completely different results when applying a particular metric to the same system. On the other extreme, when metrics are defined using some kind of mathematical formalism, the majority of software designers may not have the required background to understand the complex mathematical expressions that are used.

To illustrate these problems, consider the metrics “Number of Times a Class is Reused” [Lorenz1994] and “Count of Synchronization-based Coupled Object Types (CSCO)” [Poels2001]. The former is defined as the number of references to a class. However it is not clear what references are and how the metric should be computed. Should we count internal and external references? Should references be considered in different modules, packages or subsystem? Does the inheritance relationship count as a reference? It is visible that without clear and precise definitions it is impossible to build adequate metric extraction tools, experiments replication is hampered, and results interpretation may be flawed.

Poels define

$$\mathrm{CSCO}(P) = \#\{Q\ ?\ \ T - \{P\} \mid\ ?\ e\ ?\ \ A{:}\ (?_1(e, P) = C\ ?\ ?_1(e, Q) = E)\ \ ?\ \ (?_1(e, P) = E\ ?\ ?_1(e, Q) = C)\}.$$

Finding out the meaning of this formula, even knowing each of the components involved, is probably not an easy nut to crack, for most software designers. In other words, these two metrics solve the problem of metric definition, but introduce others, difficult to solve.

In this work, we propose an approach to metrics definition that combines understandability with formality. The remaining part of this article is organized as follows. In section 2, the problem of formalization in Object-Oriented paradigm is presented, introducing one approach that has been used as formal method. In section 3, the formalization of metrics is explained and illustrated with the GOODLY language [Abreu2001]. Section 4 shows the architecture that supports the approach being proposed. Section 5 outlines our conclusions and further work. It enunciates our intention of setting up a framework for metrics definition and comparison, creating a meta-model for the metrics construction. Finally, some references used in this work are shown.

2. Formalization in Object-Oriented Design

The problem of the lack of formalization in the metrics field has always been an Achilles' heel in the area. Although the basic elements of the diagrammatic specifications used in this area are very powerful and obvious, several modeling details (such as uniqueness and referential restraints, limitations and other constraints) are expressed ambiguously, or even cannot be conveyed at all by those graphical notations.

As pointed out in [Cook1994], there is a difference between precision and detail. In object-oriented modeling the details are often left out but at the same time, precision is needed. Our approach uses the Object Constraint Language (OCL) to guarantee precision. Section 2.1 introduces OCL. In section 2.2 we utilize OCL to formalize some Object-Oriented metrics.

2.1 The Object Constraint Language (OCL)

Accuracy and non-ambiguity in specification have been, for many years, the aims of the branch of computer science known as “formal methods”. Several attempts have been made to combine them with object-oriented modeling [Abreu2001]. The most recent and promising approach, which bridges formal methods with object-orientation is called OCL - Object Constraint Language [Warmer1999], and it is a part of the UML standard, published by the OMG (Object Management Group) [OMG1997].

OCL is a formal, yet simple notation, to be used jointly with UML diagrams and whose syntax has some similarities to those of object-oriented languages such as Smalltalk, C++ or Eiffel. It is underpinned by mathematical set theory and logic, as it happens with formal languages, but was designed for usability and is easily grasped by anybody familiar with object-oriented modeling concepts in general, and UML notation in particular. OCL is the result of an effort to combine formalism soundness with usability and has its roots in the Syntropy method [Cook1994]. It allows expressing three kinds of constraints on parts of object-oriented models: invariants, pre-conditions an post-conditions which are predefined in UML as standard stereotypes: «invariant», «precondition» and «postcondition». Constraints convey a number of benefits, namely improved precision and better design documentation, resulting in better (unambiguous) communication among the parties involved, such as designers, users, programmers, testers or managers.

The context of both pre and post-conditions is an operation. As an example, consider the *prepend* operation included in the *Sequence* type definition:

```
Sequence::prepend(object: T): Sequence(T)
post: result->size() = self@pre->size() +1
post: result->at(1) = object
```

OCL is a declarative typed language whose expressions are free of side effects. This means that the state of the objects does not change by the application of an OCL expression. These expressions can range from simple comparisons (e.g. an attribute having an upper limit) to complex navigations in a class diagram through their associations. Since it is a typed language, it is possible to check expressions for validity during modeling. To obtain a better description of the syntax and semantics of OCL, refer to [OMG1997].

2.2 Object-Oriented Metrics Formalization

To bring precision into design, one has to fully understand the semantics of the modeling concepts, which is usually described through what is called a *meta-model*. A meta-model is a language for describing models. It describes the model objects and the relationships that can exist between them. For instance, the UML meta-model is the description of UML elements, produced with the UML notation itself. Both the UML notation and meta-model can be found in [OMG1997].

In our research we proposed a textual Object-Oriented design language, named GOODLY (a Generic Object-Oriented Design Languague? Yes!), and we produced its corresponding respective meta-model [Abreu2001]. GOODLY is used as a basis for the extraction of quantitative data (metrics). During the construction of the GOODLY meta-model, OCL is used to provide precision, as shown next.

In GOODLY, one *specification* is a set of linked parts of the design. It is controlled by someone (the owner of the *specification*) and it can use services of other *specifications*. For example, one application can be built using available parts of others. Figure 1 shows the *Specification meta-class*. Following the figure, some OCL expressions are exemplified.

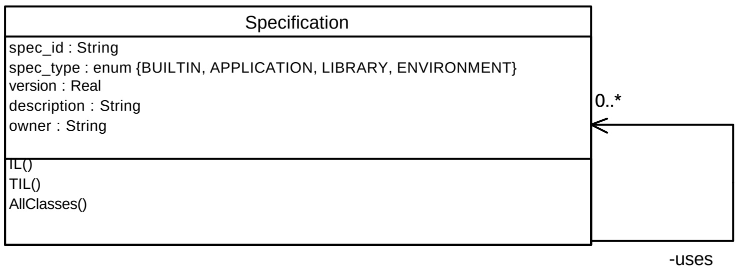

Figure 1 – *Part of the GOODLY meta-model*

The identifiers of one *specification* must be unique. This restriction can be expressed as:

```
Specification
Specification.allInstances -> forAll (s1, s2: Specification |
                                  s1 <> s2 implies s1.spec_id <> s2.spec_id)
```

Each *specification* can use others, but one *specification* cannot use itself. This invariant can be formalized in OCL as:

```
Specification
not (uses -> includes (self))
```

In GOODLY, four types of *specification* were defined (*spec_type* attribute on figure 1). The next invariant indicates that one *specification* of the type *Application* must have a *main* clause not empty.

```
Specification
spec_type = #APPLICATION implies (main_spec -> notEmpty())
```

The complete meta-model of GOODLY can be seen in Appendix B.

3. Formalization of Object-Oriented Metrics

In this work, our intention is to formalize different sets of metrics using OCL. We believe that even when expressed mathematically, some metrics may be ill defined and we want to verify this hypothesis.

The ill definition problem may happen due to two reasons:

i) Metrics definitions are usually presented without the corresponding context, that is, without expressing which is the corresponding meta-model where the entities of interest and their interrelationships are expressed;

ii) Metrics definition is done without an underlying formal specification approach that uses the former meta-model as contextual input. This formal specification should specify under which conditions the metrics are applicable.

We formalized the set of the MOOD2 metrics [Abreu1998] using the GOODLY meta-model as background. Now, we want to formalize this set and others using the UML meta-model and OCL to express different metrics as meta-model operations. The metrics applicability limitations are defined with OCL pre-conditions. The metrics result itself is formally defined with OCL post-conditions. As an example, we present the formalization of two metrics over the GOODLY meta-model. The DIT – Depth of Inheritance Tree - metric belongs to the MOOSE set [Chidamber1994] and corresponds to the maximum depth of the inheritance chain above the considered class, i.e., the number of inheritance relations between the considered class and the root class.

```
Class::DIT():Integer
post: result = (if self.IsRoot() then
                   0
                else
                   1 + self.Parents()->iterate(elem:Class; acc:Integer=0 |
                                    if elem.DIT() > acc then
                                          elem.DIT()
                                    else
                                          acc
                                    endif)
                endif)
```

Here, *elem* is an iterator. There is an iteration over the set of direct ascendant classes of the current class - self. The variable *acc* is the accumulator, initialized with zero. The iteration allows to calculate the maximum DIT() among the parent classes of the class (taking into account multiple inheritance). In each iteration the accumulator receives the value indicated after the "|", which in this case can be the value of a recursive call to DIT() or its previous value (when the accumulator is not altered).

The EIF – External Inheritance Factor – belongs to the MOOD2 group [Abreu1999] and is expressed as a ratio. The numerator, represented as IL – Inheritance Links, is the number of inheritance relations where the derived class belongs to the current specification and the base class belongs to the specification passed in the parameter. The denominator, represented as TIL – Total Inheritance Links, is the total number of inheritance relations where the derived class belongs to the current specification. This is formalized as:

```
Specification::EIF(s: Specification): Percentage
pre: self.uses -> includes (s)
pre: self.TIL() > 0
post: result = self.IL(s) / self.TIL()


Specification::IL(s: Specification): Integer
post:  result = AllClasses().Parents() -> select(IsInternal(s)) -> size()

Specification::TIL(): Integer
post:  result = AllClasses() -> iterate(elem: Class; acc: Integer = 0 |
                                                          acc + elem.PC())
```

In the complete version of this paper, we will describe several examples of design metrics, expressed upon the UML meta-model instead of the GOODLY one.

4. Architecture of the Solution

The current state of art for giving precision to Object-Oriented modeling is depicted in figure 2. Commercial UML modeling tools (e.g.: Rational Rose, Objectory, Visio, etc.) provide some graphic diagram editors that allow building models of systems. The models (represented by X, Y and Z in the picture) are stored in the tool repository.

Nowadays, those modeling tools do not offer facilities for the evaluation of OCL expressions. Nevertheless, several tools (Use, Cybernetic Parser, Elixer, etc.) emerging from undergoing research projects can be obtained and used to formalize the models designed with the commercial tools, provided that the models can be exported with an appropriate input format to the OCL tools.

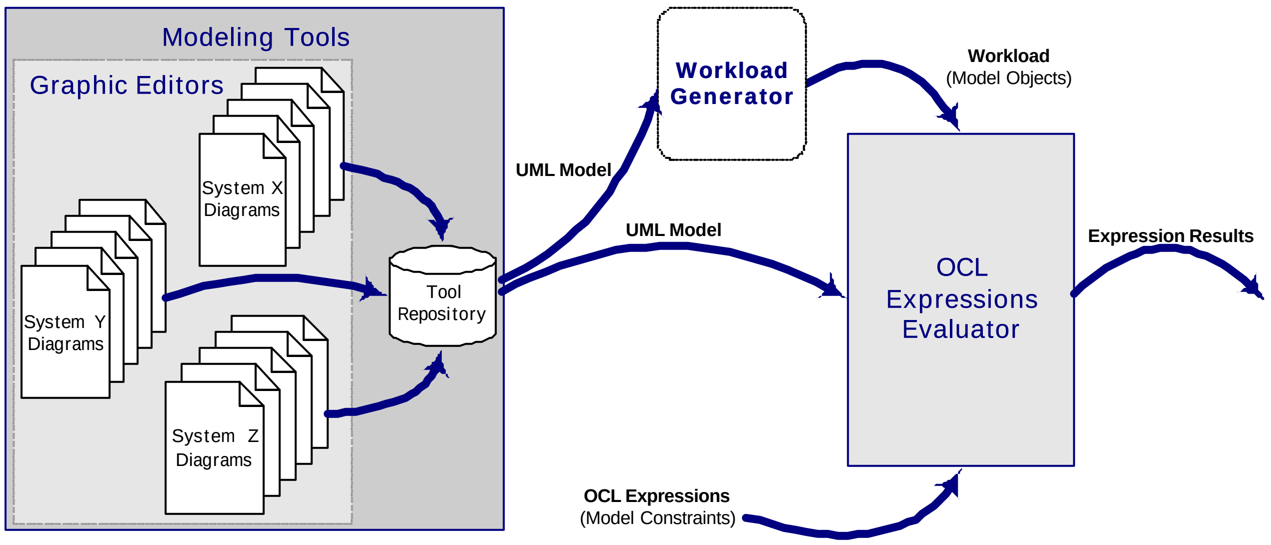

Figure 2 – *Model-Level Architecture*

After the file conversion (of the model to a representation that can be understood by OCL tools), the real instances of the entities in the diagram are created and the model is populated (i.e., a plenty of objects, corresponding to the entities in the model, is created). These instances are the base of the assertions that are constructed with OCL. Until now, the workload (creation of the objects) is done "by hand", but it can be automated.

The diagrams that compose the models serve as input to an OCL evaluation tool, which take the converted representation of the diagram (as GOODLY, for example), the added OCL constraints and the instances of the model, and evaluate each of the constraints, showing the results. Each assertion is tested and its result is showed to the user.

While the architecture depicted in figure 2 corresponds to a model-level evaluation, the one depict in figure 3 is related to a meta-model-level evaluation.

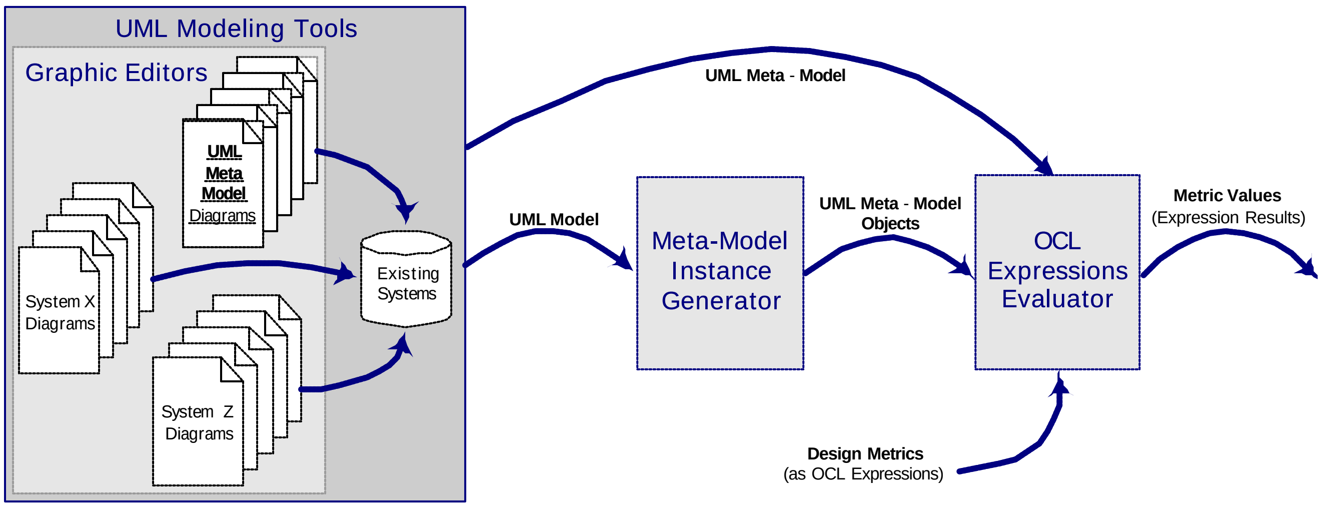

Figure 3 – *Meta-Model-Level Architecture*

In the meta-model level architecture, all the functionalities of the previous one are preserved. Notwithstanding, there are two main additions: one is the introduction of the class diagram of the UML meta-model. Another is the introduction of an automatic instance generator, that will take the meta-model and automatically generate all the instances to populate it. Using these features (meta-model and corresponding instances) we will formalize and test a relatively large sample of metrics that can be found in the literature, expressed as pre and post-conditions, over the UML meta-model.

We used a modeling tool to create the UML meta-model diagram, and with the architecture in figure 2, we can already convert it to a textual format. We are currently working on the generation of the automatic instances and on the formalization of the metric sets. We hope to clarify the metrics definitions through formalization, and also to compare the existing sets and abstract the best characteristics of each one of them. Our idea is to introduce a framework that allows practitioners to build new metrics or improve old ones and afterwards, to create the meta-metrics. We also expect to detect the limitations of the sets of metrics during the formalization process.

5. Conclusions and further work

We used the Object Constraint Language, a part of the UML standard, to define object-oriented design metrics in a very natural and understandable way. The precision granted by the formality of OCL comes at a much lower cost, for both practitioners and tool builders, than when using other formal specification constructs. Since UML has also become a *de facto* standard both in academia and industry, more and more people are expected to master OCL and use it currently it their designs.

Besides formalizing some metrics sets using the UML meta-model, we also plan to make a similar effort based upon the OML (*OPEN Modeling Language*) meta-model. OML emerged from the OPEN (*Object-oriented Process, Environment and Notation*) consortium [Firesmith1996, Henderson-Sellers1996b, Henderson-Sellers1996c, Henderson-Sellers1998]. The latter is supported by a large group of well-known methodologists such as *Brian Henderson-Sellers* (author of the *MOSES* method [Henderson-Sellers1991, Henderson-Sellers1994]), *Ian Graham* (author of *SOMA – Semantic Object Modelling Approach* [Graham]), *Donald Firesmith* or *Jim Odell*.

After the formalization efforts, we will test and compare the metric sets, performing empirical studies. Furthermore, we will abstract the common characteristics of those sets in order to build a high level meta-model. The latter will be a framework for describing, classifying and accessing existing metric sets, as well as a basis for the production of new ones.

6. References

[Abreu2001] Fernando Brito Abreu : *Using OCL to Formalize Object-Oriented Metrics Definition*, Report ES007/01 of the Software Engineering Group, INESC, Portugal, 2001.

[Abreu1998] Fernando Brito Abreu : *The MOOD2 Metrics Set* (In Portuguese), Report R7/98 of the Software Engineering Group, INESC, Portugal, 1998.

[Abreu1996a] Fernando Brito Abreu, Rita Esteves & Miguel Afonso Goulão : “The Design of Eiffel Programs: Quantitative Evaluation Using the MOOD Metrics”, actas de *TOOLS'96 (Technology of Object Oriented Languages and Systems)*, Santa Barbara, CA, EUA, Julho, 1996a.

[Abreu1995] Fernando Brito Abreu, Miguel Afonso Goulão & Rita Esteves : “Toward the Design Quality Evaluation of Object-Oriented Software Systems”, actas de *5th International Conference on Software Quality*, pp.44-57, Austin, Texas, EUA, Outubro, 1995.

[Araújo1998] João Araújo & P. Sawyer : “Integrating Object-Oriented Analysis and Formal Specification”, *Journal of Brazilian Computer Society*, July, 1998.

[Chidamber1994] Shyam R. Chidamber & Chris F. Kemerer : “A Metrics Suite for Object Oriented Design”, *IEEE Transactions on Software Engineering*, vol.20, nº6, pp.476-493, IEEE, Junho, 1994, publicado também em: Center of Information Systems Research, MIT, EUA, WP No. 249, July 1993.

[Cook1994] Steve Cook & John Daniels : *Designing Object Systems: Object Oriented Modeling with Syntropy*, Prentice Hall, Hemel Hempstead (U.K.), 1994.

[Duke1991] D. Duke, P. King, G. A. Rose & G. Smith : “The Object-Z Specification Language", *Department of Computing Science, University of Queensland, Australia*, relatório 91-1, 1991.

[Firesmith1996] Donald Firesmith, Brian Henderson-Sellers & Ian Graham : “OPEN Modeling Language (OML) - Core Notation Specification", *OPEN Consortium*, relatório Version 0.1, Julho, 1996.

[Henderson-Sellers1991] Brian Henderson-Sellers : *A BOOK of Object-Oriented Knowledge*, Prentice Hall PTR, Sydney, Australia, 1991.

[Henderson-Sellers1994] Brian Henderson-Sellers & Julian M. Edwards : *BOOK TWO of Object-Oriented Knowledge: the Working Object*, Prentice Hall, Sydney, Australia, 1994.

[Henderson-Sellers1996a] Brian Henderson-Sellers : *Object-Oriented Metrics - Measures of Complexity*, série: The Object-Oriented Series, Prentice Hall PTR, Upper Saddle River, NJ, EUA, ISBN 0-13-

239872, 1996a.

[Henderson-Sellers1996b] Brian Henderson-Sellers : “The OPEN Methodology”, *Object Magazine*, vol.6, nº9, pp.56-59, Novembro, 1996b.

[Henderson-Sellers1996c] Brian Henderson-Sellers & Ian Graham : “OPEN: Towards Method Convergence?”, *IEEE Computer*, vol.29, nº4, pp.86-89, IEEE / Object Technology Department, Abril, 1996c.

[Henderson-Sellers1998] Brian Henderson-Sellers, Tony Simons & Houman Younessi : *The OPEN Toolbox of Techniques*, Addison-Wesley Publishing Company, ISBN 0-201-33134-9, 1998.

[Jones1990] Cliff B. Jones : *Systematic Software Development Using VDM*, 2ª edição, Prentice-Hall International, Hemel Hempstead (U.K.), 1990.

[Lorenz1994] Mark Lorenz & Jeff Kidd: *Object-Oriented Software Metrics*, Prentice-Hall, Englewood Cliffs, NJ, EUA, 1994.

[Meyer1985] Bertrand Meyer : “On Formalism in Specifications”, *IEEE Software*, vol.2, nº1, pp.6-26, Janeiro, 1985, publicado também em: T. Colburn, J. Fetzer, and T. Rankin (eds.), Program Verification: Fundamental Problems in Computer Science, Kluwe r Academic Publishers, Dordrecht (The Netherlands), 1993.

[Meyer1995] Bertrand Meyer : “Beyond Design by Contract: Putting More Formality into Object-Oriented Development”, actas de *TOOLS EUROPE*, Versailles, France, 1995.

[Meyer1997] Bertrand Meyer : *Object-Oriented Software Construction*, 2ª edição, Prentice Hall PTR, Upper Saddle River, NJ, EUA, ISBN 0-13-629155-4, 1997.

[Moreira1996] Ana Moreira & R. Clark : “Adding Rigour to Object-Oriented Analysis”, *Software Engineering Journal*, vol.11, nº5, pp.270-280, July, 1996.

[OMG1997] OMG : *Object Constraint Language Specification (version 1.1),* Rational et al. (ed.), ad97-08-08, Object Management Group, 1997.

[Poels2001] Geert Poels & Guido Dedene : *Measuring Event-Based Object-Oriented Conceptual Models*. L'Object Magazine, 2001.

[Rumbaugh1991]James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy & William Lorensen : *Object-Oriented Modelling and Design*, Prentice Hall, Englewood Cliffs, NJ, EUA, 1991.

[Spivey1992] J. Michael Spivey : *The Z Notation: A Reference Manual*, 2ª edição, Prentice Hall, Hemel Hempstead, Reino Unido, 1992.

[Waldén1995] Kim Waldén & Jean-Marc Nerson : *Seamless Object-Oriented Software Architecture: Analysis and Design of Reliable Systems*, Prentice Hall, Hemel Hempstead, Reino Unido, 1995.

[Wand1989] Yair Wand : “A Proposal for a Formal Model of Objects”, em Kim WoneFrederick H. Lochovsky (eds.): *Object-Oriented Concepts, Databases, and Applications*, Addison-Wesley, Reading, MA, EUA, 1989.

[Warmer1999] Jos Warmer & Anneke Kleppe : *The Object Constraint Language: Precise Modeling with UML*, Addison-Wesley Publishing Company, ISBN 0-201-37940-6, 1999.

Appendix A – The GOODLY Meta-Model

To clarify the examples presented through this document, the GOODLY meta-model is illustrated in this section. For a complete description, refer to [Abreu2001].

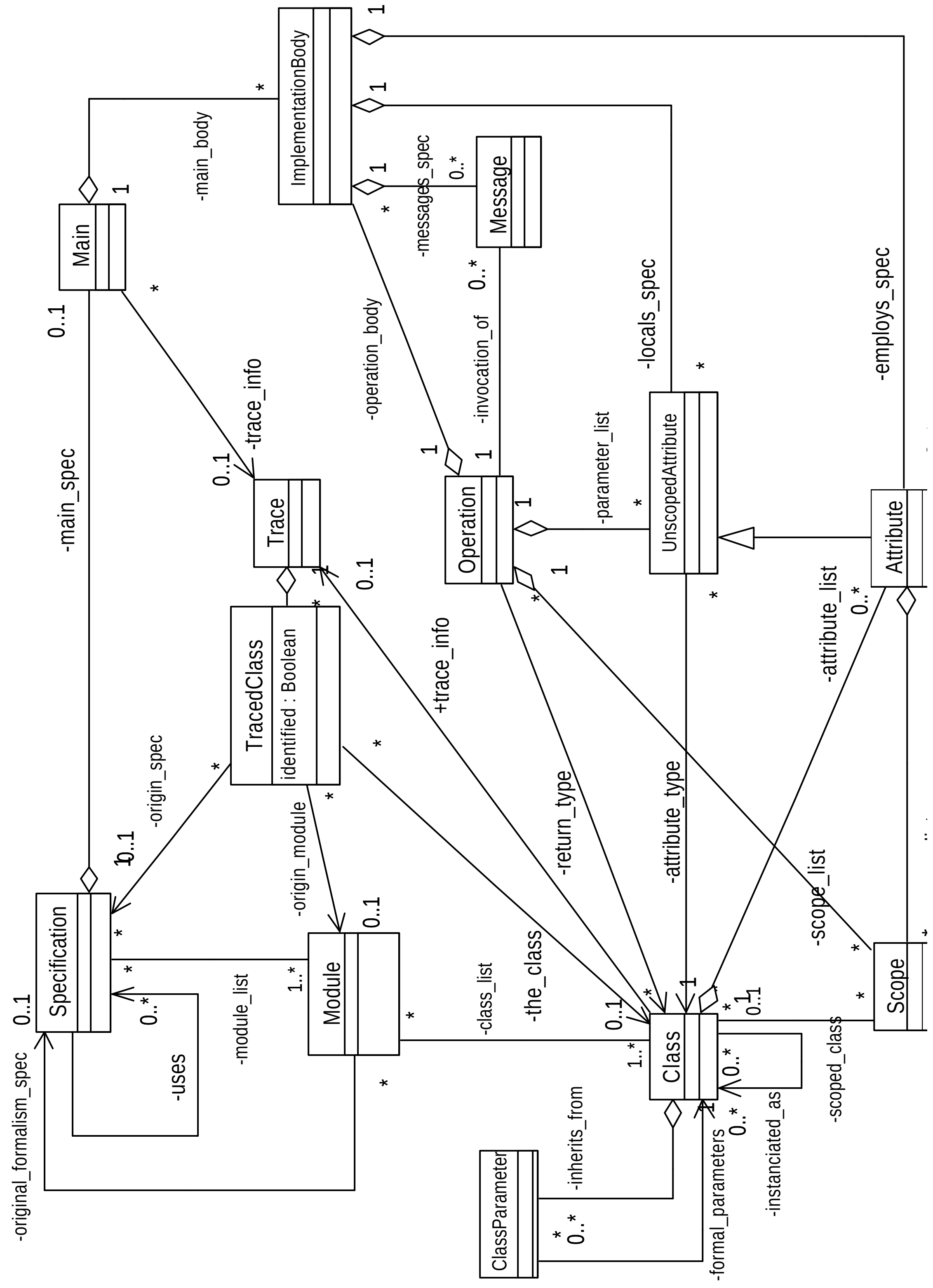

Figure 4 – *The GOODLY Meta-Model*