

SMML: Software Measurement Modeling Language

Beatriz Mora, Félix García, Francisco Ruiz, Mario Piattini

Alarcos Research Group, Department of Computer Science,
University of Castilla-La Mancha
{Beatriz.Mora | Felix.Garcia | Francisco.Ruiz | Mario.Piattini}@uclm.es

Abstract

Domain Specific Languages (DSLs) and Software Measurement are at present increasingly important in Software Engineering research.

Domain Specific Languages (DSLs) and Software Measurement are at present increasingly important in Software Engineering research

They have, in fact, become important aspects of the software industry. Domain languages facilitate the software development process in a specific domain, and measurement can help to address certain critical issues in software development and maintenance by facilitating the making of decisions. This work presents a language which allows users to define software measurement models based on the Software Measurement Ontology. Syntactically and semantically correct models in this language conform to a specific measurement metamodel, which is aligned with the aforementioned ontology.

Keywords SMML, DSL, Software Measurement

1. Introduction

Software Measurement has become a fundamental aspect of Software Engineering [1]. Measurement is proving to be highly effective in, among other things, the construction of high quality prediction systems for large-scale data base projects [2]; in the understanding and improvement of software development and maintenance projects [3]; in the evaluation and guarantee of system quality (by highlighting problematic areas) [4]; and in the determination of better work practices with the goal of assisting users and investigators in their work [4]. Moreover, software measures assist in the evaluation and institutionalization of Software Process Improvement in those organizations which develop them. Software Measurement is, in fact, a key element in initiatives such as SW-CMM (Capability Maturity Model for Software), ISO/IEC 15504 (SPICE, Software Process Improvement and Capability dEtermination) and CMMI (Capability Maturity Model Integration) [5]. The ISO/IEC 90003:2004 standard [6] also highlights the importance of measurement in managing and guaranteeing quality. Various methods and standards with which to carry out measurements in a precise and systematic manner exist, of which the most representative are:

- **Goal Question Metric (GQM):** The basic principle of GQM is that the carrying out of the measurement must always be oriented towards an objective. GQM defines an objective, refines that objective into questions and defines measures which attempt to answer those questions.

- **Practical Software and Systems Measurement (PSM):** The PSM methodology [7] is based upon the experience obtained from organizations through which the best manner in which to implement a software measurement programme with guarantees of success is discovered.
- **IEEE 1992 (Methodology for Software Quality Measures):** according to the IEEE 1992 standard, software quality can be considered as the extent to which the software possesses a clearly defined and desirable combination of quality attributes.
- **ISO/IEC 15939:** this international standard [8] identifies the activities and tasks which are necessary to successfully identify, define, select, apply and improve software measurement within a general project or within a business measurement structure.

The availability of a language which allows users to represent those elements which must be taken into account in the measurement processes might, therefore, be important in decision making and in process improvement.

It is thus of interest to consider the use of Domain Specific Languages (DSLs). DSLs appear in the context of Domain Specific Modeling (DSM). Domain-Specific Modeling raises the level of abstraction beyond programming by specifying the solution with the direct use of domain concepts. The final products are generated from these high-level specifications. This automation is possible because both the language and generators need to fit the requirements of only one company and domain. Industrial experiences of DSM consistently show it to be 5-10 times more productive than current software development practices [9], including current UML-based implementations of MDA. DSM does to code what compilers did to assembly language. Besides this vision, more investigation is needed in order to advance the acceptance and viability of DSM. Selection of a domain is a first step towards development of domain-specific languages which implies trade offs between more general applicability of the DSL and more specificity [10]. In other words, a trade off between the focus and size of the language is needed. A language which represents a larger domain can be weakly specialized to any particular aspect of the domain. On the contrary, a language which represents a small domain may have a limited number of target users [11].

These aspects constitute the main interest of this paper, whose objective is to propose the Software Modeling Measurement Language (SMML) which will permit software measurement models to be created in a simple and intuitive manner. This language has been done by using the Software Measurement Metamodel (SMM) [12] (for greater detail see [13]) as the Domain Definition Metamodel (DDMM). This language belongs to the Software Measurement Framework (SMF) presented in [14] and also discussed in Section 3 of this paper. SMF allows stakeholders to

obtain generic measurement through transformations by using two initial models as a starting point: that of software measurement and that domain. The task of the SMML is to facilitate the definition of software measurement models, which is the starting point of generic software measurement processes.

The remainder of the paper is organized as follows. Section 2 provides an overview of related works and Section 3 briefly describes SMF. In Section 4 SMML is explained, including the definition of the abstract syntax, concrete syntax and semantics. Section 5 illustrates the use of SMML in the context of a case study. Finally, conclusions and future work are outlined in Section 6.

2. Related Work

There are numerous works related to the development of DSLs. On the one hand we can find publications which present methodologies, proposals, tools and patterns with which to facilitate the development of DSLs [15-20].

In [15] is proposed that the next step towards developing a technology for software manufacturing is the development of DSLs.

In order to aid the DSL developer, [16] identifies patterns in the decision, analysis, design, and implementation phases of DSL development. These patterns improve and extend earlier work on DSL design patterns. They also discuss domain analysis tools and language development systems which may help to speed up DSL development.

In [17] is presented a partial requirements analysis for DSLs in general, focusing on relevant stakeholders, the system boundary (i.e., where DSLs end and general purpose languages start), and a core set of requirements which are relevant for any DSL. They then discuss open questions, focusing particularly upon requirements refinement, in which more specific domain information needs to be used. Their discussion is intended to be generic: they do not distinguish between domain-specific modeling and programming languages (except where noted). They therefore refer to descriptions as the construct produced by using a DSL. Specific instances of descriptions may be models or programmes.

A study of the literature available on the topic of DSLs as used for the construction and maintenance of software systems is presented in [19]. The authors list a selection of 75 key publications in the area, and provide a summary for each of the papers. Moreover, they discuss terminology, risks and benefits, example domain-specific languages, design methodologies, and implementation techniques.

Numerous works presenting DSL exist: ATL (ATLAS Transformation Language) [21], a QVT-like model transformation language [22] and its execution environment which is based on the Eclipse framework; KM3 (Kernel MetaMetaModel) [23] which is a DSL for describing metamodels; etc.

With regard to DSLs for Software Measurement, Guerra et al. [24] present a framework for the creation of domain specific visual languages (DSVL). In this work a language called SLAMMER was developed as a case study. This language is part of the suite of model management tools that Guerra et al. have defined using graph grammars and graph transformations, in which the evaluation and measurement of software artefacts is an essential element. The goal is to facilitate the task of defining measurements and redesigns for any DSVL.

The Software Metrics Meta-Model [25] developed by the OMG also exists. The Software Metrics Meta-Model, promotes a common interchange format which allows interoperability between existing modernization tools, services and their respective models. This common interchange format can be applied equally

well to development and maintenance tools, services and models. In spite of the existence of this Metamodel, we have opted to define our own language owing to the fact that the Software Measurement Ontology [26] exists. This ontology permits us to establish and clarify the elements (concepts and relationships) involved in the software measurement domain. We have, therefore, based the definition of SMML on this ontology. We have verified that the use of this ontology provides important advantages, particularly given the importance of the solid conceptual base that the problem domain (ontology) provides with which to be able to tackle the solution domain (metamodel). The ontologies are, moreover, potentially useful when developing DSLs during the analysis phase in which knowledge capture and knowledge representation are the key elements [27].

3. Software Measurement Framework

The Software Measurement Framework (SMF) (for greater detail see [14]) permits us to measure any type of software entity. In this framework, any software entity in any domain can be measured with a common Software Measurement metamodel and QVT transformations. SMF has three fundamental elements: conceptual architecture, technological aspects and method. These elements have all been adapted to the MDE paradigm and to MDA technology, taking advantage of their benefits within the field of software measurement. The Software Measurement Framework (SMF) is the evolution of the FMESP [28], but is adapted to the MDE paradigm and uses MDA technology.

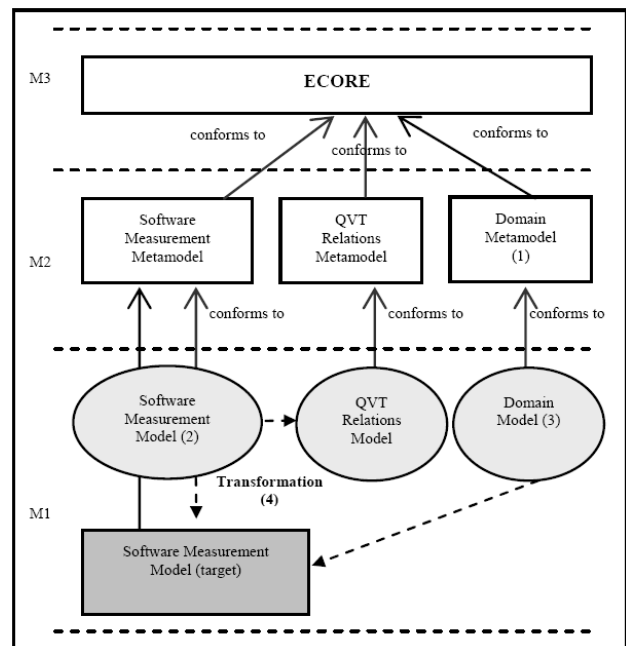


Figure 1. Elements of the SMF.

In Figure 1 the necessary elements for the adaptation of FMESP to MDA are presented according to MOF levels.

4. Software Measurement Modeling Language (SMML)

SMML is a language which permits software measurement models to be built in a simple and intuitive manner. The SMML de-

velopment requires both domain knowledge and language development expertise [16].

Feilkas [29] cites the tasks which must be carried out to make a DSL usable: Definition of an abstract syntax, Definition of a concrete syntax and Definition of semantics. The following subsection describes how these stages have been used to develop the Software Measurement Modeling Language (SMML) [30].

4.1 Definition of an abstract syntax (Domain definition metamodel)

One of the defining entities of a DSL is a Domain Definition MetaModel (DDMM) [30]. This introduces the basic entities of the domain and their relationships. This base ontology plays a central role in the definition of the DSL. Such a DDMM plays the role of the abstract syntax for a DSL.

In order to develop SMML, a Domain Definition Metamodel is therefore necessary. The Software Measurement Metamodel (SMM) exists, which is derived from the Software Measurement Ontology (SMO). This metamodel is the Domain Definition Metamodel used to define the abstract syntax of SMML.

The Software Measurement Metamodel includes the packages which are alignments with the sub-ontologies of SMO (Basic, Characterization and Objectives, Measures Software, Measurement Approaches and Measurement Action). However, for the development of the Language, all the packages are of interest, with the exception of Measurement Action. This has been excluded as it contains the elements which are relative to measurement but not to the problem domain. Figure 2 shows the structure of the packages upon which the SMML language is based.

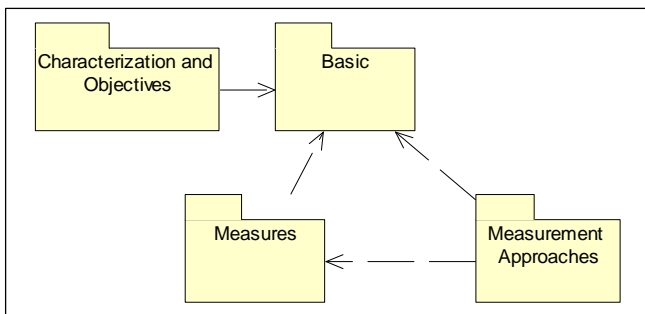


Figure 2. Structure of the packages in the Software Measurement Metamodel.

As shown in Figure 2, the metamodel is made up of a basic package which represents the general characteristics of the basic constructors of the measurement models, and three other packages (Characterization and Objectives, Measurement Approaches and Measures), in accordance with the three sub-ontologies of the SMO. The conceptualization established in the Software Measurement Ontology has been taken into account in the construction of this metamodel, but the specific constructors have been added from the point of view of implementation.





All of the elements identified in the ontology (Measure, Information need, Measurable concept, etc.) are potential elements of the Software Measurement Metamodel on which the SMML

language is based. On the other hand, the relationships which exist in the ontology do not correspond with the relationships which are necessary for the language. All of the Measurement Metamodel packages maintain the original definition of [12] with the exception of the basic package, which has had to be adapted to represent the measurement relationships in SMML.

In [13] is given a detailed description of the relationships of the Software Measurement Metamodel which correspond with the relationships in the SMO ontology. As [13] shows, all the types of relationships which are identified in the ontology, and which have been defined for the metamodel, have been studied. The elements involved (a source and a target) are indicated for each relationship. In total, 4 types of Measurement Associations have been identified: association, nonnavigable association, aggregation and dependency. These relationships have been defined in the Basic package.

In the following table a selection of relationships in the “software measurement characterization and objectives” package are shown. Note that for each relationships in the SMO we have related a new Measurement Association (see Table 1).

Table 1. A selection of the SMML elements and icons

Relationships	Description	Source
Includes 	An <i>entity class</i> may include several other <i>entity classes</i> . An <i>entity class</i> may be included in several other <i>entity classes</i> ,	UML Aggregation
Defined for 	A <i>quality model</i> is defined for a certain <i>entity class</i> . An <i>entity class</i> may have several <i>quality models</i> associated	UML Dependency
Relates 	A <i>Measurable concept</i> relates one or more <i>attributes</i> . An <i>Attribute</i> is related with one or more <i>measurable concepts</i> .	UML Association
Has 	An <i>entity class</i> has one or more <i>attributes</i> . An <i>attribute</i> can only belong to one <i>entity class</i> .	UML nonnavigable Association

We shall now describe the packages of which the Software Measurement Metamodel is made up (for greater detail see [13]):

- **Basic Package:** this basic package has been defined in order to identify and to establish the general features of the constructor necessary to define measurement model. With regard to the Software Measurement Metamodel defined in [12], 4 types of *Measurement Association* have been added: association, non-navigable association, aggregation and dependency. Figure 3 shows the UML diagram which displays the structure of this package.

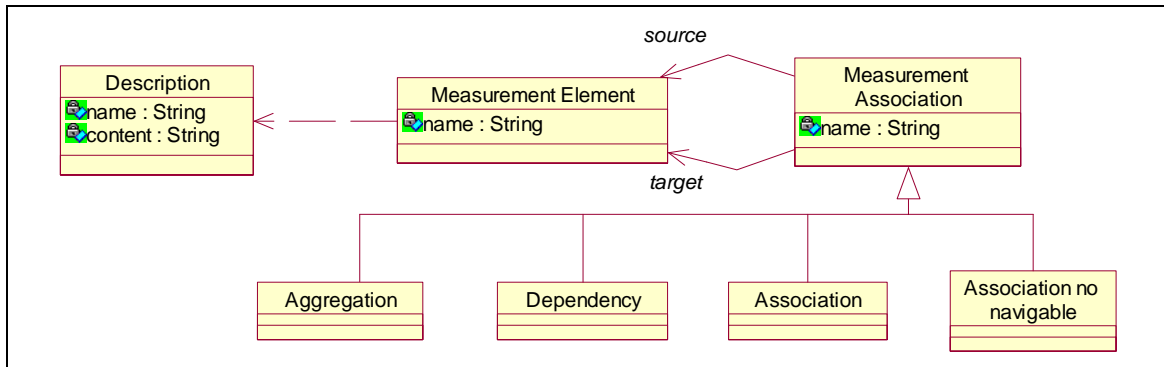


Figure 3. Basic package.

As can be observed in Figure 3, the general element from which measurement models are constructed is the “Measurement Element” constructor, and the general element from which the relationships of the models are constructed is the “Measurement Association” constructor. A measurement element has a name and can be described through elements of the “Description” type, which give additional information about the measurement elements, and this facilitates a better understanding of the measurement models developed. The measurement element is used as a starting point from which to specialize the measure’s fundamental constructors, obtained from the Software Measurement Ontology concepts. A Measurement Element relates two measurement elements, a source element and a target element. The Measurement Association is used to specialize the relationship constructors defined for the metamodel: Association, Nonnavigable association, Aggregation and Dependency.

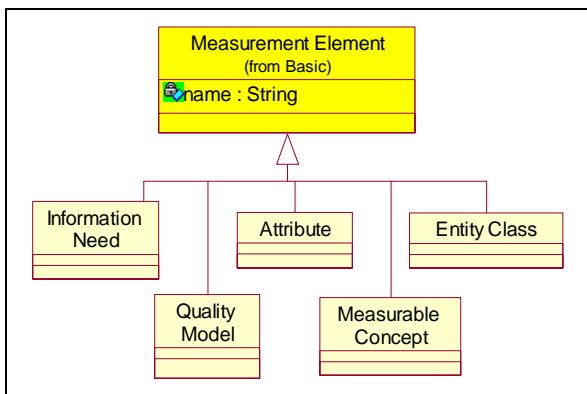


Figure 4. Characterization and objectives Package.

- **Characterization and objectives Package:** this package includes the constructors required to establish the scope and objectives of the software measurement process. Figure 4 shows the UML diagram which displays the structure of this package.
- **Software Measures Package:** this package includes the constructors needed to establish and to clarify the key elements in the definition of a software measure. Figure 5 shows the UML diagram which displays the structure of this package.

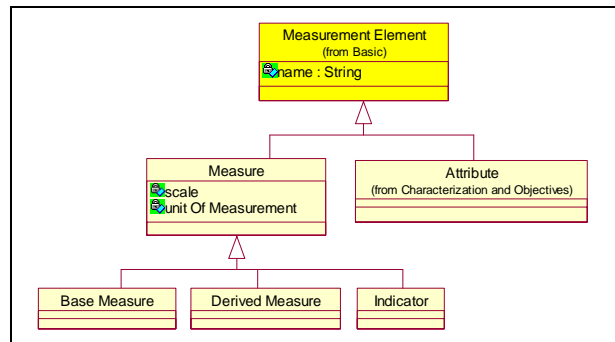


Figure 5. Software Measures Package.

- **Measurement Approaches Package:** this package includes the constructors needed to generalize the different ‘approaches’ used by the three kinds of measures to obtain their respective measurement results. Figure 6 shows the UML diagram which displays the structure of this package.

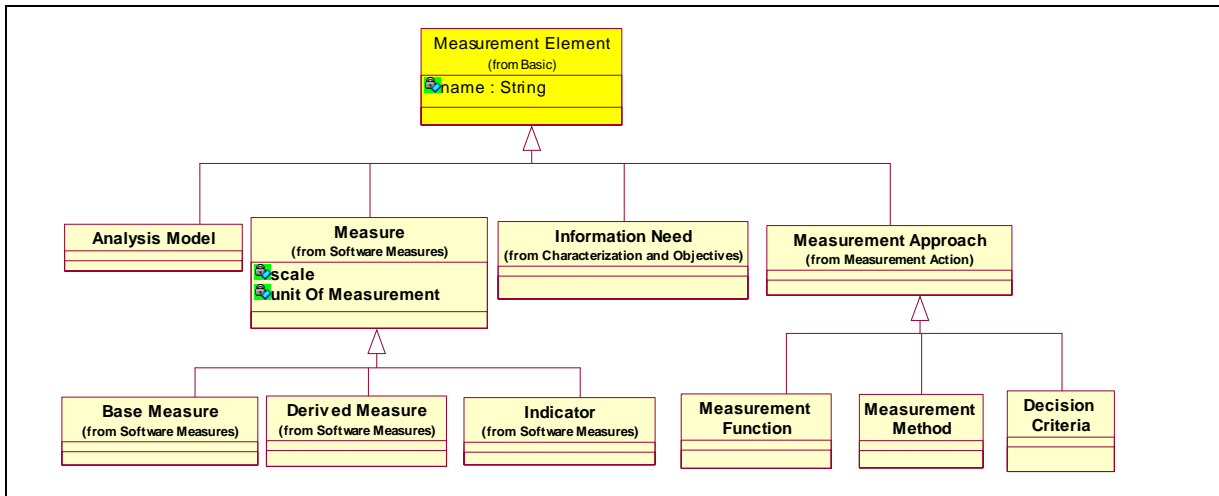


Figure 6. Measurement Approaches Package.

4.2 Definition of a concrete syntax


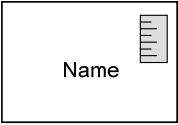

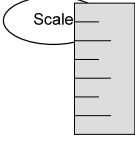
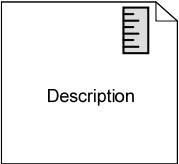
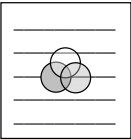

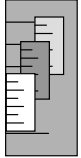
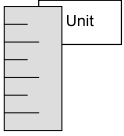
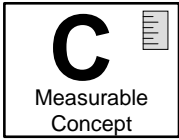
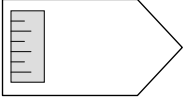

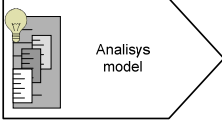
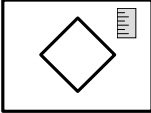
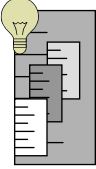
In order to make the language usable, a concrete syntax must be defined. All of the elements are defined in the basic package (see Figure 3).

Each of these elements of the language must be associated with a graphical icon which represents the element of the abstract model. Each language element and relationship has been associated with a representative icon in the SMML. Icons which are

familiar to software engineers have been used in order to facilitate its use. For example, the Description element is very similar to the UML note element, the difference being that the former includes a ruler icon while the latter does not (as a symbol of measurement) in its top right-hand corner. In a similar manner, the Entity element is taken from the Entity Class in an E/R Diagram.

Table 2 shows a selection of the language elements. For further information, see [13]:

Table 2. A selection of the SMML elements and icons

Information need	Entity	Base Measure	Scale	Description
 Information need	 Name			
Quality Model	Attribute	Derived Measure	Unit	Measurable Concept
 Quality Model	 Attribute			
Measurement Method	Measurement Function	Analysis model	Decision Criteria	Indicator
				

4.3 Definition of semantics

The most important aspect of language specification is possibly the definition of its semantics. An informal description of the language must be given in a natural language which describes its domain. The semantics of the language have been defined by using OCL constraints on the metamodel. These constraints define the cardinality and the elements involved in the associations. These constraints are considered too as being part of the abstract syntax because they are part of the metamodel. An example of OCL Constraints relating to Measures is shown in Table 3:

Table 3. A selection of SMML OCL Constraints.

OCL Constraint
Element: Nonnavigable Association self.source.oclIsTypeOf(<i>EntityClass</i>) and self.target.oclIsTypeOf(<i>Attribute</i>)
Element: Association self.source.oclIsTypeOf(<i>MeasurableConcept</i>) and self.target.oclIsTypeOf(<i>Attribute</i>) or self.source.oclIsTypeOf(<i>DerivedMeasure</i>) and self.target.oclIsTypeOf(<i>MeasurementFunction</i>) or self.source.oclIsTypeOf(<i>BaseMeasure</i>) and self.target.oclIsTypeOf(<i>MeasurementMethod</i>) or self.source.oclIsTypeOf(<i>Indicator</i>) and self.target.oclIsTypeOf(<i>AnalysisModel</i>)
Element: Agregation self.source.oclIsTypeOf(<i>EntityClass</i>) and self.target.oclIsTypeOf(<i>EntityClass</i>)
Element: Dependency (self.source.oclIsTypeOf(<i>QualityModel</i>) and self.target.oclIsTypeOf(<i>EntityClass</i>)) or (self.source.oclIsTypeOf(<i>QualityModel</i>) and self.target.oclIsTypeOf(<i>MeasurableConcept</i>)) or (self.source.oclIsTypeOf(<i>MeasurableConcept</i>) and self.target.oclIsTypeOf(<i>InformationNeed</i>)) or (self.source.oclIsTypeOf(<i>AnalysisModel</i>) and self.target.oclIsTypeOf(<i>DecisionCriteria</i>)) or (self.source.oclIsTypeOf(<i>Indicator</i>) and self.target.oclIsTypeOf(<i>InformationNeed</i>)) or (self.source.oclIsTypeOf(<i>Measure</i>) and self.target.oclIsTypeOf(<i>Attribute</i>))

As can be observed, the preceding table (Table 3) contains the OCL constraints which verify whether the Measurement Elements involved in each Measurement Association (source and target) are correct.

5. Case Study

To illustrate the benefits of the SMML, consider the following two case studies: the development and maintenance of database applications in a software company and the definition of a Data Quality Model for Web Portals.

The first case of study is detailed in [31]. This paper presents the results and lessons learned in the application of the Framework for the Modeling and Measurement of Software Processes (FMESP) [28] in a software company dedicated to the development and maintenance of software for information systems.

All the information concerning the problem is defined in each Software Measurement Package: Characterization and Objectives, Software Measures and Measurement Approaches. This case will

only show the modeling of the Characterization and Objectives package.

In this example, we wish to illustrate how a measurement model would be represented with SMML. Figure 7 shows all the information that is needed to represent the Characterization and Objectives Instance. The Measurement Elements used are: Information Need, Quality Model, Measurable Concept, and Attribute. This model has been defined by using diagrams of UML objects.

We shall, furthermore, present how the same example would be defined with SMML (Figure 8).

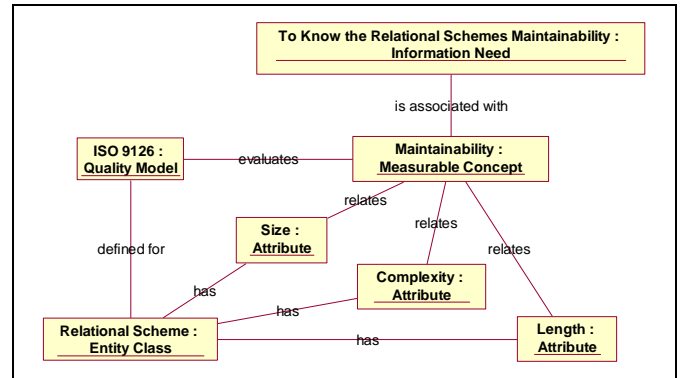


Figure 7. Characterization and Objectives Instance with UML.

As will be observed from the following figure, the representation is easier and more intuitive with the SMML language. Moreover, during the measurement model definition, no issues were found in the constructors metamodel, and no lacks were detected in the Language.

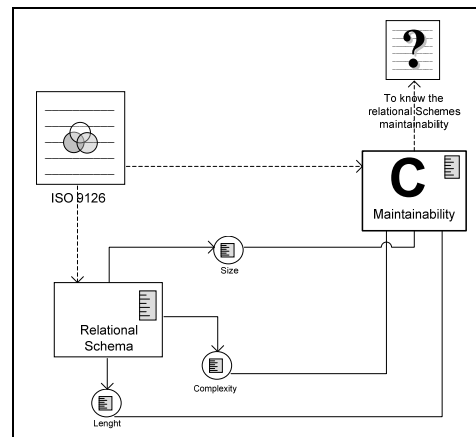


Figure 8. Characterization and Objectives Instance with SMML.

The second case study is shown in [32]. This paper shows how the SMO can be instantiated to define a Data Quality Model for Web Portals, and can also be used to define a DSL for measuring software entities.

Figure 9 shows all the information that is needed to represent the Measurement Model of PDQM. The Measurement Elements used are: *Information Need, Quality Model, Measurable Concept, Attribute, Base Measure, Derived Measure, Indicator, Measurement Method, measurement Function* and *Analysis Model*.

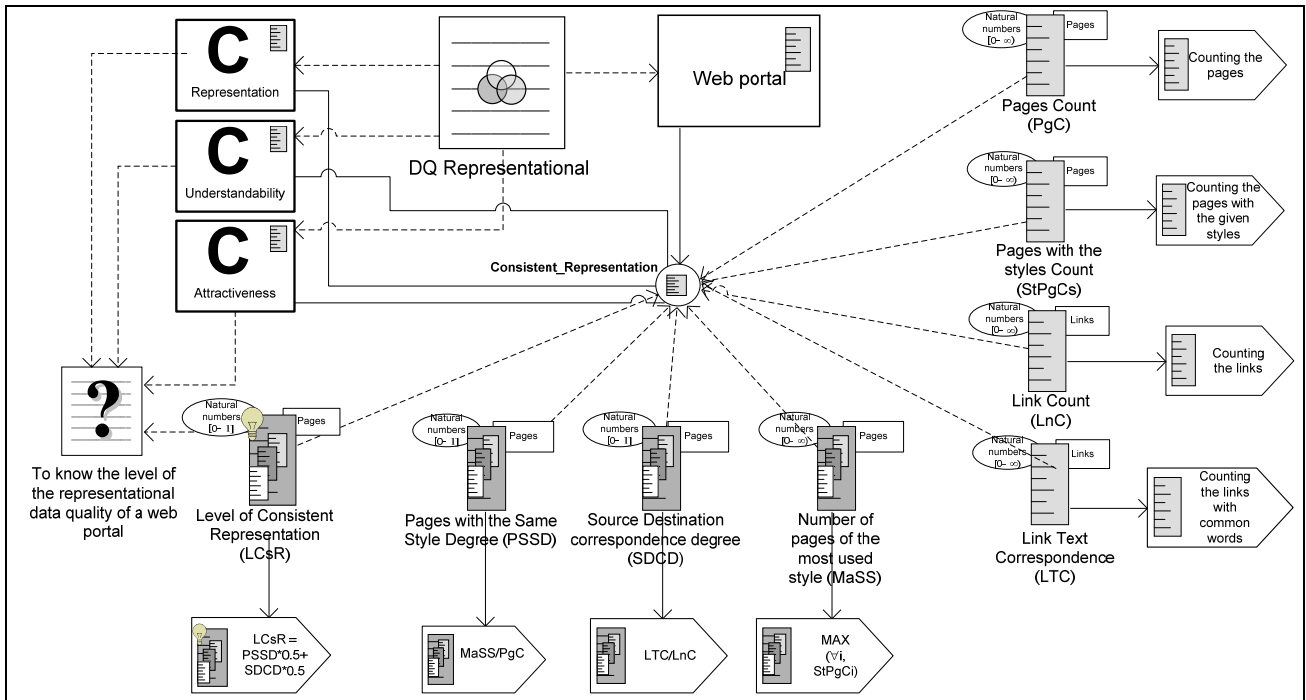


Figure 9. Measurement Model of PDQM represented with SMML.

In this case study, in spite of having to define numerous Measurement Elements, the representation continues to be easy and intuitive. What is more, it is easier to identify Measurement Elements by using this model than by using another General Purpose Language such as UML

With regard to expected requirements [17], we shall now show the requirements which are valid in our Language:

- **Conform:** the language constructs correspond to important domain concepts.
- **Orthogonal:** Each language construct is used to represent exactly one distinct concept (Attribute, Base Measure, etc.) in the domain.
- **Supportable:** The SMML language is supported by tools such as MS/DSL Tools or GMF [33].
- **Simple:** the DSL is simple in order to express the domain concepts and to support its users.
- **Usable:** DSL constructs are expressive and easy to understand.

6. Conclusions and Future Work

SMML permits software measurement models to be defined in a manner which is easy and intuitive for the user. The set of icons which form a part of the language have been selected in order for them to be as familiar as possible to Software engineers. These engineers will thus be able to use the language to define measurement models with ease. The use of general purpose languages to define domain measurement models is thus avoided. Until this moment, no graphic representation permitting a better representation of the model was available.

SSML is a complete language, with a clear syntactic and semantic definition and a solid ontological base. It, moreover, fulfils the following requirements of a DSL: it is usable, it conforms, and it is orthogonal, supportable and simple.

SMML allows users to represent measurement models in various domains.

This language plays a fundamental role in SMF [14] as it allows users to define the measurement models which are the input for the software measurement process. The visual representation of the measurement models mean that SMF is a more usable and intuitive framework for the user. In other words, it makes the measurement process more comfortable.

Among related future works, one important work is that of the extension of SMMM with the Measurement Approach package hierarchy included in the Software Metrics Meta-Model [25].

We shall, moreover, test the usability of the language through a series of experiments based on the ISO 9126 standard. Our study will focus on usability and maintainability. Our idea is to select a group of modeling experts and to test the usability of this new language on them in order to define measurement models.

Finally, we shall apply SMF to real complex environments in order to obtain further refinements and validation.

Acknowledgments

This work has been partially financed by the following projects: INGENIO (Junta de Comunidades de Castilla-La Mancha and Consejería de Educación y Ciencia. PAC08-0154-9262) and ES-FINGE (Ministerio de Educación y Ciencia, TIN2006-15175-C05-05).

References

- [1] N. Fenton and S. L. Pfleeger, *Software Metrics: A Rigorous & Practical Approach*, Second Edition: PWS Publishing Company, 1997.
- [2] S. G. MacDonell, M. J. Shepperd, and P. J. Sallis, "Metrics for Database Systems: An Empirical Study," in *Proceedings of the 4th International Symposium on Software Metrics*. IEEE Computer Society, Albuquerque, 1997.

- [3] L. C. Briand, S. Morasca, and V. R. Basili, "An Operational Process for Goal-Driven Definition of Measures," *IEEE Trans. Softw. Eng.*, vol. 28, pp. 1106-1125, 2002.
- [4] D. Champeaux, *Object-oriented Development Process and Metrics*: Prentice-Hall, 1997.
- [5] M. B. Chrissis, M. Konrad, and S. Shrum, *CMMI: guidelines for process integration and product improvement*, vol. 1: Pearson, 2006.
- [6] ISO/IEC, "Software and Systems Engineering - Guidelines for the application of ISO/IEC 9001:2000 to Computer Software," International Standards Organization, 2004.
- [7] J. McGarry, D. Card, C. Jones, B. Layman, E. Clark, J. Dean, and F. Hall, *Practical Software Measurement. Objective Information for Decision Makers*: Addison-Wesley, 2002.
- [8] ISO/IEC, "ISO 15939: Software Engineering - Software Measurement Process.," 2002.
- [9] "DSM (Domain-Specific Modeling) Forum Main Page", 2007, <http://www.dsmforum.org/>.
- [10] M. Völter, "A categorization of DSLs", 2006, <http://www.voelterblog.blogspot.com/2006/10/categorization-of-dsls.html>.
- [11] N. A. Allen, C. A. Shaffer, and L. T. Watson, "Building modeling tools that support verification, validation, and testing for the domain expert," in *Proceedings of the 37th conference on Winter simulation*, pp. 419-426, Orlando, Florida, 2005.
- [12] F. García, M. Serrano, J. Cruz-Lemus, F. Ruiz, and M. Piattini, "Managing Software Process Measurement: A Metamodel-Based Approach," *Information Sciences*, vol. 177, pp. 2570-2586, 2007.
- [13] B. Mora, F. Ruiz, F. García, and M. Piattini, "SMML: Software Measurement Modeling Language," Department of Computer Science. University of Castilla - La Mancha Technical Report UCLM-TSI-003, 2008.
- [14] B. Mora, F. García, F. Ruiz, M. Piattini, A. Boronat, A. Gómez, J. Á. Carsí, and I. Ramos, "Software Measurement by using QVT Transformation in an MDA context," in *Proceedings of the 10th International Conference on Enterprise Information Systems - ICEIS 2008*, vol. DISI, pp. 117-124, Barcelona (Spain), 2008.
- [15] S. Cook and D. S. Frankel, "Domain-Specific Modeling and Model Driven Architecture," *MDA Journal*, 2004.
- [16] M. Mernik, J. Heering, and A. M. Sloane, "When and how to develop domain-specific languages," *ACM Computing Surveys (CSUR)*, vol. Volume 37, pp. 316-344, 2005.
- [17] D. S. Kolovos, R. F. Paige, T. Kelly, and F. A. C. Polack, "Requirements for Domain-Specific Languages," in *Proceedings of the First ECOOP Workshop on Domain-Specific Program Development (ECOOP'06)*, Nantes, France, 2006.
- [18] V. Pelechano, M. Albert, M. Javier, and C. Carlos, "Building Tools for Model Driven Development comparing Microsoft DSL Tools and Eclipse Modeling Plug-ins," in *Proceedings of the Desarrollo de Software Dirigido por Modelos - DSDM'06 (Junto a JISBD'06)*, Sitges (Barcelona) España, 2006.
- [19] A. v. Deursen, P. Klint, and J. Visser, "Domain-Specific Languages: An Annotated Bibliography," *SIGPLAN Notices*, vol. 35, pp. 26-36, 2000.
- [20] T. Özgür, "Comparison of Microsoft DSL Tools and Eclipse Modeling Frameworks for Domain-Specific Modeling In the context of the Model Driven Development," in *School of Engineering. Ronneby, Sweden: Blekinge Institute of Technology*, 2007, pp. 56.
- [21] F. Jouault, F. Allilaire, J. Bézivin, I. Kurtev, and P. Valduriez, "ATL: a QVT-like Transformation Language," 2006.
- [22] OMG, "QVT Standard Specification," 2005.
- [23] F. Jouault and J. Bézivin, "KM3: a DSL for Metamodel Specification," 2006.
- [24] E. Guerra, J. d. Lara, and P. Díaz, "Visual specification of measurements and redesigns for domain specific visual languages," *Journal of Visual Languages and Computing*, 2008.
- [25] OMG, "Architecture-Driven Modernization (ADM): Software Metrics Meta-Model (SMM). OMG Document: dmtf/2007-08-01," Object Management Group 01-08-2007 2007.
- [26] F. García, M. F. Bertoa, C. Calero, A. Vallecillo, F. Ruíz, M. Piattini, and M. Genero, "Towards a consistent terminology for software measurement," *Information and Software Technology* vol. 48 (8), pp. 631-644 2006.
- [27] M. Denny, "Ontology building: A survey of editing tools", 2003, <http://www.xml.com/pub/a/2002/11/06/ontologies.html>.
- [28] F. García, M. Piattini, F. Ruiz, G. Canfora, and C. A. Visaggio, "FMESP: Framework for the modeling and evaluation of software processes," *Journal of Systems Architecture - Agile Methodologies for Software Production*, vol. 52, pp. 627-639, 2006.
- [29] M. Feilkas, "How to represent Models, Languages and Transformations?," in *Proceedings of the Proceedings of th 6th OOPSLA Workshop on Domain-Specific Modeling (DSM'06)*, pp. 204-213, 2006.
- [30] I. Kurtev, J. Bézivin, F. Jouault, and P. Valduriez, "Model-based DSL Frameworks," 2006.
- [31] G. Canfora, F. García, F. Ruiz, and C. A. Visaggio, "Applying a framework for the improvement of software process maturity," *Software: Practice & Experience*, vol. 36, pp. 283-304, 2006.
- [32] F. García, F. Ruiz, C. Calero, M. F. Bertoa, A. Vallecillo, B. Mora, and M. Piattini, "On the Effective Use of Ontologies in Software Measurement," *The Knowledge Engineering Review (in press)*, vol. 0, pp. 1-24, 2008.
- [33] Eclipse, "Eclipse Graphical Modeling Framework (GMF) Main Page", 2007, <http://www.eclipse.org/gmf/>.