The Knowledge Engineering Review, Vol. 24:1, 23–40. © 2009, Cambridge University Press
doi:10.1017/S0269888909000125 Printed in the United Kingdom

Effective use of ontologies in software measurement

FÉLIX GARCÍA[1], FRANCISCO RUIZ[1], CORAL CALERO[1], MANUEL F. BERTOA[2], ANTONIO VALLECILLO[2], BEATRIZ MORA[1] and MARIO PIATTINI[1]

[1]*Alarcos Research Group—Institute of Information Technologies & Systems, Department of Information Technologies & Systems—Escuela Superior de Informática, University of Castilla-La Mancha, Spain;*
e-mail: Felix.Garcia@uclm.es, Francisco.RuizG@uclm.es, Coral.Calero@uclm.es, Beatriz.Mora@uclm.es, Mario.Piattini@uclm.es
[2]*Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga. 29071 Málaga, España, Spain;*
e-mail: bertoa@lcc.uma.es, av@lcc.uma.es

Abstract

Ontologies are frequently used in the context of software and technology engineering. These can be grouped into two main categories, depending on whether they are used to describe the knowledge of a domain (*domain ontologies*) or whether they are used as *software artifacts* in software development processes. This paper presents some experiences and lessons learnt from the effective use of an ontology for Software Measurement, called software measurement ontology (SMO). The SMO was developed some years ago as a result of a thorough analysis of the software measurement domain. Its use as a domain ontology is presented first, a description of how the SMO can serve as a conceptual basis for comparing international standards related to software measurement. Second, the paper describes several examples of the applications of SMO as a software artifact. In particular, we show how the SMO can be instantiated to define a data quality model for Web portals, and also how it can be used to define a Domain-Specific Language (DSL) for measuring software entities. These examples show the significant role that ontologies can play as software artifacts in the realm of model-driven engineering and domain-specific modeling.

1 Introduction

In recent years, we have witnessed the evolution of software production from a near-craft activity to its formalization as a new engineering discipline. However, software engineering, when compared with other well-established engineering fields, suffers from some of the problems of any relatively young discipline. This is especially true for software measurement, one of the software engineering fields that is gradually acquiring more relevance and attracting increased interest, but which is still evolving and has not yet fully matured.

The difficulty of measuring software lies in its intangible nature and on the fact that its production costs depend on both the engineering processes and the product design. In addition, software measurement is still in the phase in which terminology, principles, and methods are being defined, consolidated, and agreed upon. In particular, there is no consensus yet on the concepts and terminology used in this field.

The first step then is to try to formalize and agree on the vocabulary and concepts used in the measurement of software products and processes. Terms such as 'measure', 'measurement', or 'attribute' do not have a uniform definition accepted by all software measurement researchers and practitioners. Other terms such as 'metric', used to refer to the measuring of a product property, are used only in the context of software measurement, and differ from the terms commonly used in other scientific areas.

https://doi.org/10.1017/S0269888909000125 Published online by Cambridge University Press

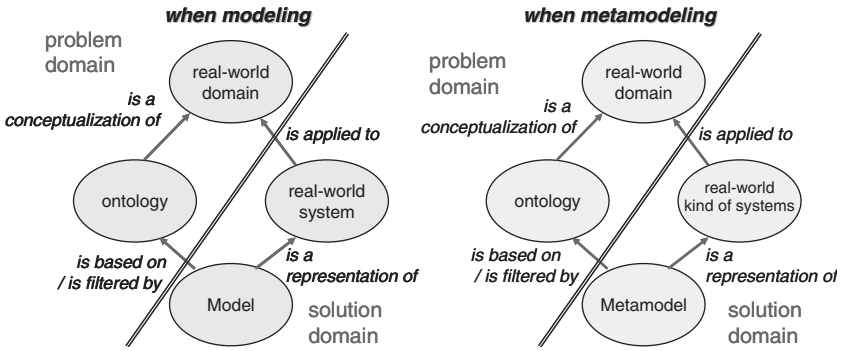


Figure 1 Relationships between the ontology, model, and metamodel concepts

One way to achieve this goal is by developing an ontology that not only provides a formal description of the entities and their properties, but also defines a shared terminology for the objects of interest in the domain—along with definitions for its terms, processes, and goals. Such an ontology should be able to capture the collective knowledge of the domain (in this case, software measurement) in a generic and formal manner, so that it can be reused and shared by different stakeholders (software engineers, providers, customers, etc.).

These facts motivated us to develop a software measurement ontology (called SMO), which is presented in Section 2. However, we discovered that merely defining an ontology may not be enough. Examples and guidelines on how to use it are also needed to illustrate its possible uses and to show its potential applications. In addition, these examples can serve as a proof of concept for the proposal, and can also help to evaluate the usefulness of the ontology in a variety of situations and use cases.

In this paper, we present a number of case studies where the SMO has been successfully used: (1) as a conceptual basis for comparing the terminology used in different proposals (including the related IEEE and ISO/IEC international standards), with the aim of achieving their harmonization; (2) to help in the process of defining a Data Quality Model (DQM) for Web portals; and (3) to develop a visual Domain-Specific Language (DSL) for software measurement.

In all these case studies, three inter-related but different basic concepts are normally used: ontology, metamodel, and the conceptual model. In the Software Engineering field there is some confusion regarding these three concepts, mostly because of the fact that they are normally represented using the same notations (diagrams), and therefore they are mistakenly considered as synonyms (Assmann *et al.*, 2006). Here, we follow the ontology definition provided by Gruber (1993): 'an explicit specification of a conceptualization', but adding that such a conceptualization belongs to the problem domain, whereas metamodels and models, which can be found in M2 and M1 levels of the 'Meta-Object Facility' (MOF) conceptual architecture (OMG, 2006), respectively, belong to the solution domain. Figure 1 summarizes the relationships between these three concepts, both for the classic modeling activity and the new metamodeling activity, which is typical in the MDE context.

In this work, we follow the Ruiz and Hilera (2006) taxonomy regarding the different types of usage of ontologies in the Software Engineering and Technology field:

- *Domain Ontologies*, which describe the knowledge of the Software Engineering and Software Technology domains.
- Ontologies as *Software Artifacts*, which are used as artifacts of several types in software processes at development time or at run time.

Using this taxonomy, the three case studies presented in this work can be classified as follows:

1. Domain ontologies/software engineering/specific (sub-domain)/software engineering management: The SMO is proposed as a vehicle to achieve a consensus on the terminology used in the software measurement field, which is one of the key aspects of Process Management within the Software Engineering discipline.

https://doi.org/10.1017/S0269888909000125 Published online by Cambridge University Press

2. Ontologies as software artifacts/at development time/for other processes/management processes: The SMO is used as an artifact to define a special kind of model, namely, a model for Web portal data quality, by instantiation. The instantiated models are used in the 'quality assurance' management process at software development time. This case study reflects the 'When modeling' scenario (left part of Figure 1).
3. Ontologies as software artifacts/at development time/for other processes/management processes: This last case study reflects the 'When metamodeling' scenario (right part of Figure 1). In this case, the SMO is used as an artifact for designing a metamodel and an associated DSL.

The structure of this paper is as follows: After this introduction, Section 2 summarizes the SMO. In Section 3, the three aforementioned case studies in which the SMO has been successfully applied are presented. Finally, some conclusions, lessons learnt, and future lines of work are outlined in Section 4.

2 The software measurement ontology

The SMO was originally developed to facilitate harmonization efforts in software measurement terminology (see Section 3.1). The SMO was initially proposed to address the lack of consensus on Spanish software measurement terms (García *et al.*, 2004). Once the Spanish ontology was defined, it evolved to cope with the English terms too. Finding the correct translation for each Spanish term became a rather difficult task and was done by comparing the existing standards and proposals again, selecting the most appropriate terms in each case.

After analyzing several formalisms for representing ontologies, Representation Formalism for Software Engineering Ontologies (REFSENO) (Tautz & Von Wangenheim, 1998) was chosen to describe SMO. REFSENO provides constructs to define concepts (each concept represents a class of experience items), their attributes, and their relationships. Three tables are used to represent these elements: one with the glossary of concepts, one with the attributes, and one with the relationships. REFSENO also allows the description of similarity-based retrievals, and incorporates integrity rules such as cardinalities and value ranges for the attributes, and assertions and preconditions on the element instances.

REFSENO was used to define SMO for a number of reasons. First, REFSENO was specifically designed for software engineering, and allows several representations for software engineering knowledge—whereas other approaches, for example, Uschold and Gruninger (1996); Gómez-Pérez (1998) and Staab *et al.* (2001), provide representations that are less intuitive for people not familiar with first-order predicate logic. Second, REFSENO has a clear terminology, differentiating between conceptual and context-specific knowledge, thus enabling the management of knowledge coming from different contexts. REFSENO also helps build consistent ontologies because of the use of consistency criteria. Unlike other approaches, REFSENO uses constructs from case-based reasoning (CBR). Finally, REFSENO stores experience in the form of documents, and not as coded knowledge. This results in an important reduction of the learning effort required, something typically associated with knowledge-based systems (Althoff *et al.*, 1999).

We also followed the following steps recommended by REFSENO to define the SMO:

1. Define the concept glossary from the knowledge sources.
2. Define the semantic relationships between the concepts by representing them in the Unified Modeling Language (UML) and create the relationships class tables.
3. Analyze the concepts, that have some kind of relationship in order to identify the commonalities among two or more concepts, and decide if these commonalities are concepts (inserted by modeling reasons) and, if so, include them in the glossary of concepts.
4. Identify the terminal attributes of all the concepts and include them in the UML diagrams; every time a new attribute type is identified, it has to be included in the type table.
5. Complete the attributes concept tables by including the non-terminal attributes.
6. Check the completeness of all the attribute tables.

https://doi.org/10.1017/S0269888909000125 Published online by Cambridge University Press

Figure 2 describes the SMO concepts and relationships represented in UML. As shown in the figure, SMO is organized into four main sub-ontologies: *Software Measurement Characterization and Objectives*, which establishes the context and goals of the measurement; *Software Measures*, which defines the terminology used in the definition of measures; *Measurement Approaches*, which describes the different ways of obtaining the measurement results for the defined measures; and *Measurement*, which contains the concepts related to performing the measurement process. The SMO concepts and their definitions are detailed in Table 1.

Tables 2 and 3 provide, respectively, an excerpt of relationships and attributes tables for the SMO. A complete description of all the tables can be found in Bertoa *et al.* (2006). Additionally, the SMO has been represented using the Web Ontology Language (OWL), and its representation can be found at `http://alarcos.inf-cr.uclm.es/ontologies/smo`.

3 SMO case studies

This section describes three usage scenarios of SMO. They serve as illustrative examples of the possible uses of the ontology. Furthermore, they have helped us to validate the proposal by tackling several problems of a diverse nature related to software measurement.

3.1 Terminology harmonization

As stated in Section 2, the SMO was initially developed to establish a common vocabulary in the Software Measurement field, which facilitates interoperability and communication among stakeholders. For the development of SMO, we analyzed sources from both existing international standards and research proposals that deal with software measurement concepts and terminology—including, among others, the ISO International Vocabulary of Basic and General Terms in Metrology (VIM) and all related ISO and IEEE standards. As a result, the SMO was built with a coherent software measurement terminology that has been agreed upon by consensus, and without contradictions or disparities in the definitions.

The first application of SMO was to provide a thorough comparative analysis of the aforementioned selected sources with the following goals: (a) to locate and identify synonyms, homonyms, gaps, and conflicts; (b) to generalize the different approaches to measuring attributes and (c) to provide a smooth integration of the concepts from the three groups, so that measurement processes can be built using clearly defined measures, while quality models identify the target and goals of the measurement processes. The analysis clearly shows all similarities, discrepancies, shortcomings, and weaknesses in the terminology used in SMO compared with the main standards and proposals. To illustrate the results of the analysis, Table 4 shows an excerpt of the comparative analysis for the terms 'Measure' and 'Information Need'. The table index is on the left column (SMO term). Its second and third columns show, respectively, the source (SMO or standard) where the term appears, and its definition according to that source. Multiple rows for a given term indicate different (normally discrepant) definitions. Synonyms are shown in brackets before term definitions. A complete description of the analysis can be found in García *et al.* (2006).

In general, we found that there is no single standard that embraces the whole area of software measurement: software measurement concepts are (re)defined in most standards and, normally, with conflicting definitions. Without an overall reference framework managing these standards, discrepancies and inconsistencies are commonplace. This fact has been explicitly acknowledged by most standardization bodies and organizations (including ISO/IEC and the IEEE), which have started working on the harmonization of software measurement terms. In particular, ISO has created a working group for the harmonization of Systems Engineering Standards within its Joint Technical Committee 1 (ISO-JTC1). There is also an agreement in place since the year 2002 between the IEEE Computer Society and ISO/SC7-JTC1 Subcommittee 7 (the one in charge of software and systems engineering) to harmonize the concepts and terminology used in their standards, which includes the terminology on measurement.

https://doi.org/10.1017/S0269888909000125 Published online by Cambridge University Press

https://doi.org/10.1017/S0269888909000125 Published online by Cambridge University Press

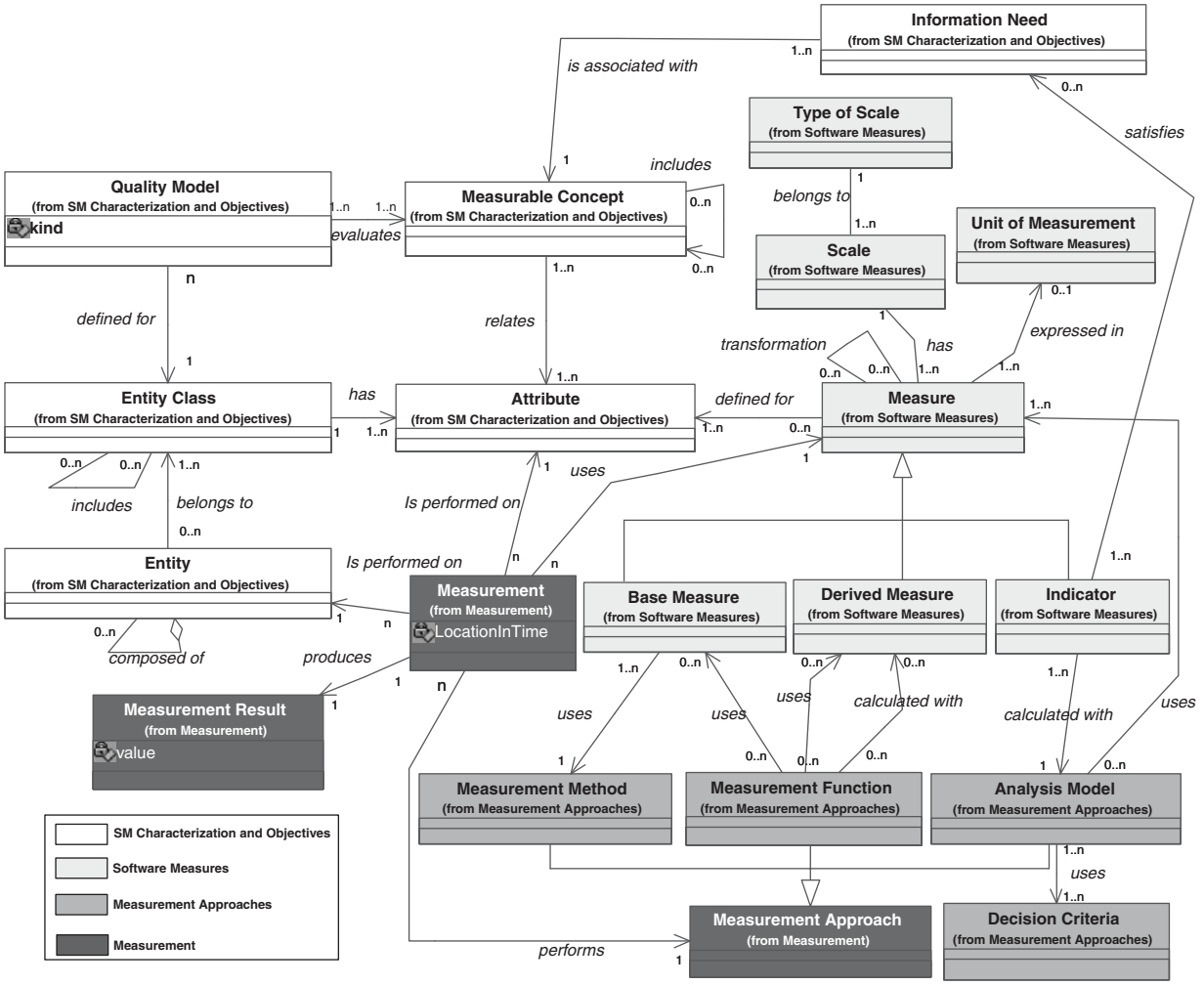


Figure 2 UML representation of SMO concepts, attributes, and relationships

Table 1 Concepts of the SMO

Concept	Superconcept	Definition
Information need	Concept	Insight necessary to manage objectives, goals, risks, and problems
Measurable concept	Concept	Abstract relationship between attributes of entities and information needs
Entity	Concept	Object that is to be characterized by measuring its attributes
Entity class	Concept	The collection of all entities that satisfy a given predicate
Attribute	Concept	A measurable physical or abstract property of an entity that is shared by all the entities of an entity class
Quality model	Concept	The set of measurable concepts and the relationships between them, which provide the basis for specifying quality requirements and evaluating the quality of the entities of a given entity class
Measure	Concept	The defined measurement approach and the measurement scale (a measurement approach is either a measurement method, a measurement function or an analysis model)
Scale	Concept	A set of values with defined properties
Type of scale	Concept	The nature of the relationship between values on the scale
Unit of measurement	Concept	Particular quantity, defined and adopted by convention, with which other quantities of the same kind are compared in order to express their magnitude relative to that quantity
Base measure	Measure	A measure of an attribute that does not depend upon any other measure, and whose measurement approach is a measurement method
Derived measure	Measure	A measure that is derived from other base or derived measures, using a measurement function as measurement approach
Indicator	Measure	A measure that is derived from other measures using an analysis model as measurement approach
Measurement method	Measurement approach	Logical sequence of operations, described generically, used in quantifying an attribute with respect to a specified scale. (A measurement method is the measurement approach that defines a base measure)
Measurement function	Measurement approach	An algorithm or calculation performed to combine two or more base or derived measures. (A measurement function is the measurement approach that defines a derived measure)
Analysis model	Measurement approach	Algorithm or calculation combining one or more measures with associated decision criteria. (An analysis model is the measurement approach that defines an indicator)
Decision criteria	Concept	Thresholds, targets, or patterns used to determine the need for action or further investigation, or to describe the level of confidence in a given result
Measurement approach	Concept	Sequence of operations aimed at determining the value of a measurement result. (A measurement approach is either a measurement method, a measurement function or an analysis model)
Measurement	Concept	A set of operations whose objective is to determine the value of a measurement result, for a given attribute of an entity, using a measurement approach
Measurement result	Concept	The number or category assigned to an attribute of an entity as a result of a measurement

In this respect, the development of an ontology has been shown to be a sound and worthwhile approach to achieving such terminology harmonization, identifying all concepts, providing precise definitions for all the terms, and clarifying the relationships among them. In addition, SMO aims at providing an important communication vehicle to companies when interoperating with others in the area of software measurement, and also tries to serve as a basis for discussion from where the software measurement community can start paving the way to future agreements. What we are completely sure of is that without these agreements, all the standardization and research efforts may be wasted, and the potential benefits that they may bring to all users (software developers, ICT suppliers, tools vendors, etc.), may never materialize.

https://doi.org/10.1017/S0269888909000125 Published online by Cambridge University Press

Table 2 Relationships table for the *Measurement Approaches* sub-ontology

Name	Concepts	Description
Uses	Base measure—measurement method	Every base measure uses one measurement method. Every measurement method defines one or more base measures
Calculated with	Indicator—analysis model	Every indicator is calculated with one analysis model. Every analysis model may define one or more indicators
Calculated with	Derived measure—measurement function	Every derived measure is calculated with one measurement function. Every measurement function may define one or more derived measures
Satisfies	Indicator—information Need	An indicator may satisfy several information needs. Every information need is satisfied by one or more indicators
Uses	Measurement function—base measure	A measurement function may use several base measures. A base measure may be used in several measurement functions
Uses	Measurement function—derived measure	A measurement function may use several derived measures. A derived measure may be used in several measurement functions
Uses	Analysis model—measure	An analysis model uses one or more measures. A measure may be used in several analysis models
Uses	Analysis model—decision criteria	An analysis model uses one or more decision criteria. Every decision criteria is used in one or more analysis models

Table 3 Attributes table for the *Measurement* sub-ontology

Concept	Attribute	Description	Type	Card.
Measurement	Location in time	Time instant where measurement is carried out	Time/date	1
Measurement result	Value	Value which represents the result of the measurement action	Variant	1

3.2 Defining a data quality model for Web portals using SMO

A Web portal is a site that aggregates information from multiple sources on the World Wide Web, and organizes this material in an easy and user-friendly manner (Xiao & Dasgupta, 2005). Over the past decade, the number of organizations that own and maintain Web portals has significantly grown. These companies and organizations have developed portals to complement, substitute, or widen the services they provide to their clients, and the way in which they provide them (Yang *et al*., 2004). This has resulted in many people using the data obtained from Web portals to carry out their work and help them make decisions. Thus, the quality of data collected from these portals should be guaranteed or, at least, evaluated.

To be able to measure the quality of data provided by a Web portal, the first step is to define a quality model that identifies the main information needs, quality characteristics, measurable concepts, attributes, measures, etc. This is precisely where ontologies, and in particular the SMO, can be very useful. The case presented in this section is a DQM for Web portals that focuses on the data consumer's perspective, which was developed using the SMO.

The portal data quality model (PDQM) was defined from scratch because of the lack of other quality models defined for data portals, although some DQMs defined for other contexts were taken into account—for further details see Caro *et al*. (2007 and 2008). The first step was the definition of a theoretical model, named PDQM(t), that contains 33 DQ attributes (Table 5, right column). This theoretical model was then transformed into an operative one by means of the probabilistic approach proposed in Malak *et al*. (2006) which involves Bayesian belief

https://doi.org/10.1017/S0269888909000125 Published online by Cambridge University Press

Table 4 Comparison of some terms in the 'software measures' sub-ontology

Term	Source	Definition
Measure	SMO	The defined measurement approach and the measurement scale. (A measurement approach is either a measurement method, a measurement function or an analysis model)
	14598-1	[Metric] The defined measurement method and the measurement scale
	610.12	[Metric] A quantitative measure of the degree to which a system, component, or process possess a given attribute
	IEEE 1061	[Metric] A function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which software possesses a given attribute that affects its quality
Base measure	SMO	A measure of an attribute that does not depend upon any other measure, and whose measurement approach is a measurement method
	VIM	[Base quantity] One of the quantities that, in a system of quantities, are conventionally accepted as functionally independent of one another
	15939	Measure defined in terms of an attribute and the method for quantifying it. (Note: a base measure is functionally independent of other measures)
	14598-1	[Direct measure] Measure of an attribute that does not depend upon a measure of any other attribute
	IEEE 1061	[Direct metric] A metric that does not depend upon a measure of any other attribute
Derived measure	SMO	A measure that is derived from other base or derived measures, using a measurement function as measurement approach
	VIM	[Derived quantity] Quantity defined, in a system of quantities, as a function of base quantities of that system
	15939	Measure that is defined as a function of two or more values of base measures
	14598-1	[Indirect measure] A measure of an attribute that is derived from measures of one or more other attributes
Indicator	SMO	A measure that is derived from other measures using an analysis model as measurement approach
	15939	An estimate or evaluation of specified attributes derived from a model with respect to defined information needs
	14598-1	A measure that can be used to estimate or predict another measure

Table 5 Data quality attributes of PDQM

Data quality category	Data quality attributes
Intrinsic: Denotes that data has quality in its own right	Accuracy, objectivity, believability, reputation, currency, duplicates, expiration, and traceability
Accessibility: Emphasizes the importance of the role of systems, i.e., the system must be accessible but secure	Accessibility, security, interactivity, availability, customer support, ease of operation, and response time
Contextual: Data quality must be considered in the context of the task in hand	Applicability, completeness, flexibility, novelty, reliability, relevancy, specialization, timeliness, validity, value-added
Representational: The system must present data so that it is interpretable and easy to understand, as well as concisely and consistently represented	Interpretability, understandability, concise representation, consistent representation, amount of data, attractiveness, documentation, and organization

https://doi.org/10.1017/S0269888909000125 Published online by Cambridge University Press

networks (BBN), indicators, measures, and fuzzy logic. As a result, a BBN was built, which represents, in a hierarchical structure, the DQ attributes of PDQM according to the categories shown in Table 5, adapted from the proposal of Wang and Strong (1996)—the most widely known model among those that are currently available within the DQ field, and which is used as a '*de facto*' standard.

To provide inputs for the entry nodes of the BBN, some quality indicators were required. The SMO was also used for this purpose. To illustrate the application of SMO in this case, we describe here the definition of the DQ_Representational part. The corresponding BBN is shown in Figure 3.

To develop this BBN, we started by connecting the quality attributes to the final node, which represents the category we want to measure. Synthetic nodes were added to the network to reduce the number of parents of each node (nodes with more than four parents should generally be avoided in BBNs; the introduction of synthetic nodes is recommended in these cases). Then we defined quantifiable variables for the entry nodes (input nodes) of the network, and finally we established a Node Probability Table (NPT) for each node.

The use of SMO was fundamental for the definition of measures for the quantifiable variables of entry nodes. One indicator was defined for each entry node (upper nodes in Figure 3), on the basis of the aggregation of several base and derived measures. The calculation methods for these measures were automated, so that input figures for the entry nodes of the network could be computed objectively by a tool for any given portal.

Table 6 provides the description of the Model for the DQ_Representational fragment of the Quality Model. It shows the instances of the *Characterization and objectives* sub-ontology of the SMO. Tables 7 and 8 provide the definition of the base and derived measures, respectively.

The indicators required to satisfy the information need of the PDQM measurement model were obtained through the aggregation (using an analysis model) of the appropriate base and derived measures. For example, the 'Level of Consistent Representation (LCsR)' indicator evaluates the extent to which data is always presented in the same format, is compatible with previous data, and is consistent with other sources (i.e., it measures the 'consistent representation' attribute of Web portals). The measures used to obtain this indicator are based on the presentation styles of the portal Web pages (PSSD), and on the correspondence between the text used in the source link and the destination page (SDCD) (see Table 8). The analysis model for the LCsR indicator is shown on the left-hand side of Table 9.

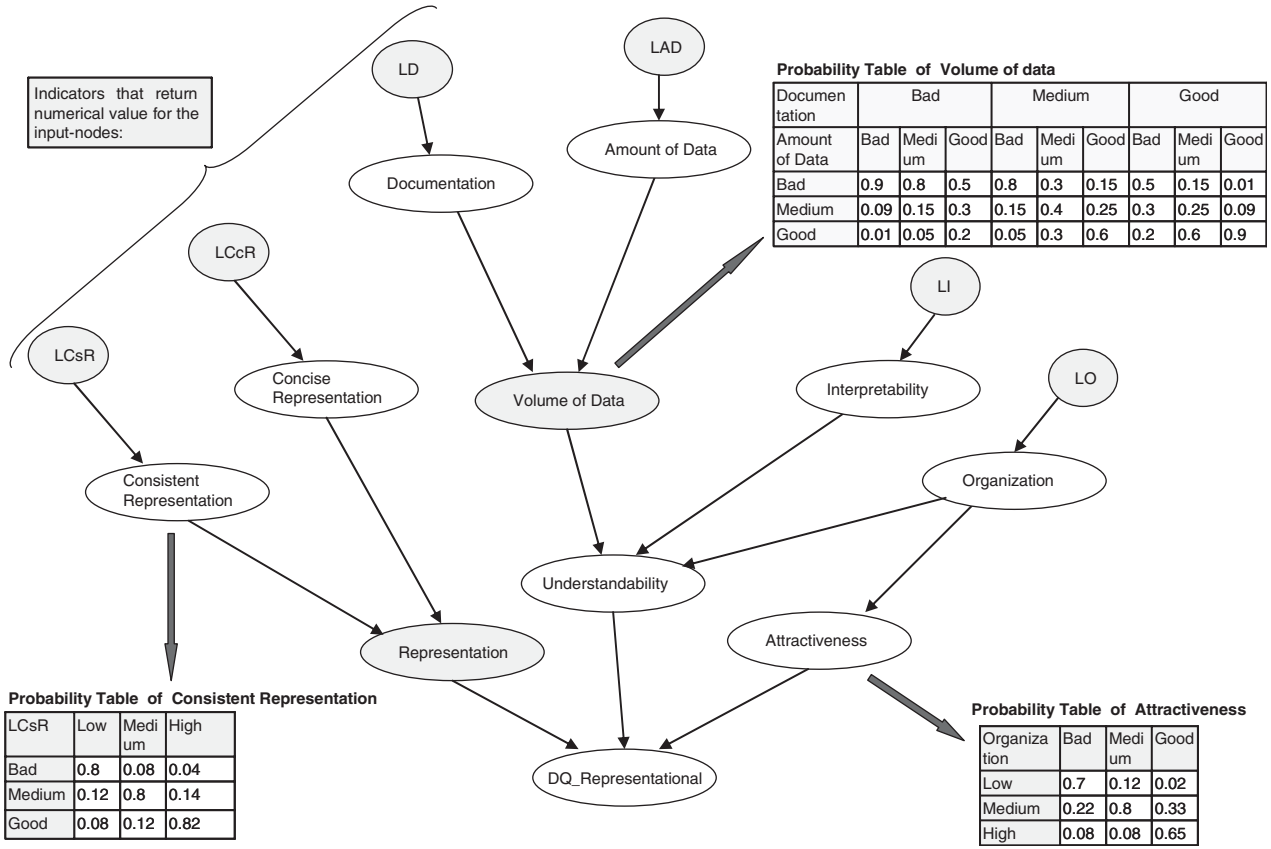
All our indicators (and in particular the one we are showing here, LCsR) are numerical values between 0 and 1, to simplify the definition of probabilities and to normalize their values. These values were later converted into discrete variables using fuzzy logic and membership functions that transform the indicator values into a set of probabilities, each of them corresponding to a label/class. As an example of such membership functions, the right part of Table 9 shows the decision criteria defined for the LCsR indicator, which permits to derive a value (low, medium or high) from the initial value of the indicator. These probability values are known as 'evidences', and are propagated through the network through causal links (applying the corresponding probability tables defined for the intermediate nodes), until the level of representational DQ in the Web portal is finally obtained.

This process has been automated using a tool named PoDQA. The tool asks for the URL of a portal and then applies the defined measures and indicators on its pages and elements. The results are transformed (also by the tool) as valid values for the Bayesian Network input nodes. After this, the network is re-calculated using the new evidences for the entry nodes, and the DQ evaluation level is generated. The PoDQA tool is available at `http://podqa.webportalquality.com`.

The use of SMO for defining the base, derived, and indicator measures of the PDQM model brings significant advantages. First, the measures can be properly defined, that is, without ambiguity and in a complete and objective manner. Second, a proper definition of the measures, including a detailed description of their calculation methods, enables and facilitates their implementation. This was essential in our case because one of our objectives was to define a DQM with fully automated tool support.

https://doi.org/10.1017/S0269888909000125 Published online by Cambridge University Press

https://doi.org/10.1017/S0269888909000125 Published online by Cambridge University Press



Documentation	Bad			Medium			Good		
Amount of Data	Bad	Medium	Good	Bad	Medium	Good	Bad	Medium	Good
Bad	0.9	0.8	0.5	0.8	0.3	0.15	0.5	0.15	0.01
Medium	0.09	0.15	0.3	0.15	0.4	0.25	0.3	0.25	0.09
Good	0.01	0.05	0.2	0.05	0.3	0.6	0.2	0.6	0.9

LCsR	Low	Medium	High
Bad	0.8	0.08	0.04
Medium	0.12	0.8	0.14
Good	0.08	0.12	0.82

Organization	Bad	Medium	Good
Low	0.7	0.12	0.02
Medium	0.22	0.8	0.33
High	0.08	0.08	0.65

Figure 3 Bayesian network for the DQ_Representational category

Table 6 DQ_Representational measurement model: characterization and objectives instance

Concept	PDQM instances
Information need	To know the level of the representational data quality of a Web portal
Measurable concept	Representation, understandability, attractiveness
Entity class	Web portal
Entity	Web portal of the University of Castilla-La Mancha, Web portal of the University of Malaga
Attribute	Concise representation, consistent representation, documentation, amount of data, interpretability, and organization
Quality model	PDQM

Table 7 DQ_Representational base measures

Name	Measure	Unit	Scale	Type of scale	Description	Measurement method
PgC	Page count	Pages	Natural number	Ratio	Number of pages in the portal	Counting the pages
$StPgC_s$	Pages with style S count	Pages	Natural number	Ratio	Number of pages in the portal with a given style S	Counting the pages with style S
LnC	Link count	Links	Natural number	Ratio	Number of links used in the portal	Counting the links
LTC	Link text correspondence	Links	Natural number	Ratio	Number of links with common words between the text of the link and the text of the destination page	Counting the links with common words

Table 8 PDQM derived measures

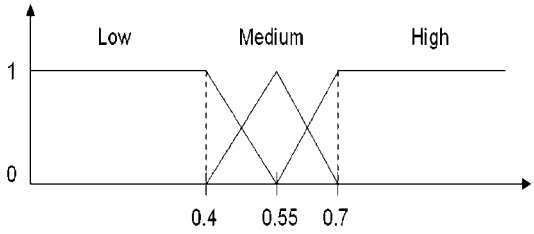
Name	Measure	Unit	Scale	Type of scale	Description	Measurement method
MaSS	Number of pages with the most used style	Pages	Natural number	Ratio	Maximum of pages with the same style	$max(\forall i, StPgC_i)$ being i each of the styles used by the portal
SDCD	Source–destination correspondence degree	Pages	Real number between 0 and 1	Ratio	Correspondence degree between source and destination pages in a portal	LTC/LnC
PSSD	Pages with the same style degree	Pages	Real number between 0 and 1	Ratio	Degree to which the portal pages have the same page style	MaSS/PgC

3.3 Developing a DSL for software measurement

In any mature engineering discipline, models are the core artifacts that allow the design and development of prototypes first, and then of the complete engineering system (Sprinkle *et al.*, 2001). The new MDE paradigm follows this approach for designing and building software systems (Bézivin *et al.*, 2005). The Model-Driven Architecture (MDA®) is the OMG proposal for implementing MDE principles and practices (OMG, 2003). In MDA, as in MDE, models are the

https://doi.org/10.1017/S0269888909000125 Published online by Cambridge University Press

Table 9 LCsR analysis model

LCsR (Level of Consistent Representation)	
Formula	Decision criteria
$LCsR = PSSD * 0.5 + SDCD * 0.5$	

essential artifacts, used to direct the course of understanding, design, construction, testing, deployment, operation, maintenance, and modification of systems. MDA raises the level of abstraction by enabling specifications that use different models to focus on different concerns, and by automating the production of such specifications and the software that meets them. In particular, MDA distinguishes between platform-independent models and platform-specific models. In addition, MDA permits the definition of further models of the system, each one focusing on a specific concern, and at the appropriate level of abstraction. These specific models are described using DSLs and related by model transformation (MT) specifications, which act as viewpoint correspondences.

In the context of MDE, domain-specific modeling (DSM) is a way of designing and developing systems, which involves the systematic use of DSLs to represent the various facets of a system. Such languages tend to support higher-level abstractions than general-purpose modeling languages, and are closer to the problem domain than to the implementation domain. Thus, a DSL follows the domain abstractions and semantics, allowing modelers to perceive themselves as working directly with domain concepts. Furthermore, the rules of the domain can be included in the language as constraints, preventing the specification of illegal or incorrect models. In general, defining a modeling language involves at least two aspects: the domain concepts and rules (abstract syntax), and the notation used to represent these concepts (concrete syntax—be it textual or graphical). Each model is written in the language of its metamodel (we normally say that a model conforms to its metamodel). Thus, a metamodel will describe the concepts of the language, the relationships between these concepts and the structuring rules that constrain the model elements and their combinations, in order to respect the domain rules.

MDA is based on a set of OMG standards, among which the MOF allows the specification of metamodels and defines a conceptual architecture with four levels of abstraction (OMG, 2006). At the lowest level (M0) we have the instances of the models, which represent the real-world entities. Then we have the model level (M1), which allows describing system models. Metamodels live at the M2 level, and define the languages in which models are written. But metamodels are also models, and therefore they need to be written in another language, which is described by its meta-metamodel. This recursive definition normally ends at that level (M3), as meta-metamodels conform to themselves.

In recent years, we have been working on a project the main goal of which was to develop a framework (called SMF) to support the software measurement process, using the MDE principles and ideas (García *et al.*, 2007; Mora *et al.*, 2008). Thus, we distinguish between the problem domain (the software measurement domain) and the solution domain (the tools that measure software products and processes in a generic and automated way) in the SMF framework. Assmann *et al.* (2006) provide a detailed proposal of the different roles played by ontologies and metamodels from a perspective based on the MDE paradigm. In that context, the core element of the SMF framework in the problem domain side is the SMO, whereas from the perspective of the solution domain, a software measurement metamodel (SMM) has been developed.

https://doi.org/10.1017/S0269888909000125 Published online by Cambridge University Press

The SMM provides the abstract syntax for the language. To provide the concrete syntax we developed a textual and graphical notation for describing measurement models (what to measure, how, who, when, etc.), in the easiest possible way. This language is called the Software Measurement Modeling Language (SMML) and is based on the original concepts defined in the SMO, that is, the SMO has been the conceptual basis during the development of the language. This approach to defining DSLs is in line with the basic principles of DSM, and follows the recommendations of several experts in this field, such as Mernik *et al*. (2005), who state that the development of a DSL requires both domain knowledge and language development expertise; and Denny (2003), who thinks that ontologies are potentially useful when developing DSLs during the analysis phase, where knowledge capture and knowledge representation are the key elements.

The package structure of the SMM metamodel that provides the abstract syntax for SSML is shown in Figure 4. The metamodel is organized around four packages. All the constructors of the Measurement metamodel have been obtained from the concepts of SMO, with the exception of the concepts of the Measurement Action sub-ontology. The fact that we have been able to reuse all the concepts of SMO has allowed us to save considerable effort. In addition, our metamodel is fully aligned with an ontology already tested and validated, therefore allowing us to build a robust metamodel. Furthermore, the resulting metamodel can seamlessly interoperate with other languages and tools on the basis of the same ontology: for example, with the SLAMMER language defined by Guerra *et al*. (2008), which also uses SMO as a conceptual basis to define a visual DSL for software measurement. 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 artifacts is an essential element.

Some of the concepts of SMML (namely, those defined in the Measurement Action metamodel) do not come from the SMO, because these concepts belong to the domain of the execution of the measurement process. It is important to note that the SMO deals with those concepts related to the definition and specification of software measures, but does not include others, such as those concepts related to the measurement process execution. This is why SMM extends the SMO with some new concepts. The fact that this extension is conservative with respect to the original ontology (i.e., it just adds new elements, but respects the structure, semantics, and relationships of the original one) guarantees that the interoperability with the methods, tools, and proposals that make use of the original ontology is maintained.

The SMM supports the graphical language to represent software measurement models in an intuitive manner. Table 10 shows some of the most representative graphical elements of SMML.

An example of the representation of a measurement model with SMML is illustrated in Figure 5, where the measurement model for Web portals (described in Section 3.2) is graphically represented.

After examining diverse tools and considering existing analysis (Pelechano *et al*., 2006), SMML has been implemented using the GMF Eclipse Project (Eclipse Graphical Modeling Framework (GMF), 2007) that supports the definition of graphic DSL editors. SMML satisfies the expected requirements of DSL (Kolovos *et al*., 2006) as follows:

Conformity: The language constructs correspond to relevant domain concepts.
Orthogonality: Each construct in the language is used to represent exactly one distinct concept in the domain.
Supportability: SMML is supported by tools including Microsoft DSL tools and GMF.
Integrability: SMML and its tools can be used in concert with other languages and tools with minimal effort (e.g., SLAMMER, as mentioned before). This is because of the way in which it has been defined and implemented, using SMO as its conceptual base.
Longevity: We hope SMML will be used long enough to justify its definition; the feedback from its initial users seems to support this claim, but it remains to be seen.
Simplicity: The language has been defined as simply as possible in order to express the concepts of interest, and to support the users and stakeholders in their preferred ways of working, avoiding unnecessary complexity.

https://doi.org/10.1017/S0269888909000125 Published online by Cambridge University Press

https://doi.org/10.1017/S0269888909000125 Published online by Cambridge University Press

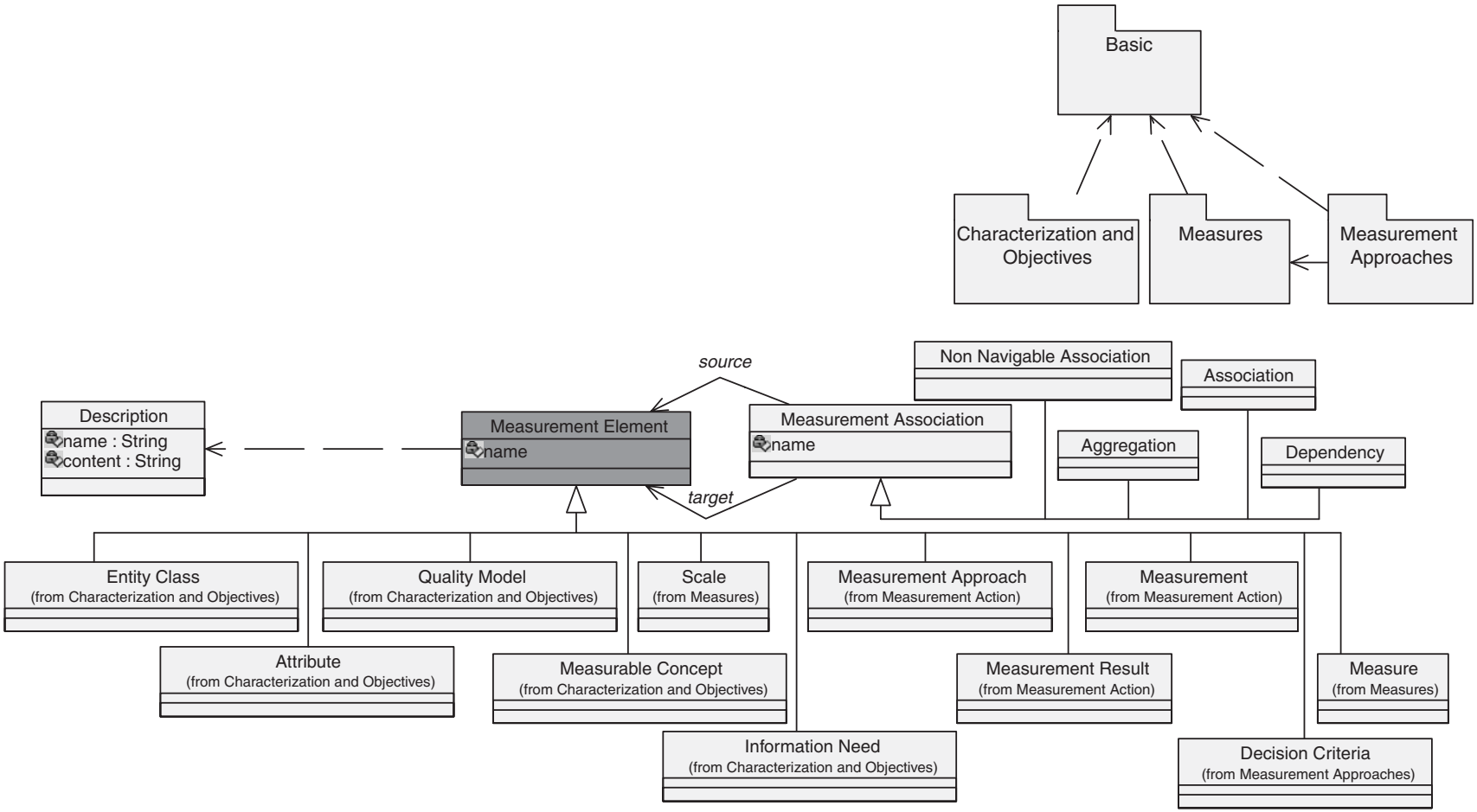
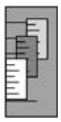
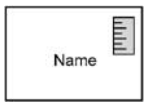
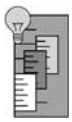
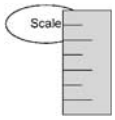
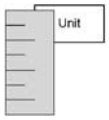
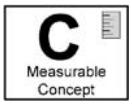
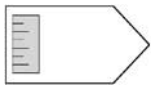
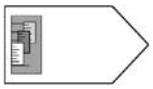
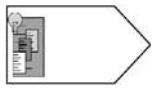


Figure 4 Overview of the software measurement metamodel

Table 10 selection of the SMML elements and icons

Ontology concept	Metamodel constructor	SMML icon	Ontology concept	Metamodel constructor	SMML icon
—	Description		Base measure	Base measure	
Information need	Information need		Derived measure	Derived measure	
Entity class	Entity class		Indicator	Indicator	
Attribute	Attribute		Scale	Scale	
Quality model	Quality model		Unit	Unit	
Measurable concept	Measurable concept		Measurement method	Measurement method	
Measurement function	Measurement function		Analysis model	Analysis model	

Quality: The language has been designed to provide some mechanisms for ensuring system quality, enforcing, for example, that all defined measures are properly and completely constructed.
Scalability: This is provided by SMML supporting tools.
Usability: DSL constructs have been designed to be expressive and easy to understand.

In summary, SMO has been extremely useful for the development of both SMM, which is being used to implement tools that manage software measurement models, and SMML, which is aimed at software measurement engineers. Furthermore, SMO has been the key to ensuring that the DSL fulfils the conformity, orthogonality, and simplicity requirements. Usability will be validated in future work by developing experiments where expert engineers in the field of software measurement will participate.

4 Conclusions and future work

It is our claim that creating an ontology for software measurement will enable the collection of the agreed knowledge in this domain and will allow agreements to be reached, something that is still

https://doi.org/10.1017/S0269888909000125 Published online by Cambridge University Press

https://doi.org/10.1017/S0269888909000125 Published online by Cambridge University Press

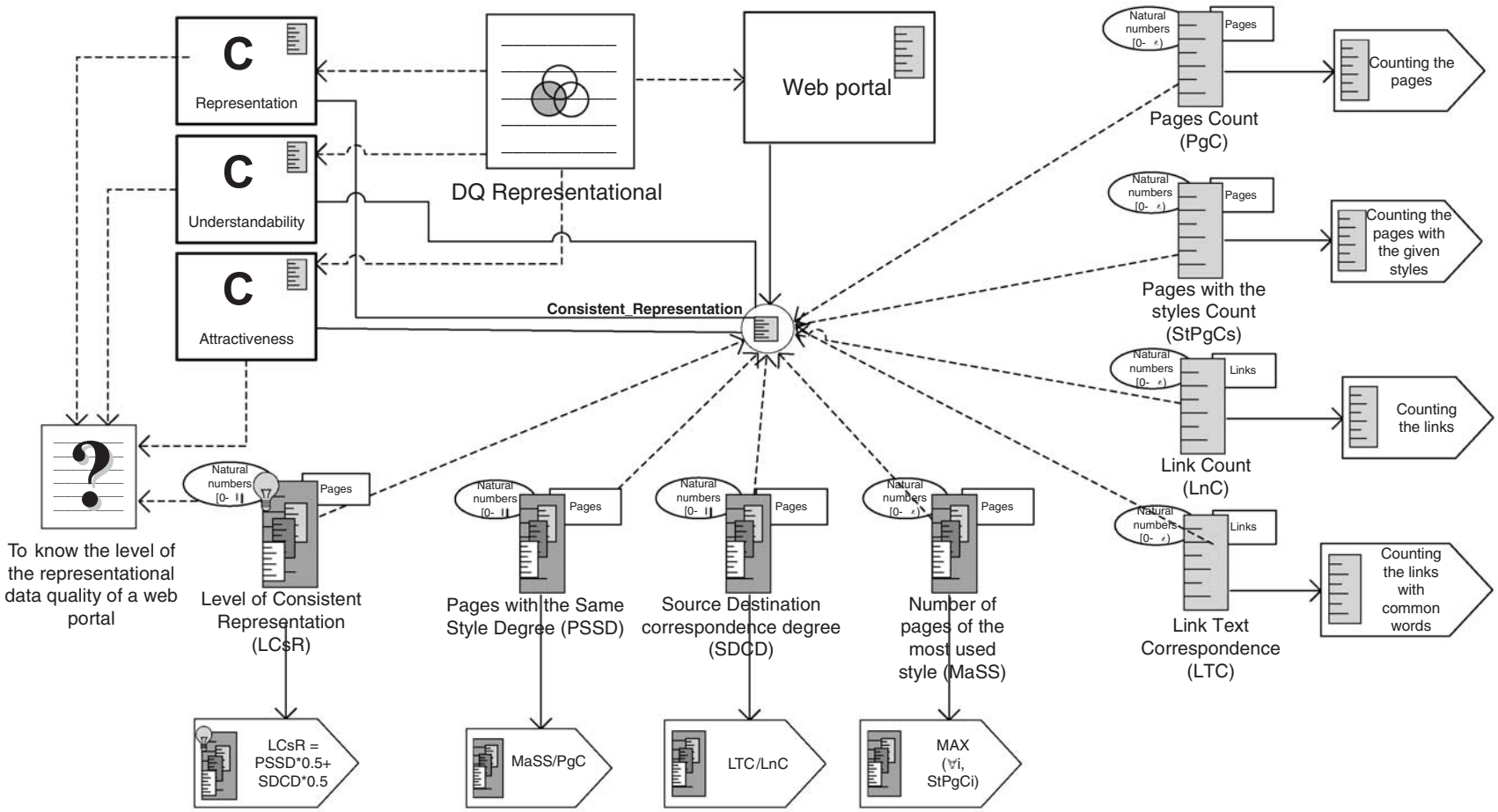


Figure 5 Measurement model of PDQM represented with SMML

far from being achieved in this field. In this sense, SMO can serve as a basis for discussion to achieve the necessary consensus and to contribute to the harmonization of existing (and future) standards and proposals within the software measurement domain. Furthermore, it can be used as a vehicle for achieving the interoperability required between the ever-increasing number of groups and organizations working on languages and tools for software measurement.

In the case studies presented in this paper, we have seen different use cases for the SMO where it has shown its usefulness. In the first case, the ontology has served as the basis for comparing and analyzing the terminology used in several international standards related to software measurement, and it has been put forward as a reconciling proposal for reaching future agreements. The second case makes use of SMO to define a quality model for Web portal data. The SMO allowed us to identify and define all the elements of the quality model, from the information needs and measurable concepts, to the base and derived measures used to evaluate the quality attributes. In this example, BBN were used to measure the quality characteristics of the model, using the values of the indicators as input evidences. SMO allowed defining, without ambiguity, the base and derived measures that were used to synthesize the indicators, making them amenable to automation. The last case study shows the development of a SMM and a textual and graphical concrete syntax for it (the SMML language), which allows representing software measurement models. The SMO has been used as a conceptual model during the development of this language.

Our future plans for SMO include its integration and alignment with the new and revised terms of VIM 3.0, in order to be fully compatible with the way in which most science and engineering disciplines deal with Measurement. In addition, the SMO may need to evolve in order to take into consideration the new versions of ISO and IEEE standards (e.g., ISO/IEC 15939, ISO/IEC 25000 (SquARE), etc.). These actions are aimed at maintaining a complete and up-to-date ontology, that can offer a useful reference for software measurement, and a framework for harmonizing the terminology used in this domain.

We are also conducting new case studies in which the ontology is used as a Software Artifact. This will allow us to obtain useful feedback for future improvements of the ontology. In the same spirit, we plan to thoroughly validate the visual language SMML through a family of experiments, with the aim of verifying its usability and full applicability in this context.

Acknowledgements

We thank Prof. Alain Abran for his useful suggestions and comments, which have helped us to improve the first versions of the ontology. We are also grateful to the anonymous reviewers of this work, for their valuable and detailed comments. This work has been funded by the following research projects: ESFINGE (TIN2006-15175-C05-05), MECENAS (PBI06-0024), MOVIS (P07-TIC-03184), IVISCUS (PAC08-0024-5991), VIASCO (PET2006-0682-00), CALIPSO (TIN2005-24055-E) and TIN2005-09405-C02-01.

References

Althoff, K., Birk, A., Hartkopf, S. & Muller, W. 1999. Managing software engineering experience for comprehensive reuse. In *Proceedings of the International Conference on Software Engineering (ICSE'99)*. Kaiserslautern, Germany.

Assmann, U., Zschaler, S. & Wagner, G. 2006. Ontologies, meta-models, and the model-driven paradigm. In *Ontologies for Software Engineering and Technologies*, Calero, C., Ruiz, F. & Piattini, M. (eds). Springer-Verlag, 49–102.

Bertoa, M., García, F. & Vallecillo, A. 2006. An ontology for software measurement. In *Ontologies for Software Engineering and Technologies*, Calero, C., Ruiz, F. & Piattini, M. (eds). Springer-Verlag, 175–196.

Bézivin, J., Jouault, F. & Touzet, D. 2005. Principles, standards and tools for model engineering. In *Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'2005)*. IEEE Computer Society, Shanghai, China, 28–29.

Caro, C., Calero, C. & Piattini, M. 2007. Development of the operational version of PDQM. In *Proceedings of the 8th International Conference on Web Information Systems Engineering (WISE 2007)*. France.

https://doi.org/10.1017/S0269888909000125 Published online by Cambridge University Press

Caro, C., Calero, C., Caballero, I. & Piattini, M. 2008. A proposal for a set of attributes relevant for Web portal data quality. *Software Quality Journal* **16**(4), 513–542.

Denny, M. 2003. *Ontology Building: A Survey of Editing Tools*. Available at `http://www.xml.com/pub/a/2002/11/06/ontologies.html`

Eclipse Graphical Modeling Framework (GMF). 2007. Available at `http://www.eclipse.org/gmf/`

García, F., Ruiz, F., Bertoa, M., Calero, C., Genero, M., Olsina, L. A., Martín, M. A., Quer, C., Condori, N., Abrahao, S., Vallecillo, A. & Piattini, M. 2004. *An Ontology for Software Measurement*. Technical report, UCLM DIAB-04-02-2, Computer Science Department, University of Castilla-La Mancha, Spain.

García, F., Bertoa, M., Calero, C., Vallecillo, A., Ruiz, F., Piattini, M. & Genero, M. 2006. Towards a consistent terminology for software measurement. *Information and Software Technology* **48**(8), 631–644.

García, F., Serrano, M., Cruz-Lemus, J., Ruiz, F. & Piattini, M. 2007. Managing software process measurement: a metamodel-based approach. *Information Sciences* **177**, 2570–2586.

Gómez-Pérez, A. 1998. *Knowledge Sharing and Reuse*. CRC Press.

Gruber, T. R. 1993. A translation approach to portable ontologies. *Knowledge Acquisition* **5**(2), 199–220.

Guerra, E., de Lara, J. & Díaz, P. 2008. Visual specification of measurements and redesigns for domain specific visual languages. *Journal of Visual Languages and Computing* **19**(8), 399–425.

Kolovos, D. S., Paige, R. F., Kelly, T. & Polack, F. A. C. 2006. Requirements for Domain-Specific Languages. In *First ECOOP Workshop on Domain-Specific Program Development (ECOOP'06)*. Nantes, France.

Malak, G., Sahraoui, H., Badri, L. & Badri, M. 2006. Modeling Web-based applications quality: a probabilistic approach. In *Proceedings of the 7th International Conference on Web Information Systems Engineering*, Lecture Notes in Computer Science **4255**, 398–404. Springer.

Mernik, M., Heering, J. & Sloane, A. M. 2005. When and how to develop domain-specific languages. *ACM Computing Surveys* **37**(4), 316–344.

Mora, B., García, F., Ruiz, F., Piattini, M., Boronat, A., Gómez, A., Carsí, J. & Ramos, I. 2008. Proceedings of the Tenth International Conference on Enterprise Information Systems (ICEIS2008), Volume DISI, Barcelona, Spain, 117–124.

Object Management Group (OMG). 2003. *MDA Guide*, Version 1.0.1, June 2003. Available at `http://www.omg.org/mda/specs.htm`

Object Management Group (OMG). 2006. *Meta Object Facility (MOF) Core Specification*, Version 2.0, January 2006. OMG document formal/2006-01-01. Available at `http://www.omg.org/docs/formal/06-01-01.pdf`

Pelechano, V., Albert, M., Javier, M. & Carlos, C. 2006. Building tools for model driven development comparing microsoft DSL tools and eclipse modeling plug-ins. In *Proceedings of Desarrollo de Software Dirigido por Modelos—DSDM'06*. Sitges, Spain.

Ruiz, F. & Hilera, J. R. 2006. Using ontologies in software engineering and technology. In *Ontologies in Software Engineering and Software Technology*, Calero, C., Ruiz, F. & Piattini, M. (eds). Springer-Verlag, 49–102.

Sprinkle, J. M., Ledeczi, A., Karsai, G. & Nordstrom, G. 2001. The new metamodeling generation. In *Proceedings of the 8th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, 275–279.

Staab, S., Schnurr, H. & Sure, Y. 2001. Knowledge processes and ontologies. *IEEE Intelligent Systems* **16**(1), 26–34.

Tautz, C. & Von Wangenheim, C. 1998. *REFSENO: A Representation Formalism for Software Engineering Ontologies*. Technical report, N 015.98/E, version 1.1. Fraunhofer IESE.

Uschold, M. & Gruninger, M. 1996. Ontologies: principles, methods, and applications. *Knowledge Engineering Review* **11**(2), 93–196.

Wang, R. & Strong, D. 1996. Beyond accuracy: what data quality means to data consumers. *Journal of Management Information Systems* **12**, 5–33.

Xiao, L. & Dasgupta, S. 2005. User satisfaction with Web portals: An empirical Study. In *Web Systems Design and Online Consumer Behavior*, Gao, Y. (ed.). Idea Group Publishing, 193–205.

Yang, Z., Cai, S., Zhou, Z. & Zhou, N. 2004. Development and validation of an instrument to measure user perceived service quality of information presenting Web portals. *Information and Management* **42**, 575–589.

https://doi.org/10.1017/S0269888909000125 Published online by Cambridge University Press