Received: 20 May 2015 | Revised: 23 December 2016 | Accepted: 3 March 2017
DOI: 10.1111/exsy.12213

WILEY Expert Systems

ARTICLE

An ontology to document a quality scheme specification of a software product

María Julia Blas | Silvio Gonnet | Horacio Leone

INGAR, National University of Technology, CONICET, Santa Fe 3000, Argentina

Correspondence
María Julia Blas, INGAR, National University of Technology, CONICET, Avellaneda 3657, Santa Fe 3000, Argentina.
Email: mariajuliablas@santafe-conicet.gov.ar

Funding Information
CONICET, Grant/Award Number: PIP 112-20110100906; Universidad Tecnológica Nacional, Grant/Award Number: PID 25/O144 and PID 25/O156

Abstract
Pressman's (2010) definition of software quality is the conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software. The achievement of quality is a complex activity that is related with the process quality and product quality. However, still, it is not clear how product quality should apply in the development process. Frequently, the development team does not know what characteristics influence a specific entity and how these characteristics compose the quality of the product. This lack of knowledge reflects the need to define some document that serve as a valid quality specification that should be evaluated along the development process. This contribution addresses this need by introducing an ontology for document a quality scheme based on the product quality model of ISO/IEC 25010. The proposal follows this model with aims to clarify the properties that are normally present in a product and its meaning. The ontology incorporates the metric concept to represent the way in which quality should be measure. The ontology was implemented using Protégé, and it was evaluated using a set of metrics that estimate the required structural characteristics.

KEYWORDS
knowledge representation, ontology, quality model, requirement engineering, software engineering, software quality

1 | INTRODUCTION

Software quality is the degree to which software possesses a desired combination of attributes (IEEE STD. 1061, 1998). However, the complex nature of software makes achieving this property a complicated issue. Moreover, the delimitation of what defines an adequate level of quality in a software system is a highly context-dependent question (Kitchenham & Pfleeger, 1996). This problem changes with the product and perspective of the stakeholders. As long as each stakeholder group has its own perspective on what is important, software product quality can easily become an area of problems and conflict.

More generally, software quality encompasses many different product and process factors (Pressman, 2010). System quality attributes have been of interest to the software community at least since the 1970s (Bass et al., 2012). In the past few years, this interest has increased and a lot of research work has been done in this area. The understanding of software priorities in industrial contexts (Barney & Wohlin, 2009; Barney et al., 2014) and the study of computational technics to develop quality evaluation methods (Papas & Tjortjis, 2014) are examples of these researches.

Everyone involved in the software engineering process is responsible for quality (Pressman, 2010). Achieving quality attributes must be considered throughout design, implementation, and deployment (Bass et al., 2003). However, maintaining the traceability of this attributes in the development process is very difficult. Frequently, the attributes get lost between different stages because there is no mechanism that supports the quality decisions made in the previous phases. Or worse, the development team does not know which attributes are related and how they impact in the overall quality of the software product. So the precise knowledge of software quality is usually not available until very late (usually during operations and maintenance) in the software life cycle (Khoshgoftaar, Liu, & Seliya, 2004). In this context, software quality is still considered to be one of the most important concerns of software production teams (Ampatzoglou, Frantzeskou, & Stamelos, 2012). The quality specification of software products is a valuable addition to functional specification because it allows clarifying product properties such as learnability and availability (Van Zeist & Hendriks, 1996). Then, a documentation mechanism that provides the basic quality information to define a quality document could become a feasible solution to the quality specification problem of the software community.

To this purpose, this paper presents an ontological approach that helps in the specification of a quality scheme based on the quality model defined in ISO/IEC 25010. The proposed ontology is based on the combination of three semantic models: quality semantic model, metric semantic model, and software semantic model. The quality semantic model represents the product quality model presented in the standard ISO/IEC 25010. The metric semantic model represents the main concepts associated with the definition of a metric. Finally, the software semantic model represents the content of a software product and links the quality attributes with the metrics. Each model is complemented with a set of Semantic Web Rule Language (SWRL) rules that enrich its definition. The SWRL is used to express rules in the form of an implication between an antecedent (body) and consequent (head; Horrocks et al., 2004). The incorporation of these rules in each ontology definition allows to predict new knowledge and verify the integrity of the instances created from it. The proposed ontology is implemented in Web Ontology Language (OWL) 2 (W3C OWL, 2012). The final model is complemented with a set of questions that helps to explore a set of defined instances. These questions allow using the quality schemes instantiated in order to know which are the main quality artifacts, metrics, and subcharacteristics used in a specific context. All the questions were implemented in SPARQL (Harris, Seaborne, & Prud'hommeaux, 2013).

The major contribution of this proposal is to introduce into the software engineering community a new mechanism that allows developers to specify a quality document for a specific software product. The information included on this new document will be based on a defined set of quality aspects identified on the requirements specification. Those aspects will be mapped to characteristics of a quality model that contextualizes the relationships between quality concepts. By mean of the final ontology instantiation, the development team will know which elements must be created in order to treat quality properties and, therefore, will increase the visibility and understanding of the different quality aspects. Also, the final customer could be aware of the strategies used to guarantee its satisfaction. Then, the use of the ontological approach proposed in the development process will give multiple benefits, not only to the development team but also to the final customer. The existence of a single document that defines all quality aspects related to a software product will help to manage quality.

The remainder of this paper is organized as follows. A background of software quality evaluation and ontology engineering is given in Section 2. Sections 3, 4, and 5 present the semantic models. Section 6 describes the final ontology, specifying the implementation and verification process carried out. Section 7 shows the application of the proposed ontology in a case study and analyses the resultant quality scheme. Finally, Section 8 is devoted to conclusions.

2 | BACKGROUND AND MOTIVATION

The quality of a software system is directly related to the ability of the system to satisfy its functional, nonfunctional, implied, and specified requirements (Albin, 2003). This ability can be evaluated in any product that results of some activity of the development process. An intermediate product is any model, specification, document, or source code that is prepared or created in support of constructing an executable system (Albin, 2003). The result of evaluating this type of products provides a reasonable prediction of some of the target system's quality attributes. Researchers have work on this topic over different intermediate products. According to Dargan, Campos-Nanez, Fomin, and Wasek (2014), it is possible to predict the performance of a software system using the requirements document. In other work, Roshandel, Medvidovic, and Golubchik (2007) show a way to predict reliability by mean of the architectural design. Related works on this research area are presented in the literature (Bogado, Gonnet, & Leone, 2014; Meiappane, Chithra, & Venkataesan, 2013; Rech & Bunse, 2008). However, all these researches are applied to a defined set of quality attributes. The use of an attribute set hides the overall quality aspects that impact in the software product and leaves out the existing relationships between attributes.

In this context, the use of quality models is a good proposal. A standard taxonomy of quality attributes that serves as a framework for system specification and testing is called quality model (Albin, 2003). Numerous models have been developed to support software quality (Milicic, 2005). Examples of these models include McCall quality model, Boehm's quality model, Dromey's quality model, and ISO 9126. All these models are discussed in Al-Badareen, Selamat, Jabar, Din, and Turaev (2011). More recently, a new standard (ISO/IEC 25010) has been developed as an evolution of ISO 9126. Boukouchi, Marzak, Benlahmer, and Moutachaouik (2013) compared this new model with the previous ones. Because the quality is related to all the activities of the development process, the first specification must be elaborated after the elicitation process (when the architectural design has place). That is, software architect can use a quality model to develop a quality specification for a software product. This specification can be used as a base to estimate and evaluate a specific component of the software product (i.e., the architectural design) and to know how it adjusts to the desired quality level. Then, the specification involves the adoption of a coherent set of properties that must be present in the system and that can be predicted by the components. However, this type of specification only includes properties that previously must be understood by the architect. Ideally, a quality specification must have the set of attributes of interest and the way in which these attributes must be measured. The use of metrics to develop strategies for improving quality of the end product is a good practice (Pressman, 2010).

A software metric is a measure of some property of a piece of software code or its specifications (El-Haik & Shaout, 2010). Software metrics can be classified into three categories: product metrics, process metrics, and project metrics (Kan, 2003). Product metrics describe the characteristics of the product. Process metrics can be used to improve software development and maintenance. Project metrics describe the project characteristics and execution. Specifically, software quality metrics refer to end-product metrics or in-process metrics. This type of metrics is useful to register the current quality state of an end product or process associated to the software. Although a lot of metrics have been proposed over the years, the main source of this content corresponds to the ISO 9126 standard (ISO/IEC TR 9126-2, 2003; ISO/IEC TR 9126-3, 2003). Then, quality models and software metrics are related only when a specific software product is considered.

2.1 | Quality scheme foundations

In the traditional development process, software requirement specification (SRS) is the artifact that establishes all the software product requirements. By using its content, the development team can isolate the defined quality properties and use them in order to specify the way in which these properties should be evaluated in the software product. This specification is called Quality Scheme (QS). A QS is a set of triplets over a software product definition where each element is composed by a software attribute, a software metric that should be used to its measurement, and a quality subcharacteristic that should be evaluated over it. That is, QS connects quality properties and software metrics over a software product taken as a base of the SRS. For example, think of an online toy store as a software product. Then, two software attributes can be measured to evaluate the software product data: "Secure Access" and "Secure Storage." In order to define a QS, a software metric must be defined for each identified software attribute, for example, "Number of Unauthorized Access" and "Number of Access." Also, each identified software attribute must be associated with a quality subcharacteristic, for example, "Confidentiality" and "Integrity." So the following set of triplets can be used to describe part of QS proposed for the online toy store: $QS_{onlinetoystore}$ = {(Secure Access, Number of Unauthorized Access, Confidentiality); (Secure Storage, Number of Access, Integrity)}.

Given that this artifact is obtained as an outcome of the development process, a QS can be used, improved, and traced while the development process is carried out. Then, the QS can be used as a support document to manage quality. However, its definition must follow specific concepts and relationships. Therefore, its structure could not be left up to the team expertise. Instead of documenting the concepts in the traditional way, this paper proposes use of ontologies to represent the quality scope using a set of defined elements and relationships.

2.2 | Ontologies to support quality schemes

An ontology is an explicit specification of a conceptualization, that is, an abstract, simplified view of the world that includes the objects, concepts, and the relationships between them in a domain of interest (Gruber, 1993). Its definition allows an unambiguous specification of the structure of knowledge in a domain, enables knowledge sharing and reuse and, consequently, makes automated reasoning about ontologies possible (Orgun & Meyer, 2008).

The use of ontologies to support different aspects of software engineering has increase in the last years. A lot of the research work done refers to the elicitation process (Al Balushi, Sampaio, & Loucopoulos, 2013; Couto, Ribeiro, & Campos, 2014; Jwo & Cheng, 2010; Pires et al., 2011) and to the specification of ontologies for intermediate products (Abebe & Tonella, 2015; De Graaf, Liang, Tang, Van Hage, & Van Vliet, 2014). Other works focus on the development of tools based on ontologies that help to support the development process (García-Peñalvo, Colomo-Palacios, García, & Therón, 2012; Henderson-Sellers, 2011; Reinhartz-Berger, Sturm, & Wand, 2013). In this context, ontologies are a good mechanism to specify the set of concepts and relationships required in the software product quality domain (i.e., the QS domain).

The software product quality domain consists of three different and independent domains: quality model domain, software metric domain, and software product domain. Given that a unique conceptualization of all of these domains does not exist, a new ontology must be created. Each identified domain represents a specific aspect of the QS. Considering that there are ontologies to characterize the software metric domain, the approach proposed in this paper comprises the representation of each domain as an individual semantic model that, subsequently, is linked to the others by means of relationships. According to this modular design strategy, each semantic model can be built leaving aside the conceptualization of other domains. Then, the domains can be treated individually. This gives multiple advantages. First, the available ontologies of specific domains can be used or adapted to the purposes of the QS representation. This is the case of the software metric domain. Several authors have proposed semantic models for this domain across the years (Bertoa, Vallecillo, & García, 2006; Kitchenham, Hughes, & Linkman, 2001; Martin & Olsina, 2003; Olsina & Martín, 2003). Although these models are useful, its conceptualization does not reflect the required content. Then, in order to build the software metric ontology, the existing models were used as guidelines of the ontology development process. The same approach was used to the software product ontology. As opposed, when a domain does not have related ontologies, the modular strategy allows to define new semantic models using the representation needed. The ISO/IEC 25010 quality model belongs to this domain type. Given that the software product quality model specification included in the standard has recently emerged, semantic models of this domain are still under developing. Therefore, in order to conceptualize this domain, this work makes an exhaustive analysis of the standard to build a new ontology of the quality domain.

Another advantage of dividing the final model in three ontologies is reutilization. The approach proposed allows to reuse the developed models in different context, for example, (a) software product model can be used as part of a software description; (b) software metric model can be used as support model in measurement process to know which properties should be obtained for an specific metric; and (c) quality model ontology can be used as tool in the quality management process, highlighting the characteristics and their relationships.

The final document created by following the proposed method will be an instantiation of ontology elements that reflect the quality requirements of the software product. This instantiation must follow all the mandatory relationships between concepts. However, in some cases, not all components will be created. The instantiation of all the components included in the quality model ontology will only take place if the SRS identifies requirements for all the quality aspects considered. Actually, the generation of this document type must be done using ontology tools. In the future, this creation process will be supported by a software tool based on ontologies developed specifically with aim to help in the quality management activity.

2.3 | Semiformal ontologies and completeness

Ontologies can be classified using several dimensions (Roussey, Pinet, Kang, & Corcho, 2011). Following the classification based on the degree of formality proposed by Hadzic, Chang, Dillon, Kacprzyk, and Wongthongtham (2009), ontologies can be classified in four types: highly informal, semi-informal, semiformal, and rigorously formal. An ontology is highly informal if is expressed in natural language. Usually, the term definitions used in this type of ontologies are ambiguous due to the ambiguity of natural language. Semi-informal ontologies try to solve this problem by restricting and structuring the natural language. Then, this type of ontologies is expressed in a restricted and structured form of natural language in order to improve clarity. When the language used to express the ontology is not natural language, the ontology can be considered semiformal or formal. Semiformal ontologies are expressed in an artificial and formally defined language. Rigorously formal ontologies provide meticulously defined terms with formal semantics, theorems, and proofs of properties such as soundness and completeness.

According to Gómez-Pérez, Fernandez-Lopez, and Corcho (2010), an ontology is sound if and only if it does not allow deducing invalid conclusions. Furthermore, an ontology is complete if and only if it allows deducing all the possible valid conclusions starting from the ontology vocabulary and applying the deduction rules permitted. The proof of these two properties can be only done in formal ontologies. The lack of formality in other ontology types makes it impossible to prove the completeness of the proposed model. Then, neither the completeness of an ontology nor the completeness of its definitions can be proved. However, the incompleteness of an individual definition can be proved, and therefore, the incompleteness of the ontology can be deduced if at least one definition is missing in the established reference framework (Gómez-Pérez et al., 2010). This reference framework can be, for example, the requirements specifications or the real world. Using this approach, an ontology is complete if and only if all that is supposed to be in the ontology is explicitly stated in it (or can be inferred) and each definition is complete (all the entities of the world required are explicitly represented or can be inferred using other representations and axioms).

As already stated, the designed ontologies were described using OWL and SWRL. Considering that both languages are artificial and formally defined languages, each of the semantic models proposed belong to the semiformal ontology category. Then, the proof of completeness can be only done following the incompleteness approach mentioned above. Therefore, the completeness of the semantic models is evaluated by checking the existence of all the entities required (given by its domain description) over the ontology definition.

3 | QUALITY ONTOLOGY

A quality model is a model with the objective to describe, assess, and/or predict quality (Deissenboeck, Juergens, Lochmann, & Wagner, 2009). Software architect can use a quality model to develop quality specifications for different components of a software product. Then, software quality models are a well-accepted means to support quality management of software systems.

Many models have been proposed to support stakeholders in dealing with software quality. However, by novelty and completeness, the product quality model presented in ISO/IEC 25010 (ISO/IEC 25010, 2011) is the most rigorous and complete of all. Although the standard has the difficulties and ambiguity of all documents written in natural language, the conceptualization using ontologies provides the semantic context needed to overcome these problems. Then, this quality model was taken as foundation of this work.

The ontology proposed enriches the ISO/IEC 25010 quality definition by showing how the concepts relate to other concepts and revealing the implicit relationships between concepts.

3.1 | Domain description

The ISO/IEC 25010 is a quality standard developed by ISO (International Organization for Standardization) and IEC (International Electrotechnical Commission) as part of the SQuaRE series of International Standards (SQuaRE, 2005). This standard belongs to the “Quality Models Division” and is called “System and Software Quality Models” because it defines two different and complementary quality models applicable to software systems: a quality in use model and a product quality model.

The product quality model identifies the main characteristics of a software product in different levels of hierarchy (Figure 1). It is composed of eight characteristics that are further subdivided into subcharacteristics, trying to relate static properties of software and dynamic properties of the computer system. A characteristic represents an external quality view (i.e., a property that can be seen by the user). The model identifies as quality characteristics the following properties: portability, reliability, usability, compatibility, functional suitability, performance efficiency, security, and maintainability. Each characteristic is divided in a set of subcharacteristics that references to properties that can be evaluated when the software is used as a part of a system. By example, the characteristic “functional suitability” is divided into three subcharacteristics: functional correctness, functional appropriateness, and functional completeness. Each subcharacteristic is decomposed in attributes. An attribute is an entity that can be verified or measured in the software product. Attributes are not defined in the standard, as they vary between different software products.

The characteristics and subcharacteristics' hierarchies provide a consistent terminology for specifying, measuring, and evaluating system and software product quality. They also provide a way to recognize the quality properties and compare it with the stated quality requirements to see its completeness. The value of each of these characteristics comprises the total quality of the system. However, the standard does not specify how to the quality should be considered along the hierarchy level.

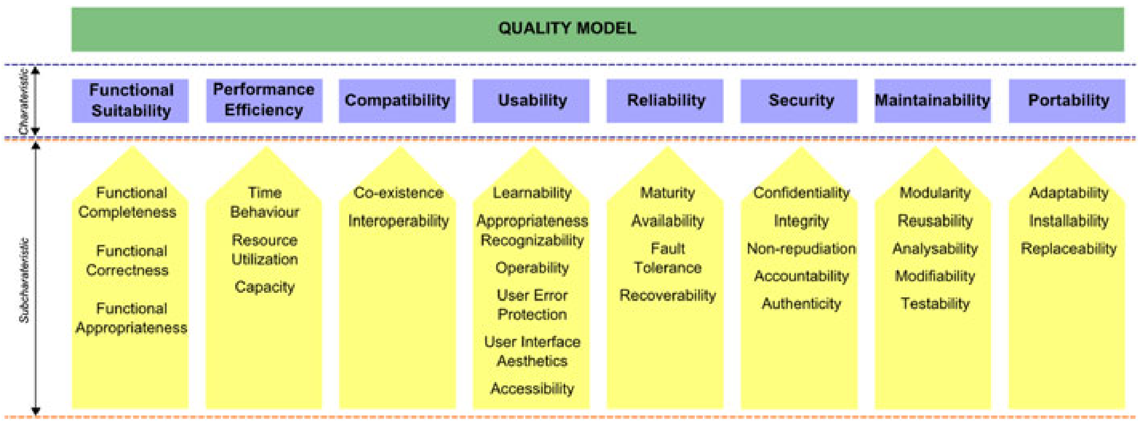


FIGURE 1 Product quality model proposed in ISO/IEC 25010, adapted from ISO/IEC 25010 (2011)

3.2 | Semantic model

The representation of the product quality model presented in the previous section was made following the taxonomy defined in ISO/IEC 25010 (Figure 1). Only the quality properties detailed were included.

Table 1 resumes the set of transformations made. Each characteristic was transformed in a concept (#T1). The same transformation was used for subcharacteristics (#T2). In both cases, the quality concepts represent individual properties that must be linked between them. Then, its transformation to ontology concepts allows to show its essence giving a way to relate them using relationships. Relationships between these concepts were modeled defining links between them (#T3). This definition took the form "is-decomposed-in." For example, the characteristic functional suitability is related with subcharacteristics functional appropriateness, functional completeness, and functional correctness. Therefore, this concept must have three relationships: "is-decomposed-in-functional-completeness," "is-decomposed-in-functional-correctness," and "is-decomposed-in-functional-appropriateness." Also, other concepts were included in the model with the aim to help in the understanding of it. The first concepts included were "Characteristic" and "Subcharacteristic" (#T4). These definitions were linked with the related concepts using an "is-a" relationship (#T5). By means of this classification, the different concepts (that represent quality elements of ISO/IEC 25010) are grouped in different categories (which give a concepts' typification). In order to organize all the concepts, the "Quality Model" concept was also included as part of the semantic model (#T6). The relationship of this concept with each characteristic was made by defining a new relationship that took the name "contains," for example, "contains-performance-efficiency" (#T7). Also, the incorporation of the Quality Model concept helps with the definition of a quality scheme. As a quality scheme involves a set of quality subcharacteristics defined by a quality model, the schemes created using the quality model of ISO/IEC 25010 (defined in the Quality ontology) will respond to its definition. Therefore, the Quality Model concept englobes all the quality properties defined in a specific scheme. Once the main concepts were defined, a set of properties was included in the model with the aim to refine the represented semantic. These properties include characteristic description, subcharacteristic description, and source (#T8). All of them are defined as strings.

TABLE 1 Mapping between ISO/IEC 25010 quality elements and ontology components

Transf. #	ISO/IEC 25010 quality elements	Type of component	Example
T1	Characteristic type	Concept	The "compatibility" characteristic becomes the "compatibility" concept.
T2	Subcharacteristic type	Concept	The "interoperability" subcharacteristic becomes the "interoperability" concept.
T3	Decomposition of each characteristic into subcharacteristic elements	Relation	The relation between the "compatibility" characteristic and "interoperability" subcharacteristic becomes the "is-decomposed-in-interoperability" relation.
T4	Characteristic and subcharacteristic definition	Concept	The "characteristic" definition becomes the "characteristic" concept.
T5	Characteristic and subcharacteristic hierarchies	Relation	The relation between the "compatibility" characteristic and "characteristic" concept becomes the "is-a" relation.
T6	Quality model	Concept	The "quality model" definition becomes the "quality model" concept.
T7	Decomposition of quality model into characteristic elements	Relation	The relation between the "quality model" concept and "compatibility" characteristic becomes the "contains-compatibility" relation.
T8	Source, characteristic description, and subcharacteristic description	Property	The "description" attribute related with a "characteristic" definition becomes the "description" property of the "characteristic" concept.

Figure 2 shows the ontology developed. The type of representation used is similar to the proposed in Gómez-Pérez et al. (2010). The light gray nodes are the concepts whereas the white nodes refer to the relationships. The arrow head indicated the direction of the relationship. The empty arrows model the relation is-a, disjoint, and complete. The dark gray boxes model the data type of the attributes.

The ontology definition includes the specification of some SWRL rules related to the contains relationship. These rules allow defining new links between concepts at abstract level (which are derived from the explicit relationships established). That is, according to #T1, #T2, and #T3 (Table 1), each specific characteristic is related with its subcharacteristics by an is-decomposed-in relationship. Then, each characteristic contains to all its subcharacteristics (no matter the

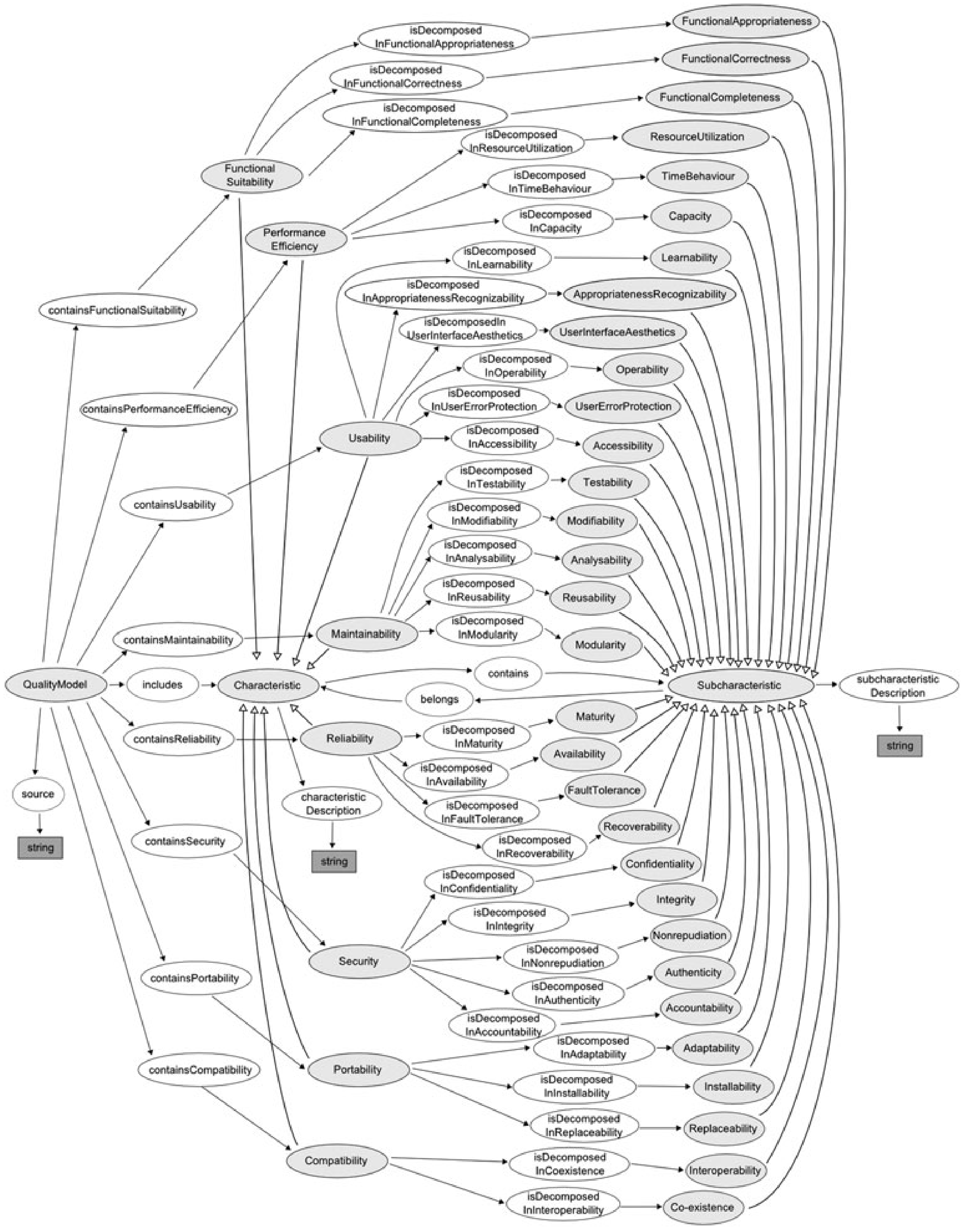

QualityModel
includes
Characteristic
contains
belongs
Subcharacteristic
subcharacteristic Description
string
source
string
characteristic Description
string
containsFunctionalSuitability
containsPerformanceEfficiency
containsUsability
containsMaintainability
containsReliability
containsSecurity
containsPortability
containsCompatibility
Functional Suitability
Performance Efficiency
Usability
Maintainability
Reliability
Security
Portability
Compatibility
isDecomposed InFunctionalAppropriateness
isDecomposed InFunctionalCorrectness
isDecomposed InFunctionalCompleteness
isDecomposed InResourceUtilization
isDecomposed InTimeBehaviour
isDecomposed InCapacity
isDecomposed InLearnability
isDecomposed InAppropriatenessRecognizability
isDecomposedIn UserInterfaceAesthetics
isDecomposed InOperability
isDecomposed InUserErrorProtection
isDecomposed InAccessibility
isDecomposed InTestability
isDecomposed InModifiability
isDecomposed InAnalysability
isDecomposed InReusability
isDecomposed InModularity
isDecomposed InMaturity
isDecomposed InAvailability
isDecomposed InFaultTolerance
isDecomposed InRecoverability
isDecomposed InConfidentiality
isDecomposed InIntegrity
isDecomposed InNonrepudiation
isDecomposed InAuthenticity
isDecomposed InAccountability
isDecomposed InAdaptability
isDecomposed InInstallability
isDecomposed InReplaceability
isDecomposed InCoexistence
isDecomposed InInteroperability
FunctionalAppropriateness
FunctionalCorrectness
FunctionalCompleteness
ResourceUtilization
TimeBehaviour
Capacity
Learnability
AppropriatenessRecognizability
UserInterfaceAesthetics
Operability
UserErrorProtection
Accessibility
Testability
Modifiability
Analysability
Reusability
Modularity
Maturity
Availability
FaultTolerance
Recoverability
Confidentiality
Integrity
Nonrepudiation
Authenticity
Accountability
Adaptability
Installability
Replaceability
Interoperability
Co-existence

FIGURE 2 Quality ontology

subcharacteristic type). So the contains relationship is a new relation that can be derived by using the is-decomposed-in relationship. Equation 1 describes an example SWRL restriction, where "?x" and "?y" refer to individuals (i.e., concept instances), "Security()" represents the security characteristic concept, and "isDecomposedInIntegrity()" relates two individuals by means of the specific established relationship. Then, the equation shows that all instance of security characteristic decomposed in an integrity subcharacteristic must contain this subcharacteristic. Similar restrictions were added to the model to make that all characteristics contain the appropriate set of subcharacteristics. Also, the "belongs" relation was specified using SWRL. Given a contains relationship (i.e., a link between a quality characteristic and subcharacteristic), a belongs relationship must be created in order to represent the belonging of the subcharacteristic to the characteristic. Then, these two relations are inverses.

$$\text{Security}(?x) \wedge \text{isDecomposedInIntegrity}(?x, ?y) \rightarrow \text{contains}(?x, ?y), \quad (1)$$

The semantic model specified along with the set of SWRL rules proposed helps to instantiate the lower level quality concepts (i.e., characteristic type and subcharacteristic type identified in Table 1) and then derivate the missing relationships at upper level (i.e., between characteristic and subcharacteristic definitions—Table 1).

In order to prove the completeness of the model, the list of terms defined in ISO/IEC 25010 along with the domain description of the software product quality model was used. An exhaustive test was performed to check the presence of all the concepts required as part of the model. A table with two columns was designed in order to be used across the test. Each term defined in ISO/IEC 25010 was included as a row of the first column. Once all the terms were listed, each concept of the semantic model was added to some row of the second column according to the criteria "represents-the-first-column-term." At the end of the test, all rows of the table have two elements. Then, all ISO/IEC 25010 terms were represented as part of the semantic model.

4 | METRIC ONTOLOGY

A lot of research work has been done in order to contribute in the measure of the software product quality. As part of the SQuaRE series, ISO and IEC propose a measurement reference model and a guide for measuring the quality characteristics defined in Quality Model Division: ISO/IEC 25020. However, this standard only sets the requirements for the selection and construction of quality measures but does not describe which metrics should be applied in different cases (ISO/IEC 25020, 2007). Although ISO/IEC 25023 (called "measurement of system and software product quality") will define metrics to measure software products quality, this standard is not available yet. However, because this standard will revise ISO 9126-2 and ISO 9126-3, the structure of the metric concept used on these standards can be taken as reference.

For these reasons, the metric ontology proposed in this work focuses in the traditional metric definition (Fenton & Bieman, 2014; Kan, 2003; Pressman, 2010) incorporating some concepts related to the current normative.

4.1 | Domain description

A product metric gives the properties of the software product at any point of its development. According to ISO 9126, a metric is basically defined by the specification of its name, purpose, application method, measurement formula, interpretation, scale type, measure type, input to measurement, and target audience. The identification of the metric is given by its name. For this reason, the metric name must be related with the information obtained when the metric is applied. The purpose of the metric basically is expressed as the question to be answered by the application of the metric. The metric application method provides an outline of application whereas the measurement formula stipulates the mathematical expression used for the calculation and explains the meanings of the used data elements. The interpretation of the measured value supplies the range and preferred values. The scale type defines the dimension of the metric. Scale types used are nominal

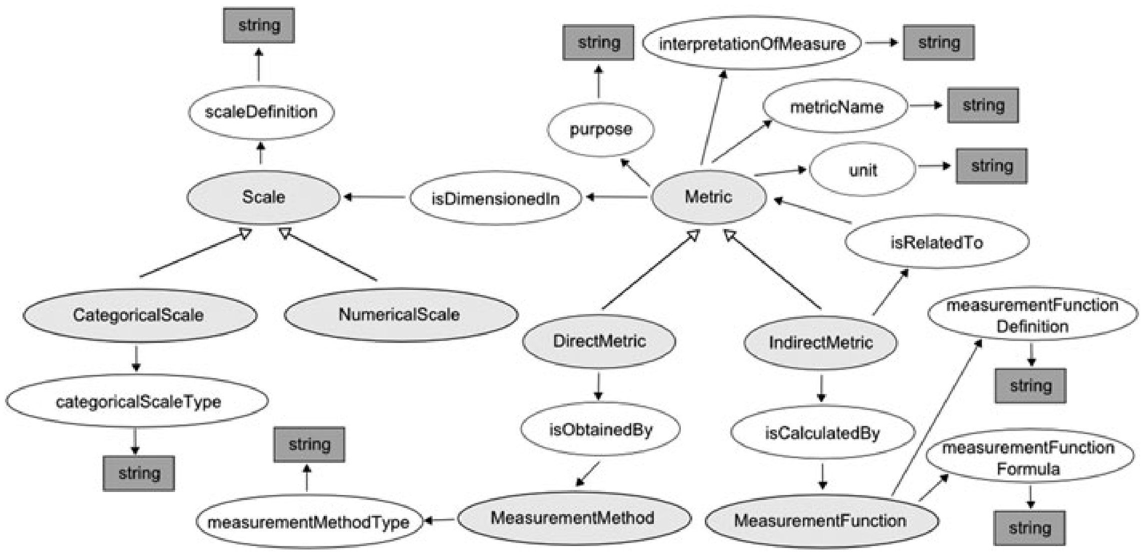


FIGURE 3 Metric base ontology

scale, ordinal scale, interval scale, ratio scale, and absolute scale. The measure type involves the specification of the way in which the metric is obtained. Types used are size type, time type, and count type. The input to the measurement process refers to the source of data used in the measurement. Finally, the target audience identifies the users of the measurement results.

Although all this concepts define a metric, only a set of them was used as part of the metric ontology. The selection was made taking into account the final objective: document the metrics related with different quality attributes that should be used to evaluate a specific software product. In order to do this, the input to measurement and the target audience are concepts not represented in the model. The reason to exclude the input to measurement involves to the need to know in which stage of the development process the metric will be applied. The documentation activity only represents the way to calculate and estimate the value of the measure. The specific source of information is not important in the context of this activity because the measurement process is not executed. This characteristic is only useful when the metric is used in a specific context (which happens when the process of measurement has place). On the other hand, the

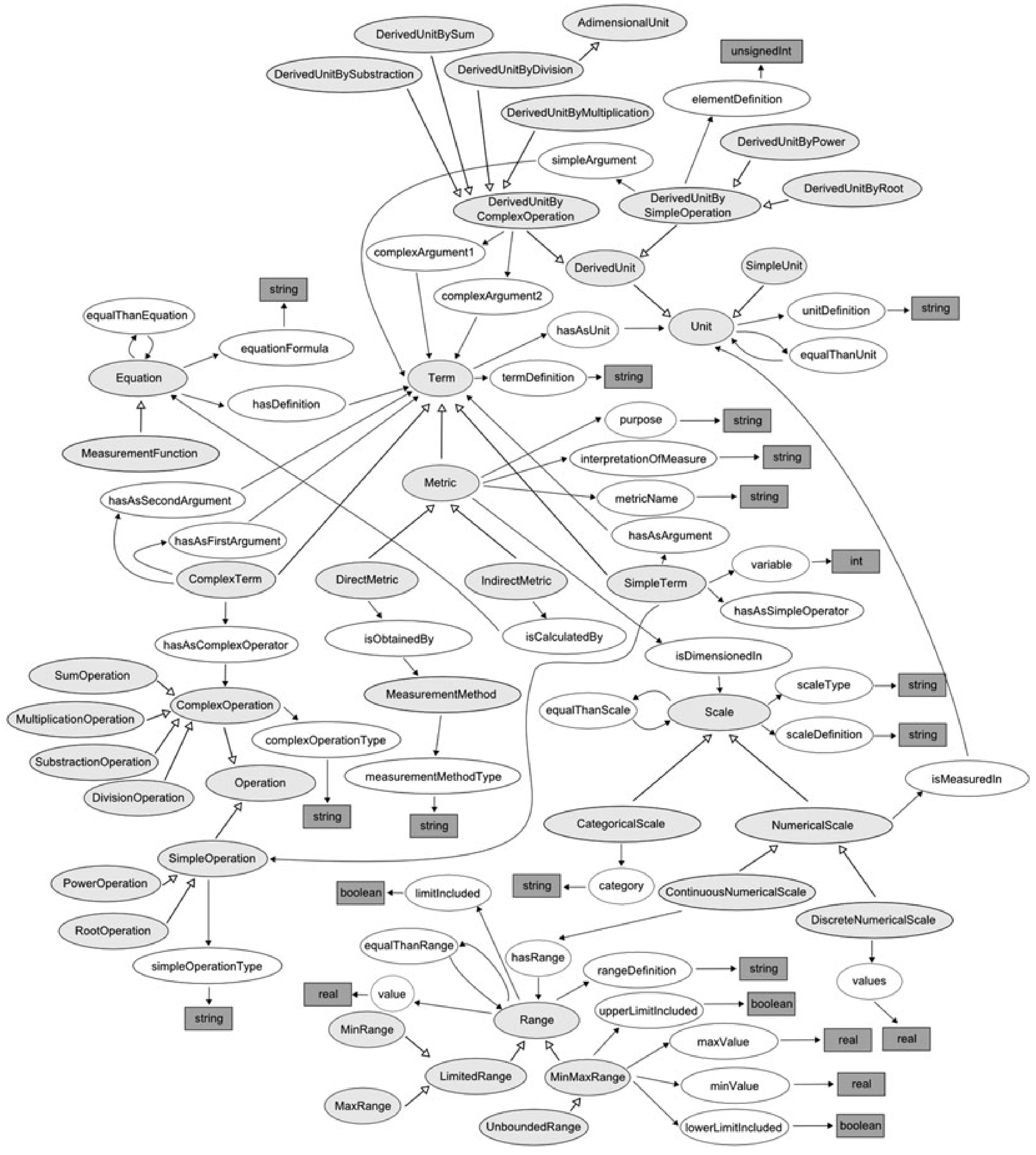


FIGURE 4 Metric ontology (final)

target audience is also a concept that shows a dependency with the use of the metric. Some metrics can be used by not identified audience, or worst, none of the identified audiences really uses the metric. The rest of the concepts were taken as they are defined to compose the semantic model.

4.2 | Semantic model

The development of the metric ontology was made in two phases. First, the development focused on the modeling of a base ontology. Then (in phase two), the focus was improving this ontology by refining some of the concepts previously identified.

Figure 3 shows the base ontology developed using as basis of the concepts identified in ISO 9126 and the approach presented in García et al. (2006). All the concepts refer to the characteristics detailed in the previous section, adding a few details in some on them. The attributes also represent properties of the main concepts. These elements have been modeled using the string data type.

However, the base ontology model is not complete. Some of its concepts are too general whereas some of its attributes are more important than what in the beginning seemed to be. Because of this, the model was refined. Figure 4 shows the final metric ontology. The improvements performed were substitution of the "unit" attribute by the "Unit" concept, modification of the "Scale" hierarchy, and incorporation of the "Equation" concept.

The identification of unit as a metric attribute brings two problems. First, a unit can be assigned into a metric with categorical scale. Second, a metric may have an adimensional unit. In order to fix these issues, the unit attribute was substituted by a new set of concepts, which derives from the Unit concept. The proposal of Rijgersberg, Wigham, and Top (2011) was taken as reference for this improvement process. The incorporated concepts include Simple Unit and Derived Unit. A "Simple Unit" represents a base unit (i.e., a unit that can only be obtained by definition). A "Derived Unit" represents a unit that is obtained operating other units. The operations used for the derivation can be simple or complex. A "Derived Unit By Simple Operation" represents units of operations that have only one variable argument. This category includes power and root operations ("Derived Unit By Power Operation" and "Derived Unit By Root Operation," respectively). Although mathematically, both operations are binary, the context of software metrics usually uses these operations as a mechanism to modify one variable by a numerical factor. For example, a metric that measures the structural complexity of a software module can be defined as $S(i) = f_{out}^2(i)$, where $f_{out}(i)$ is the module's expansion that is another metric. For this reason, the metric ontology represents these units as part of the hierarchy of simple operations. Finally, a "Derived Unit By Complex Operation" represents units of operations that have two variable arguments. This category involves sum, subtraction, multiplication, and division operations ("Derived Unit By Sum," "Derived Unit By Subtraction," "Derived Unit By Multiplication," and "Derived Unit By Division," respectively). Given that an adimensional unit can only be derived by a division operation, the "Adimensional Unit" concept derives from the "Derived Unit By Division" concept. All this new hierarchy of concepts is included in the semantic model using two relationships: "is-measured-in" and "has-as-unit." The first one represents the need of units in numerical scales whereas the second one is a derivative relation that only exists if the term is simple or complex or if the metric has a "Numerical Scale" concept assigned (see Equation 2). The Numerical Scale concept is useful to describe the numerical nature of the value obtained from the evaluation of a metric.

The hierarchy derived from the "Scale" concept (from the base ontology) was also modified in the final ontology using the approach described at Olsina and Martín (2003). Concepts and attributes were added, including "Discrete Numerical Scale" and "Continuous Numerical Scale" to represent specific types of numerical scales. The incorporation of this last concept brings with it the integration of the "Range" hierarchy in attempt to model the collection of valid values in a continuous scale. The Range hierarchy includes concepts to model lower and upper limits ("MinRange" and "MaxRange," respectively) and intervals ("MinMaxRange").

The final refinement of the base ontology refers to the incorporation of the Equation concept to the semantic model. Because the "Measurement Function" concept represents a mathematical formula composed of mathematical terms, operators, and variables, the final ontology propose a collection of concepts to model more appropriately this definition. To this purpose, the Equation concept was included. All measurement function is an equation composed of different types of terms ("Term" concept). Each term can be represented as a simple or complex mathematical operation with its arguments ("SimpleTerm" and "Complex Term," respectively). However, a term of an equation can be an expression that refers to a metric. Indirect metrics use other metrics to its own calculation process. For this reason, the "Metric" concept is included in the Term hierarchy. As complement of these concepts, the final ontology includes the "Operation" hierarchy with the objective to model the most frequent operations used in metrics. The main concepts proposed by Castro, Rico, and Castro (1995) were used as reference model for the final specification. The properties detailed in concepts "Simple Operation" and "Complex Operation" allow differentiating the specific type of operation.

The metric ontology's definition included some SWRL rules related with derivation of knowledge (Equations 2–4). These rules are useful to reason on individuals. Equation 2 describes an example rule that refers to one previously mentioned property: a unit must be assigned into a metric only if its scale is numerical, then the unit will be the one specified in the scale. Specifically, "?m" refers to an instance of Metric, "?s" refers to an instance of "NumericalScale," and "?u" refers to an instance of Unit. Then, if a metric "?m" and a numerical scale "?s" are related by the "isDimensionedIn" relationship as long as the numerical scale "?s" and unit "?u" are linked by the "isMeasuredIn" relationship, a new relation between the metric "?m" and the unit "?u" must be established: "hasAsUnit," In this case, the SWRL rule allows establishing a new relation between instances of specific concepts.

$$\begin{aligned}&\text{Metric}(?\text{m})\wedge\text{NumericalScale}(?\text{s})\wedge\text{isDimensionedIn}(?\text{m}, ?\text{s})\\&\quad\wedge\text{isMeasuredIn}(?\text{s}, ?\text{u})\wedge\text{Unit}(?\text{u})\rightarrow\text{hasAsUnit}(?\text{m}, ?\text{u}),\end{aligned}\tag{2}$$

$$\begin{aligned}&\text{Range}(?\text{r1})\wedge\text{rangeDefinition}(?\text{r1}, ?\text{d1})\wedge\text{Range}(?\text{r2})\\&\quad\wedge\text{rangeDefinition}(?\text{r2}, ?\text{d2})\wedge\text{equals}(?\text{d1}, ?\text{d2})\\&\qquad\rightarrow\text{equalThanRange}(?\text{r1}, ?\text{r2}),\end{aligned}\tag{3}$$

$$\begin{aligned}&\text{Equation}(?\text{e1})\wedge\text{equationFormula}(?\text{e1}, ?\text{f1})\wedge\text{Equation}(?\text{e2})\\&\quad\wedge\text{equationFormula}(?\text{e2}, ?\text{f2})\wedge\text{equals}(?\text{f1}, ?\text{f2})\\&\qquad\rightarrow\text{equalThanEquation}(?\text{e1}, ?\text{e2}).\end{aligned}\tag{4}$$

A similar case is given in the "equal-than" relationships. In order to identify equality between different instances of the same concept, the model includes SWRL rules that derivate the equal-than relations (equal-than-unit, equal-than-scale, equal-than-equation, and equal-than-range). These relations have the property of be reflexive, symmetric, and transitive. Its definition can be derived from the attributes related with the different instances. Then, these SWRL rules were added to the model in order to generate consistent individuals. Also, its incorporation allows a later comparison between different individuals that have the same properties. Equations 3 and 4 show as example the "equal-than-range" and "equal-than-equation" inference rule. Equation 3 uses "?r1," "?r2," "?d1," and "?d2" in order to refer individuals (i.e., concept instances). A similar definition is used in Equation 4 with "?e1," "?e2," "?f1," and "?f2." The predicates "Range()" and "Equation()" represent, respectively, the range and equation concept modeled in the metric ontology. The "rangeDefinition()" and "equationFormula()" elements formalize the "rangeDefinition" and "equationFormula" links proposed in the semantic model in order to establish a relation between individuals. Finally, "equalThanRange()" and "equalThatEquation()" are new relations between the existent individuals that can be derived using the proposed associations. Equation 3 shows that if the ranges ?r1 and ?r2 have the definitions ?d1 and ?d2 respectively and both definitions are equal, then the ranges ?r1 and ?r2 are equal. A similar approach is proposed in Equation 4 in order to determine if two equations are equal. Other SWRL rules were added to enrich operations types, adimensional units, and range delimitations. For space reasons, these rules are not included in the paper. The set of terms described in this work only refers to the set of rules presented. For these rules, an exhaustive definition of terms is not required.

The approach used to evaluate completeness over quality ontology was also used in metric ontology. However, given that there is none list of terms applicable to the domain, the model was check using literature (García et al., 2006; Olsina & Martín, 2003; Rijgersberg et al., 2011). Again, details about the check process are beyond the scope of this paper.

5 | SOFTWARE ONTOLOGY

The software ontology was developed in order to represent a software product specification. The main purpose of this ontology is to model the domain of software systems products including all the components than usually are develop along the development process.

5.1 | Domain description

A software product is a set of computer programs, procedures, and possibly associated documentation and data, designed for delivery to a specific user. It always includes the development of one or more computer programs. The development process carried out for the construction of each computer program usually involves the creation of different artifacts. An artifact is a product produced during the development of software that contains information of some part of it. Some artifacts help to describe the function, architecture, and design of the software whereas others are concerned with the process of development itself. All artifacts can be divided in a set of entities. An entity is an object that can be characterized by measuring its attributes. Therefore, an attribute is a measurable physical or abstract property of an entity. Formal definition specifies that an attribute is an inherent property or characteristic of an entity that can be distinguished quantitatively or qualitatively by human or automated means.

5.2 | Semantic model

Each of the main concepts identified in the software domain was transformed into an ontology concept. Concepts include "Software Product," "Computer Program," "Artifact," "Entity," and "Attribute." The links between the concepts were modeled as relationships. The name of these relationships describes the way in which the concepts are related (e.g., "is-divided-in" between the Artifact and Entity concepts). To allow a correct identification of the instances derived from a concept, the model includes some ontology attributes labeled as "name" and "type" according to the case (i.e., "attributeName" and "attributeType" for the Attribute concept). Figure 5 shows the semantic model developed.

6 | FINAL ONTOLOGY: INTEGRATION OF THE SEMANTIC MODELS

The ontologies described in Sections 2, 3, and 4 represent different related domains. To document a quality scheme for a software product is necessary to unify these domains in a single model that represents the relationships between them.

Figure 6 shows how the main concepts of each ontology are related. An attribute of a software product needs to be described by a quality subcharacteristic and has to be measured by a metric. In this

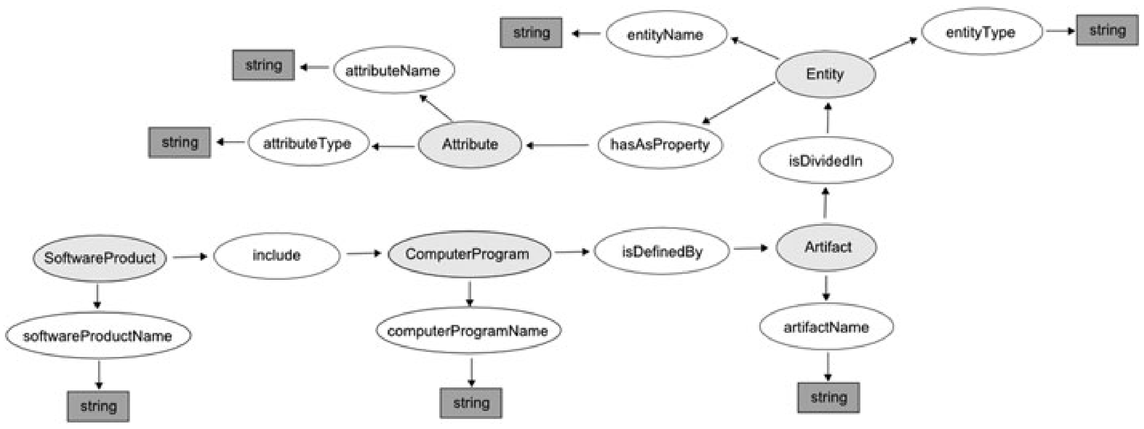


FIGURE 5 Software ontology

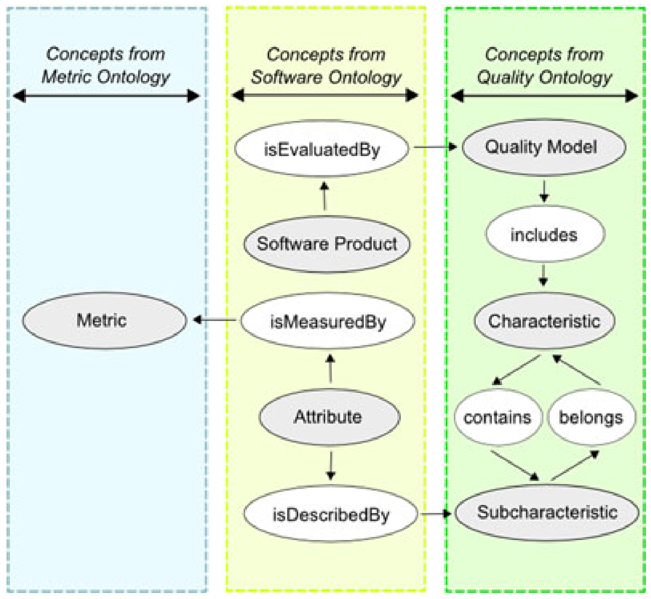


FIGURE 6 Relations between concepts of quality, metric, and software ontologies

sense, the three elements define one specification of the set of triples included at quality scheme. Therefore, for each quality subcharacteristic, the capability of a software product is determined by a set of internal attributes that can be measured.

To define a quality scheme for a software product is necessary to specify which artifacts have to be evaluated. Not all the artifacts generated in the development process have to be evaluated in terms of the product quality. Only the artifacts related with final product (directly or indirectly) should be associated with quality subcharacteristics and metrics. Two or more artifacts may have the same quality scheme. However, they also may have different quality schemes assigned. The differences depend on the stage of the development process in which the quality scheme of the artifact is specified. If its specification is made in early stages (as a first approach to the quality evaluation), the scheme probably will not incorporate all the quality features required. In contrast, if the quality specification is made with the creation of the artifact, the scheme probably will be attached to the vision of the developer. None of these situations is ideal. In fact, the recommendation is to specify one quality scheme in early stages of the development process and then refine it as the process is carried out. A quality scheme developed following this mechanism will be much more specific than one developed according the other options. In this context, a single quality scheme for a software product may be derived by unifying all the quality schemes defined for its artifacts.

6.1 | Implementation

The designed ontologies were implemented using Protégé.[1] Protégé is an extensible tool that provides a plug-and-play environment, which makes it a flexible base for rapid prototyping and application development (Knublauch et al., 2005). Protégé ontologies can be exported into different formats including Resource Description Framework (RDF) Schema and OWL. Ontologies defined in OWL 2 provide classes, properties, individuals, and data values, which are primarily exchanged as RDF documents.

Figure 7 resumes the implementation process carried out to. In order to obtain the final ontology, the domain ontologies developed were implemented in OWL format. Each semantic model was built individually in order to maintain the original designs. All the elements defined in the ontologies were specified in English and Spanish to allow multiple language support. Also, an annotation component was included for each defined element with the purpose to describe the modeled concept. The implementations made for the domain ontologies were imported in a new document with aim to model the final ontology. This new model requires the creation of the relationships defined in the final ontology (Figure 6) to relate the semantic models.

Figure 8 shows the final ontology implemented over the logical view window of Protégé. The selected class (shown in the left side of the view) corresponds to the Attribute concept whereas the sections presented in the right side of the view show the properties related with this concept. Over the annotation, section two properties are used. The label property represents a synonymous (in other language, in this case, Spanish) of the concept modeled whereas the comment property represents a colloquial definition of the concept. Both properties are based on the string data type. On the other hand, in the description section, the set of axioms that specify the concept is included. Each "Equivalent to" axiom represents one link from the "Attribute" class to another class or data type, that is, the relationships modeled in order to relate the concept with the rest of the ontology's concepts. For example, the "isDescribedBy" relationship relates the Attribute concept with the Subcharacteristic concept whereas the attributeName relationship relates the Attribute concept with the string data type. The axioms included in this section match with the links modeled for the Attribute concept in Figures 5 and 6.

The SWRL rules defined in the ontologies were specified using Protégé. Figure 9 shows the rules implemented over the window of Protégé. The first rule is equivalent to the SWRL rule exemplified in Equation 1 but instead of describing the security characteristic, it describes the reliability characteristic. The rest of the rules are applicable to other aspects of the Quality ontology and to the description of the Metric ontology.

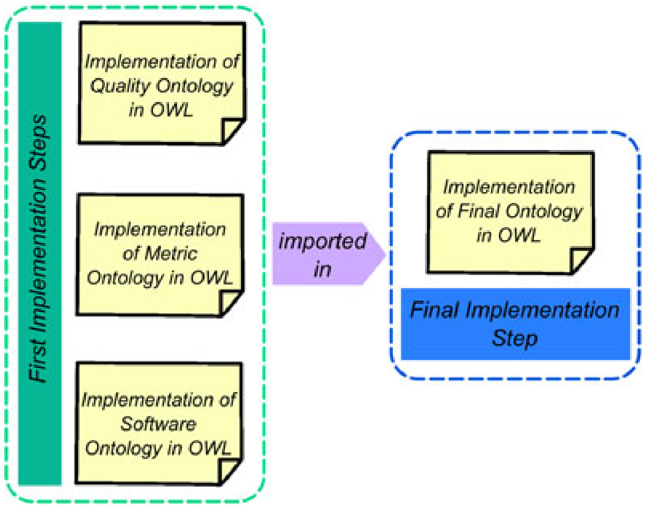
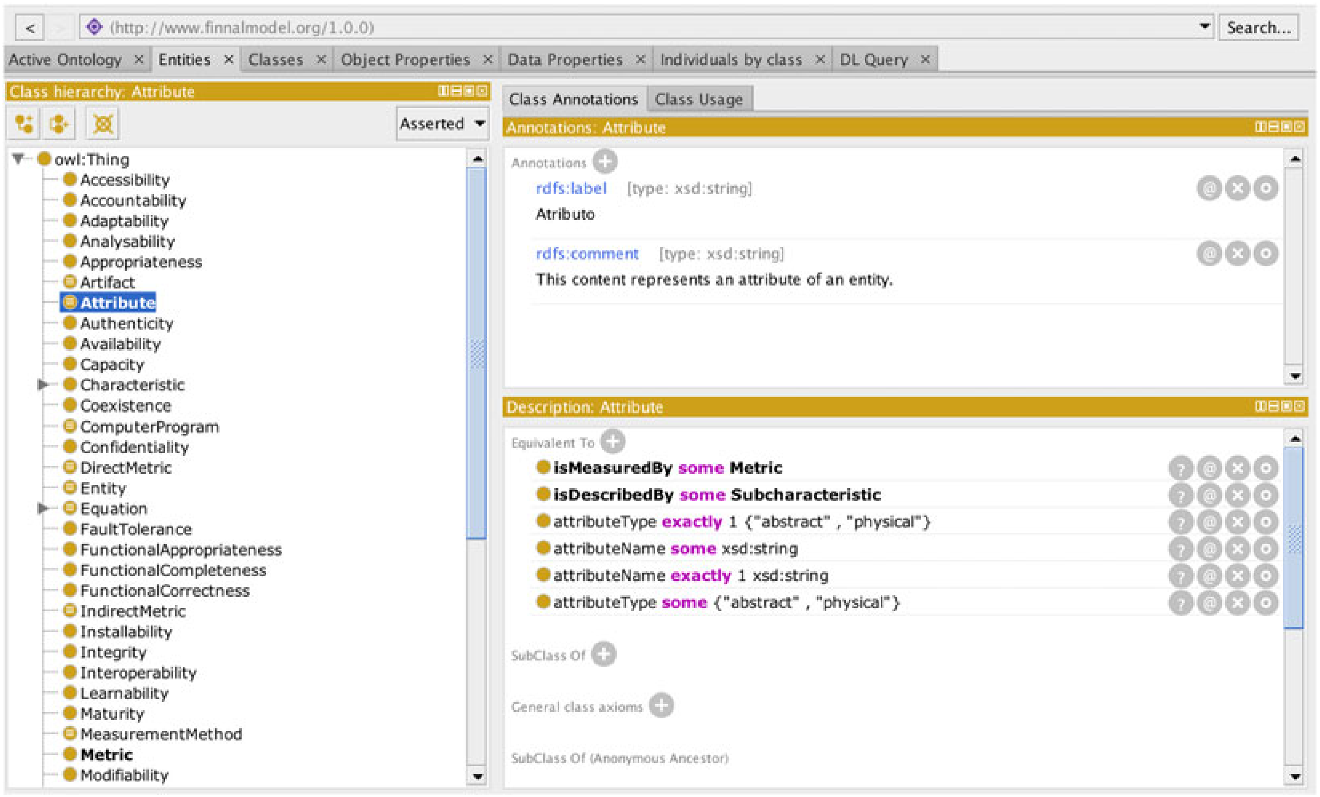

First Implementation Steps
Implementation of Quality Ontology in OWL
Implementation of Metric Ontology in OWL
Implementation of Software Ontology in OWL
imported in
Implementation of Final Ontology in OWL
Final Implementation Step

FIGURE 7 Implementation process carried out to build the final ontology

[1]Available at: http://protege.stanford.edu/



owl:Thing
- Accessibility
- Accountability
- Adaptability
- Analysability
- Appropriateness
- Artifact
- **Attribute**
- Authenticity
- Availability
- Capacity
- Characteristic
- Coexistence
- ComputerProgram
- Confidentiality
- DirectMetric
- Entity
- Equation
- FaultTolerance
- FunctionalAppropriateness
- FunctionalCompleteness
- FunctionalCorrectness
- IndirectMetric
- Installability
- Integrity
- Interoperability
- Learnability
- Maturity
- MeasurementMethod
- **Metric**
- Modifiability

Annotations

rdfs:label [type: xsd:string]
Atributo

rdfs:comment [type: xsd:string]
This content represents an attribute of an entity.

Equivalent To

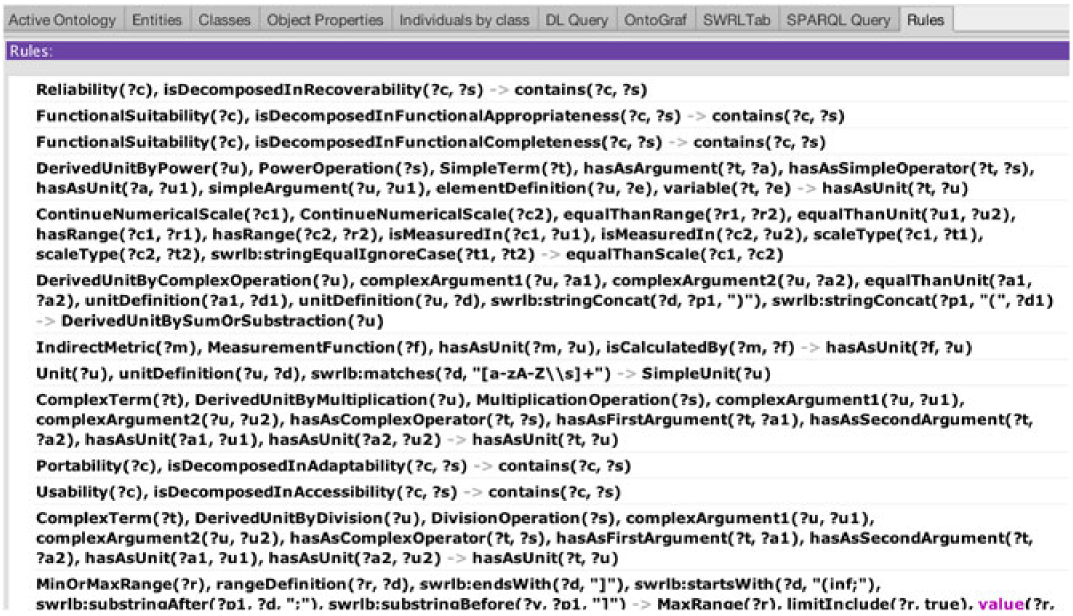
- **isMeasuredBy some Metric**
- **isDescribedBy some Subcharacteristic**
- attributeType exactly 1 {"abstract" , "physical"}
- attributeName some xsd:string
- attributeName exactly 1 xsd:string
- attributeType some {"abstract" , "physical"}

SubClass Of

General class axioms

SubClass Of (Anonymous Ancestor)

FIGURE 8 Final ontology implemented over the logical view window of Protégé



```
Reliability(?c), isDecomposedInRecoverability(?c, ?s) -> contains(?c, ?s)
FunctionalSuitability(?c), isDecomposedInFunctionalAppropriateness(?c, ?s) -> contains(?c, ?s)
FunctionalSuitability(?c), isDecomposedInFunctionalCompleteness(?c, ?s) -> contains(?c, ?s)
DerivedUnitByPower(?u), PowerOperation(?s), SimpleTerm(?t), hasAsArgument(?t, ?a), hasAsSimpleOperator(?t, ?s),
hasAsUnit(?a, ?u1), simpleArgument(?u, ?u1), elementDefinition(?u, ?e), variable(?t, ?e) -> hasAsUnit(?t, ?u)
ContinueNumericalScale(?c1), ContinueNumericalScale(?c2), equalThanRange(?r1, ?r2), equalThanUnit(?u1, ?u2),
hasRange(?c1, ?r1), hasRange(?c2, ?r2), isMeasuredIn(?c1, ?u1), isMeasuredIn(?c2, ?u2), scaleType(?c1, ?t1),
scaleType(?c2, ?t2), swrlb:stringEqualIgnoreCase(?t1, ?t2) -> equalThanScale(?c1, ?c2)
DerivedUnitByComplexOperation(?u), complexArgument1(?u, ?a1), complexArgument2(?u, ?a2), equalThanUnit(?a1,
?a2), unitDefinition(?a1, ?d1), unitDefinition(?u, ?d), swrlb:stringConcat(?d, ?p1, ")"), swrlb:stringConcat(?p1, "(", ?d1)
-> DerivedUnitBySumOrSubstraction(?u)
IndirectMetric(?m), MeasurementFunction(?f), hasAsUnit(?m, ?u), isCalculatedBy(?m, ?f) -> hasAsUnit(?f, ?u)
Unit(?u), unitDefinition(?u, ?d), swrlb:matches(?d, "[a-zA-Z\\s]+") -> SimpleUnit(?u)
ComplexTerm(?t), DerivedUnitByMultiplication(?u), MultiplicationOperation(?s), complexArgument1(?u, ?u1),
complexArgument2(?u, ?u2), hasAsComplexOperator(?t, ?s), hasAsFirstArgument(?t, ?a1), hasAsSecondArgument(?t,
?a2), hasAsUnit(?a1, ?u1), hasAsUnit(?a2, ?u2) -> hasAsUnit(?t, ?u)
Portability(?c), isDecomposedInAdaptability(?c, ?s) -> contains(?c, ?s)
Usability(?c), isDecomposedInAccessibility(?c, ?s) -> contains(?c, ?s)
ComplexTerm(?t), DerivedUnitByDivision(?u), DivisionOperation(?s), complexArgument1(?u, ?u1),
complexArgument2(?u, ?u2), hasAsComplexOperator(?t, ?s), hasAsFirstArgument(?t, ?a1), hasAsSecondArgument(?t,
?a2), hasAsUnit(?a1, ?u1), hasAsUnit(?a2, ?u2) -> hasAsUnit(?t, ?u)
MinOrMaxRange(?r), rangeDefinition(?r, ?d), swrlb:endsWith(?d, "]"), swrlb:startsWith(?d, "(inf;"),
swrlb:substringAfter(?p1, ?d, ";"), swrlb:substringBefore(?v, ?p1, "]") -> MaxRange(?r), limitInclude(?r, true), value(?r,
```

FIGURE 9 SWRL rules implemented using Protégé. SWRL, Semantic Web Rule Language

6.2 | Detection of modeling problems

The ontology evaluation process is an important activity that allows the detection of potential problems derived by a lack of modeling experience. This process includes the verification, validation, and assessment of the ontology developed. The verification activity refers to the correct implementation (i.e., if the ontology has been correctly developed). The detection of problems in the model can be realized as part of this activity, along with the fix of the problems detected.

In order to verify the final ontology to guarantee its correctness over the common modeling errors, each of the domain specific ontologies developed must be evaluated. The verification was made using "OOPS!" tool.[2] OOPS! (OntOlogy Pitfall Scanner) is a web application that helps to detect some of the most common pitfalls that appear when ontologies are developed. Its purpose is to help developers during the diagnose activity of the validation process by describing the

[2]Available at: http://oops.linkeddata.es/

problems detected in a specific ontology. Because not all pitfalls are equally important, the tool attached an importance level to each problem. There are three possible levels: critical, important, and minor. A critical pitfall is one that is crucial to correct or, otherwise, it could affect the consistency, reasoning, and applicability of the ontology. An important pitfall is one that, though not critical for ontology function, it is important to correct. Finally, a minor pitfall is not really a problem but its correction will improve the ontology. For each detected problem, the tool also shows an example and indicates which part of the semantic model is the one that causes the issue. The reparation activity of the validation process is not supported by the tool. The problems indicated as troubles should be fixed by the developer (taking into account the description of the problem detected).

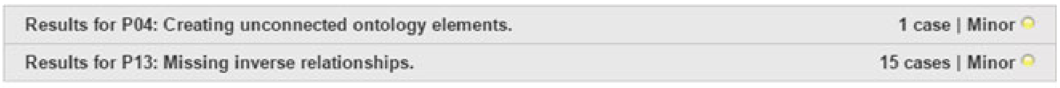
The verification of the ontologies developed was made using the OWL documents. Figure 10 shows the list of pitfalls and elements detected as problems in the metric (Figure 10a), quality (Figure 10b), and software (Figure 10c) ontologies, respectively. As it can be seen, in all cases, the pitfalls detected were classified with a minor importance level.

In all the ontologies, the "missing inverse relationships" problem (pitfall identified as P13) was detected. However, this issue is not a real problem. The need of inverse relationships in an ontology is related to the nature of the represented domain. In the software and metrics domain, the relationships between concepts are unidirectional and, therefore, is not necessary to define inverse relationships in the ontology. The same occurs in the product quality model defined by ISO/IEC 25010. But, in this case, some relationships require both navigability senses and, therefore, some inverse relationships were specified. An extra pitfall was detected in the metric ontology. The problem refers to the creation of an element that is isolated from the others. Specifically, this issue is related to the "Operation" concept. The Operation concept is a representation that englobes all the mathematical operations that usually are used in the metrics contexts. However, the links between the operations and its arguments were modeled between each specific operation and the term and unit corresponding. Therefore, the main concept is only used to model the common characteristics of different operations. The lack of connections with other concepts is not a real problem.

The verification of the final ontology also was performed using OOPS!. However, because its OWL file only saves the classes, object properties, and data properties defined in the final ontology, the information associated with the imported ontologies is not available to evaluation. Therefore, the tool only examines the relationships defined in order to relate the three developed models. For this reason, each domain ontology was evaluated individually. The result of this verification only showed the pitfall of missing inverse relationships. As in the previous cases, this pitfall is not a real problem because the relationships defined do not require bidirectional navigability.

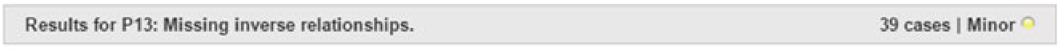
6.3 | Evaluation

It is difficult to quantify the quality of domain ontologies due to the absence of a formal method that allows its evaluation. A lot of research work on this area focus on the analyses of the structural dimensions of the ontology (e.g., number of classes, relations, and instances defined), given a way to estimate the complexity of the semantic model. In order to evaluate the complexity of the final ontology, the set of metrics used by Vegetti, Leone, and Henning (2011) was selected. These metrics (Table 2) are used to estimate the structural characteristics of the ontol-

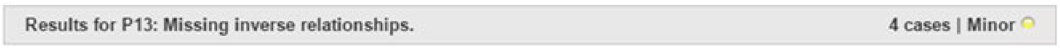

Results for P04: Creating unconnected ontology elements. 1 case | Minor ●

Results for P13: Missing inverse relationships. 15 cases | Minor ●

(a): *Metric ontology evaluation – Pitfalls detected.*


Results for P13: Missing inverse relationships. 39 cases | Minor ●

(b): *Quality ontology evaluation – Pitfalls detected.*


Results for P13: Missing inverse relationships. 4 cases | Minor ●

(c): *Software ontology evaluation – Pitfalls detected.*

FIGURE 10 Pitfalls detected by the OOPS! tool in the ontologies developed. OOPS!, OntOlogy Pitfall Scanner

TABLE 2 Set of metrics applied to the final ontology to estimate its complexity in terms of structural characteristics

Definition	Abbrev.	Description	Value
Number of classes	NOC	Count of the number of classes defined in the ontology.	87
Number of relations	NOR	Count of the number of relationships defined in the ontology.	140
Number of root classes	NORC	Number of classes without super classes.	15
Number of leaf classes	NOLC	Number of classes without subclasses.	72
Relationship richness	RR = |P|/(|H| + |P|)	Variety of relationships in the ontology. Consider |P| as the number of noninheritance relationships. |H| as the number of inheritance relationships.	68/140 = 0.485
Inheritance richness	IR = |H|/|NOC|	Average number of subclasses per class. Consider |NOC| as the total number of classes |H| as the number of hierarchical relations.	72/87 = 0.827
Depth of the subsumption hierarchy	DOSH	Length of the longest path from a given class C to the root class in a given ontology subsumption hierarchy.	5
Attribute richness	AR = |NAT|/|NOC|	Average number of attributes per class. Consider |NAT| as the number of attributes for all classes. |NOC| as the number of classes.	32/87 = 0.367

TABLE 3 Set of queries developed to obtain information about the composition of a quality scheme

Ans.	Id.	Question
Metric	Q1	Which metrics are useful to evaluate the "X" quality characteristic?
	Q2	Which metrics are useful to evaluate the "Y" quality subcharacteristic?
	Q3	How many different metrics have been related with the "X" quality characteristic?
	Q4	How many different metrics have been related with the "Y" quality subcharacteristic?
Characteristic, subcharacteristic, or quality scheme	Q5	To which quality characteristic is associated the "Z" metric?
	Q6	How many times has related the "Z" metric with the "X" quality characteristic?
	Q7	To which quality characteristic is more often associated the "Z" metric?
	Q8	To which quality subcharacteristic is associated the "Z" metric?
	Q9	How many times has related the "Z" metric with the "Y" quality subcharacteristic?
	Q10	To which quality subcharacteristic is more often associated the "Z" metric?
	Q11	Which is the quality scheme related to the "V" artifact of the "W" software product?
	Q12	Which is the quality scheme related to the "W" software product?
Artifact or computer program	Q13	To which artifact has been associated the "Z" metric?
	Q14	To which computer program has been associated the "Z" metric?
	Q15	How many artifacts have been associated the "Z" metric?
	Q16	How many computer programs have been associated the "Z" metric?

ogy. It is important to note that quantitative measurement of complexity can help ontology development but it cannot be used to prove completeness, coherence, correctness, and reusability features.

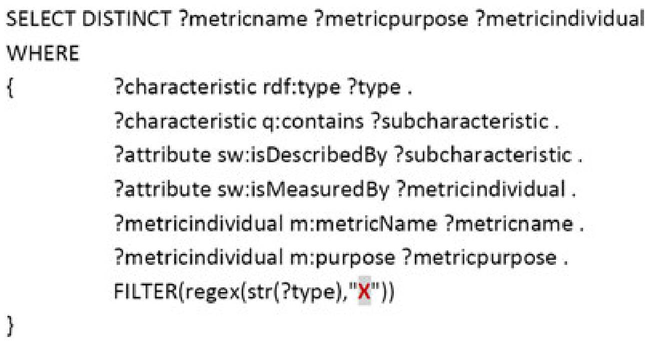
The results of applying these metrics to the final ontology are summarized in the "Value" column. Given that each individual domain ontology contains 42, 40, and 5 classes (respectively for quality, metric, and software ontologies), the total count of classes in the final ontology is 87. These 87 classes are divided in 72 leaf classes and 15 root classes. On the other hand, the final ontology contains 140 relations, from which 72 correspond to hierarchical relationships and 68 to other types of relationships. By computing the other metrics, the final ontology has an IR of 0.827 which together with the DOSH value indicate that the proposed ontology is of a vertical nature. This means that represents knowledge of a specific domain, allowing to instantiate schemes that fit to the quality specification. In consequence, the value of AR gives as average 0.367 attributes per concept. This average shows that the attributes of the ontology are one of the characteristics that allow restricting the domain. Finally, the RR is very close to the average. A value of 48.5% implies that the number of hierarchical relationships is a bit greater than the number of the other kind of associations. In this sense, the ontology maintains an adequate balance between inheritance relations and associations.

6.4 | SPARQL queries

SPARQL (SPARQL Protocol and RDF Query Language) is a semantic query language for databases, able to retrieve and manipulate data stored in RDF format (and therefore OWL 2 documents). The results of SPARQL queries can be result sets or RDF graphs.

In order to obtain information about the composition of a quality scheme, a set of questions was designed and implemented. Sixteen questions were developed. Each question was classified in one category according to the type of answer expected. The categories used are metric, characteristic, subcharacteristic, or quality scheme, and artifact or computer program. Table 3 resumes the questions developed.

New questions can be added to the set of proposed queries if and only if the model has the information required to reply them. In other case, the model should be restructured in order to include the missing content.

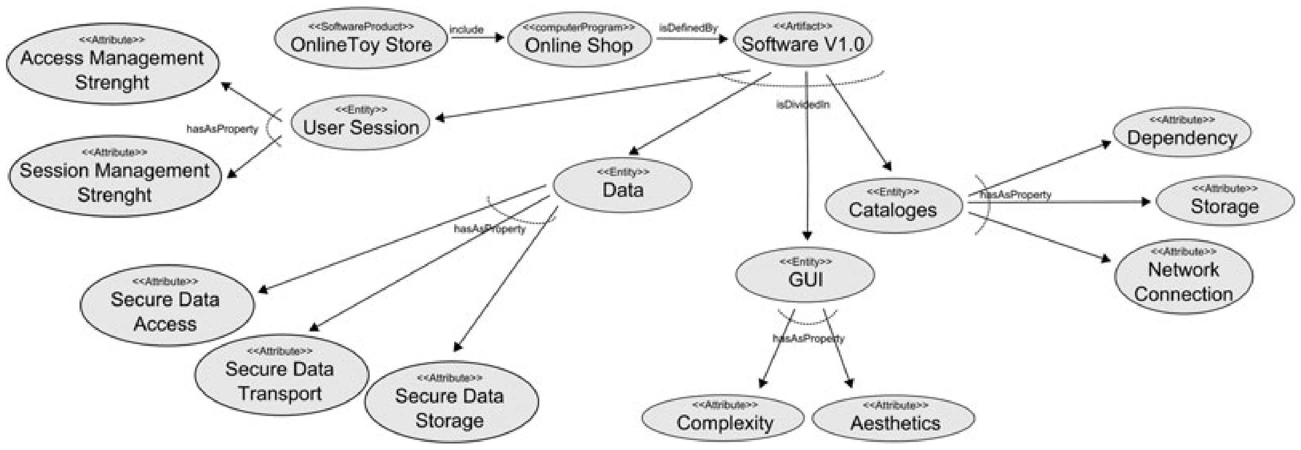


```
SELECT DISTINCT ?metricname ?metricpurpose ?metricindividual
WHERE
{         ?characteristic rdf:type ?type .
          ?characteristic q:contains ?subcharacteristic .
          ?attribute sw:isDescribedBy ?subcharacteristic .
          ?attribute sw:isMeasuredBy ?metricindividual .
          ?metricindividual m:metricName ?metricname .
          ?metricindividual m:purpose ?metricpurpose .
          FILTER(regex(str(?type),"X"))
}
```

FIGURE 11 SPARQL query that models the question Q1

TABLE 4 Functional requirements, quality goals, and specific quality requirements related with the online toy store

Functional requirement	Quality goal	Quality requirement (subcharacteristic)
Log-in	Secure access and user validation	Authenticity Integrity
	Friendly interface	User interface aesthetics Learnability
Create account	Communication between other systems	Interoperability
	Friendly interface	User interface aesthetics Learnability
	Secure access and user validation	Authenticity Integrity
Search for toys	Communication between catalogues at any time	Interoperability Availability
	Fast responses	Time behavior
	Friendly interface	User interface aesthetics Learnability
Checkout	Fast communication between other systems	Time behavior Interoperability
	Reliable and secure payment operations	Confidentiality
	Friendly interface	User interface aesthetics Learnability



<<SoftwareProduct>> OnlineToy Store
include
<<computerProgram>> Online Shop
isDefinedBy
<<Artifact>> Software V1.0
<<Attribute>> Access Management Strenght
<<Entity>> User Session
hasAsProperty
<<Attribute>> Session Management Strenght
isDividedIn
<<Entity>> Data
<<Entity>> Cataloges
<<Attribute>> Dependency
<<Attribute>> Storage
<<Attribute>> Network Connection
<<Entity>> GUI
<<Attribute>> Secure Data Access
<<Attribute>> Secure Data Transport
<<Attribute>> Secure Data Storage
<<Attribute>> Complexity
<<Attribute>> Aesthetics

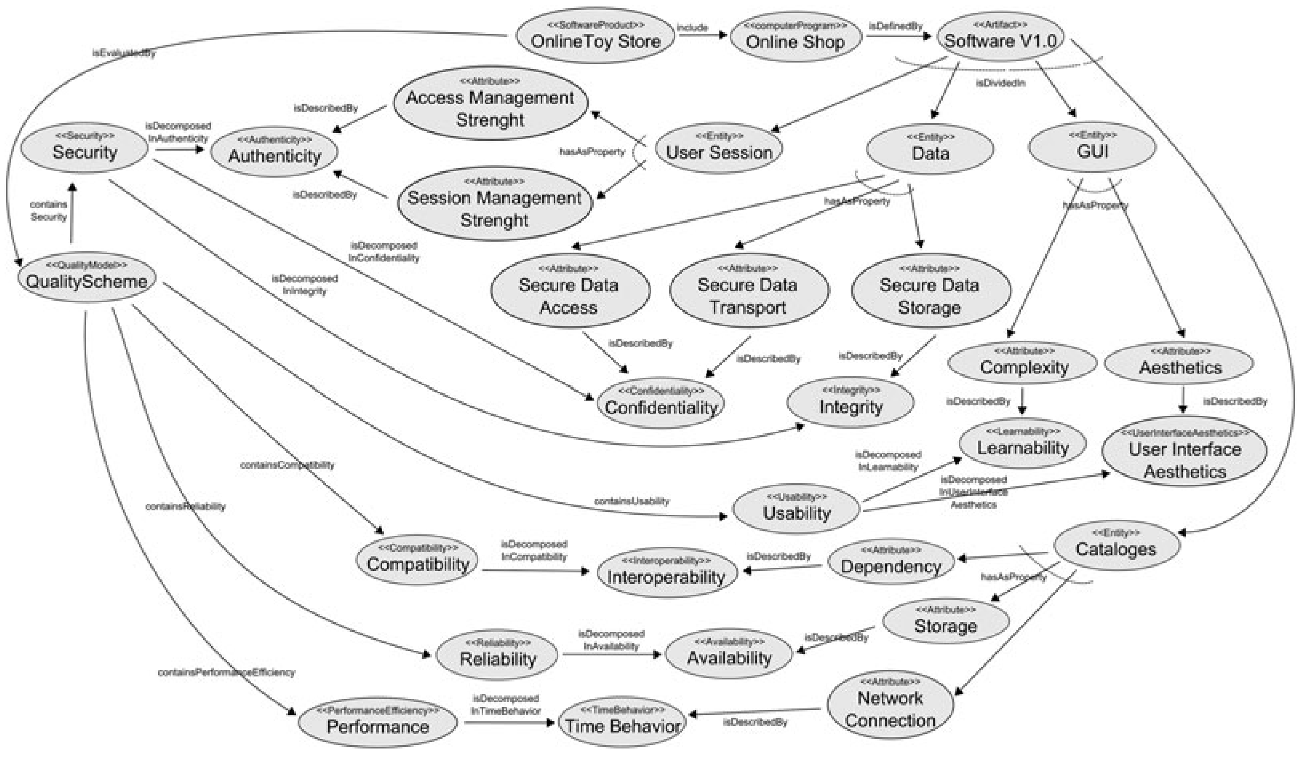
FIGURE 12 Software product representation of the online toy store using the final ontology developed

With aims to enrich the final ontology, each question was implemented as a query using SPARQL language. Figure 11 shows as example the SPARQL query that models the question Q1. The query contains three main clauses: SELECT, WHERE, and FILTER. The SELECT clause defines three variables (?metricname, ?metricpurpose, and ?metricindividual) from which a valid RDF triple could be created in the WHERE clause. In the first line of the WHERE clause for all the individuals, its type is retrieved. Then, in the second line, only for the individuals in which the contains relationship exists, the set of subcharacteristics contained is retrieved. The third line obtains the attributes associated with the subcharacteristics found previously, whereas the fourth line obtains the metrics that measure the attributes found. Finally, in the fifth and sixth lines, the name and the purpose of each metric found are retrieved. The FILTER clause helps with the search specifying the type of characteristic to evaluate.

The rest of the questions were transformed in SPARQL queries in the same way that Q1.

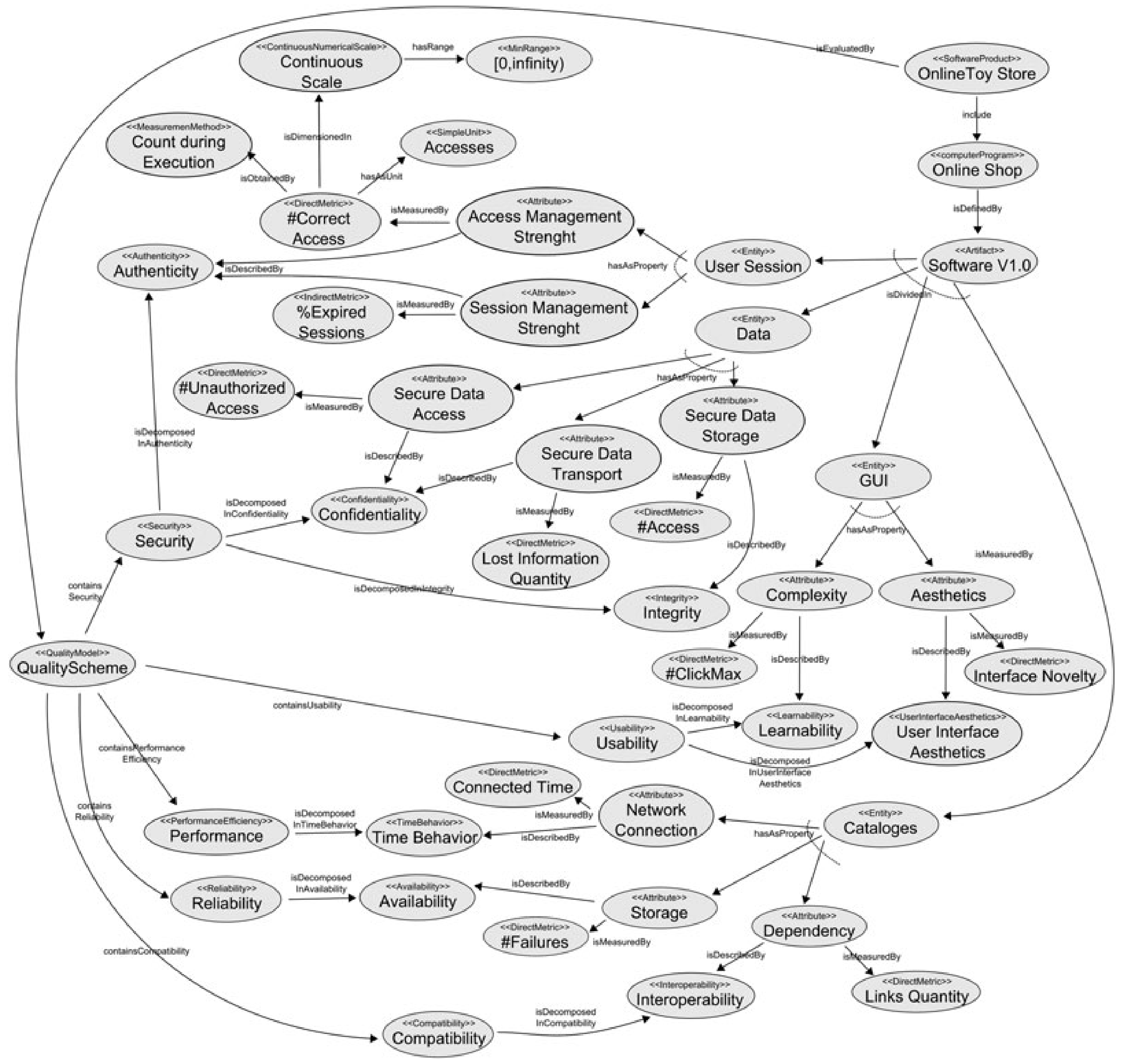
7 | CASE STUDY: INSTANTIATING THE FINAL ONTOLOGY

In order to exemplify the creation of a quality scheme using the ontology developed as support mechanism, a case study is presented. The case study proposed is based on a simplified version of an online toy store (an e-commerce web-based application). This case is introduced in previous study (Castillo, Losavio, Matteo, & Boegh, 2010) with aim to obtain a common terminology to develop models associated with requirements, aspects, and quality of a software product.



<<SoftwareProduct>> OnlineToy Store
include
<<computerProgram>> Online Shop
isDefinedBy
<<Artifact>> Software V1.0
isEvaluatedBy
isDividedIn
<<Attribute>> Access Management Strenght
isDescribedBy
<<Security>> Security
isDecomposed InAuthenticity
<<Authenticity>> Authenticity
hasAsProperty
<<Entity>> User Session
<<Entity>> Data
<<Entity>> GUI
<<Attribute>> Session Management Strenght
contains Security
<<QualityModel>> QualityScheme
isDecomposed InConfidentiality
isDecomposed InIntegrity
<<Attribute>> Secure Data Access
<<Attribute>> Secure Data Transport
<<Attribute>> Secure Data Storage
<<Attribute>> Complexity
<<Attribute>> Aesthetics
<<Confidentiality>> Confidentiality
<<Integrity>> Integrity
<<Learnability>> Learnability
<<UserInterfaceAesthetics>> User Interface Aesthetics
isDecomposed InLearnability
isDecomposed InUserInterface Aesthetics
containsCompatibility
containsUsability
<<Usability>> Usability
containsReliability
<<Compatibility>> Compatibility
isDecomposed InCompatibility
<<Interoperability>> Interoperability
<<Attribute>> Dependency
<<Entity>> Cataloges
<<Attribute>> Storage
<<Reliability>> Reliability
isDecomposed InAvailability
<<Availability>> Availability
containsPerformanceEfficiency
<<PerformanceEfficiency>> Performance
isDecomposed InTimeBehavior
<<TimeBehavior>> Time Behavior
<<Attribute>> Network Connection

FIGURE 13 Software product and quality model representation of the online toy store using the final ontology developed



<<ContinuousNumericalScale>> Continuous Scale
hasRange
<<MinRange>> [0,infinity)
isEvaluatedBy
<<SoftwareProduct>> OnlineToy Store
include
<<MeasuremenMethod>> Count during Execution
isDimensionedIn
<<SimpleUnit>> Accesses
isObtainedBy
hasAsUnit
<<computerProgram>> Online Shop
isDefinedBy
<<DirectMetric>> #Correct Access
isMeasuredBy
<<Attribute>> Access Management Strenght
<<Authenticity>> Authenticity
isDescribedBy
hasAsProperty
<<Entity>> User Session
<<Artifact>> Software V1.0
isDividedIn
<<IndirectMetric>> %Expired Sessions
isMeasuredBy
<<Attribute>> Session Management Strenght
<<Entity>> Data
<<DirectMetric>> #Unauthorized Access
isMeasuredBy
<<Attribute>> Secure Data Access
hasAsProperty
<<Attribute>> Secure Data Storage
isDecomposed InAuthenticity
isDescribedBy
<<Attribute>> Secure Data Transport
isDescribedBy
isMeasuredBy
<<Entity>> GUI
isDecomposed InConfidentiality
<<Confidentiality>> Confidentiality
isMeasuredBy
<<DirectMetric>> #Access
<<Security>> Security
<<DirectMetric>> Lost Information Quantity
isDescribedBy
hasAsProperty
isMeasuredBy
isDecomposedInIntegrity
<<Integrity>> Integrity
<<Attribute>> Complexity
<<Attribute>> Aesthetics
contains Security
isMeasuredBy
isMeasuredBy
<<QualityModel>> QualityScheme
<<DirectMetric>> #ClickMax
isDescribedBy
isDescribedBy
<<DirectMetric>> Interface Novelty
containsUsability
isDecomposed InLearnability
<<Learnability>> Learnability
<<UserInterfaceAesthetics>> User Interface Aesthetics
containsPerformance Efficiency
<<Usability>> Usability
isDecomposed InUserInterface Aesthetics
<<DirectMetric>> Connected Time
contains Reliability
isDecomposed InTimeBehavior
isMeasuredBy
<<Attribute>> Network Connection
<<Entity>> Catalogues
<<PerformanceEfficiency>> Performance
<<TimeBehavior>> Time Behavior
isDescribedBy
hasAsProperty
isDecomposed InAvailability
isDescribedBy
<<Reliability>> Reliability
<<Availability>> Availability
<<Attribute>> Storage
<<Attribute>> Dependency
<<DirectMetric>> #Failures
isMeasuredBy
containsCompatibility
isDescribedBy
isMeasuredBy
<<Interoperability>> Interoperability
<<DirectMetric>> Links Quantity
isDecomposed InCompatibility
<<Compatibility>> Compatibility

FIGURE 14 Quality scheme developed by the instantiation of the final ontology in order to describe the for online toy store case study

TABLE 5 Description of the instance "%ExpiredSession"—indirect metric instantiation

Element	Class	Object properties	
%ExpiredSessions	Indirect metric	isCalculatedBy isDimentionedIn	Session/ExpiredSessions %Scale
Session/ExpiredSessions	Measurement function	hasDefinition	FormulaExpiredSessions
FormulaExpiredSessions	Complex term	hasAsUnit hasAsFirstArgument hasAsSecondArgument hasAsComplexOperator	%Unit Session Expired Session Division
%Unit	Adimensional unit	complexArgument1 complexArgument2	Session unit Session unit
Session unit	Simple unit		
%Scale	ContinuousNumericalScale	hasRange	[0,1]
Division	DivisionOperation		
Session	Direct metric	(see Table 6)	
ExpiredSession	Direct metric	(see Table 7)	
[0,1]	MinMaxRange		

7.1 | Description

In the online toy store, the customers (web user) can buy a toy online by registering an account and specifying a log-in. The software system must provide a set of options to browse through the toy store online catalogue. The customers can add or remove toys to its shopping car by using a selection mechanism.

When a customer wants to checkout, its goal is to buy the current contents of their shopping car (including paying for the content and arranging the delivery of the toys). A customer that has generated an order can check its status and even can cancel it (only if it has not been processed).

Besides the functional operations, the demands of the customers include secure transactions, fast processing, and easy browsing through the catalogue and also require that the system operates 365 days of the year.

7.2 | Functional requirements and quality goals

Castillo et al. (2010) identifies the main functional concerns and related quality goals of the online toy store from the case study description. Their work also defines specific quality requirements related with each quality goal identified. However, the defined quality requirements refer to the quality model proposed in ISO/IEC 9126-1 (i.e., the quality standard used as base of ISO/IEC 25010). Therefore, the definition of the specific quality requirements was modified in order to refer to the quality model of ISO/IEC 25010. Table 4 summarizes the functional requirements, quality goals, and quality requirements identified.

7.3 | Quality scheme

In order to represent the quality scheme of the online toy store using the ontology developed, it is necessary to explicit its composition in terms of software product (i.e., using the elements defined in the software ontology). With aim to capture the quality requirements identified, only the attributes related with these requirements are included. The limitation in a unique artifact was adopted in order to obtain a smaller quality scheme. However, the ontology allows defining multiple artifacts in relation to one computer program. Figure 12 illustrates the instantiation made to represent the case study. The diagram only includes instances of classes and object properties.

Because each quality requirement identified refers to a specific subcharacteristic and each subcharacteristic belongs to a specific characteristic, the instantiation of the quality ontology was a simple step. After that, the specification of the isDescribedBy relationships was done (in order to relate the quality subcharacteristics with the software attributes). The quality concepts incorporated to the software scheme (illustrated previously) are shown in Figure 13.

For each attribute identified, the final quality scheme must incorporate a software metric that represents the way in which the attribute should be measured. In order to evaluate the software quality, the metrics must be expressed with the concepts prescribed in the metric ontology. Figure 14 shows the quality scheme of the online toy store with the metrics for each attribute. The figure includes a complete description of the "Number of Correct Access" software metric (instance #CorrectAccess) to show how a metric should be described. The rest of the descriptions are not present in the diagram in order to maintain its clarity. However, with the purpose of describe an indirect metric, Tables 5, 6, and 7 exemplify the representation of the components used for describe the "Percentage of expired sessions" metric (instance %ExpiredSessions).

The case study quality scheme was implemented using Protégé as an instance of the ontology implemented.

7.4 | Executing queries over the quality scheme

Using the quality scheme of the online toy store implemented in Protégé, a set of SPARQL queries was executed in order to show how the specification responds to the questions.

The selected queries were taken from two different groups in order to show different types of answers (metric and characteristic, subcharacteristic, or quality scheme). Specifically, the queries Q1, Q3, Q8, and Q11 were selected as examples. Figure 15 shows the parameters used in each question and the result of its execution over Protégé. Although Q3 shows a quantity, Q1, Q8, and Q11 show the instances that answer to the criteria indicated in the questions. For example, the answer to the question Q8 refers to an instance of the "User Interface Aesthetics" concept that is related with an instance of the Metric concept named "Interface Novelty."

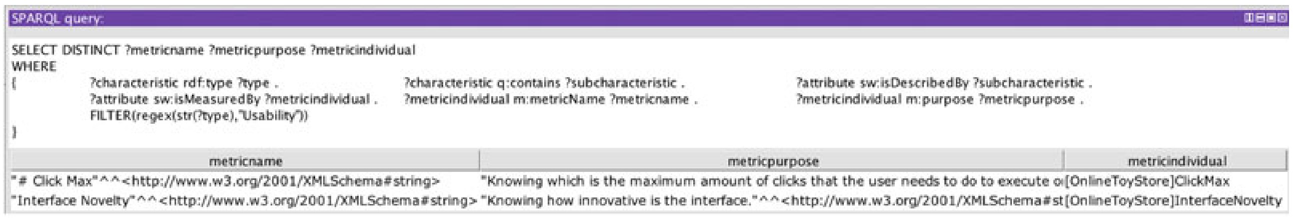
The main purpose of the queries is to allow extract information about the composition of the quality scheme developed for a specific software product. The result of these queries can be used as a reference to build new quality schemes or to improve the existing ones taking into account the composition of other schemes. In this way, the realization of quality schemes not only improves the overall

TABLE 6 Description of the instance "Session" used in "%ExpiredSession"—direct metric instantiation

Element	*Class*	*Object properties*	
Session	Direct metric	isObtainedBy isDimentionedIn hasAsUnit	Count during execution ContinuousScale Session unit
Count during execution		(see Figure 14)	
ContinuousScale		(see Figure 14)	
Session unit		(see Table 5)	

TABLE 7 Description of the instance "ExpiredSession" used in "%ExpiredSession"—direct metric instantiation

Element	*Class*	*Object properties*	
ExpiredSession	Direct metric	isObtainedBy isDimentionedIn hasAsUnit	Count during execution ContinuousScale Session unit
Count during execution		(see Figure 14)	
ContinuousScale		(see Figure 14)	
Session unit		(see Table 5)	

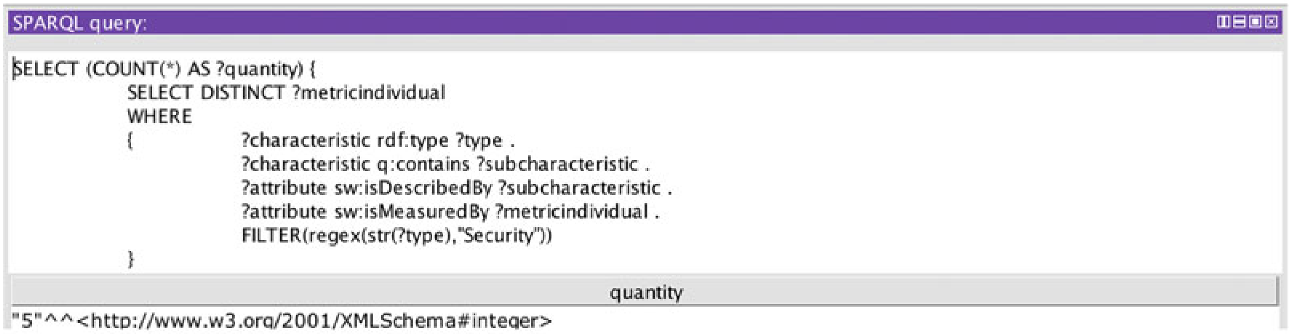


SPARQL query:

```
SELECT DISTINCT ?metricname ?metricpurpose ?metricindividual
WHERE
{            ?characteristic rdf:type ?type .                    ?characteristic q:contains ?subcharacteristic .            ?attribute sw:isDescribedBy ?subcharacteristic .
             ?attribute sw:isMeasuredBy ?metricindividual .      ?metricindividual m:metricName ?metricname .               ?metricindividual m:purpose ?metricpurpose .
             FILTER(regex(str(?type),"Usability"))
}
```

metricname	metricpurpose	metricindividual
"# Click Max"^^<http://www.w3.org/2001/XMLSchema#string>	"Knowing which is the maximum amount of clicks that the user needs to do to execute o	[OnlineToyStore]ClickMax
"Interface Novelty"^^<http://www.w3.org/2001/XMLSchema#string>	"Knowing how innovative is the interface."^^<http://www.w3.org/2001/XMLSchema#st	[OnlineToyStore]InterfaceNovelty

(a): *Q1 – X=Usability.*

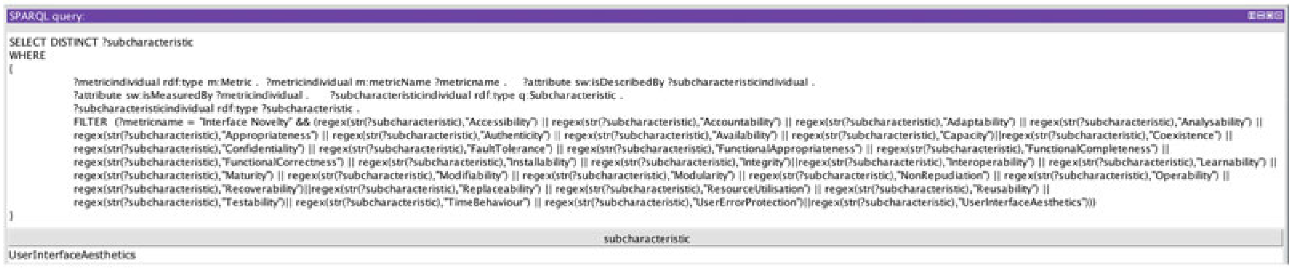


SPARQL query:

```
SELECT (COUNT(*) AS ?quantity) {
        SELECT DISTINCT ?metricindividual
        WHERE
        {       ?characteristic rdf:type ?type .
                ?characteristic q:contains ?subcharacteristic .
                ?attribute sw:isDescribedBy ?subcharacteristic .
                ?attribute sw:isMeasuredBy ?metricindividual .
                FILTER(regex(str(?type),"Security"))
        }
```

quantity
"5"^^<http://www.w3.org/2001/XMLSchema#integer>

(b): *Q3 – X=Security.*

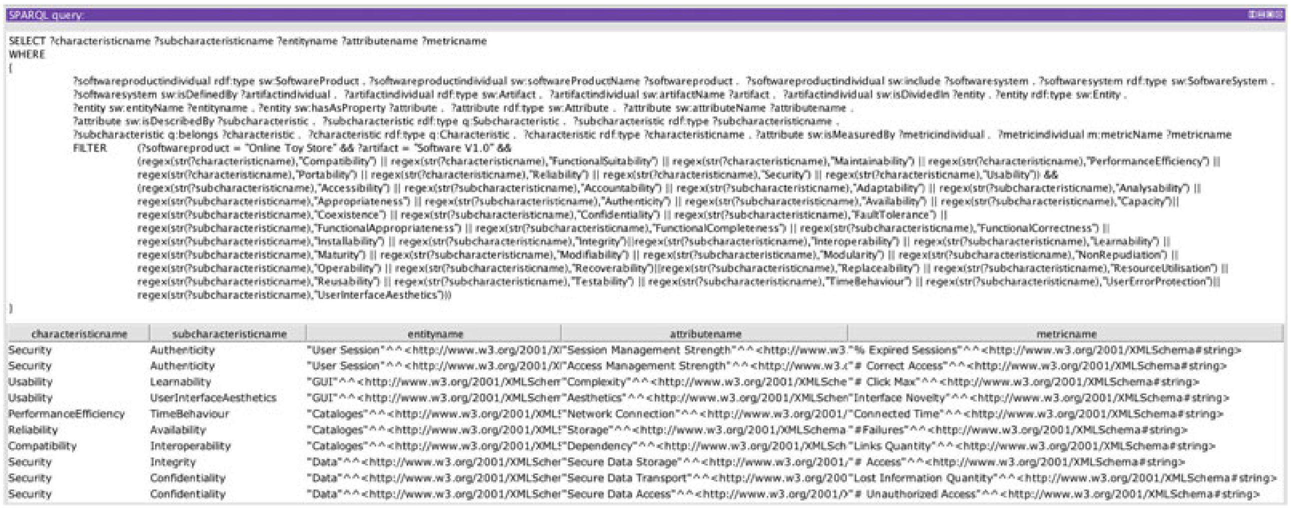


SPARQL query:

```
SELECT DISTINCT ?subcharacteristic
WHERE
{
        ?metricindividual rdf:type m:Metric .  ?metricindividual m:metricName ?metricname .   ?attribute sw:isDescribedBy ?subcharacteristicindividual .
        ?attribute sw:isMeasuredBy ?metricindividual .    ?subcharacteristicindividual rdf:type q:Subcharacteristic .
        ?subcharacteristicindividual rdf:type ?subcharacteristic .
        FILTER  (?metricname = "Interface Novelty" && (regex(str(?subcharacteristic),"Accessibility") || regex(str(?subcharacteristic),"Accountability") || regex(str(?subcharacteristic),"Adaptability") || regex(str(?subcharacteristic),"Analysability") ||
        regex(str(?subcharacteristic),"Appropriateness") || regex(str(?subcharacteristic),"Authenticity") || regex(str(?subcharacteristic),"Availability") || regex(str(?subcharacteristic),"Capacity")||regex(str(?subcharacteristic),"Coexistence") ||
        regex(str(?subcharacteristic),"Confidentiality") || regex(str(?subcharacteristic),"FaultTolerance") || regex(str(?subcharacteristic),"FunctionalAppropriateness") || regex(str(?subcharacteristic),"FunctionalCompleteness") ||
        regex(str(?subcharacteristic),"FunctionalCorrectness") || regex(str(?subcharacteristic),"Installability") || regex(str(?subcharacteristic),"Integrity")||regex(str(?subcharacteristic),"Interoperability") || regex(str(?subcharacteristic),"Learnability") ||
        regex(str(?subcharacteristic),"Maturity") || regex(str(?subcharacteristic),"Modifiability") || regex(str(?subcharacteristic),"Modularity") || regex(str(?subcharacteristic),"NonRepudiation") || regex(str(?subcharacteristic),"Operability") ||
        regex(str(?subcharacteristic),"Recoverability")||regex(str(?subcharacteristic),"Replaceability") || regex(str(?subcharacteristic),"ResourceUtilisation") || regex(str(?subcharacteristic),"Reusability") ||
        regex(str(?subcharacteristic),"Testability")|| regex(str(?subcharacteristic),"TimeBehaviour") || regex(str(?subcharacteristic),"UserErrorProtection")||regex(str(?subcharacteristic),"UserInterfaceAesthetics")))
}
```

subcharacteristic
UserInterfaceAesthetics

(c): *Q8 – Z=Interface Novelty.*



SPARQL query:

```
SELECT ?characteristicname ?subcharacteristicname ?entityname ?attributename ?metricname
WHERE
{
        ?softwareproductindividual rdf:type sw:SoftwareProduct . ?softwareproductindividual sw:softwareProductName ?softwareproduct .  ?softwareproductindividual sw:include ?softwaresystem . ?softwaresystem rdf:type sw:SoftwareSystem .
        ?softwaresystem sw:isDefinedBy ?artifactindividual .  ?artifactindividual rdf:type sw:Artifact .  ?artifactindividual sw:artifactName ?artifact .  ?artifactindividual sw:isDividedIn ?entity . ?entity rdf:type sw:Entity .
        ?entity sw:entityName ?entityname . ?entity sw:hasAsProperty ?attribute .  ?attribute rdf:type sw:Attribute .  ?attribute sw:attributeName ?attributename .
        ?attribute sw:isDescribedBy ?subcharacteristic .  ?subcharacteristic rdf:type q:Subcharacteristic .  ?subcharacteristic rdf:type ?subcharacteristicname .
        ?subcharacteristic q:belongs ?characteristic .  ?characteristic rdf:type q:Characteristic .  ?characteristic rdf:type ?characteristicname . ?attribute sw:isMeasuredBy ?metricindividual .  ?metricindividual m:metricName ?metricname
        FILTER        (?softwareproduct = "Online Toy Store" && ?artifact = "Software V1.0" &&
                      (regex(str(?characteristicname),"Compatibility") || regex(str(?characteristicname),"FunctionalSuitability") || regex(str(?characteristicname),"Maintainability") || regex(str(?characteristicname),"PerformanceEfficiency") ||
                      regex(str(?characteristicname),"Portability") || regex(str(?characteristicname),"Reliability") || regex(str(?characteristicname),"Security") || regex(str(?characteristicname),"Usability")) &&
                      (regex(str(?subcharacteristicname),"Accessibility") || regex(str(?subcharacteristicname),"Accountability") || regex(str(?subcharacteristicname),"Adaptability") || regex(str(?subcharacteristicname),"Analysability") ||
                      regex(str(?subcharacteristicname),"Appropriateness") || regex(str(?subcharacteristicname),"Authenticity") || regex(str(?subcharacteristicname),"Availability") || regex(str(?subcharacteristicname),"Capacity")||
                      regex(str(?subcharacteristicname),"Coexistence") || regex(str(?subcharacteristicname),"Confidentiality") || regex(str(?subcharacteristicname),"FaultTolerance") ||
                      regex(str(?subcharacteristicname),"FunctionalAppropriateness") || regex(str(?subcharacteristicname),"FunctionalCompleteness") || regex(str(?subcharacteristicname),"FunctionalCorrectness") ||
                      regex(str(?subcharacteristicname),"Installability") || regex(str(?subcharacteristicname),"Integrity")||regex(str(?subcharacteristicname),"Interoperability") || regex(str(?subcharacteristicname),"Learnability") ||
                      regex(str(?subcharacteristicname),"Maturity") || regex(str(?subcharacteristicname),"Modifiability") || regex(str(?subcharacteristicname),"Modularity") || regex(str(?subcharacteristicname),"NonRepudiation") ||
                      regex(str(?subcharacteristicname),"Operability") || regex(str(?subcharacteristicname),"Recoverability")||regex(str(?subcharacteristicname),"Replaceability") || regex(str(?subcharacteristicname),"ResourceUtilisation") ||
                      regex(str(?subcharacteristicname),"Reusability") || regex(str(?subcharacteristicname),"Testability") || regex(str(?subcharacteristicname),"TimeBehaviour") || regex(str(?subcharacteristicname),"UserErrorProtection")||
                      regex(str(?subcharacteristicname),"UserInterfaceAesthetics")))
}
```

characteristicname	subcharacteristicname	entityname	attributename	metricname
Security	Authenticity	"User Session"^^<http://www.w3.org/2001/XI	"Session Management Strength"^^<http://www.w3.	"% Expired Sessions"^^<http://www.w3.org/2001/XMLSchema#string>
Security	Authenticity	"User Session"^^<http://www.w3.org/2001/XI	"Access Management Strength"^^<http://www.w3.c	"# Correct Access"^^<http://www.w3.org/2001/XMLSchema#string>
Usability	Learnability	"GUI"^^<http://www.w3.org/2001/XMLScher	"Complexity"^^<http://www.w3.org/2001/XMLSche	"# Click Max"^^<http://www.w3.org/2001/XMLSchema#string>
Usability	UserInterfaceAesthetics	"GUI"^^<http://www.w3.org/2001/XMLScher	"Aesthetics"^^<http://www.w3.org/2001/XMLSchen	"Interface Novelty"^^<http://www.w3.org/2001/XMLSchema#string>
PerformanceEfficiency	TimeBehaviour	"Cataloges"^^<http://www.w3.org/2001/XML!	"Network Connection"^^<http://www.w3.org/2001/	"Connected Time"^^<http://www.w3.org/2001/XMLSchema#string>
Reliability	Availability	"Cataloges"^^<http://www.w3.org/2001/XML!	"Storage"^^<http://www.w3.org/2001/XMLSchema	"#Failures"^^<http://www.w3.org/2001/XMLSchema#string>
Compatibility	Interoperability	"Cataloges"^^<http://www.w3.org/2001/XML!	"Dependency"^^<http://www.w3.org/2001/XMLSch	"Links Quantity"^^<http://www.w3.org/2001/XMLSchema#string>
Security	Integrity	"Data"^^<http://www.w3.org/2001/XMLScher	"Secure Data Storage"^^<http://www.w3.org/2001/	"# Access"^^<http://www.w3.org/2001/XMLSchema#string>
Security	Confidentiality	"Data"^^<http://www.w3.org/2001/XMLScher	"Secure Data Transport"^^<http://www.w3.org/200	"Lost Information Quantity"^^<http://www.w3.org/2001/XMLSchema#string>
Security	Confidentiality	"Data"^^<http://www.w3.org/2001/XMLScher	"Secure Data Access"^^<http://www.w3.org/2001/>	"# Unauthorized Access"^^<http://www.w3.org/2001/XMLSchema#string>

(d): *Q11 – V=Software V1.0, W=Online Toy Store.*

FIGURE 15 SPARQL queries execution over the quality scheme of the online toy store

quality of the product but also helps the developers to understand which software aspects are directly related to quality.

7.5 | Discussion

The case study shows how a QS can be built using the requirements specification of a software product (in this case, an online toy store). From the functional requirements (Table 4), by specifying quality goals, a set of quality subcharacteristics is defined. Each subcharacteristic identified refers to a concept of the quality model ontology. Then, using the final ontology elements, an appropriate QS is instantiated (Figure 14).

By having the QS mechanism, software quality can be documented uniformly along all the development process. Now, the quality specification is visible to all the stakeholders. So the use of quality schemes as a documentation technique will improve the quality understanding of all the participants. During maintenance phase, a QS will give a support document that will help to analyze the changes impact over the quality properties specified. Then, benefits in this type of mechanism are multiple, helping to improve the development process. The absence of QS will cause a wrong track of quality requirements, confusing the development team over its traceability.

8 | CONCLUSIONS AND FUTURE WORK

Documentation is an integral part of a software system. It contains the information that is necessary to effectively and successfully develop, use, and maintain a system. In practice, however, the creation of appropriate documentation is largely neglected (Bayer & Muthig, 2006).

In this paper, an ontology to document quality schemes is proposed. The final model represents a specific domain that focuses on the evaluation of a software product quality. One instantiation of this semantic model gives a quality scheme for a specific software product. The created scheme can be used as a document that specifies the measurement mechanism that should be applied over different artifacts through the development process.

In addition, a case study has been presented. This case study is based on the specification of a quality scheme to an online toy store. The instantiation of the concepts proposed in the ontology for the definition of the quality scheme is one of the main results obtained. The execution of some SPARQL queries over the quality scheme created shows how information contained in this type of document is useful. This document can contribute to the understanding of the main quality aspects associated with a specific software product and to develop of new ways to measure these quality aspects.

The implementation of a tool that automates the elaboration of the quality schemes is the next step in this direction. The ontology designed can be taken as base of this tool, using an ontology-based approach. The main purpose of this tool should be the creation, storage, modification, and query of the existent quality schemes. With this technological support, the development team could easily understand the quality aspects related with a specific software artifact and see how the quality should be measured.

In the future, quality in use can be added to the final ontology in order to analyze other aspects of the software product. The ontology can be adapted to other types of quality models because the quality semantic model is an independent model. All the specific characteristics defined in the quality in use model should be modeled as a new semantic model and, then, replace the quality semantic model with this new specification (maintaining the relationships between different models). By simply changing the quality model ontology, the approach proposed allows to reuse the existing models and, then, define a new type of document in order to specify other quality properties. As the metric concept is modeled in universal way, a metric can be defined to any type of quality property. Then, the independence of the semantic models gives modularity over the ontological approach.

ACKNOWLEDGEMENTS

The authors wish to acknowledge the financial support received from CONICET (PIP 112-20110100906) and Universidad Tecnológica Nacional (PID 25/O144 and PID 25/O156).

REFERENCES

Abebe, S., & Tonella, P. (2015). Extraction of domain concepts from the source code. *Science of Computer Programming*, *98*(4), 680–706.

Al Balushi, T., Sampaio, P., & Loucopoulos, P. (2013). Eliciting and prioritizing quality requirements supported by ontologies: A case study using the ElicitO framework and tool. *Expert Systems*, *30*, 129–151.

Al-Badareen, A., Selamat, M., Jabar, M., Din, J., & Turaev, S. (2011). Software quality models: A comparative study. *Software Engineering and Computer Systems*, *Communications in Computer and Information Science*, *179*, 46–55.

Albin, S. (2003). *The art of software architecture: Design methods and techniques* (1st ed.). John Wiley & Sons.

Ampatzoglou, A., Frantzeskou, G., & Stamelos, I. (2012). A methodology to assess the impact of design patterns on software quality. *Information and Software Technology*, *54*(4), 331–346.

Barney, S., Mohankumar, V., Chatzipetrou, P., Aurum, A., Wohlin, C., & Angelis, L. (2014). Software quality across borders: Three case studies on company internal alignment. *Information and Software Technology*, *Special sections on International Conference on Global Software Engineering – August 2011 and Evaluation and Assessment in Software Engineering – April 2012*. 56, p. 20–38.

Barney, S., & Wohlin, C. (2009). Software product quality: Ensuring a common goal. *Trustworthy Software Development Processes*, *Lecture Notes in Computer Science*, *5543*, 256–267.

Bass, L., Clements, P., & Kazman, R. (2012). *Software architecture in practice* (3rd ed.). Addison-Wesley Professional.

Bayer, J., & Muthig, D. (2006). A view-based approach for improving software documentation practices. *13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems, 2006.* P.10.

Bertoa, M. F., Vallecillo, A., & García, F. (2006). An ontology for software measurement. In *Ontologies for software engineering and software technology* (pp. 175–196). Berlin Heidelberg: Springer.

Bogado, V., Gonnet, S., & Leone, H. (2014). Modeling and simulation of software architecture in discrete event system specification for quality evaluation. *SIMULATION*, *90*, 290.

Boukouchi, Y., Marzak, A., Benlahmer, H., & Moutachaouik, H. (2013). Comparative study of software quality models. *IJCSI International Journal of Computer Science Issues*, 309–314.

Castillo, I., Losavio, F., Matteo, A., & Boegh, J. (2010). REquirements, aspects and software quality: The REASQ model. *Journal of Object Technology*, *9*(4), 69–91.

Castro, E., Rico, L., & Castro, E. (1995). *Estructuras aritméticas elementales y su modelización*. Grupo Editorial Iberoamericana, p. 45–79.

Couto, R., Ribeiro, A., & Campos, J. (2014). Application of ontologies in identifying requirements patterns in use cases. *11th International Workshop on Formal Engineering approaches to Software Components and Architectures (FESCA)*. 147, p. 62–76.

Dargan, J., Campos-Nanez, E., Fomin, P., & Wasek, J. (2014). Predicting systems performance through requirements quality attributes model. *Procedia Computer Science*, *28*, 347–353.

De Graaf, K., Liang, P., Tang, A., Van Hage, W., & Van Vliet, H. (2014). An exploratory study on ontology engineering for software architecture documentation. *Computers in Industry*, *65*(7), 1053–1064.

Deissenboeck, F., Juergens, E., Lochmann, K., & Wagner, S. (2009). Software quality models: Purposes, usage scenarios and requirements. *ICSE Workshop on Software Quality 2009 (WOSQ '09)*. p. 9–14.

El-Haik, B., & Shaout, A. (2010). *Software design for six sigma: A roadmap for excellence* (1st ed.). John Wiley & Sons.

Fenton, N., & Bieman, J. (2014). *Software metrics: A rigorous and practical approach* (3rd ed.). Boca Raton: CRC Press.

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

García-Peñalvo, F., Colomo-Palacios, R., García, J., & Therón, R. (2012). Towards an ontology modeling tool. A validation in software engineering scenarios. *Expert Systems*, *39*, 11468–11478.

Gómez-Pérez, A., Fernandez-Lopez, M., & Corcho, O. (2010). *Ontological engineering: With examples from the areas of knowledge management, e-commerce and the semantic web* (1st ed.). Springer London.

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

Hadzic, M., Chang, E., Dillon, T., Kacprzyk, J., & Wongthongtham, P. (2009). *Ontology-based multi-agent systems*. Germany: Springer Berlin Heidelberg.

Harris, S., Seaborne, A., & Prud'hommeaux, E. (2013). *SPARQL 1.1 Query Language*. W3C recommendation, 21.

Henderson-Sellers, B. (2011). Bridging metamodels and ontologies in software engineering. *Journal of Systems and Software*, *84*(2), 301–313.

Horrocks, I., Patel-Schneider, P., Boley, H., Tabet, S., Grosof, B., & Dean, M. (2004). *SWRL: A semantic web rule language combining OWL and RuleML.* W3C member submission, 21, 79.

IEEE Standard 1061:1998, *IEEE Standard for a Software Quality Metrics Methodology – Description*.

ISO/IEC 25010:2011, *System and software quality models*.

ISO/IEC 25020:2007, *Measurement reference model and guide*.

ISO/IEC TR 9126-2:2003, *Software engineering - Product quality - Part 2: External metrics*.

ISO/IEC TR 9126-3:2003, *Software engineering - Product quality - Part 3: Internal metrics*.

Jwo, J., & Cheng, Y. (2010). Pseudo software: A mediating instrument for modeling software requirements. *Journal of Systems and Software*, *83*(4), 599–608.

Kan, S. (2003). *Metrics and models in software quality engineering* (2nd ed.). Addison-Wesley Professional.

Khoshgoftaar, T., Liu, Y., & Seliya, N. (2004). A multiobjective module-order model for software quality enhancement. *IEEE Transactions on Evolutionary Computation*, *8*(6), 593–608.

Kitchenham, B. A., Hughes, R. T., & Linkman, S. G. (2001). Modeling software measurement data. *IEEE Transactions on Software Engineering*, *27*(9), 788–804.

Kitchenham, B., & Pfleeger, S. (1996). Software quality: The elusive target. *IEEE Software*, *13*, 12.

Knublauch, H., Horridge, M., Musen, A., Rector, R., Stevens, N., Drummond, P. … Wang, H. (2005). The Protégé OWL experience. *Workshop on OWL: Experiences and Directions, 4th International Semantic Web Conference*.

Martin, M., & Olsina, L. (2003). Towards an ontology for software metrics and indicators as the foundation for a cataloging web system. *Proceedings of the 2003 international conference on Web engineering, July 14–18, 2003, Oviedo, Spain*, pp. 103–113.

Meiappane, A., Chithra, B., & Venkataesan, P. (2013). Evaluation of software architecture quality attribute for an internet banking system. *International Journal of Computer Applications*, *62*(19), 21.

Milicic, D. (2005). Software quality models and philosophies. *Software Quality Attributes and Trade-Offs*, 3–19.

Olsina, L., & Martín, M. (2003). Ontology for software metrics and indicators. *Journal of Web Engineering*, *2*(4), 262–281.

Orgun, M. A., & Meyer, T. (2008). Introduction to the special issue on advances in ontologies. *Expert Systems, The Journal of Knowledge Engineering*, *25*(3), 175–178.

Papas, D., & Tjortjis, C. (2014). Combining clustering and classification for software quality evaluation. *Artificial Intelligence: Methods and Applications, Lecture Notes in Computer Science.* pp. 273–286.

Pires, P., Delicato, F., Cóbe, R., Batista, T., Davis, J., & Song, J. (2011). Integrating ontologies, model driven, and CNL in a multi-viewed approach for requirements engineering. *Requirements Engineering*, *16*(2), 133–160.

Pressman, R. (2010). *Software engineering: A practitioner's approach* (7th ed.). McGraw-Hill.

Rech, J., & Bunse, C. (2008). Evaluating performance of software architecture models with the Palladio component model. *IGI Global. p.* 95–118.

Reinhartz-Berger, I., Sturm, A., & Wand, Y. (2013). Comparing functionality of software systems: An ontological approach. *Data & Knowledge Engineering*, *87*, 320–338.

Rijgersberg, H., Wigham, M., & Top, J. (2011). How semantics can improve engineering processes: A case of units of measure and quantities. *Advanced Engineering Informatics*, *25*(2), 276–287.

Roshandel, R., Medvidovic, N., & Golubchik, L. (2007). A Bayesian model for predicting reliability of software systems at the architectural level. *Quality of Software Architectures; Software Architectures, Components and Applications: QoSA.* pp. 108–126.

Roussey, C., Pinet, F., Kang, M., & Corcho, O. (2011). An introduction to ontologies and ontology engineering. *Ontologies in Urban Development Projects, Springer London*, *1*, 9–38.

Software Product Quality Requirements and Evaluation SQuaRE (2005) *Guide to SQuaRE*, 1° Ed.

Van Zeist, R., & Hendriks, P. (1996). Specifying software quality with the extended ISO model. *Software Quality Journal*, *5*(4), 273–284.

Vegetti, M., Leone, H., & Henning, G. (2011). PRONTO: An ontology for comprehensive and consistent representation of product information. *Engineering Application of Artificial Intelligence, Elsevier*, *24*, 1305–1327.

W3C OWL Working Group (2012). OWL 2 web ontology language document overview. [Online] Available from: http://www.w3.org/TR/owl2-overview/ [Accessed January 26, 2015].

María Julia Blas received her Information Systems Engineering degree from Universidad Tecnológica Nacional (UTN), Santa Fe, Argentina, in 2014. She is currently a PhD student in Information Systems Engineering at Universidad Tecnológica Nacional (UTN). She also has a Research Fellowship from the National Council for Scientific and Technical Research of Argentina (CONICET), to work at Instituto de Desarrollo y Diseño (INGAR). Her research interests focus on software quality evaluation using software architecture.

Silvio Gonnet received an Engineering degree in Information Systems from Universidad Tecnológica Nacional (UTN), Santa Fe, Argentina, in 1998 and obtained his PhD degree in Engineering from Universidad Nacional del Litoral (UNL) in 2003. He currently holds a Researcher position at the National Council for Scientific and Technical Research of Argentina (CONICET), to work at Instituto de Desarrollo y Diseño (INGAR). Also, he works as an Assistant Professor at Universidad Tecnológica Nacional. His research interests are in models to support the design process, software architectures, and semantic web.

Horacio Leone is a full Professor at the Department of Information Systems Engineering of the Facultad Regional Santa Fe, Universidad Tecnológica Nacional (Santa Fe, Argentina). He also holds a Researcher position at the National Council for Scientific and Technical Research of Argentina (CONICET), working at Instituto de Desarrollo y Diseño. He obtained his PhD degree in Chemical Engineering from Universidad Nacional del Litoral (Santa Fe, Argentina) in 1986 and was a Postdoctoral Fellow at the

Massachusetts Institute of Technology (1986–1989). His current research activities focus on software architectures, models for supporting the design process, semantic web applications to supply chain information systems, and enterprise modelling. He has supervised several PhD students.

How to cite this article: Blas MJ, Gonnet S, Leone H. An ontology to document a quality scheme specification of a software product. *Expert Systems*. 2017;e12213. https://doi.org/10.1111/exsy.12213