A MOF-Compliant Approach to Software Quality
Modeling
Burgués, X. and Franch, X. and Ribó, J. M.
Research Report LSI-04-54-R

Departament de Llenguatges i Sistemes Informàtics

UNIVERSITAT POLITÈCNICA DE CATALUNYA

A MOF-Compliant Approach to Software Quality Modeling

Xavier Burgués¹, Xavier Franch¹, Josep M. Ribó²

¹ Universitat Politècnica de Catalunya (UPC)
c/ Jordi Girona 1-3 (Campus Nord, C6) E-08034 Barcelona (Catalunya, Spain)
{diafebus, franch}@lsi.upc.edu

² Universitat de Lleida (UdL)
C. Jaume II, 69 E-25001 Lleida (Catalunya, Spain)
josepma@eps.udl.es

Abstract. Software quality is a many-faceted concept that depends on the kind of artifact to be measured, the context where measurement takes place, the quality framework used, and others. Furthermore, there is a great deal of standards, white papers, and in general proposals of any kind related to software quality. Consequently, a unified software quality framework seems to be needed to compare, combine or select these proposals and to define new ones. We propose a MOF-compliant approach for structuring quality models in order to formalise software quality issues and deal with quality information modelling. We propose two types of models: a generic model, situated in the M2 MOF layer; and a hierarchy of reference models, defined in the M1 and M0 MOF layers. The generic model elements are derived from the UML metamodel by specialization. Then, we can instantiate them to get reference models that can be further refined for defining quality frameworks to be used in different experiences. Each of these models is divided into three parts, namely fundamental concepts, metrics and context. We illustrate our proposal providing a multi-level reference model in the context of collection libraries quality evaluation.

1. Introduction

Quality assessment and management (QA&M) plays currently a crucial role in all the facets of software development. This means that not only the software process and the system-to-be are targets of QA&M, but also subprocesses such as specification, design and testing, and software-related artifacts such as system requirements, specifications and software architectures. As a result, we may find a great deal of proposals aiming at the study of QA&M issues in those contexts, so diverse in nature such as software process assessment and improvement [SEI95, ISO98], analysis of data models like UML class diagrams [GPP02] or ER models [Moo98], measurement of OO designs [BM02], and so on. Furthermore, the tendency seems not to converge into more compact, general-purpose frameworks but on the contrary, to provide new, specialized proposals.

All of these proposals share a core of common concepts, e.g. metrics, quality factor, etc., but it is not obvious to identify similarities and differences between them. This difficulty hampers the understanding of the quality frameworks, their further extension or evolution, and their comparison when it becomes necessary to choose one in a given context.

Several authors claim that ontologies, conceptual models or similar descriptions are needed in order to precisely define the concepts, processes, languages and tools related to software quality [KHL01, OM04]. The goal is the definition of a framework useful to analyse the variety of approaches, to define new ones and to adapt the

existing ones to new contexts. As a result, it becomes necessary to work on the foundations, to obtain a set of widely accepted general concepts with a clear structure to be used as the basis of particular methods and tools.

In [BF04] we proposed a 3-level hierarchy of quality models. Each level was related to a different abstraction degree: the generic model provided a universal unified framework; reference models allowed the definition of operational frameworks, ready to be used as the concepts coming from the generic model were instantiated; domain models fit in the specificities of concrete QA&M experiences and set a comfortable way to deal with quality. This hierarchy was a good starting point for stating a quality framework but some serious problems were identified:

- Our proposal was *ad hoc*, without being integrated into any existing and consolidated metamodel, architecture or ontology. This was a serious drawback considering 3 different aspects: semantics of the models; reuse of existing concepts, methods and tools; and dissemination of the approach.
- The frontier among reference and domain models was too fuzzy and arbitrary. In fact, for some applications, we found that the last thing to refine was not the domain but other parts of the generic model, for instance the type of artifact itself or the metrics to be used.
- We just allowed one level of reference and domain models. This was in fact a serious limitation, since it was impossible to refine or combine models hampering thus quality knowledge structure and reuse.
- We included in the generic model a dimension (the language dimension) that is not present in all the approaches, unlike the other three dimensions that we consider.
- Our experiences were a few and then the proposal was still unstable. Once we acquired more knowledge we discovered some minor flaws, specially in the generic model.

In this paper we propose a conceptual framework for structuring quality models that overcomes these drawbacks. The framework is presented in section 2. The proposal is integrated into the MOF architecture [MOF03] as an extension of the UML metamodel [UML03], it supports a hierarchical structure of reference models without imposing any particular refinement order, it removes the language dimension and it has been validated with a greater number of cases (i.e., proposals about quality available in the literature). The core of the proposal is presented in sections 5 and 6. Previously, sections 3 and 4 provide a short summary about the MOF architecture and UML metamodel extensions. Section 7 provides the conclusions.

2. A Hierarchy of Quality Models

We present a framework for dealing with software quality that consists of a hierarchy of two types of quality models:

- *Generic model*. The root of the hierarchy. It introduces the fundamental concepts that are present in every single approach to QA&M. “Quality model”, “artifact” and “metrics” are some of these concepts. It is abstract enough to be used in several software engineering activities: specification, design, development, certification, selection, etc.
- *Reference models*. They provide particular interpretations of the generic model fundamental concepts in a particular setting. As an example, a reference model could be built up following the ideas of the ISO/IEC 9126 quality standard, part 1

[ISO01] and incorporating metrics-related notions coming from the theory in [FP98, Zus98]. We may have different degrees of refinement which means that reference models can be structured in hierarchies until we obtain leaves, which stand for reference models that can be used in particular QA&M experiences. Also the hierarchy may contain models that are not obtained by refinement but by composition, because different models may focus on just one part of the generic model.

Although diverse, these two models are structured into three different parts:

- *Fundamental concepts.* It embraces the concepts and relationships that form the quality models and system requirements about quality. The concepts therein stem from general quality standards [IEEE92, ISO01] and widespread catalogues of quality factors and requirements [Fir03, KKP90].
- *Metrics.* Here we define the types of metrics to be used to measure the items defined by the model and to state the satisfaction of requirements. Classic proposals [FP98, Zus98] and quality standards again [ISO01] are the foundations of this part.
- *Context.* It has to do with the software domains which the quality models will be attached to; the structure of artifacts to be measured; and the environment in which they operate (a type of organisation, a particular one, a department, a project, etc.). Domains may be structured as a taxonomy as proposed in [GV95, CFQT04]. The artifacts may be aggregations or compositions of others.

Figure 1 illustrates the evolution from the generic model to reference ones. It shows also the recommended use of our framework. From the generic model, we obtain virtually hundreds of reference models, one for each consolidated proposal that has been defined in the literature for the concepts present in the generic model (we depict just three in the figure). For instance, we get a reference model for the ISO/IEC 9126 standard, other for the software domains as defined in the INCOSE taxonomy, other for the Stevens set of scales [Ste46], and so on. The important thing is that each of these reference models takes as few assumptions as possible, not compromising therefore its use unnecessarily; in fact, a great deal of these reference models just refine one of the three parts of the generic model sometimes even not completely.

These first-level reference models can then be refined to introduce details, as shown in fig. 1. This allows to structure quality proposals in such a way that details are introduced progressively, making understanding easier. A general strategy we have adopted is to use first-level reference models to represent the general structure of the approach, and second-level ones to indicate particular elements. Another strategy consists on using this refinement concept for distinguishing among normative or mandatory parts of a proposal (defined at the first-level) from optional or recommended parts (defined at lower levels). We present examples of both situations in section 6.

Once the target level of detail has been reached, we can combine the lower-level reference models to obtain new ones embracing all the aspects of quality. This combination can be made for several reasons: to put together widespread proposals for further use, as we will show in section 6; to construct *ad hoc* frameworks for particular experiences; or to create a collection of reusable frameworks that can be used in lots of quality-related experiences. Fig. 1 also illustrates this combination.

3. The MOF Architecture and the UML Metamodel

UML [UML03] is currently a *de facto* standard in object-oriented modeling. As it is a general purpose notation, it has to be tailored to specific contexts, in particular to different software domains. The extensions provide some clear advantages: the integration of the domain in a standard framework, a potential usage by the software engineering community and the existence of a large number of support tools. Moreover, integration to the widespread MOF metamodeling architecture [MOF03] –adopted by OMG– is also provided.

Fig. 1. A framework for constructing quality models.

This metamodeling architecture consists of four layers: M0, M1, M2 and M3, such that layer M_i contains instances of elements defined in M_{i+1} . M0 contains runtime objects, M1 is the model level and contains the classes for M0 objects; UML models are defined in M1. M2 is the metamodel level and contains elements used to build models in M1 (e.g., *Class*, *Association* and *Dependency*). The UML metamodel and other metamodels are defined in M2. Finally, M3 is the meta-metamodel level. MOF is the meta-metamodel shared by all the metamodels defined in the OMG framework. Its main contents are the essential elements of the UML metamodel (in particular, the infrastructure library of the UML 2.0 metamodel is reused in the MOF model definition [UML03]).

As mentioned in the introduction, in this paper we propose to extend the UML metamodel with concepts aimed at the generation of quality models of software-related artifacts. We want the result to be fully integrated in the MOF metamodeling architecture and coherent with the standard UML extension mechanism in order to end up with a UML-consistent result. There are two ways to tailor UML: (1) define a so-called *heavyweight extension*, which provides a first-class metamodel extension mechanism for extending the metamodel and (2) define a *UML profile*. While the first approach is more expressive and comprehensible, the second one provides compatibility with UML modelling tools. Therefore, in our work we have adopted both of them following the methodology defined in [FR04], which consists in two steps:

1. Proposal of a heavyweight extension of the UML metamodel, providing a properly built metamodel, fully integrated in the MOF architecture and preserving the semantics of the UML metamodel elements.
2. Transformation of the previous extension into a UML profile. Since the extension built in step 1 preserves the semantics of the original UML metamodel and since all its metaelements are based on already existing UML metaelements, this transformation can be done in a semi-automatic way using any of the methodologies presented in [JSZ04, FR04, UML03].

4. Induced Associations

In the previous section we were talking about heavyweight extensions which must be defined in layer *M2* of the metamodeling hierarchy. In such an extension we may add new metaclasses and metaassociations between them.

Instances¹ at layer *M1* of a metaassociation *a* connecting metaclasses *MC1* and *MC2* are collections of pairs of elements $\langle e1, e2 \rangle$ being *e1* an instance of *MC1* and *e2* an instance of *MC2*. Of course, the extension of these pairs must respect the cardinality of each role of the metaassociation.

Figure 2(a) is taken from our generic model, the extension of the UML metamodel we are going to present in section 5 (thus, in layer *M2*). You can see two metaclasses: *Attribute* and *IndirectAttribute*. You can also see a metaassociation *dependsOn* between them.

An instantiation of this fragment will take place in layer *M1* as it is shown in figure 2(b). In this layer we find classes which are instances of the metaclasses in *M2* (e.g., *Characteristic* -a kind of quality factor defined in standard ISO/IEC 9126-, an instance of *IndirectAttribute* and *Subcharacteristic* -another kind of factor defined by the same standard-, an instance of *Attribute*) and we find also an extension of the metaassociation *dependsOn* (e.g., the pair $\langle \textit{Characteristic}, \textit{Subcharacteristic} \rangle$). We follow the notation proposed in UML 2.0; thus, we represent the extension of *dependsOn* in *M1* through annotations in classes *Characteristic* and *Subcharacteristic*. Note that the existence of the metaassociation in the metamodel in figure 2(a) does not require the presence of an association in layer *M1*. If we want to

¹ Instances of an association are usually called *extensions* and refer to the collection of pairs of objects, each one instance of each of the classes participants in the association, connected by it. Not to be confused with the term “metamodel extension” referring the addition of new elements to the metamodel. We will use the term “association extension” as it is the standard way to refer association’s instances.

express that the instantiation must contain an association between the instances of the metaclasses, metaassociation *dependsOn* should induce an association at layer *M1*, *dependsOn_M1* as it is depicted in figure 3; this new association will record the connections established in *M0* between instances of *Characteristic* and *Subcharacteristic*.

Fig. 2. Instantiation of a metaassociation.

Fig. 3. Association *dependsOn_M1*.

4.1 Definition.

We define precisely the above introduced concept of *association induced by a metaassociation*.

The main idea, continuing with the same example, is based on: 1) add a new metaclass *DependsOn_Ass* to the metamodel as a subclass of *Association* (coming from the UML metamodel) and 2) a constraint attached to the metaassociation *dependsOn* forcing every model in layer *M1* to contain an association, instance of *DependsOn_Ass*, connecting each pair of classes in the extension of the metaassociation *dependsOn*. As a consequence of this constraint, in the example we are considering it is mandatory to establish, at layer *M1*, an association connecting classes *Characteristic* and *Subcharacteristic* as you may see in figure 3. This association, like any instance of *DependsOn_Ass*, is an association induced by the metaassociation *dependsOn*. Thus, the model in figure 2(b) is no longer a correct instantiation of the metamodel after the addition of the metaclass and the restriction.

Definition. Association induced by a metaassociation.

Given an extension of the UML metamodel at layer *M2* containing two metaclasses *MC1* and *MC2* and a metaassociation between them *M2A*, *M2A* induces associations at layer *M1* if the extension of the metamodel includes: (see fig. 4):

- A subclass *M1A* of the UML metaclass *Association*. The instances of *M1A* (hence, at layer *M1*) will constitute associations induced by *M2A*.

- A constraint [c1] attached to *M2A* establishing:

```
[c1]: MC1.allInstances()->forall(c1|
      MC2.allInstances()->forall(c2| c2.mc1=c1 implies
      M1A.allInstances()->exists(a|
        a.memberEnd->exists(e|e.class=c1) and
        a.memberEnd->exists(e|e.class=c2))))
```

That is, for all instances *c1* of *MC1* and *c2* of *MC2* such that $\langle c1, c2 \rangle$ is in the extension of metaassociation *M2A*, there is an instance of association *M1A* connecting *c1* and *c2*.

- A constraint [c2] attached to *M1A* establishing:

```
[c2]: M1A.allInstances()->forall(a|
      a.memberEnd->size()==2 and
      a.memberEnd->exists(e|e.class.oclIsTypeOf(MC1)) and
      a.memberEnd->exists(e|e.class.oclIsTypeOf(MC2)))
```

In other words, all instances of the association *M1A* will occur in *M1* between two classes which must be instances of *MC1* and *MC2*.

Fig. 4. Graphical representation of induced associations.

We may express that a metaassociation induces associations at layer *M1* in a simpler way defining a boolean function *inducesAssociation(Metaassociation, Association)* with the meaning coming from [c1] and [c2]. The definition of such a function is the following one:


```

context inducesAssociation(M2A: Association, M1A: Class):
Boolean
post: let MC1: Class=M2A.memberEnd->at(1).class,
      MC2: Class=M2A.memberEnd->at(2).class in
Result=M1A.ocIsKindOf(Association) and
      M1A.allInstances()->forall(a|
        a.memberEnd->size()=2 and
        a.memberEnd->exists(e|e.class.ocIsTypeOf(MC1)) and
        a.memberEnd->exists(e|e.class.ocIsTypeOf(MC2)))
and
      MC1.allInstances()->forall(c1|
        MC2.allInstances()->forall(c2| c2.mc1=c1 implies
          M1A.allInstances()->exists(a|
            a.memberEnd->exists(e|e.class=c1) and
            a.memberEnd->exists(e|e.class=c2))))

```

Figure 5 is an example of the use of a boolean function of this kind. It refers to the same situation we were considering at the beginning of this section.

Fig. 5. Induced associations (simplified notation).

4.2 Induced association classes.

In some cases we require the metamodel to induce association classes in the instantiating models. For instance, in our generic model we have the concepts *Domain* (any category of software artifacts to which we want to attach a quality model like word processors) and *Environment* (circumstances under which the products will be used like medium-sized enterprises). Of course, the metamodel also includes the concept *Quality Model*. It also contains a metaassociation between metaclasses *Domain* and *Environment* (denoting that a given domain of products may be used in several environments). We want to induce in all instances of the metamodel the following condition: a different quality model will be obtained for each pair $\langle domain, environment \rangle$ in the extension of the previously cited metaassociation. In other words, we want to induce an association class of type *QualityModel* attached to each association established in *M1* between a domain and an environment.

This situation leads to the definition of induced association classes, analogous to the previous concept of induced associations. The definition is similar to the one already introduced in section 4.1 for induced associations and, hence, we don't repeat it. Figure 6 shows the situation in a graphical way.

Fig. 6. Induced association classes.

5. A Generic Model for Quality as an Extension of the UML Metamodel

We describe in this section the generic model of our framework as an extension of the UML metamodel. Hence, its elements are metaclasses and metaassociations at layer M2. Our target is the construction of a generic model in such a way that it provides a common quality metamodel which any reference model can be an instance of.

Figure 7 shows the package layout of the integration of the generic model into the UML metamodel and figure 8 shows the model itself. Table 1 lists the UML metamodel elements from which the generic model elements derive by specialization. Some of the elements inherit from more than one UML metamodel element.

Fig. 7. Integration of the reference model into the UML metamodel.

Table 1. Connection among the generic model and the UML metamodel.

Generic model	UML metamodel
Domain, Environment, Attribute (and heirs), Metrics	Classifier
MeasurableArtifact	Artifact
Scale, Unit	Enumeration
QualityType	DataType
Value, QualityStatement, AppliedQualityStatement	ValueSpecification
QualityModel, AppliedQualityModel, AppliedQualityStatement, Measurement, Influences	AssociationClass

Fig. 8. The generic model.

The elements are presented next. We structure the presentation into the three parts enumerated in section 2. In fact, the three parts are also structured as packages, but since this structure is not essential for the goal of the paper, we do not address this issue.

Context. *Domain* captures the knowledge about any kind of (parts of) *MeasurableArtifacts* for which a quality model is required. The concept of “artifact” is utterly comprehensive, embracing different things that can be as general as the notion of “office tool”, as specific as a “sorting algorithm”, or related not to the software itself but to software descriptions and documentations as models or specifications. It is important to remark that an artefact may *belong* to several domains. For instance, the quality of a geographical information system may be viewed from the point of view of information systems and from the point of view of geographical software.

The class *Environment* represents the circumstances in which the domains are *used*. It may define the kind of organization (for instance, a huge supermarket, a non-governmental organization, ...), a kind of project (like a CMM level 3 one), etc.

Taking into account that the three classes have been defined as UML classifiers, we can compose instances of these three elements and refine them, therefore we may construct taxonomies of domains and packages of artefacts, we can tailor quality concepts in each single department of an organization, etc. As a consequence, we may define in a reference model, for instance, that the reliability of a component is a combination of the reliability of its subcomponents.

Fundamental concepts. A *QualityModel* will be used for a domain in a given environment (e.g., there will be a quality model for groupware systems in medium-size companies and a quality model for mail servers in ISPs). As an artifact may belong to several domains and a domain may be used in several environments, several quality models may be applicable to each artifact. Quality models *include* quality *Attributes* (such as efficiency, reliability, etc.). As *Attribute* is an heir of *Classifier*, we may organize them as a hierarchy, which is the usual case in most proposals. Therefore, they can be *Direct* or *Indirect*; indirect attributes *depend on* others. Attributes are not mutually independent and so their *influence* on each other is also represented. *Influences* allows categorizing and grading these dependencies (providing kind, intensity, etc.). The class *QualityStatement* represents quality requirements, constraints, criteria and other elements that *concern* a quality model and that may be needed to use the model in an evaluation process.

A quality model is applied on an artefact to obtain an *AppliedQualityModel* in which its attributes will be *evaluated*. This concept is the cornerstone of a particular QA&M experience. It allows for instance to obtain a quality model for a particular mail server, ERP system, UML class diagram, or whatever. Quality statements over an applied quality model are called *AppliedQualityStatements* and are the refinement of the quality statements (e.g., “The page shall be downloaded in less than 5 secs.”) of a particular quality model *involving* attributes from this model (e.g., *throughput* or *refreshing time*).

Metrics. The class *Metrics* represent possible *methods* to perform *measurements* of attributes to find the *value* to *assign*. Metrics are mainly characterised by the category of their *Scale* (which define their *properties*), their *Unit* of measurement and the *QualityType* of their *Values* (that of course shall be *compatible* with the scale).

Measurement appears as the *assignment* of values to the attributes of the artifacts under consideration by applying a particular *method* of measurement in the form of a metric that applies to the attribute. It is possible for a given attribute to be measured with different methods in different quality models. For this reason, it would have been more natural to model this measurement as a ternary relationship between quality models, artefacts and attributes, but since ternary associations are not supported by the MOF at level M2 [MOF03], we have decomposed them into binaries.

As we stated in section 4, the presence of a metaassociation in the metamodel between the metaclasses MA and MB does not require the presence of an association in layer M1 between the instances of MA and MB. This behaviour does not align well with the special nature of the metaassociations that come up in our generic model: all of them should induce associations or association classes in the reference and domain models. As an example, consider in figure 3 the metaassociation *dependsOn* between

the metaclasses *IndirectAttribute* and *Attribute* (an indirect attribute depends on other attributes). The extension of this association occurs in the reference model (M1) and links a specific instance of *IndirectAttribute* (e.g., the instance *Characteristic* that corresponds to the concept of characteristic in the ISO/IEC9126-1 standard) with another of *Attribute* (e.g., the *Subcharacteristic* from the same standard). However, the reference model should contain an M1 association between the classes *Characteristic* and *Subcharacteristic*, in order to state which specific *Subcharacteristic* instances at M0 (e.g., *Suitability*) depend on which specific *Characteristic* instances (e.g., *Functionality*). Unfortunately, this M1 association is not implied by the *dependsOn* metaassociation. Since it must be enforced, we will make the metaassociation of the generic model induce it by means of the notion of *induced association*.

All the associations that come up in the generic model (fig. 8) have been identified to induce associations in the reference models at level M1; therefore, for each of them, an instance of the *Association* UML metaclass has been defined with the same name. In the particular case of *AssociationClass* heirs (listed in table 1), table 2 shows the metaclasses that, along with the corresponding meta-associations of the generic model, induce association classes in the reference models. For example, we will see in the next section that the metaclass *QualityModel* together with the metaassociation *usedIn* (between *Domain* and *Environment*) induces an association class as shown in fig. 10.

Table 2. Metaassociation and the metaclasses they connect.

Metaclass	Corresponding metaassociation
QualityModel	usedIn
AppliedQualityStatement	concerns
AppliedModel	applicable
Measurement	evaluates
Influences	influences

6. A Reference Model for Stating the Quality of Collection Libraries

In this section we aim at building a reference model for the evaluation of collection libraries such as JCF [AGH00], STL [MS96] and LEDA [MN99]. This experience could be necessary in some situations, remarkably as a particular case of Commercial Off-The-Shelf (COTS) component selection [FSR96].

According to the guidelines outlined in the introduction, we adopt a strategy such that reference models are fine-grained (i.e., each of them refines just those elements directly implied by the approach) making easier their reuse in many contexts. The general strategy is depicted at fig. 9. We decide to divide the experience into two parts. In the first part we build a reference model for COTS selection, called *COTS selection framework* in the figure. This is build upon:

- The ISO/IEC 9126 standard, parts 1 (the quality model structure [ISO01]) and 2 (proposal of external metrics for software artifacts [ISO03]).
- The NFR framework [CNYM00] for establishing different degrees of relationships among quality factors.

- Fenton’s metrics framework [FP98] for presenting metrics fundamental concepts.
- A commercial classification of COTS components coming from the Gartner consulting group to define software domains of interest [Gar04].

At the first level of the hierarchy, we provide separate refinements for each of the parts (fundamentals, metrics and domain) of the generic model focusing on the general layout of each proposal. The three resulting models stay at the M1 MOF layer, with their elements obtained as instantiations of the M2 generic model; in particular, each association in M2 induces an association or association class in M1 as explained at the end of the previous section.

In the second level we include concrete values for the concepts, i.e.: the quality factors provided by the ISO/IEC 9126-1 (efficiency, suitability, ...); the NFR type of links (hurt, some, ...); Fenton’s scales (nominal, ordinal, ...) with simple types; and the concrete elements of the Gartner hierarchy (CRM, ERP, DCM, ...). The added parts of these models are at M0, which means that their elements are instances of the former M1 reference models (which of course are preserved at this level).

In the third level we add more detail to the metrics part refining by means of the ISO/IEC 9126-3 (including types, units, etc.). We did not include this part in the former level again for enhancing reuse: we can propose other concrete metrics starting from the level-two metrics-related model. Also, at this third level, we merge the two level-2 reference models that refined the fundamental concepts part. Finally, at the fourth level, we join the three reference models for fundamental concepts, metrics and context and we obtain the target COTS selection framework. This reference model may be used as starting point for building proposals for supporting COTS selection processes at any kind of software domain, such as ERP systems and collection libraries.

The reference model for collection libraries is based on two basic manipulations. On the one hand, the internal structure of the collection libraries shall be represented in the reference model. This is done at the first two levels, by declaring the artifacts as composable (M1) and then by defining a collection library as composed by collections and algorithms, the former composed of data structures and operations (M0). On the other hand, the COTS selection framework is specialized not only by including this new second-level model but also by modifying the instances in M0 for customizing to this domain. Once we have the model built, we evaluate particular collection libraries such as JCF, STL and LEDA.

Fig. 9. General structure of the COTS selection reference model (FC: fundamental concepts; M: metrics; C: context).

Let's take next a closer look to the resulting models. We present directly the COTS selection framework at fig. 10 (M1) and 11 (M0). We may observe the following:

- The inheritance hierarchy classifying attributes as direct and indirect is combined in this particular model with the classification of quality factors in attributes², subcharacteristics and characteristics made by the standard ISO/IEC 9126. The refined model reflects ISO/IEC's directives concerning the hierarchy (see <<dependsOn>> associations): characteristics may be decomposed into subcharacteristics, subcharacteristics into other subcharacteristics and attributes and, finally, attributes may be decomposed into attributes.
- Metrics will be classified as *observations* and *formula* depending on the way to obtain attribute's values (measuring or calculating).

² Please do not be get confused with the general term "attribute" that we have introduced in the generic model and the particular term "attribute" as defined in the ISO/IEC 9126-1 estandard.

- [illegible]

Fig. 11. The reference model for the COTS selection framework, level M0.

Concerning the customization for the collections library case, fig. 12 summarizes the changes in both levels. At level M1, it is just required to declare a recursive association over *Artifact* to represent the notion of artifact composition. At level M0:

- A new type of value is introduced for measuring the efficiency of data structures and algorithms using the big-Oh notation [BB96]. Correspondingly, the values for this type are also introduced: $O(1)$ or constant; $O(n)$ or lineal; and so on.
- The structure of collection libraries is depicted at three levels: library; collections and algorithms; data structures and operations. Although not shown in the figure, this structure may also be used to define hierarchical metrics. For instance, we may define the efficiency of a collection as a calculation over the efficiency of its components.
- The ISO/IEC 9126-1 model is customized for this particular domain. For instance, we show how the subcharacteristic *Attractiveness* disappears, because it does not apply in this context according to its definition. The same happens with others.
- The metrics for measuring efficiency with the big-Oh notation also appear at M0.

Fig. 12. The COTS selection framework customised to the collection libraries domain.

7. Conclusions

In this paper we have presented a MOF-compliant framework for dealing with software quality modeling. The contributions of the proposal are:

- The most important aspects of software quality are integrated into a single framework: the definition of quality factors, quality requirements, the structure of analysed artifacts, the metrics, and so on. Our framework has been extensively validated, which has resulted in an important improvement with respect to previous versions such as [BF04].
- The framework is compliant with the MOF architecture and defined as an extension of the UML metamodel. Therefore, it has a more clearly defined semantics and we are able to reuse concepts (e.g., classifier), methods (e.g., metamodel extension methodologies) and tools (e.g., for defining profiles) that are well known in the community. Dissemination of the proposal is also ameliorated.
- The proposal is highly structured, with a hierarchy of reference models that allow to define quality proposals in fine-grained chunks and to put them together as desired. In fact, we aim at constructing a catalogue of such chunks to be able to define quality frameworks in an easy and flexible way.
- We have identified some methodological guides in the construction of quality frameworks. The scenario presented in section 2 and illustrated in section 6 is very common and the definition of levels in the hierarchy as done in the example applies to the general case.
- As an additional point, we have defined a concept, the one of induced associations, which has been very valuable in our work. We think that it can be necessary in other contexts different than the target of this paper and therefore one future work is to formalize it and to check if it also applies to other constructs, especially inheritance.

Some other proposals have been proposed for dealing with software quality by means of generic frameworks. An important approach is the work by Kitchenham et al.

[KHL01] which defines a data model and uses it for storing experimental data. The model is not intended to be a general framework but nevertheless it is an attempt to gather quality information in a single model. More recently, there has been a joint effort in the Spanish and south-american community to produce an ontology of software quality with special focus on metrics [Ols+02, Gar+04]. Unfortunately, this work is not spread enough for the moment and to the best of our knowledge is only available in Spanish, except for the continuation [OM04]. Our proposal is more comprehensive than these approaches since it is aimed at embracing the huge variety of concepts behind quality. In addition, these proposals do not provide integration with the MOF architecture or UML metamodels and do not provide this notion of stepwise refinement for building quality frameworks presented in this paper.

References

- [AGH00] K. Arnold, J. Gosling, D. Holmes. *The Java Programming Language*. Addison-Wesley, 3rd edition, 2000.
- [BB96] G. Brassard, P. Bratley. *Fundamentals of Algorithmics*. Prentice-Hall, 1996
- [BF04] X. Burgués, X. Franch. "Formalising Software Quality using a Hierarchy of Quality Models". In *Proceedings of the 15th International Conference on Database and Expert Systems Applications (DEXA'04)*, LNCS 3180, Zaragoza (Spain), Sept. 2004.
- [BM02] A. Burton-Jones, P. Meso. "How good are these UML diagrams? An empirical test of the Wand and Weber good decomposition model". In *Proceedings 23rd International Conference on Information Systems (ICIS'02)*, 2002.
- [CNYM00] L. Chung, B. Nixon, E. Yu, J. Mylopoulos. *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers, 2000.
- [CFQT04] J.P. Carvallo, X. Franch, C. Quer, M. Torchiano. "Characterization of a Taxonomy for Business Applications and the Relationships Among Them". In *Proceedings 3rd International Conference on COTS-Based Software Systems (ICCBSS'04)*, LNCS 2959, Redondo Beach (CA, USA), Feb. 2004.
- [Fir03] D. G. Firesmith. "Using Quality Models to Engineer Quality Requirements". *Journal of Object Technology*, 2(5), 2003.
- [FP98] N. Fenton, S.L. Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. PWS, 1998.
- [FR04] X. Franch, J.M. Ribó. "A two-tiered Methodology for Metamodel Extension Applied to UML 1.4". Technical Report LSI-04-51-R, LSI-UPC, November 2004.
- [FSR96] A. Finkelstein, G. Spanoudakis, M. Ryan. "Software Package Requirements and Procurement". *Procs. 8th IEEE IWSSD*, 1996.
- [Gar04] Gartner Consulting. <http://www4.gartner.com/Init>. Last accessed November 2004.
- [Gar+04] F. García, F. Ruiz, M.F. Bertoa, C. Calero, M. Genero, L. Olsina, M. Martín, C. Quer, N. Tondori, S. Abrahao, A. Vallecillo, M. Piattini. "Una Ontología de la Medición del Software". Informe Técnico UCLM DIAB-04-04-2, Febrero 2004.
- [GPP02] M. Genero, G. Poels, M. Piattini. "Defining and Validating Measures for Conceptual Data Model Quality". In *Proceedings 14th International Conference on Advanced Information Systems Engineering (CAiSE'04)*, LNCS 2348, Riga (Latvia), June 2002.
- [GV95] R. Glass, I. Vessey. "Contemporary Application Domain Taxonomies". *IEEE Software*, 12(4), 1995.
- [IEEE92] IEEE Standard 1061-1992. *Standard for a software quality metrics methodology*. 1992.
- [ISO01] ISO/IEC Standard 9126-1 Software Engineering – *Product Quality – Part 1*, 2001.
- [ISO03] ISO/IEC Standard 9126-2 Software Engineering – *Product Quality – Part 2*, 2003.
- [ISO98] ISO/IEC Standard 15504: Information Technology - Software Process Assessment, 1998.

- [JSZ04] Y. Jiang et al. "On the Classification of UML's Meta Model Extension". In *Proceedings of the UML 2004*, LNCS 3273, 2004.
- [KHL01] B. Kitchenham, R. Hugues, S.G. Linkman. "Modeling Software Measurement Data". *IEEE Transactions on Software Engineering*, 27(9), 2001.
- [KKP90] S. Keller, L. Kahn, R. Panara. "Specifying Software Quality Requirements with Metrics". *System and Software Requirements Engineering* – IEEE Computer Society, 1990.
- [MOF03] *MOF 2.0 Core Final Adopted Specification*. Document ptc/03-10-04, available at <http://www.uml.org/>, last accessed November 2004.
- [Moo98] D.L. Moody. "Metrics for Evaluating the Quality of Entity Relationship Models". In *Proceedings of the 17th International Conference on Conceptual Modelling (ER'98)*, Singapore, LNCS 1507, November 1998.
- [MS96] D.R. Musser, A. Saini. *STL Tutorial and Reference Guide*. Addison-Wesley, 1996.
- [MN99] K. Mehlhorn and S. Näher. *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press, 1999.
- [Ols+02] L.A. Olsina, M. F. Bertoa, G.J. Lafuente, M.A. Martín, M. Katrib, A. Vallecillo. "Un Marco Conceptual para la Definición y Explotación de Métricas de Calidad". *Procs. Jornadas de Ingeniería del Software y Bases de Datos (JISBD)*, Madrid, 2002.
- [OM04] L. Olsina, M.A. Martín. "Ontology for Software Metrics and Indicators: Building Process and Decisions Taken". In *Proceedings of Web Engineering - 4th International Conference (ICWE)*, LNCS 3140, 2004.
- [SEI95] Software Engineering Institute (CMU). *The Capability Maturity Model: Guidelines for Improving the Software Process*. Addison-Wesley, 1995.
- [Ste46] S.S. Stevens. "On the theory of scale types and measurement". *Science* 103, 1946.
- [UML03] *UML 2.0 Infrastructure Final Adopted Specification*, document ptc/03-09-15 and *UML 2.0 Superstructure Final Adopted Specification*, document ptc/03-08-02, available at <http://www.uml.org/>, last accessed November 2004.
- [Zus98] H. Zuse. *Framework of Software Measurement*. De Gruyter, 1998.

**Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya**

Research Reports - 2004

- LSI-04-1-R : *Automatic Generation of Polynomial Loop Invariants: Algebraic Foundations*, Rodríguez, E. and Kapur, D.
- LSI-04-2-R : *Comparison of Methods to Predict Ozone Concentration* , Orozco, J.
- LSI-04-3-R : *Towards the definition of a taxonomy for the cots product´s market* , Ayala, Claudia P.
- LSI-04-4-R : *Modelling Coalition Formation over Time for Iterative Coalition Games*, Mérida-Campos, C. and Willmott, S.
- LSI-04-5-R : *Illegal Agents? Creating Wholly Independent Autonomous Entities in Online Worlds*, Willmott, S.
- LSI-04-6-R : *An Analysis Pattern for Electronic Marketplaces*, Queralt, A. and Teniente, E.
- LSI-04-7-R : *Exploring Dopamine-Mediated Reward Processing through the Analysis of EEG-Measured Gamma-Band Brain Oscillations*, Vellido, A. and El-Deredy, W.
- LSI-04-8-R : *Studying Embedded Human EEG Dynamics Using Generative Topographic Mapping*, Vellido, A. and El-Deredy, W. and Lisboa, P.J.G.
- LSI-04-9-R : *Similarity and Dissimilarity Concepts in Machine Learning*, Orozco, J.
- LSI-04-10-R : *A Framework for the Definition of Metrics for Actor-Dependency Models*, Quer, C. and Grau, G. and Franch, X.
- LSI-04-11-R : *QM: A Tool for Building Software Quality Models*, Carvallo, J.P. and Franch, X. and Grau, G. and Quer, C.
- LSI-04-12-R : *COSTUME: A Method for Building Quality Models for Composite COTS-based Software Systems*, Carvallo, J.P. and Franch, X. and Grau, G. and Quer, C.
- LSI-04-13-R : *Enabling Collaboration in Virtual Reality Navigators*, Theoktisto, V. and Fairén, M. and Navazo, I.
- LSI-04-14-R : *DesCOTS: A Software System for Selecting COTS Components*, Carvallo, J.P. and Franch, X. and Grau, G. and Quer, C.
- LSI-04-15-R : *Evaluation and symmetrisation of alignments obtained with the Giza++ software*, Lambert, P. and Castell, N.
- LSI-04-16-R : *A note on the use of topology extensions for provoking instability in communication networks*, Blesa, M.J.
- LSI-04-17-R : *An ISO/IEC-compliant Quality Model for ER Diagrams*, Costal, D. and Franch, X.
- LSI-04-18-R : *A Case Study on Pruning General Ontologies for the Development of Conceptual Schemas* , Conesa, J.
- LSI-04-19-R : *Adding Efficient and Reliable Access Paths to the JCF*, Marco, J. and Franch, X.

- LSI-04-20-R : *Exploiting Simple Corporate Memory in Iterative Coalition Games*, Mérida-Campos, C. and Willmott, S.
- LSI-04-21-R : *On the Semantics of Operation Contracts in Conceptual Modeling* , Queralt, A. and Teniente, E.
- LSI-04-22-R : *Complexity issues on bounded restrictive H-coloring*, Díaz, J. and Serna, M. and Thilikos, D.M.
- LSI-04-23-R : *Chromatic number in random scaled sector graphs*, Díaz, J. and Sanwalani, V. and Serna, M. and Spirakis, P.
- LSI-04-24-R : *Bounds on the bisection width for random d-regular graphs*, Díaz, J. and Serna, M. and Wormald, N.C.
- LSI-04-25-R : *Open Source environment to define constraints in route planning for GIS-T*, Pérez, L. and Silveira, A. da M.
- LSI-04-26-R : *A basic repository of operations for the refinement of general ontologies*, de Palol, X.
- LSI-04-27-R : *Tetrahedral mesh subdivision based on underlying volume data*, Rodríguez, L. and Navazo, I. and Vinacua, A.
- LSI-04-28-R : *The Price of Connectedness in Expansions*, Fomin, F.V. and Fraigniaud, P. and Thilikos, D.M.
- LSI-04-29-R : *Smaller kernels for hitting set problems of constant arity*, Nishimura, N. and Ragde, P. and Thilikos, D.M.
- LSI-04-30-R : *Searching Spatial Sense in the Ontological World: Discovering Spatial Objects*, Morocho, V and Pérez, L. and Saltor, F.
- LSI-04-32-R : *Implementation considerations of an Expert System to assess Stream Water Quality management*, Cabanillas, D. and Willmott, S.
- LSI-04-33-R : *Multisided patches*, Pla, N. and Vigo, M. and Cotrina, J.
- LSI-04-34-R : *SVMTool: A general POS tagger generator based on Support Vector Machines*, Giménez, J. and Màrquez, Ll.
- LSI-04-35-R : *A distributed and mobile component system based on the ambient calculus*, Mylonakis, N. and Orejas, F.
- LSI-04-36-R : *Developing Competitive HMM PoS Taggers Using Small Training Corpora*, Padró, M. and Padró, Ll.
- LSI-04-37-R : *The AlignmentSet Toolkit*, Lambert, P.
- LSI-04-38-R : *Integración de Fuentes de Datos espaciales: análisis e implementación de una Ontología de términos espaciales: Primera Parte - - Creación de una Ontología*, Ramos, Erik G.
- LSI-04-39-R : *Integración de Fuentes de Datos espaciales: análisis e implementación de una Ontología de términos espaciales: Segunda Parte - - Evaluación de similitudes*, Ramos, Erik G.
- LSI-04-40-R : *Kernels on Structured Domains*, Valentín, L.

- LSI-04-41-R : *Determining the Structural Events that May Violate an Integrity Constraint*, Cabot, J. and Teniente, E.
- LSI-04-42-R : *Review of Statistical Word Alignment Techniques* , Lambert, P.
- LSI-04-43-R : *Algoritmos geneticos en el problema de la solucion deseada* *Optimizacion de parametros* , Barreiro, E. and Joan-Arinyo, R. and Luzón, M.V.
- LSI-04-44-R : *Generative Topographic Mapping as a constrained mixture of Student t-distributions: Theoretical developments* , Vellido, A.
- LSI-04-45-R : *Adapting Agent Communication Languages for Web Service to Web Service Communication*, Willmott, S. and Fernández-Peña, F. O. and Mérida-Campos, C. and Constantinescu, I.
- LSI-04-49-R : *A brief on constraint solving*, Hoffmann, C.M. and Joan-Arinyo, R.
- LSI-04-50-R : *Missing data imputation through Generative Topographic Mapping as a mixture of t-distributions: Theoretical developments*, Vellido, A.
- LSI-04-51-R : *A two-tiered Methodology for Metamodel Extension Applied to UML 14*, Franch, X. and Ribó, J. M.
- LSI-04-52-R : *Virtual reality for prostate gland cryosurgery*, Joan-Arinyo, R.
- LSI-04-53-R : *High level communication functionalities for wireless sensor networks*, Àlvarez, C. and Díaz, J. and Petit, J. and Rolim, J. and Serna, M.
- LSI-04-54-R : *A MOF-Compliant Approach to Software Quality Modeling*, Burgués, X. and Franch, X. and Ribó, J. M.
- LSI-04-55-R : *Pure Nash equilibria in games with a large number of actions*, Àlvarez, C. and Gabarró, J. and Serna, M.
- LSI-04-56-R : *An Optimal Anytime Estimation Algorithm*, Gavaldà, R.

Hardcopies of reports can be ordered from:

Núria Sanchez
 Departament de Llenguatges i Sistemes Informàtics
 Universitat Politècnica de Catalunya
 Campus Nord, Mòdul C6
 Jordi Girona Salgado, 1-3
 03034 Barcelona, Spain
 nurias@lsi.upc.es

See also the Departament WWW pages, <http://www.lsi.upc.es/>