ITEA 2 Office
High Tech Campus 69-3
5656 AG Eindhoven
The Netherlands
Tel : +31 88 003 6136
Fax : +31 88 003 6130
Email : info@itea2.org
Web : www.itea2.org
ITEA 2 is a EUREKA strategic ICT cluster programme

INFORMATION TECHNOLOGY FOR EUROPEAN ADVANCEMENT

D2.1: Specification Language and Interchange Format for measurements and metrics

MEASURE

.....

Executive summary

This document describes the specification language and the interchange format for measurements and metrics chosen (and introduced) by the MEASURE project consortium, in order to express the software quality metrics in a uniform and non-ambiguous manner. The document includes the rationale behind the language of choice and it also includes the state of the art regarding the solutions and models for the formal (or semi-formal) specification of metrics and measurements. Furthermore, the document includes certain examples of metrics (properly described in the specification language) that are envisioned to be relevant to the industrial partner test cases.

Table of Contents

1. Introduction

1.1. Context

The main goal of the ITEA 3 Measuring Software Engineering (MEASURE) Project is to measure the quality and efficiency of software, as well as to reduce the costs and time-to-market of software engineering in Europe, by implementing a comprehensive set of tools for automated and continuous measurement. Therefore, this project aims at providing a toolset for future projects to properly measure their software quality and efficiency. More importantly, it opens a new field for innovation. The second innovation will be in the advanced analytics of the measurement data enabled by the project. To reach this ambitious goal, the project will iteratively and incrementally:

1. Define novel, better metrics and develop methods and tools for automated, precise, and unbiased measurement of software engineering activities and artefacts;

2. Develop methods and tools for analyzing the (big) data produced by the continuous measurement to enable continuous improvement of performance;

3. Validate the developed metrics, approaches, and measurement tools by integrating them into software development environments and processes of the project's industrial case studies, and iteratively improve them, based on the feedback gathered from the industrial partners and other industrial actors;

4. Validate the developed tools by analyzing the data gathered from the industrial partners and measuring the impact of the improvements suggested by the analysis tools. A practical example of a measurement-based suggestion could be pointing out an area of source code not covered by automated test suite and generating new targeted test automation scripts based on manual test cases recorded during continuous measurement;

5. Support management of decision making by visualizing the results of continuous measurement at targeted level of abstraction, i.e., providing different visualization or even completely different metrics for developers and managers.

1.2. Deliverable Content

In order to achieve the goals of the MEASURE project, a common specification language to define and to interchange the metrics and measurements is required. This document contains the findings of the project's consortium, in regard to which formal (or semi-formal) language specification is the best suited for the previously mentioned task. With a well-defined specification language, the goals of the MEASURE project can be achieved without ambiguity with respect to the metrics' specification and also while interchanging these metrics and measures.

In this document, we show that the Structured Metrics Meta-model (SMM) of the Object Management Group (OMG) is certainly a language that has the characteristics required to describe metrics and has associated facilities for their interchanges. Therefore, the SMM is to be adopted as the specification language of the MEASURE project. An associated language that is to be utilized is the XML Metadata Interchange (XMI). XMI defines an XML integration framework for defining, interchanging, manipulating and integrating XML data and objects; XMI-based standards are used for integrating tools, repositories, applications and data warehouses. The conclusions were made after a systematic literature review. Likewise, a simple XML-based language is considered for data exchange. Findings with respect to the

different languages to specify and exchange metrics and measurements are described in the State of the Art Section (Section 2) of this document.

The rest of the deliverable is organized as follows: Section 2 contains the state of the art of the existing languages for metric specification and interchange. Section 3 contains the description of SMM, XMI, and the integration into the consortium's tools, and examples. Finally, Section 4 concludes the deliverable.

2. State of the Art: Languages for Metrics and Measurements Specification

Measuring has become a fundamental aspect of software engineering. Accurate measurement is proving to be highly efficient in various domains. For example, measurement is vital in the construction of high quality prediction systems, in the context of the improvement of software development and maintenance projects. Further, and more important, accurate measurement is required for the evaluation to guarantee a system's quality (by highlighting problematic areas), and in the determination of better work practices with the end of assisting practitioners and researchers in their work.

Software measurers are, moreover, important tools that assist in the evaluation and institutionalization of Software Process Improvement in the organizations that develop them. Software Measurement is, in fact, a key element in initiatives such as SW-CMM (Capability Maturity Model for Software), ISO/IEC 15504 (SPICE, Software Process Improvement and Capability dEtermination) and CMMI (Capability Maturity Model Integration). The ISO/IEC 90003:2004 standard also highlights the importance of measurement in managing and guaranteeing quality.

There exist various methods and standards with how to carry out measurements in a precise and systematic manner, of which the most representative are:

- **Goal Question Metric (GQM)**: The basic principle of GQM is that the carrying out of the measurement must always be oriented towards an objective. GQM defines an objective, refines that objective into questions and defines measures which attempt to answer those questions.
- **Practical Software Measurement (PSM)**: The PSM methodology is based upon the experience obtained from organizations through which the best manner in which to implement a software measurement program with guarantees of success is discovered.
- **IEEE 1992 (Methodology for Software Quality Measures)**: according to the IEEE 1992 standard, software quality can be considered as the extent to which the software possesses a clearly defined and desirable combination of quality attributes. This standard deals with the definition of software quality for a system by using a list of software quality attributes which are required by the system itself.
- **ISO/IEC 15939**: this international standard identifies the activities and tasks which are necessary to successfully identify, define, select, apply and improve software measurement within a general project or within a business's measurement structure.

The availability of a description language that allows to represent the elements which must be taken into account in the measurement processes might, therefore, be important in decision making and in process improvement. Among these languages, we present:

- **GMSL**: Goanna Metric Specification Language
- **Slammer:** Specification LAnguage for Modelling MEasurements and Redesigns
- **SMML**: Software Measurement Modeling Language

More details about these languages are presented in the following subsections.

2.1. GMSL: Goanna Metric Specification Language

Goanna [2] is a tool that performs deep analysis of C/C++ source code using a model checking technique. It verifies the presence or absence of bugs, memory leaks and security vulnerabilities. The tool is fully path-sensitive and inter-procedural; it utilizes additional static analysis techniques such as abstract interpretation. It also provides two specification languages for defining source code verification. The first language is a tree-query language based on XPath for finding constructs and patterns of interest in the abstract syntax tree (AST), correspondingly, it is called Goanna XPath Specification Language (GXSL). The second language is

based on temporal logic expression over paths in the CFG, it is called Goanna Property Specification Language (GPSL). GPSL allows the embedding of GXSL expression. Figure 1 shows how these languages fit into the static analysis.

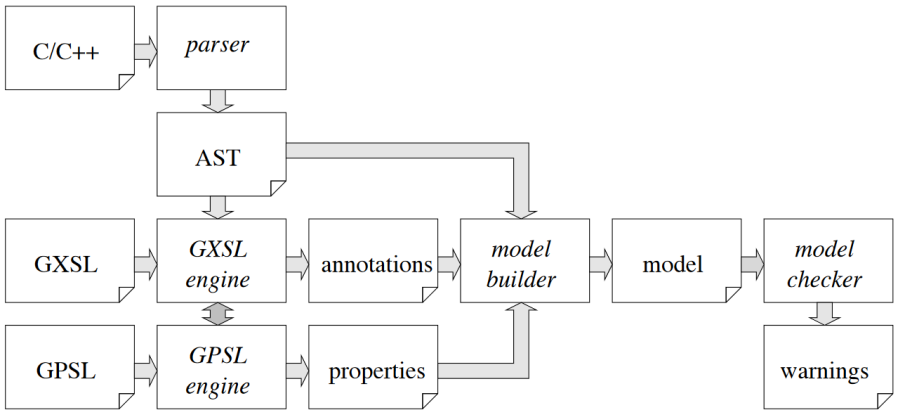

Figure 1. Goanna's model checking approach for static analysis of C/C++ code

The Goanna Metric Specification Language (GMSL) provides a way to define metrics on an abstract level. A prerequisite for the use of GMSL is a query engine that returns sets of nodes of the AST for which certain syntactic properties hold. Goanna provides a language to define functions that select certain nodes of the AST of a program. The queries are always evaluated on the entire AST but it is possible to pass parameters to the queries when referring to a particular node (or a sub-tree) in the AST. The result of a GXSL query is a set of AST nodes.

The Goanna GMSL interpreter is an extension to the existing Goanna analyzer. An overview of the extended architecture can be found in Figure 2. The metrics interpreter is situated on top of the existing GXSL query engine, i.e., it mostly uses existing library functions for pattern matching constructs of interest, and interprets the metrics' specification written in GMSL.

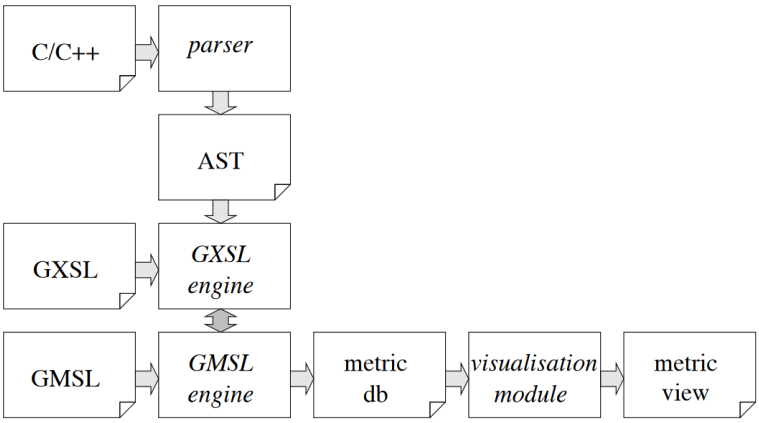

Figure 2. Goanna for computing metrics

The Goanna Metric Specification Language (GMSL) is very interesting for C/C++ code analysis. Nevertheless, in the context of MEASURE, we would like to target other languages (like Java) and also take into account metrics related to different phases of the software lifecycle including design, implementation test and operation as well as metrics related to engineering processes. For these reasons, this language is not selected.

2.2. Slammer: Specification LAnguage for Modelling MEasurements and Redesigns

SLAMMER [3] allows the customization of generic measurements and redesigns for a given DSVL (Domain Specific Visual Language). It has been defined through a meta-model (Figure 3) that contains generalizations of some of the main types of measurements. These include measurements for global model properties (such as number of cycles and size), single element features (e.g. methods of a class in object oriented languages), features of groups of elements (e.g. their similarity or coupling) and paths (e.g. hierarchies in object oriented languages, navigation paths in web design languages, etc.). SLAMMER allows customizing these generic measurements by using visual patterns, creating new ones (procedurally), and composing them in order to build more complex measurements.

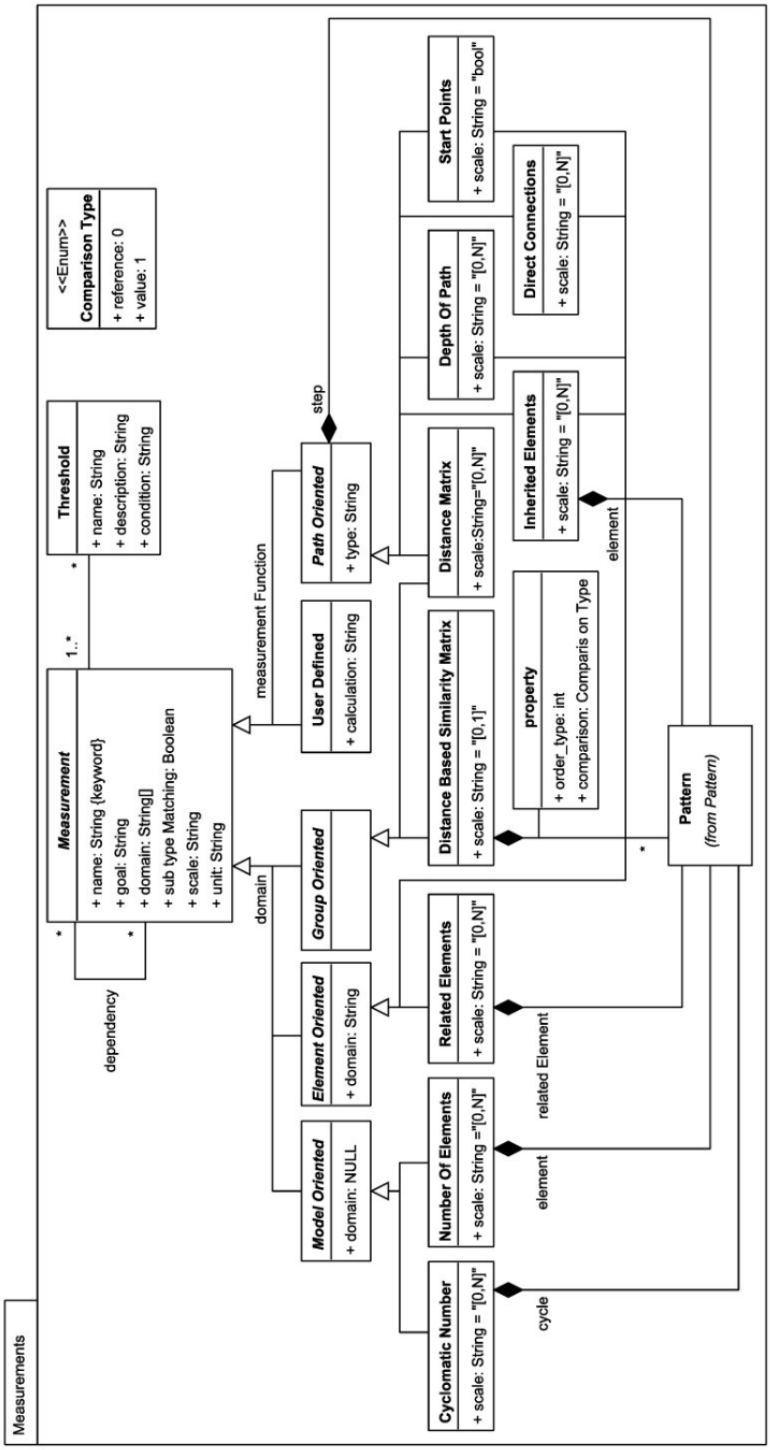

Figure 3. Slammer Measurement Metamodel

It is also possible to specify threshold values for the measurements that may have an associated action described either procedurally, by customizing a generic action template or by using a graph transformation system. This can be useful if the action performs a redesign that improves the quality of the model or modifies it with respect to the known design patterns. In order to define this language, related works on the definition of ontologies concerning software measurement [1,4] have been taken into account, as well as the international standard for software quality ISO 15939 [5].

These approaches have been implemented in the meta-modelling tool AToM [6]: A Tool for Multi-Formalism Modelling and Meta-Modelling. In this way, the DSVL designer can enrich the DSVL specification with a SLAMMER model specifying a number of measurements and redesigns for it. Starting from this definition, a modelling environment is generated for the language; this environment integrates the measurements and the previously defined actions.

The SLAMMER language looks promising given the MEASURE project requirements. Nevertheless, it provides only support for direct metrics (i.e., atomic metrics). In the context of MEASURE, more research work is planned to target complex and dynamic metrics. For example, metrics defined as the correlation of other metrics and where their definition can change over time (based on a machine learning approach, for instance). For this reason, another language has been chosen for specifying software quality metrics.

2.3. SMML: Software Measurement Modeling Language

SMML [1] is a language which permits software measurement models to be built in a simple and intuitive manner. This language has been produced by using the Software Measurement Meta-model (SMM) as the Domain Definition Meta-model that is also an OMG standard specification [7]. However, SMML is not a model, it is a language and as any language it has an abstract syntax, a concrete syntax and a semantics.

The SMML syntax is obtained from the Software Measurement Ontology (SMO). This ontology contains all the elements (concepts and relationships) of the software measurement domain. The Software Measurement Meta-model includes the packages which are alignments with the sub-ontologies of SMO (*Basic, Characterization and Objectives, Measures Software, Measurement Approaches and measurement Action*). However, for the development of the Language, all the packages are of interest, in exception of Measurement Action. This package has been excluded as it contains the elements which are related to measurement but not to the problem domain. The following figure shows the structure of the packages upon which the SMML language is based.

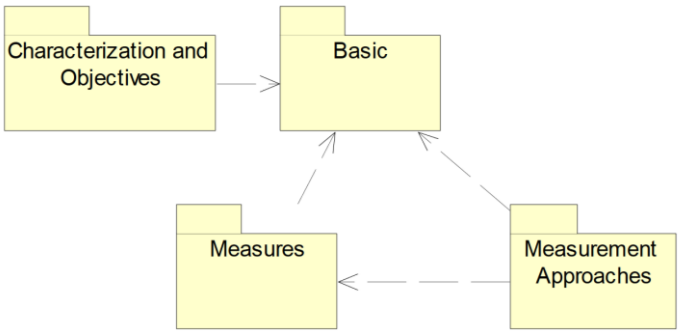

Figure 4. Structure of the package

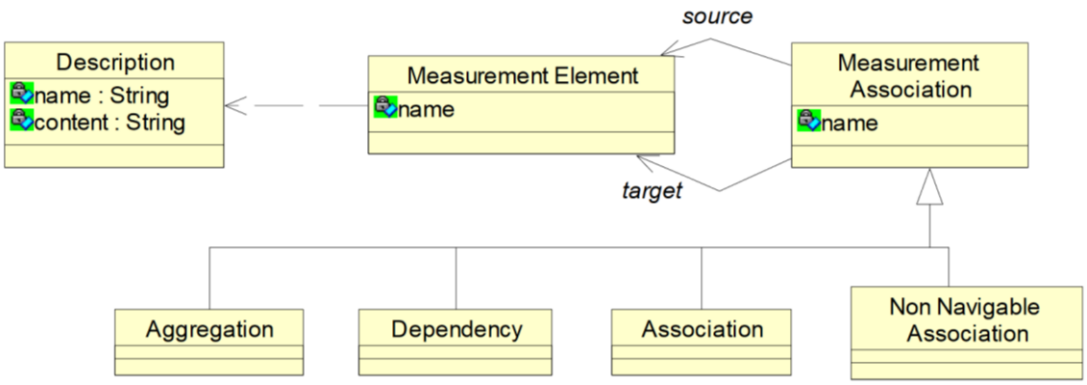

Figure 5. Basic Package

The SMML language described in this section allowed us to better understand the semantics of the SMM meta-model. Unfortunately, this language, which is the result of research work, there does not exist any maintained tool that support it. For this reason, we decided to adopt SMM natively on one of the consortium tools. The Modelio tool permits to define a DSML (domain specific modeling language) based on SMM. More details about this are presented in section 3.

2.4. Conclusion

In conclusion, in this section, we have covered some of the most common languages and formats to describe measurements of software and metrics. A summary of their features is provided in the following table.

Table 1. Comparison matrix for metric specification languages

	Software lifecycle coverage	Metric complexity	Tool	Based on standard
GMSL	(-) Only C/C++ is supported	(-) Only direct metrics	(+) Available	(-) No
SLAMMER	(+) Can covers all SLC	(-) Only direct metrics	(+) Available	(-) No
SMML	(+) Can covers all SLC	(+) Direct and complex metrics	(-) Not maintained	(+) Based on SMM
MEASURE APPROACH	**(+)** Can covers all SLC	**(+)** Direct and complex metrics	**(+)** Based on Modelio	**(+)** Based on SMM

This analysis lets us decide which language can be used along the execution of MEASURE project. The basic idea is to rely on SMM, which is an OMG standard and to use the Modelio tool to support it. The language allows targeting direct and complex metrics and can be supported by the MEASURE framework solution. More details about the integration of SMM into the tools of the consortium is presented in the next section.

3. MEASURE Specification Language and Interchange Format

3.1. MEASURE Language Proposal

Taken into account the existing languages for describing and interchanging software measurements and metrics available in the literature (and presented in the previous section), some conclusions can be drawn. First, the majority of software measurements and metrics are considered to be defined only via the source code [1,4]. If access to the software source code is not possible, measuring its quality should still be possible using other methods. For that reason, we consider that the language used to specify the measurements and metrics should be more generic. Further, as seen in [3], some approaches focus on the "main types of measurements", however, one of the MEASURE project goals is to create a very flexible architecture. With a flexible architecture, different tools and approaches can contribute to assess the quality of given software.

For the previously mentioned reasons, we have found out that the Structured Metric Meta-model (SMM) [7] language is highly flexible and is able to accurately describe the metrics and measurements of software. For the interchange format, the associated XML Metadata Interchange (XMI) [8] language can be utilized. There are many advantages in using XMI as the interchange format. One of the reasons is that frequently there exist tools which can only export simple data. Moreover, XML is a widely spread language and thus, is easy to adopt into such tools. For instance, in [9] we have exported the collected measurements of a particular software using power meter, and interchanged those measurements using the XMI standard. Nonetheless, for the MEASURE platform we have chosen a trimmed down XML version as the interchange format. Given the fact that the interchange format is 'inspired' from XMI, we the XMI description and leave the description of the MEASURE deployment format for the future. Nonetheless, important remarks about this language and the reasons why it was chosen are discussed.

In the following subsections, we briefly describe the SMM, and XMI languages. Later, we exhibit the current integrations of such languages into the consortium's tools.

3.2. Structured Metric Meta-model Language Description

The Structured Metric Meta-model (SMM) [7] is an OMG standard specification. It defines the meta-models to define all necessary concepts of a software measurement architecture. The SMM specification allows to describe the following concepts:

- the measure calculations,
- the calculations results,
- the relationship between the previous concepts,
- the measurement context.

In the SMM specification, these concepts are referred to as: *Measure*, *Measurement*, *SmmRelationship* and *Observation*, correspondingly. The definition is performed with the use of the UML language.

Figure 6 shows a model of the SMM architecture. There are the key concepts defining a software measure architecture with the SMM specification 1.1, and the links between them. Thus, one can see the parent entity *SmmElement,* which all other elements inherit from. From this entity, the hierarchy link between those concepts is described. More details about such links are given later. We first present the *SmmElement* entity, the *AbstractMeasureElement* and the Measurement elements. Later on, the *SmmRelationship* concepts, and finally, the observation properties are being described.

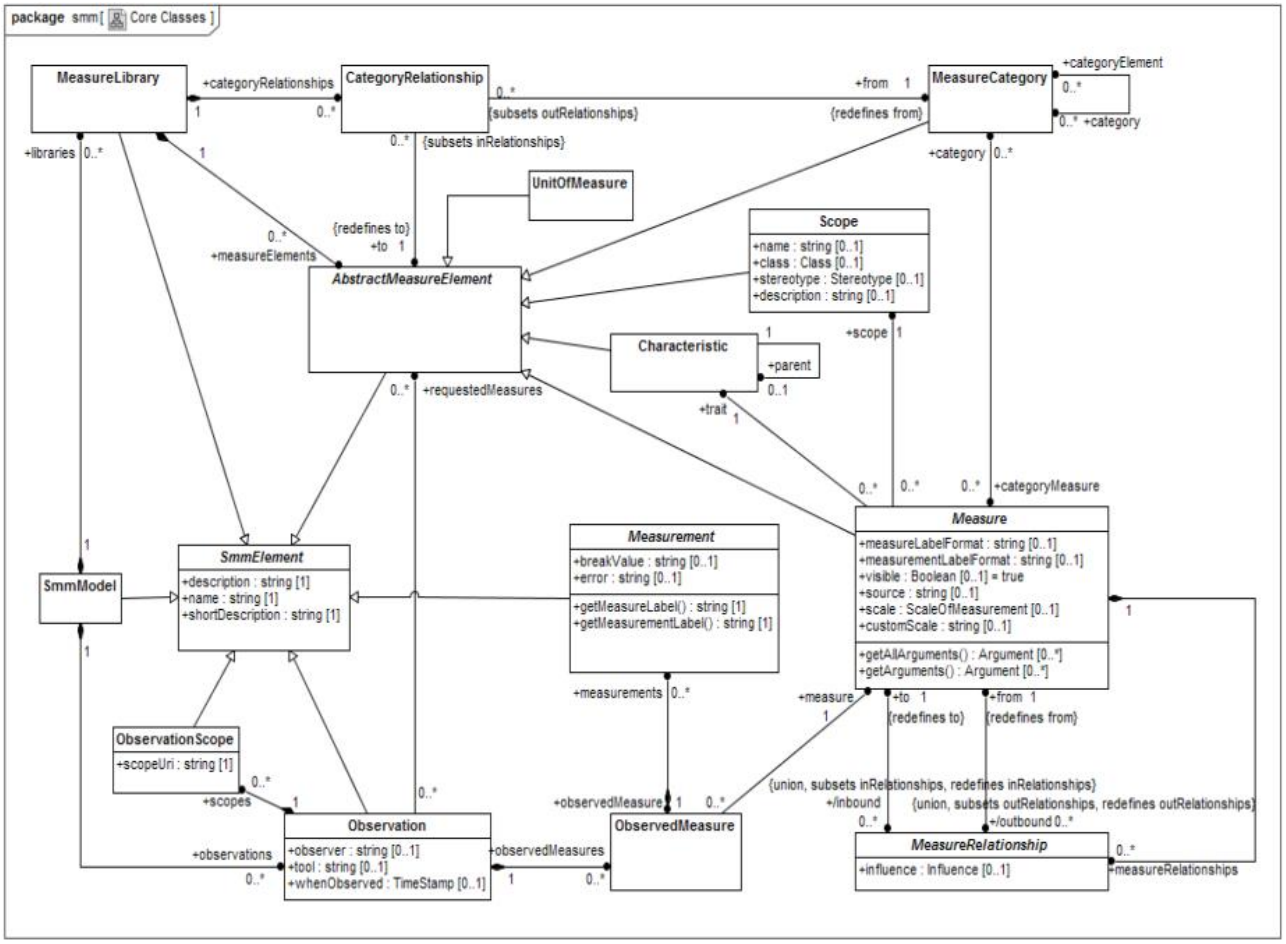

Figure 6 SMM core classes

3.2.1. SmmElement

SmmElement is the top abstract class in the SMM specification hierarchy. As shown in the Figure 7, it allows to declare a name and a description of entities specified through optional attributes: name, shortDescription, and description.

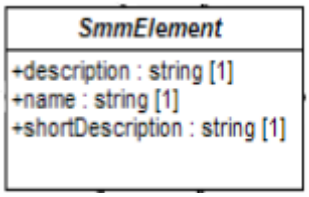

Figure 7 SMMElement class

Entities that directly inherit from *SmmElement* are:

- *SmmModel*,
- *MeasureLibrary*,
- *AbstractMeasureElement*,
- *Measurement*,

- *SmmRelationShip*,
- *Observation*,
- *ObservationScope*,
- *ObservedMeasure*,
- *Argument*.

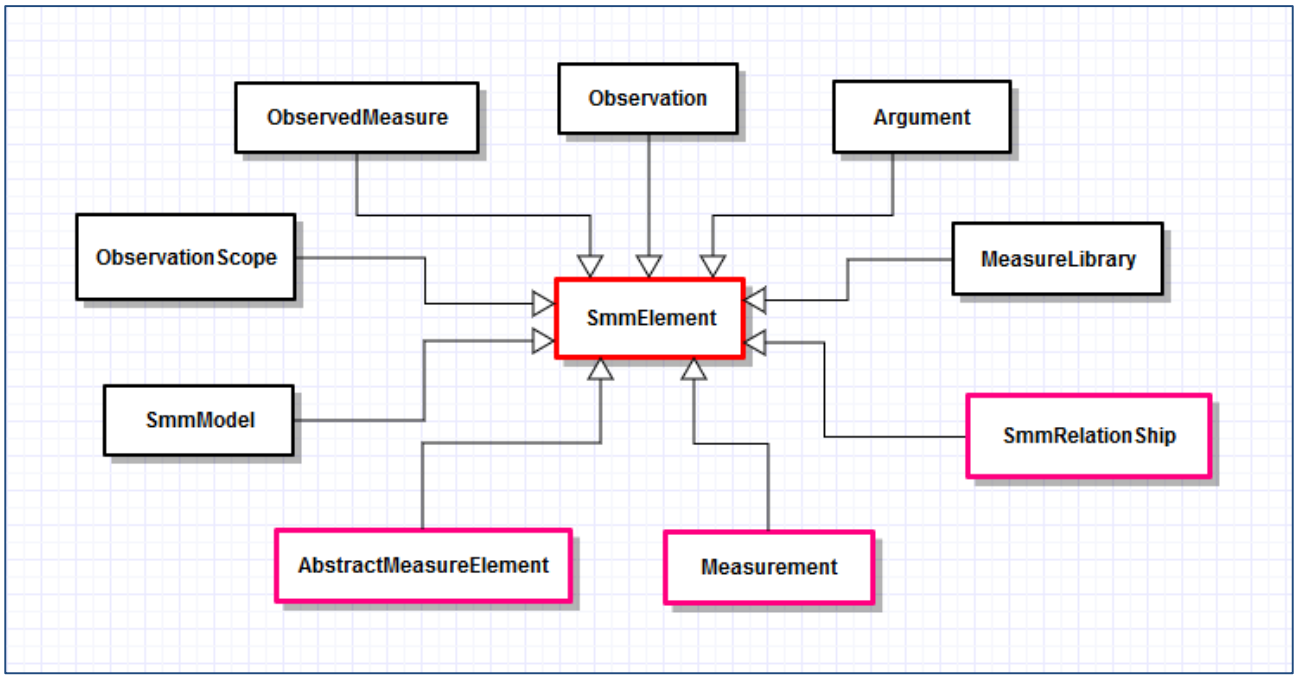

Figure 8 SMM first level hierarchy

Figure 8 presents the first level hierarchy of the SMM specification. Entities shown in this figure are described below.

SmmModel is an abstract class which defines the entry point into a *SmmModel* and provides the top-level container for all the elements of the SMM. It is associated with all *MeasureLibrary* and all observations defined in the model.

MeasureLibrary class represents a set of measure references that are independent of the measurand (SMM term to describe the measured object) and the context. It is associated with a set of *AbstractMeasure* and a set of category relationships, related to those measures.

The *AbstractMeasureElement*, *Measurement* and *SmmRelationShip* are abstract classes which allow to define measures, measurement and relations between entities. These elements will be presented in details in the following subsections.

Observation entity represents the context of the measurement model. It depends on the classes, namely *ObservationScope*, *ObservedMeasure* and *Argument* which allow to define different context aspects. Details about the observation are also given below.

3.2.2. AbstractMeasureElement

This abstract class, is the top of measure calculation definition. As shown in the Figure 9, it can be refined in five different entities:

- *Scope*,
- *Characteristic*,
- *UnitOfMeasure*,
- *Operation*,
- *OclOperation*,
- *Measure*.

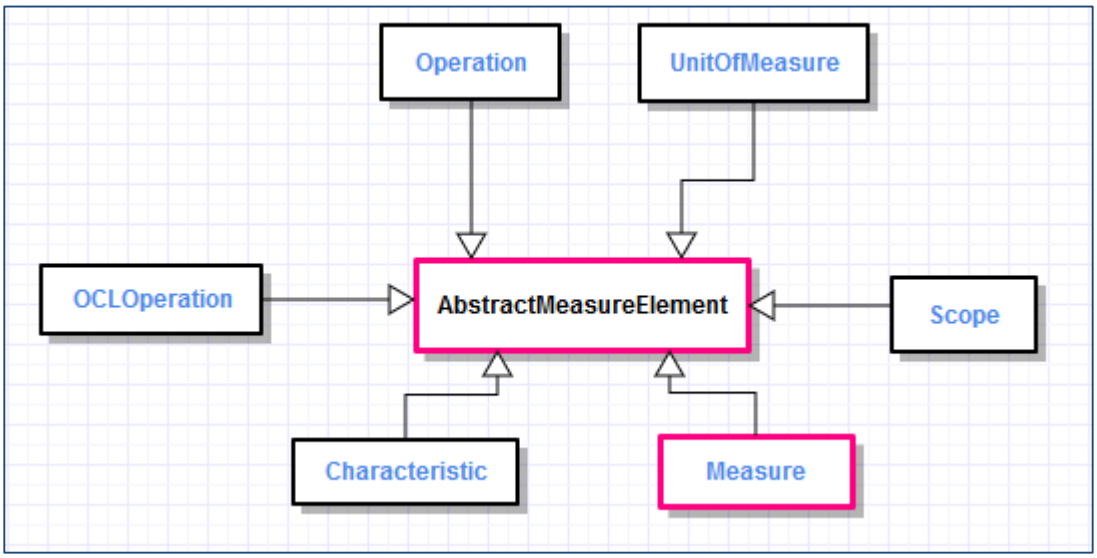

Figure 9 Measure level hierarchy (1)

Scope, *Characteristic* and *UnitOfMeasure* classes allow to have specify the *Measure* details. The *Scope* defines the domain of the measurement of the Measures set. The *Characteristic* class allows to specify the kind of the scope on which the *Measure* is applied, while the *UnitOfMeasure* class provides a representation for units of measure definition of the *Measure* class.

Operation class defines the procedure to be executed by a *Measure* with the needed parameters. *OCLOperation* class defines OCL (Object Constraint Language) helper methods for having more information about the parsing of measures.

The abstract class *Measure* allows to define the different types of calculations (see Figure 10). The abstract class *Measure* can be refined in two concepts: *GradeMeasure* and *DimensionalMeasure*.

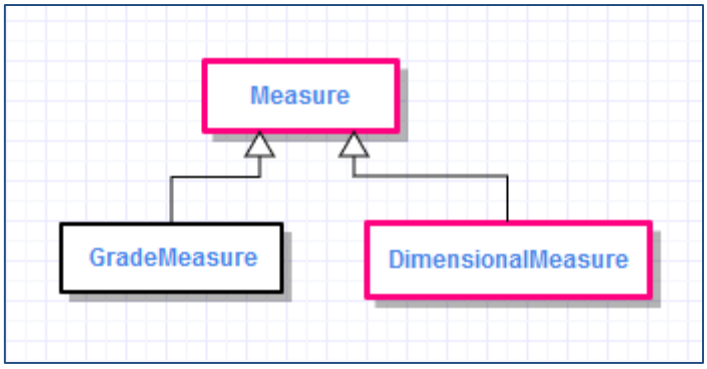

Figure 10 Measure level hierarchy (2)

The *GradeMeasure* class defines calculations that return a classification of a dimensional measure.

The *DimensionalMeasure* abstract class allows to specify measures that assign numeric values. Dimensional measures have units of measure and their values belong to a dimension.

Figure 11 depicts the measure hierarchy from *DimensionalMeasure*. This entity can be refined in:

- *DirectMeasure*,
- *CountingMeasure*,
- *RankingMeasure*,
- *CollectiveMeasure*,
- *BinaryMeasure*,
- *RatioMeasure*,
- *NamedMeasure*,
- *RescaledMeasure*.

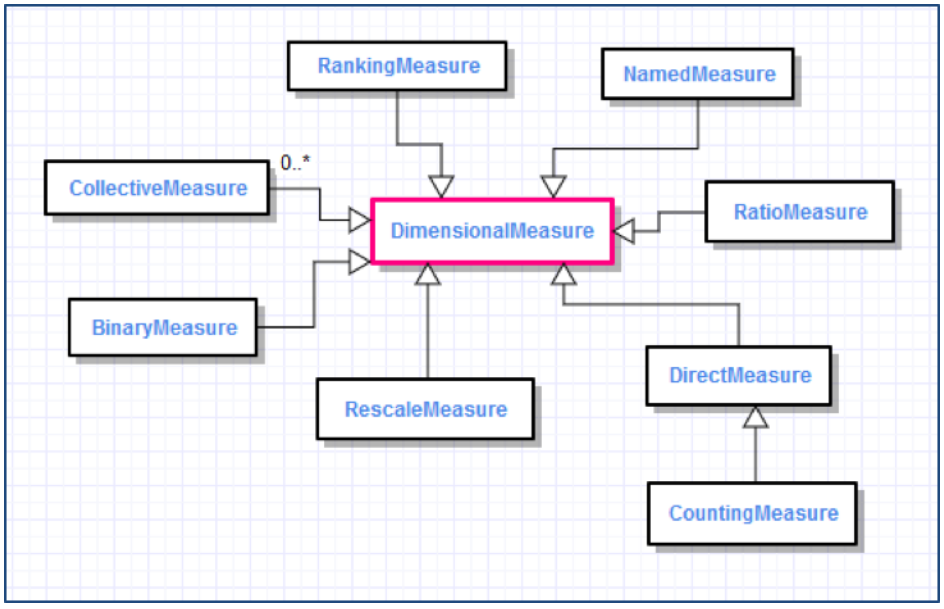

Figure 11 Measure level hierarchy (3)

There are two types of measures, the *direct measure*, directly applied to the measurand and the rest, which are applied to the result of direct measure, called *derived measure*.

DirectMeasure and *CountingMeasure* classes define the measures directly applied to the measured software. The first one is associated with an *Operation* entity that specifies the calculation operation to be executed. The second refines the first entity and it represents a *DirectMeasure* which operation returns the value of 0 or 1. Those measures are known as direct measures.

Regarding the derived measures, there exists the *RankingMeasure* class, which defines measures that classify the results of a *DirectMeasure*. On the other hand, the *CollectiveMeasure* class represents measures that collect measurements of a given measuring element and entities similarly related to this element; it has an attribute Accumulator which declares the accumulation function. In the same way, the *BinaryMeasure* class represents a measure that collects measurements of two entities related to the measuring entity. It defines a BinaryFunctor attribute which refers to the binary function that combines the two base results.

The *RatioMeasurement* class represents measures which return the ratio of two other measure results. Furthermore, the *NamedMeasure* defines well-known measures and its name is sufficient to identify the needed measure. Meanwhile, the *RescaleMeasure* class refers to the calculations that compute a measurement already defined in a unit of measure to another unit of measure.

All those measures return a result that can be modelled by the SMM specification, referred to as Measurement.

3.2.3. Measurements

SMM defines the different types of results with the *Measurement* entity. For each type of *Measure* there is the same type of *Measurement*. The *Measurement* hierarchy is the same as that of *Measure*. *Measurement* is an abstract class and can be specified in:

- *GradeMeasurement*,
- *RankingMeasurement*,
- *DimensionalMeasurement*.

GradeMeasurement and *RankingMeasurement* classes inherit directly from the Measurement abstract class. The first entity represents the result of the *GradeMeasure* and the second of the *RankingMeasure*.

The *DimensionalMeasurement* abstract class defines the result of *DimensionalMeasure* and can be refined in:

- *DirectMeasurement*,
- *CountingMeasurement*
- *CollectiveMeasurement*,
- *BinaryMeasurement*,
- *RatioMeasurement*,
- *NamedMeasurement*,
- *RescaledMeasurement*.

As explained in the previous subsection, the *DirectMeasurement* class defines the result of the *DirectMeasure* entity and *CountingMeasurement* is a subclass of *DirectMeasurement*. It represents the

CountingMeasure result. For the rest, the definition is similar, each one defines the result of the associated *Measure*, namely *CollectiveMeasure* class for *CollectiveMeasure*, *BinaryMeasurement* for *BinaryMeasure* etc.

The SMM specification defines the *SmmRelationship* entity to model the link between the derived measures and measures on which they are based. This allows to associate the derived measurements with its respective direct measurement.

3.2.4. SmmRelationship

SmmRelationship is an abstract class and a subclass of the *SmmElement*. *SmmRelationship* is divided into two types of association: *MeasureRelationship* and *MeasurementRelationship*. The first one is used to define the links between Measures, and the second - for the links between the Measurements. Each type of *SmmRelationship* is defined as a subclass of it.

MeasureRelationship is refined by:

- *EquivalentMeasureRelationship*,
- *RefinementMeasureRelationship*,
- *GradeMeasureRelationship*,
- *RescaledMeasureRelationship*,
- *BaseMeasureRelationship*.

The *EquivalentMeasureRelationship* class allows to define two measures, which are equivalent. The *RefinementMeasureRelatonship* class is used to declare that a Measure M1 is refined by a Measure M2. The GradeMeasureRelationship class represents the link, which associates the *DimensionalMeasure* graded by a *GradeMeasure*. Furthermore, the *RescaledMeasureRelationship* is used to link the *DimensionalDimensional* rescaled by a *RescaledMeasure*. The *BaseMeasureRelationship* is an abstract class to define the links of hierarchy between a derived Measure and its base Measures. This link can be refined in:

- *RankingMeasureRelationship*,
- *BaseNMeasureRelationship*,
- *Base1MeasureRelationship*,
- *Base2MeasureRelationship*.

The *RankingMeasureRelationship* class defines the association between a *DimansionalMeasure* and a *RankingMeasure* which ranks the first one.

The *BaseNMeasureRelationship* class represents the link between a *CollectiveMeasure* and a *DimensionalMeasure,* which the *CollectiveMeasure* is based on. The *Base1MeasureRelationship* and *Base2MeasureRelationship* classes define the relationships between a *BinaryMeasure* and its two base *DimensionalMeasure*. The *Base1MeasureRelationship for the first DimensionalMeasure* and *Base2MeasureRelationship for the second.*

The relationship hierarchy of *Measurement*s is the same as that of *Measure*s. The link classes are subclasses of *MeasurementRelationship*, and they are:

- *EquivalentMeasurementRelationship*,

- *RefinementMeasurementRelationship*,
- *GradeMeasurementRelationship*,
- *RescaledMeasurementRelationship*,
- *RankingMeasurementRelationship*,
- *BaseMeasurementRelationship*,
  - *BaseNMeasureRelationship*,
  - *Base1MeasureRelationship*,
  - *Base2MeasureRelationship*.

Thereby, via the UML language, SMM allows to define, an architecture of software measurement, as well as the context of the concrete measurement (observation).

3.2.5. Observations

In an SMM specification, the different aspects of a measurement environment are defined through four elements: The Observation class, which represents the contextual information as date of measurement, the tools being used and the measurer through three attributes. Tools and observer are string attributes. The date attribute is a *Timestamp* entity defined by SMM as a primitive type, which represents a point in time. The *ObservationScope* class allows to describe the subject of the related measurement through a string attribute referencing the model or a part of model measured. The *ObservedMeasure* class represents the link which associates Measures and observations. The *Argument* class defines the parameters or variable arguments that are passed to the measures of *Operation*s, which use parameters.

In the following subsection, we present a model of a measurement using SMM specification.

3.2.6. Examples

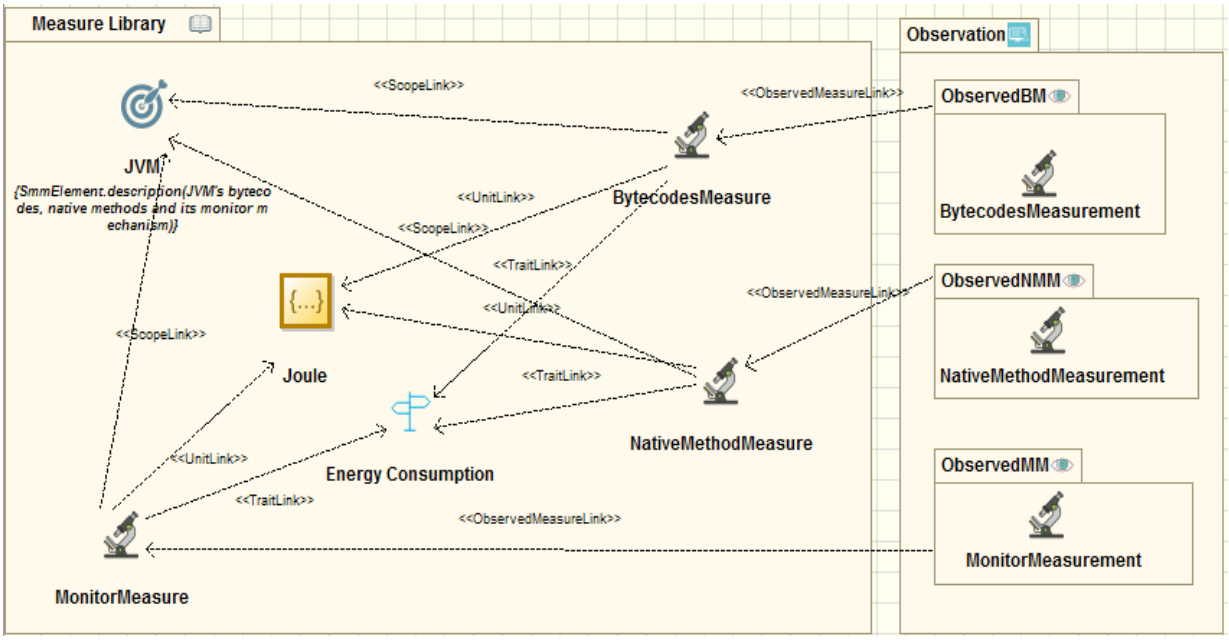

Figure 12 The Computational Energy Cost metric model in SMM

The model illustrated in Figure 12, is a model of the measurement architecture to compute the energy cost of Java-based software.

This model is composed of two SMM packages, namely 1) *Measure Library* that contains all aspects about measures and 2) *Observation* that contains the measure results. For each measure, a measurement is modelled by a proper UML class in the *Observation* package [10]. Each of them is linked to its associated measure by an association UML link, called *ObservedMeasure*. Concerning the Measure Library package, the three measures are modelled by an UML class; these measures are associated to their unit of measure class called Joule, by an association link called UnitLink. They are associated to a Scope element named JVM, that defines the domain of measurement. The three measures compute different values of the same domain and return the same unit of measure. That is the reason why they are associated to the same *Scope* with a *ScopeLink* and Unit element. The same explanations are applied to the characteristic element called EnergyConsumption. It computes the energy consumption of elements of the same domain. The measure elements are matched to their corresponding characteristics by the TraitLink.

3.3. XMI: Metadata Interchange

SMM defines an associated XMI schema for data interchange. In fact, XMI is an XML interchange format. In this section we briefly describe the foundations of XMI, for detailed information, please refer to its specification found in [8].

Meta Object Facility (MOF) is the foundation technology for describing meta-models, such as SMM. It covers a wide range of domains, and is based on a constrained subset of UML. XMI is a widely used XML interchange format. It defines the following aspects involved in describing objects in XML:

- the representation of objects in terms of XML elements and attributes,
- the standard mechanisms to link objects within the same file or across files,
- the validation of XMI documents using XML Schemas,
- object identity, which allows objects to be referenced from other objects in terms of IDs and UUIDs.

XMI describes solutions to the above issues by specifying formal grammar rules to create XML documents and Schemas that share objects consistently.

3.3.1. XML Schemas

Although XML schemas are optional in general terms, it is incumbent on standards bodies that define MOF2 instances to produce corresponding XMI 2 Schemas for them.

All XML elements defined by this International Standard are in the namespace *http://www.omg.org/spec/XMI/versionnamespace,* where version-namespace is the version of the XMI specification being used. The XML namespace mechanism can be used to avoid name conflicts between the XMI elements and the XML elements from the particular MOF models.

Every model classes are represented in the schema by an XML element whose name is the class name, as well as a complexType whose name is the class name. The declaration of the type lists the properties of the class. By default, the content models of XML elements corresponding to model classes do not impose an order on the properties.

By default, XMI allows you to serialize features using either XML elements or XML attributes; however, XMI allows to specify how to serialize them if you wish. Composite and multivalued properties are always serialized using XML elements.

One important remark is that every XMI schema contains a mechanism for extending a model class. Therefore, zero or more extension elements are included in the content model of each class.

3.3.2. XMI Example

Below we show the code of a brief XMI file. It can be seen that the structure of such file is rather simple for any user familiarized with XML and MOF.

```
<xmi:XMI xmlns:uml="http://www.omg.org/spec/UML/20110701"
xmlns:xmi="http://www.omg.org/spec/XMI/20110701">
        <uml:Class name="C1" xmi:type=”uml:Class” xmi:id=”_1”>
          <ownedAttribute xmi:type="uml:Property" xmi:id=”_2” name="a1" visibility="private"/>
        </uml:Class>
</xmi:XMI>
```

XMI is used as the interchange format for SMM. In this section, we have briefly summarized some relevant aspects of the XMI standard. It can be seen the adoption of XMI to interchange the measurements can be straightforward. In the next subsections we present an alternative and XML-based interchange format and the adoption of SMM into the consortium's tools.

3.4. Measure Deployment Format

The SMM specification provides a standardized way to exchange the definition of measures. Namely, SMM defines XMI (Xml Model Interchange) to realize this operation. As indicated by its name, this format has been designed to exchange models between modelling tools.

However, it seems that this format is not adapted to deploy measures in the analysis tools. In context of the project, we plan to use a format, which is better suited to our requirements: the deployment of measures definition in the Measure Platform, and the execution (collection and calculation) of measures. Each measure definition is composed of two elements collected in a Zip File: an XML file which stores meta-data of the Measure and another file that contains the measure implementation, as shown in Figure 13.

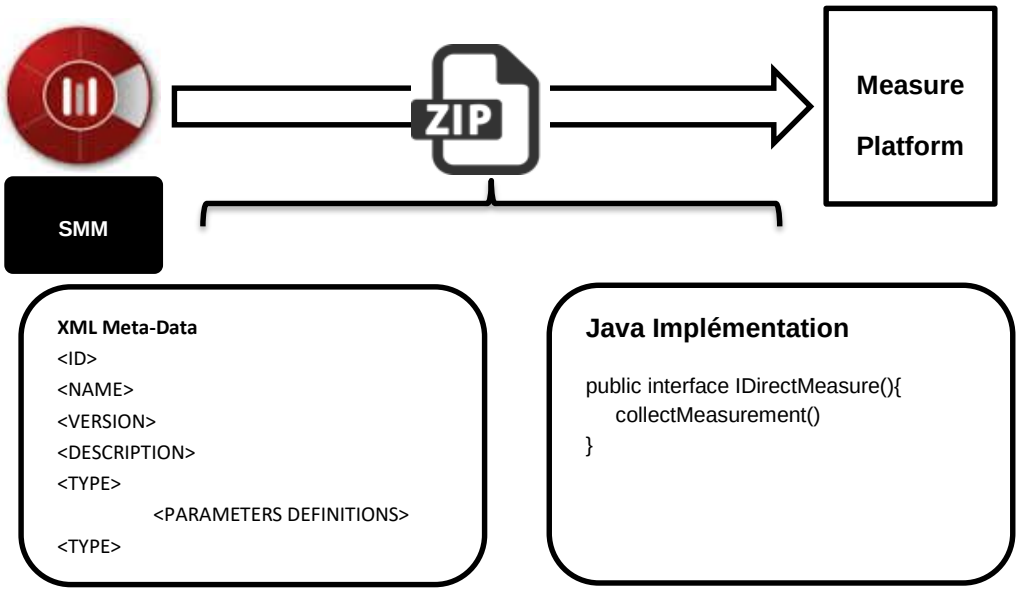

Figure 13 SMM Deployment Format

In SMM, direct and calculated measure definitions are associated with operations that represent the implementation of the measure. These operations can be expressed in natural language or mainly containing executable code. In order to be able to collect direct measures and to calculate the compound measures, a

common executable language is to be chosen. This deployment format will be defined later in the MEASURE deliverable D3.1 - First release of the measuring, analysis and visualization tools for internal evaluation.

4. Integration with the consortium's tools

4.1. Modelio SMM Module

As presented before, the Structured Metrics Meta-model (SMM) is a publicly available specification from the Object Management Group (OMG). SMM specifies a meta-model for defining, representing and exchanging both measures and measurement information related to any structured information model, such as the OMG Meta Object Facility (MOF™) standard, defining an XMI [8] interchange format between metrics extraction tools.

Most software system properties can be quantified with the application measurement processes. OMG's Structured Metrics Meta-Model (SMM) supports the meta-model agnostic definition of those measurement processes. The Modelio SMM Module is the first building block of the MEASURE project tool chain. The Modelio modeling tool, enabled with the SMM Module is based on Modelio's open source distribution to allow the specification of metrics.

As mentioned before, the goal of the MEASURE (Measuring Software Engineering) project is to provide a toolset for future projects to properly measure their quality and their impact, and in particular to develop methods and tools for analyzing the big data produced by the continuous measurement. In this context, we present the current integration into the consortium's tools.

Modelio supports the Structured Metrics-model. "This specification defines a meta-model for representing measurement information related to any model structured information with an initial focus on software, its operation, and its design. Referred to as the Structured Metrics Meta-model (SMM), this specification is an extensible meta-model for exchanging both measures and measurement information concerning artifacts contained or expressed by structured models, such as MOF." [7] This is a work in progress towards supporting SMM 1.1.1. SMM Designer requires Modelio 3.4.1 open-source. It can be obtained from the official website, namely https://www.modelio.org/downloads/download-modelio.html.

4.2. Measure modeling with the Modelio SMM Module

- **Project Organization**

An SMM modeling project is organized on top of SMM models (shown in "[1]" and "[2]" of Figure 14).

The SMM model is composed of Libraries, as shown in "[3]" of Figure 14. Each library contains some categories, as shown in "[4]" of Figure 14. A library allows organizing measures related to the same domain hierarchy. In this example, libraries are organized around phases of the software development process.

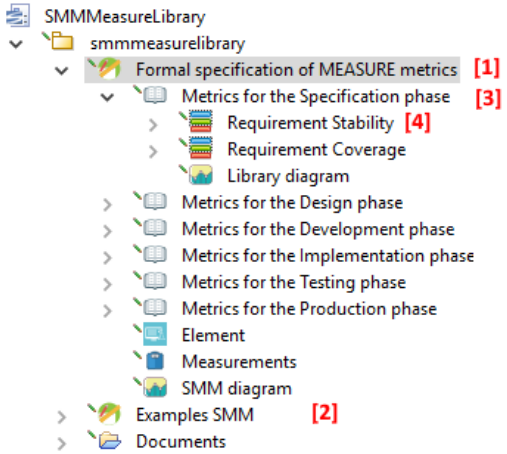

Figure 14 : SMM Project in Modelio

- **Measures' Categories**

Categories are used to group in the same context related measures. As example, if a **Collective Measure** is defined, which aggregates data of a direct measure, it is convenient to define these two measures in the same category.

The **Library diagram,** shown in "[1]" of Figure 15 provides graphical tools to create a **new Category, we** shown in **"**[2]" of Figure 15.

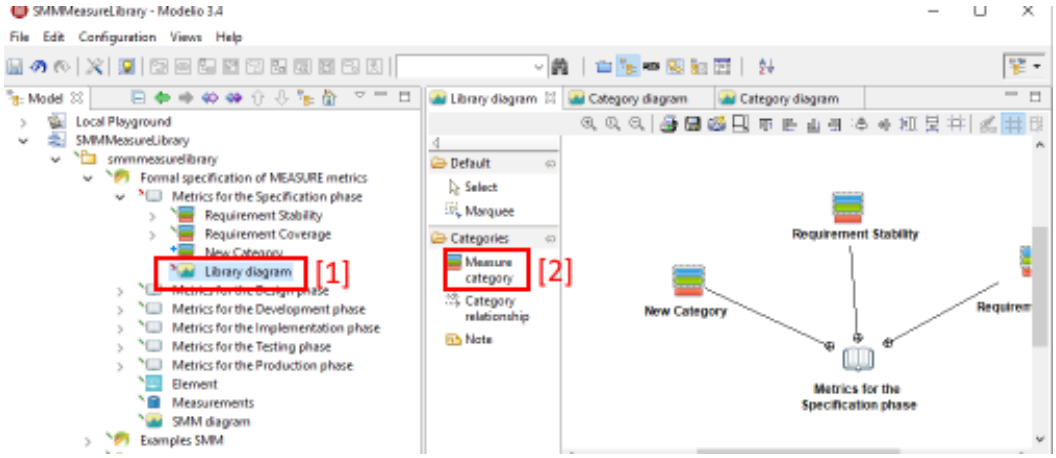

Figure 15 : Library and Category modelling

- **Measure Modelling**

A Measure can be modelled using the **Category Diagram,** shown in "[1]" of Figure 16, which is associated with each category. This diagram provides several tools allowing to create the desired measure, shown in "[2]" of Figure 16.

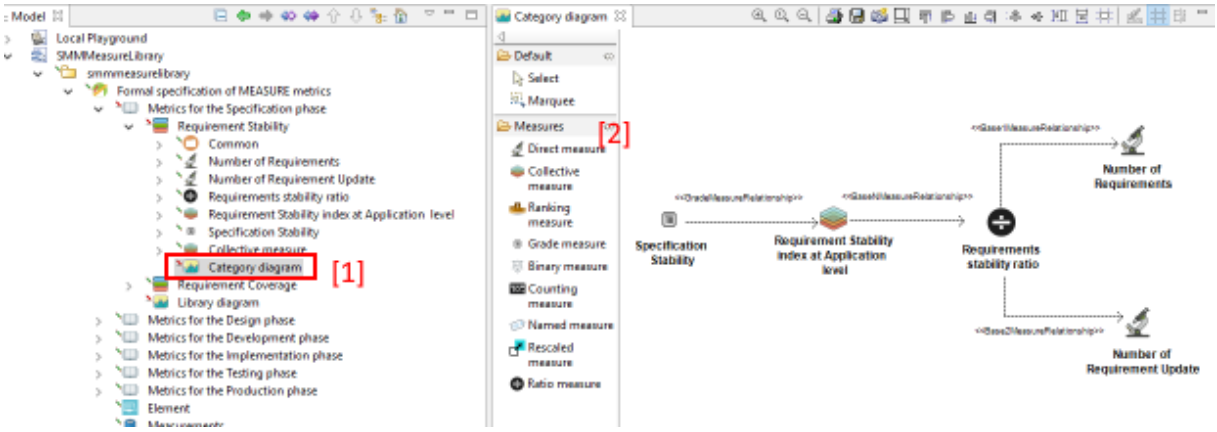

Figure 16 : Measure modelling

Once created, it is required to model the relations between these measures and other measures. Direct Measures and Named measures are the only measures that do not depend on other measures. The number and kind of relation required depend on the type of the measure.

- **Specifying the Measure's Scope, Unit and Dependencies**

To specify the Measure scope and Unit, the Measure diagram, shown in "[1]" of Figure 17, should be open. A new Scope (shown in "[2]" of Figure 17), a new Unit (shown in "[3]" of Figure 17) and a link (shown in "[4]" of Figure 17) should be created with the measure. A measure can also be linked with an existing Scope or Unit by selecting the Scope in a common directory of the explorer (shown in "[5]" of Figure 17) and dropping it on the diagram.

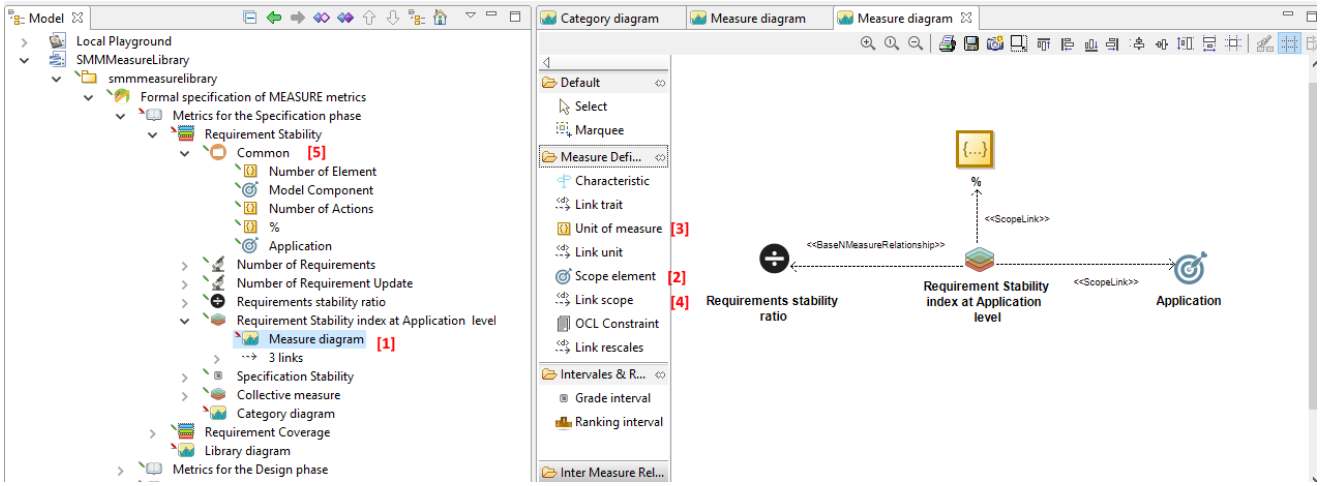

Figure 17 : Measure, scope and unite modelling

- **Specifying the Measure's description**

In order to specify the description of each measure, scope and Units. The element to describe (shown in "[1]" of Figure 18) must be selected and the Note View (shown in "[2]" of Figure 18) must be used to include the content in the associated **Description** field (shown in "[3]" of Figure 18).

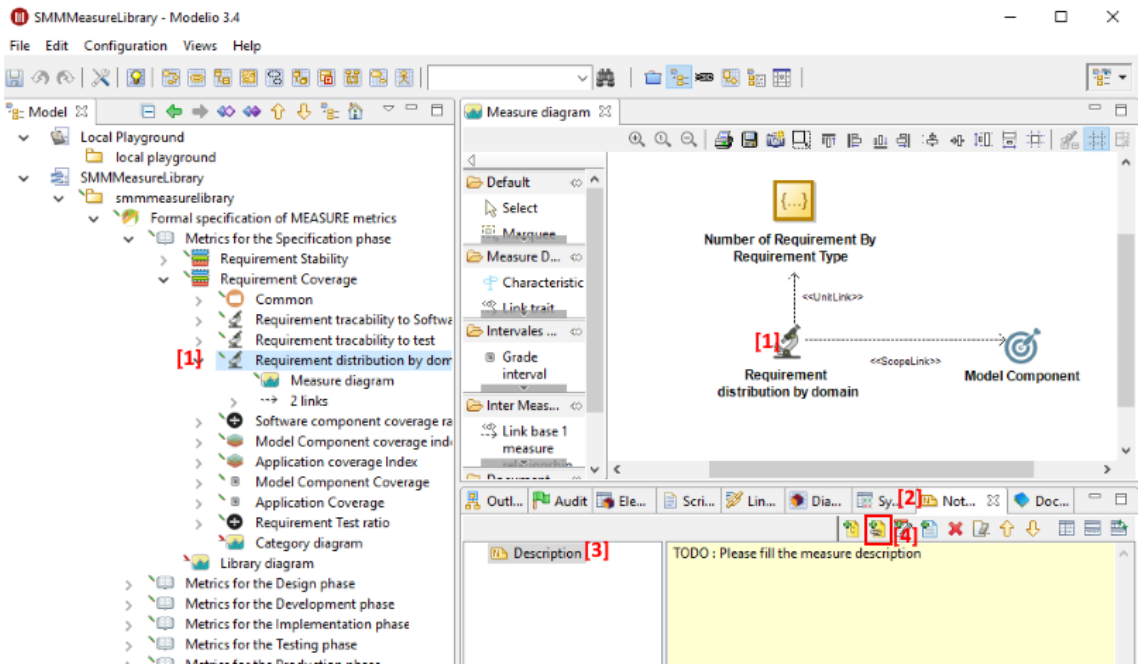

Figure 18 : Measure description

4.3. Examples for measurements and metrics specification using Modelio

Example 1 – Square example

This example is presented on page 19 of the SMM formal specification [7]. It represents the area of a square as a function of the length of its sides. For this purpose, it defines two meta-model concepts, namely Side and Square. It then defines a direct measurement measure Side Length, as a measure of the Size of a Side. Then, the binary measure Area of Square, measuring the Area of a Square as a product of the measured length of the sides of the square (Fig. 19).

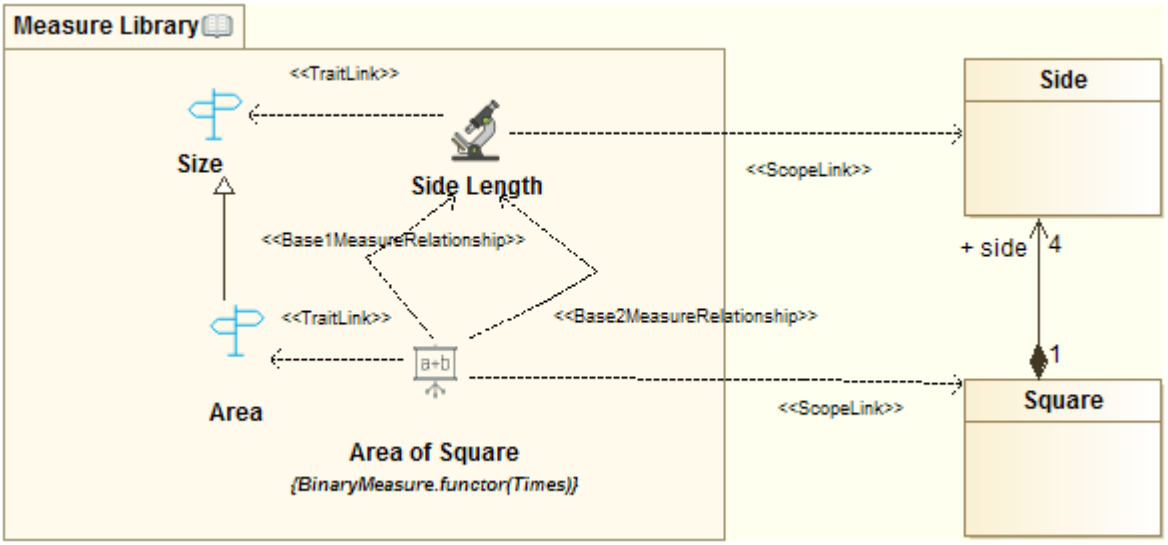

Figure 19 : Illustrating the Square example

Followed by a very basic and simple example, it is easy to see that the description of complex measurements and metrics can also be straightforward. In the next example, we illustrate a different type of measurement.

Example 2 – Measure rescaling example

This example is based on the Conversion of Information Size to Maintainability example on page 76 of the SMM formal specification [7].

As explained in the specification, the Maintainability for a collection of code modules (e.g., programs) is calculated by the formula:

171 - 5.2(ln(aveV)) - 0.23(aveV(g')) - 16.2(ln(aveLOC)) + 50(sin (sqrt(2.4(perCM))))

where aveV is the average Halstead volume, aveV(g') is the average Cyclomatic complexity, aveLOC is the count of lines of code, and perCM is the percentage of comments in the modules.

The Information Size Maintenability is computed by rescaling the average Halstead volume (represented by the AverageDiscriminations ratio measure) rescaled by the formula 50 - 5.2(ln(aveV)), where aveV is the average Halstead volume. In the image below, we depict the SMM metric description (Figure 20).

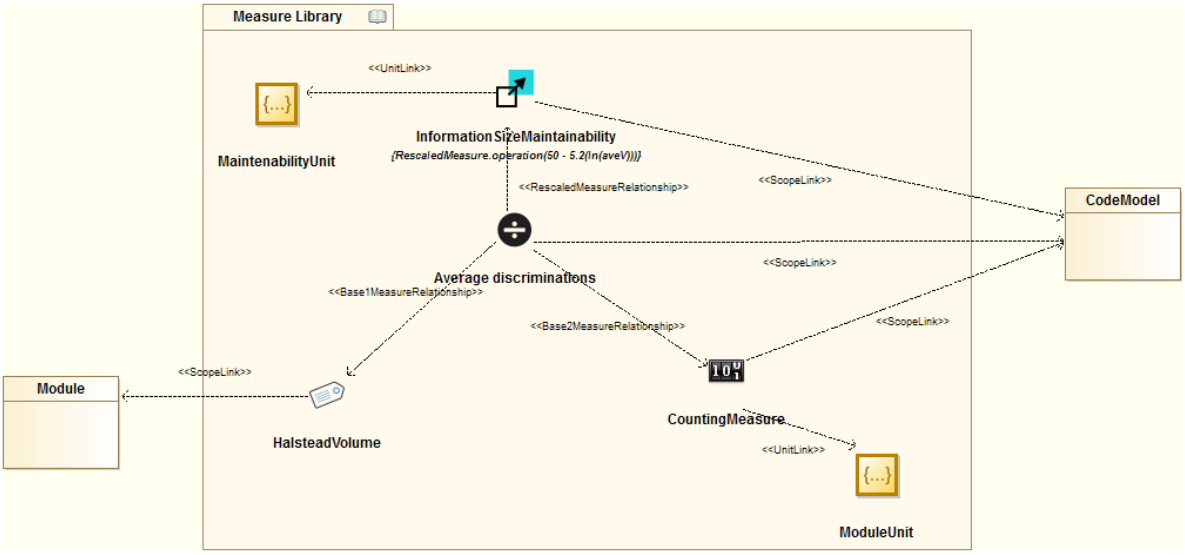

Figure 20 : Illustrating the Measure rescaling example

Example 3 – Energy Efficiency Index example

This example defines the Energy Efficiency Index of refrigerating devices according to its power consumption (Figure 21). It is based on a statistical distribution of codes on energy efficiency classes from A to F based on their power consumption.

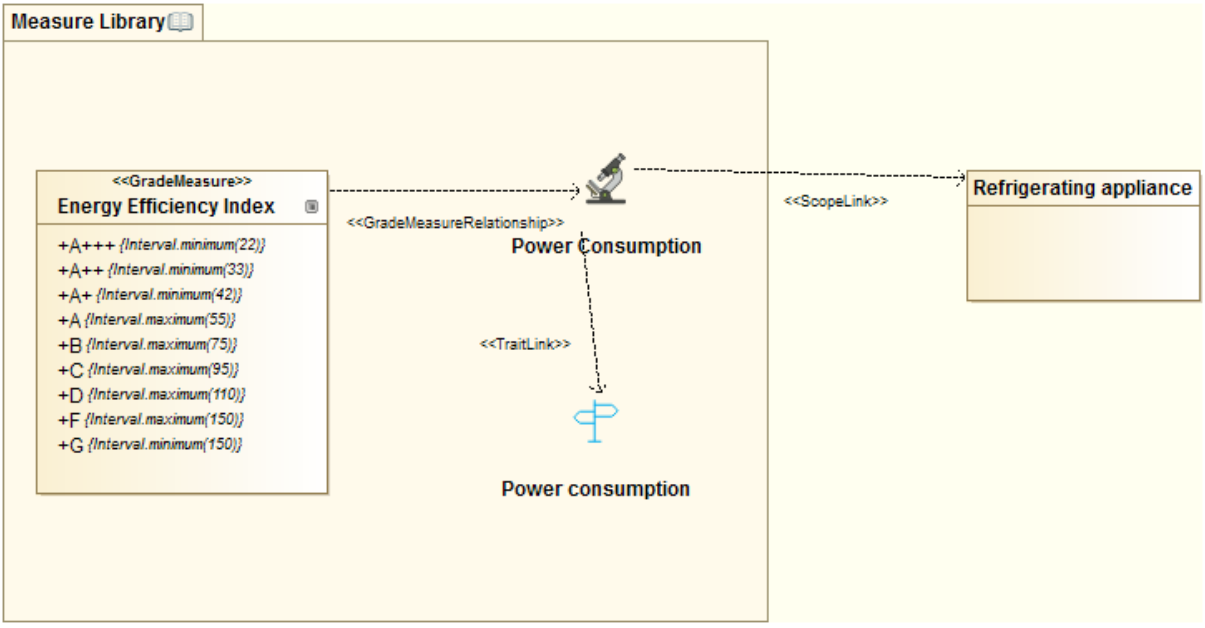

Figure 21: Illustrating the Energy Efficiency Index example

The image below depicts the European energy efficiency index table.

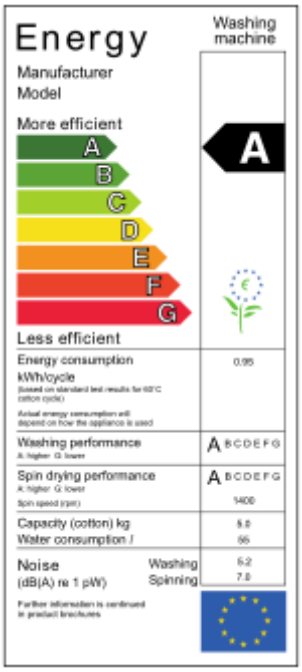

Figure 22 : European energy efficiency index table

One can see that the SMM provides the flexibility to describe different types of measurements and metrics for determining the quality of a given software. The previously presented examples are meant to emphasize the versatility of the SMM language.

4.4. Interoperability between Modelio and EMIT by utilizing SMM

In [9], the authors showcase how two partner tools have communicated the measures and measurement data exchange using SMM. These tools are Softeam's Modelio SMM modeler and ICAM's software energy consumption monitoring platform, called EMIT.

EMIT data model focuses on measurements as shown by the Figure 23. In fact, the central entity in the class diagram called *Measurement* references four other entities:

1. The *MeasurementSet* is the entity that corresponds to the set of measurements collected during one measurement acquisition. The *MeasurementSet* is specified by a measurement timestamp, via its *measured* attribute and by a measurement *duration* attribute.
2. The *Measure* entity i.e., the type of measurement that is defined by its *name* and its *unit* of measure.
3. The *Instrument* entity that corresponds to the observer which provides the measurement. The observer is defined by its Unique Resource Identifier (URI).
4. The *Environment* entity that corresponds to the *observee*, such as the operating system and the computer architecture. Hence its attribute names *sys*, *arch* and *version*.

The Measurement entity is defined by an attribute called feature that corresponds to the name of the measurement among all the measurements provided by its observer during a single acquisition. This entity is also defined by an attribute path which corresponds to the file that contains this measurement data. The latter is formatted as a list of key-value pairs respectively made of long and double values.

EMIT data model is composed of two more entities: Measurand and Observation. The first one corresponds to the monitored software process specified by its command line. The last entity makes possible to register some analysis results (values) provided by third-party tools (providers) over a given measurement. For instance, this could be used for storing some statistics, such as, the average. An observation is related to a given measure that can be different from its corresponding measurement.

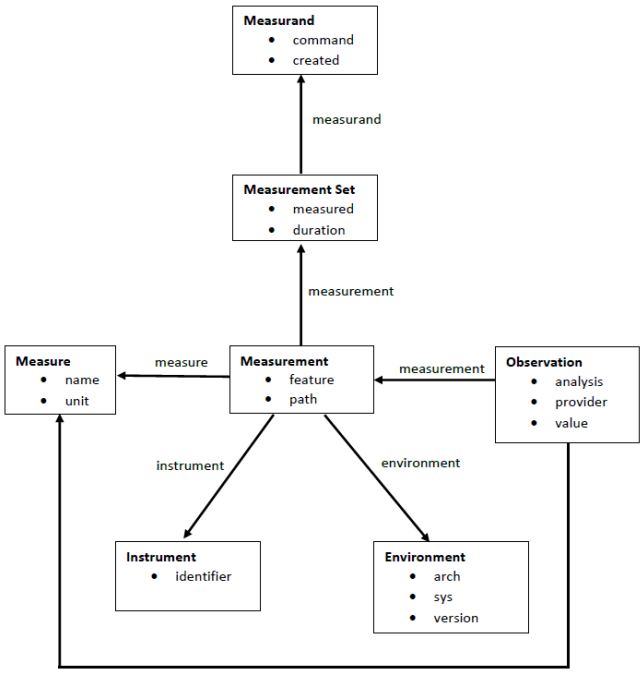

Figure 23 : EMIT Meta-Model

EMIT meta-model does not fully comply to SMM as it has been designed to suit the measurement process, which is devoted to one measurand and which produces a set of measurements. However, this data model can be embedded into an SMM model as illustrated by the Figure 24. To embed the EMIT data model into an SMM model, every measurements of a given measurement set are sliced according to the different instruments that provide these measurements. Each Measurement slice is then mapped into an SMM Observation where the when Observed attribute corresponds to the measured one of the measurement set, and where the tool attribute corresponds to the identifier of the corresponding instrument. Therefore, each measurement of such measurement slices is mapped into an SMM ObservedMeasure.

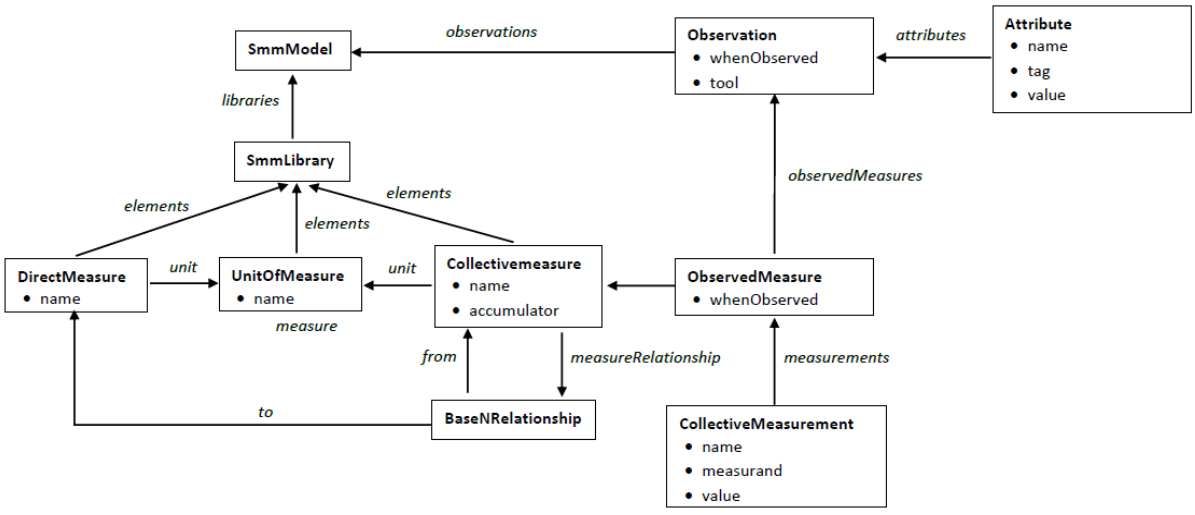

Figure 24 : EMIT SMM Embedding

Every measure related to these measurements is inserted into a unique SMM library as an SMM direct measure. Moreover, every measure related to an observation are inserted into this SMM library as SMM collective measures whose accumulator corresponds to the observation analysis. The accumulator can

range over several statistical functions such as average, minimum, maximum, standard deviation, etc. Every collective measure owns a measure relationship (defined by a BaseNRelationship) between itself and the direct measure of the measurement mapped measure. These collective measures have to be synchronized between the EMIT data model and that of SMM: it is required in order to ensure the mapping compatibility between them. Finally, each EMIT observation is mapped into a SMM CollectiveMeasurement which share its value attribute with the attribute of the EMIT observation. The measurand attribute corresponds to the command attribute of the measurand related to this observation throughout the minimum or the maximum of the measurements data values.

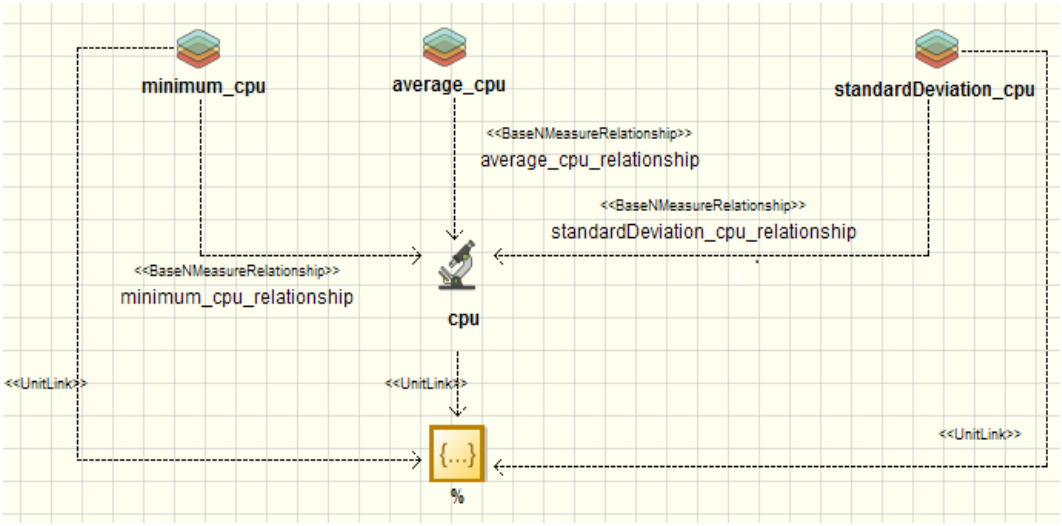

Figure 25 : SMM CPU Measure Group

The export of EMIT measures and measurements into a XMI file compliant to the SMM meta-model has been achieved thanks to a Java library developed and released by Softeam. These measures and measurements have successfully been imported into Modelio proving the interoperability between the two tools thanks to SMM. The exported file always contains fixed data with 6 groups of 5 measures (1 direct measure and 4 collective ones) and variable ones provided by the measurements of each measure of these groups. However, its file size increases with that of measurements. In fact, an export file size of 200 measurement corresponds to several hundred MB. This leads us to adopt as guideline that fact that SMM will be used in the consortium for designing and exchanging measures whereas measurements will be shared in a different manner.

5. Conclusions

The main goal of the ITEA 3 Measuring Software Engineering (MEASURE) Project is to measure the quality and efficiency of software. This project aims to provide a toolset for future projects to properly measure their software quality and efficiency. In order to achieve the goals of the MEASURE project, a common specification language to define and to interchange the metrics and measurements is required. This document presented the findings of the project's consortium, in regards to which formal (or semi-formal) language specification is the best suited for the previously mentioned task. With a well-defined specification language, the goals of the MEASURE project can be achieved without ambiguity with respect to the metric specification and also while interchanging those metrics and measures.

After a systematic literature review, the consortium found that the Structured Metrics Meta-model (SMM) language was the better suited language for the task. Further, the associated XML Metadata Interchange (XMI) model of it facilitates the interchange of information. However, the use of XMI revealed particular needs for the interchange needed by the consortium. For that reason, we envision a different and simplified XML-based language for the interchange format.

Defining the language to model and interchange metrics and measurements represents a major stepping stone towards achieving the project's goals. After the definition of the language to be used by the MEASURE project, further work is to be performed using the language decided in this report. Namely, the following work packages of the MEASURE project depend upon the chosen language presented in this deliverable: 1) the work package 3, which aims at providing tools for measurement, such tools are to be developed using the SMM model and they should export / interchange data using the XMI language; 2) the work package 5, which aims at evaluating the tools using industrial case studies will in fact produce data in the formats defined in this deliverable; and finally 3) the work package 4, which aims at analyzing the data produced by the work package 5, as mentioned before its output data follows the language and format proposed in this deliverable.

An important remark to conclude is that the work of MEASURE is being performed with the predicted timing and this allows us to envision that the problems and tasks raised in the project proposal will be solved on time, and furthermore, successfully. We will specify the MEASURE developed metrics in SMM in Deliverable D2.2 "Formal specification of MEASURE metrics ".

References

[1] Beatriz Mora, Mario Piattini, Francisco Ruiz, Felix Garcia: “Smml: Software measurement modeling language”, in the proceedings of the *8th Workshop on Domain-Specific Modeling (DSM’2008)*, 2008.

[2] Andreas Vogelsang, Ansgar Fehnker, Ralf Huuck, Wolfgang Reif: “Software Metrics in Static Program Analysis”, in the proceedings of the *12th International Conference on Formal Engineering Methods, ICFEM*, pp. 485-500, Shanghai, China, 2010.

[3] Esther Guerra, Juan de Lara, Paloma Díaz: “Visual specification of measurements and redesigns for domain specific visual languages”, *Journal of Visual Languages & Computing*, V.19, I. 3, pp. 399-425, 2008.

[4] Maria de los Angeles Martin, Luis Olsina: “Towards an ontology for software metrics and indicators as the foundation for a cataloging web system”, in the proceedings of the *First Latin American Web Congress*, pp. 103-113, 2003.

[5] International Organization for Standardization, International Electrotechnical Commission: “ISO/IEC 15939: Systems and software engineering -- Measurement process”, *ISO/IEC Standard*, *http://www.iso.org/iso/catalogue_detail.htm?csnumber=44344*, 2007.

[6] Hans Vangheluwe, Juan De Lara, Pieter J. Mosterman: “An introduction to multi-paradigm modelling and simulation”, in the proceedings of the AI, Simulation and Planning in High Autonomy Systems conference, AIS, pp. 9-20, Lisbon, Portugal, 2002.

[7] Object Management Group (OMG): “SMM: Structured Metrics Meta-Model“, Version 1.1, *OMG Standard*, URL: http://www.omg.org/spec/SMM/, 2015.

[8] Object Management Group (OMG): “XML Metadata Interchange (XMI)”, Version 2.5, OMG Standard, URL: *http://www.omg.org/spec/XMI/2.5.1*, 2015.

[9] Alessandra Bagnato, Marcos Aurélio Almeida Da Silva, Antonin Abherve, Jérôme Rocheteau, Claire-Lise Pihery, Pierre Mabit: “Measuring Green Software Engineering In the MEASURE ITEA 3 Project”, in the proceedings of the 3rd International Workshop on Measurement and Metrics for Green and Sustainable Software Systems (MeGSuS), 2016.

[10] Sarah Dahab, Stephane Maag, Alessandra Bagnato and Marcos Aurélio Almeida Da Silva: “A Learning based approach for Green Software Measurements”, in the proceedings of the 3rd International Workshop on Measurement and Metrics for Green and Sustainable Software Systems (MeGSuS), 2016.