USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE.
IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 140 Kendrick Street, Needham, MA 02494, U.S.A.

TRADEMARKS

MDA®, Model Driven Architecture®, UML®, UML Cube logo®, OMG Logo®, CORBA® and XMI® are registered trademarks of the Object Management Group, Inc., and Object Management Group™, OMG™ , Unified Modeling Language™, Model Driven Architecture Logo™, Model Driven Architecture Diagram™, CORBA logos™, XMI Logo™, CWM™, CWM Logo™, IIOP™ , IMM™ , MOF™ , OMG Interface Definition Language (IDL)™ , and OMG Systems Modeling Language (OMG SysML)™ are trademarks of the Object Management Group. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this

specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

OMG’s Issue Reporting Procedure

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page *http://www.omg.org*, under Documents, Report a Bug/Issue (http://www.omg.org/technology/agreement.htm).

Table of Contents

Preface

About the Object Management Group

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at *http://www.omg.org/*.

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. A catalog of all OMG Specifications is available from the OMG website at:

http://www.omg.org/technology/documents/spec_catalog.htm

Specifications within the Catalog are organized by the following categories:

Business Modeling Specifications

- Business Rules and Process Management Specifications

Language Mappings

- IDL/Language Mapping Specifications
- Other Language Mapping Specifications

Middleware Specifications

- CORBA/IIOP
- CORBA Component Model
- Data Distribution
- Specialized CORBA

Modeling and Metadata Specifications

- UML
- MOF
- XMI
- CWM
- Profile specifications.

Modernization Specifications

- KDM

Platform Independent Model (PIM), Platform Specific Model (PSM), and Interface Specifications

- CORBAservices
- CORBAfacilities
- OMG Domain specifications
- OMG Embedded Intelligence specifications
- OMG Security specifications

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) All specifications are available in PostScript and PDF format and may be obtained from the Specifications Catalog cited above. Certain OMG specifications are also available as ISO standards. Please consult *http://www.iso.org*

OMG Contact Information

OMG Headquarters
140 Kendrick Street
Building A, Suite 300
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
http://www.omg.org/
Email: *pubs@omg.org*

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Times/Times New Roman - 10 pt.: Standard body text

Helvetica/Arial - 10 pt. Bold: OMG Interface Definition Language (OMG IDL) and syntax elements.

`Courier - 10 pt. Bold:` Programming language elements.

Helvetica/Arial - 10 pt: Exceptions

Note – Terms that appear in *italics* are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

Issues

The reader is encouraged to report any technical or editing issues/problems with this specification to *http://www.omg.org/technology/agreement.htm*.

1 Scope

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.

The SMM include elements representing the concepts needed to express a wide range of diversified measures. The specification does include a minimal library of software measures, but it is not asserting that the listed measures constitute standards themselves; these are supplied simply as non-normative examples.

The SMM is a specification for the definition of measures and the representation of their measurement results. The measure definitions make up the library of measures and that serves to establish the specification upon which all of the measurements will be based.

The SMM is part of the Architecture Driven Modernization (ADM) roadmap and fulfills the metric needs of the ADM roadmap scenarios as well as other information technology scenarios.

The SMM specifies the representation of measures without detailing the representation of the entities measured. SMM anticipates that those entities are represented in other OMG meta-models. Measures of software artifacts or their features that are defined within the SMM, the Knowledge Discovery Metamodel (KDM), the Abstract Syntax Tree Metamodel (ASTM), another ADM roadmap meta-model or another OMG meta-model may arise as:

- Counts. (Lines of code measures exemplify the mechanism.)
- Direct applications of named measurements. (One such named measure is Cyclomatic Complexity.)
- Simple algebraic change of scales of already defined numeric measures (e.g., the translation to ‘choice points’ from Cyclomatic complexity).
- Simple algebraic aggregations of numeric artifact features, including other measures, over sets of software artifacts. (Determining the complexity of an application by summing the complexities of the application’s elements demonstrates this process.)
- Simple range-based grading or classification of already defined numeric measures. (Cyclomatic reliable/unreliable quadrants are one such grading.)
- Qualitative evaluations where the range of evaluations can be mapped to a linear order.

Useful metrics must go beyond static (or dynamic) code analysis and technical performance to include factors related to information utility and acceptance of the system by the organization(s) participating in an enterprise. To be objective and repeatable, such metrics need to be based on technical characteristics of the system. Given a meta-model representation of such characteristics, the SMM will facilitate the exchange of such measures.

Given the evolutionary nature of system development and the predicate value of metrics with respect to “downstream” problems, metrics are gathered into trends or viewed from historical perspective. As shown in Clause 17 “Historic and Trend Data,” SMM addresses the issues of trend and history to model for system development as long as the historical links of the measured entities are provided.

Consistent with other models defined by OMG, the SMM is defined using the MOF meta-modeling language. As such, it has a standard textual representation presented by XMI. Consequently, the exchange of metrics defined by SMM is in the XMI. These models are compatible with MOF repositories for storage and retrieval by various tools.

2 Conformance

The principle goal of SMM is the exchange of measurements about software. To be SMM compliant, a tool must fully support SMM as one compliance point. An implementation can provide:

- The capability to generate XMI documents based on the SMM XMI schema capturing measurements from the existing model of the tool.
- The capability to import measurements via representations based on the SMM XMI schema and to map the measurements into the existing model of the tool.

3 Normative References

The following normative documents contain provisions, which, through reference in this text, constitute provisions of this specification. For dated references, subsequent amendments to, or revisions of any of these publications do not apply.

- UML 2. Infrastructure Specification
- MOF 2.0 Specification
- OCL 2.2 Specification

4 Terms and Definitions

We assume the following definitions:

Measure: A method assigning comparable numerical or symbolic values to entities in order to characterize an attribute of the entities.

Measurement: A numerical or symbolic value assigned to an entity by a measure.

Measurand: An entity quantified by a measurement.

Unit of Measure: A quantity in terms of which the magnitudes of other quantities within the same total order can be stated.

Dimension: A totally ordered range of values which can be stated as orders of magnitude relative to one another or to an archetypal member.

Measurement Accuracy: The measurement by which another measurement may be wrong.

Measurement Scope: The domain (set of entities) to which a given measure may be applied.

Measurement Range: The range (set of comparable values) assignable by a given measure.

5 Symbols

There are no symbols/abbreviations.

6 Additional Information

6.1 Changes to Adopted OMG Specifications

There are no changes to other OMG specifications.

6.2 How to Read this Specification

The rest of this document contains the technical content of this specification.

Although the clauses are organized in a logical manner and can be read sequentially, this reference specification is intended to be read in a non-sequential manner. Consequently, extensive cross-references are provided to facilitate browsing and search.

6.3 Acknowledgments

The following companies submitted and/or supported parts of this specification:

- EDS
- Benchmark Consulting
- KDM Analytics
- Software Revolution
- Tactical Strategy Group
- NIST
- eCube Systems

The following persons were members of the core team that designed and wrote this specification: Kevin Barnes, Djenana Campara, Larry Hines, Nikolai Mansurov, Alain Picard, John Salasin, Michael Smith, and William Ulrich.

7 SMM Introduction

7.1 Overview

Measurements provide data for disciplined engineering in that engineers and their managers rely on these comparable evaluations in assessing the static and operational qualities of systems.

For example, software measurement methods produce comparable evaluations of software or application artifacts. Counts such as number of screens, lines of code, and number of methods quantify the size of artifacts along a single dimension. These evaluations readily distinguish larger artifacts from smaller ones; likewise complexity metrics such as Halstead and Cyclomatic separate the simpler artifacts from the more complex. Comparable evaluations form mappings of artifacts of a given type into a single dimension.

Such is also the case for architecture measures (coupling and cohesion); functional measures (functions defined in system, persistent data as a percentage of all data, functions in current system that map to functions in target architecture); quality measures (failures per unit time, meantime to failure, meantime between repair); performance measures (average batch window clock time, average online response time); software assurance measures; and cost measures.

Predictive metrics provide a basis for continual system-level in contrast to fixed milestone-based assessments. These metrics may indicate at some future development stage the probability that the system will or will not meet its requirements.

This specification defines a meta-model for representing measurement related to structured model assets and their operational environments referred to as the Structured Metrics Meta-model (SMM).

The SMM promotes a common interchange format that will allow interoperability between existing tools, commercial services providers, and their respective models. This common interchange format applies equally well to development and maintenance tools, services, and models. SMM complements a common repository structure and so facilitates the exchange of data currently contained within individual tool models that represent modeled assets. Given that the repository's meta-model represents the physical and logical modeled assets at various levels of abstraction as entities and relations, SMM represents the measurements of these assets.

7.1.1 Goals

The main goals for the SMM are to provide an extendable meta-model establishing a standard for the interchange of measure libraries and structured model related measurements over the entities modeled by OMG meta-models. By structured model, we mean measurements derived from the structure model artifacts (that is those artifacts that are modeled according to the MOF meta-model approach). SMM contains meta-model classes and associations to model measurements, measures, and observations. We present and explain diagrams depicting measures, then measurements and finally observations. All initial depictions are in terms of software measurement, but the specification is not limited to representing those modeled elements.

SMM supports the meta-models of the OMG by providing for extendable measurements of entities.

7.2 General Usage Notes (Non normative)

The SMM is designed to allow for both the exchange of measurement data, as well as the measures upon which those measurements were established.

Even though there exists a mechanism whereby someone can essentially exchange measurement data without providing any insight into the measures (accomplished with NamedMeasure), this approach is surely not the major trust of this specification.

The value of SMM comes from the ability of various groups and vendors to be able to define library of measures against different structured models. These libraries can then be exchanged, validated, and then used to produce measurements of specific model instances.

To exchange measure libraries, the definition of those libraries has to be precise and detailed enough to enable for their unambiguous use in carrying out measurements on models.

While SMM compliance doesn't mandate how to gather measurements from defined measures, it is clear that without any common understanding measures lose most of their value. The following clause(s) should help to facilitate the understanding of the specification and also provide some background that will help in applying the specification more uniformly.

7.3 Steps in using SMM (Non-normative)

In general, using the SMM starts with the definition of measures and their libraries. In the case of measures being applied to standard models, these measure libraries could also be pre-defined and made available to various practitioners.

How we proceed next very much depends on the type of environment that the tools are operating in. Tools that are simply using the SMM as a means of interchanging measurement data will take some measurements, along with the details about the Observation that resulted in those measurements, populate the model, and deliver the results.

Other tools that are designed more natively with the SMM in mind will take a bit of a different multi-step process.

Once we have our measures in place, the next step is to determine what we will be measuring. This is what we call defining the observation. Among other things this will include specifying the model(s) to use (ObservationScope) for taking the measures, as well as determining which measures we are interested in performing (requestedMeasures). It can also include determining and passing in any arguments that might be needed by our requestMeasure(s) and their descendants.

Next step is to apply the requested measure(s) on the model(s) in scope and to figure out the measurements. Once that is done, the resulting model is ready to be used or exchanged.

The step of applying the measure, the "measurement step" is clearly one that can take on many forms depending on the implementer. But regardless of how the process is carried out, the measure library should provide sufficient information for a tool vendor to implement "executable measuring." This "executable measuring" should enable another tool vendor, presented with the same measure libraries, observation information and instance models, to be able to apply those measures in an unambiguous fashion and to come up with the same measurements (subject to uncertainty errors).

7.4 Interpreting Measures (Informative)

Measures essentially fall into 2 "categories," there are direct measures, which are measures that are taken directly against a measurand, and all others, which we shall call derived measures, as their result is based on some other measure(s), direct or derived. Ultimately, every measure comes from a direct measure (otherwise it might end up triggering a defaultQuery for its value).

In order to support many types of measure refinement, where you have a drill-down of measures representing the collective aggregation of values in a top-down fashion, and also in order to make sure that derived measures are correctly linked to their base measure(s), the establishment of a measurement graph shall be considered to essentially be a top-down operation.

In contrast, the taking of measurements to realize such a measurement graph, is normally a bottom-up operation, where the direct measures are first calculated, in order for the various next levels of derived measures to have all of the base measures calculated prior to being calculated themselves.

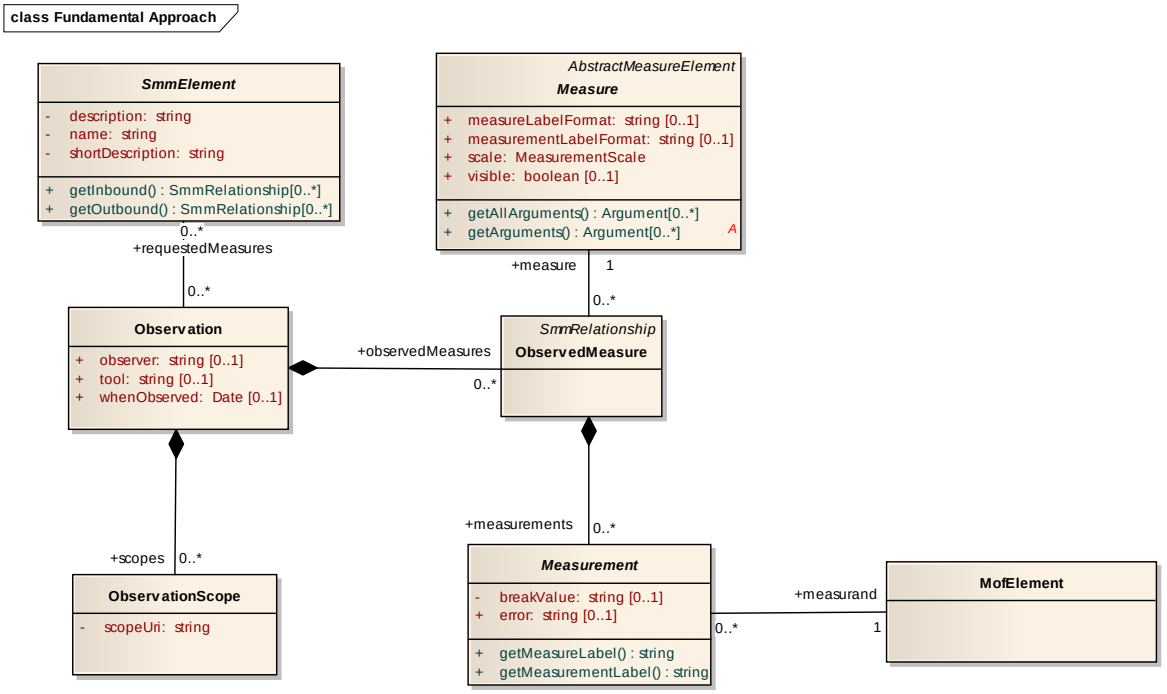

Figure 7.1 - Fundamental Approach

SMM avoids duplicating features of the measured artifact as features of the measurement. Consider as an example a log of bug reports. Possible measures are total bug count in the log, total time logged in the log, and bugs per time-period. The units of measures are a bug, a unit of time, and bugs per time interval, respectively. SMM does not provide representations for bug, start time and end time. Their representations must be provided elsewhere[1].

A measurement result is precisely identified only if its measure is identified. To understand the meaning of 1000 lines we need to know that it is the result of measuring a program's length in lines. The measured entity must be identified. That is, 1000 lines is for a particular program. Contextual information may also be needed. For example, function point counts of a program may vary depending upon the expert applying the measure.

???? presents the fundamental approach of this specification. Measurement has a value conveying the measurement results. The measurement may be of any MOF element as related by the measurand association. In this way, measurement is applicable to elements of any OMG meta-models including the Knowledge Discovery Meta-model and the Abstract Syntax Tree Meta-model. The measured entity may represent any software artifact or an aspect of an artifact.

The SMM associates an evaluation process, a measure, to each of the measurements. Measures signify functions from the domain of the modeled artifacts and aspects thereof to sets of ordered values.

Contextual information is related by Observation, such as who, where, and when. Observation may serve to distinguish distinct utilizations of a given measure on a given measurand.

1. For example, the General Ledger Specification v1.0 provides representations for start_date and end_date.

8 Core Classes

8.1 General

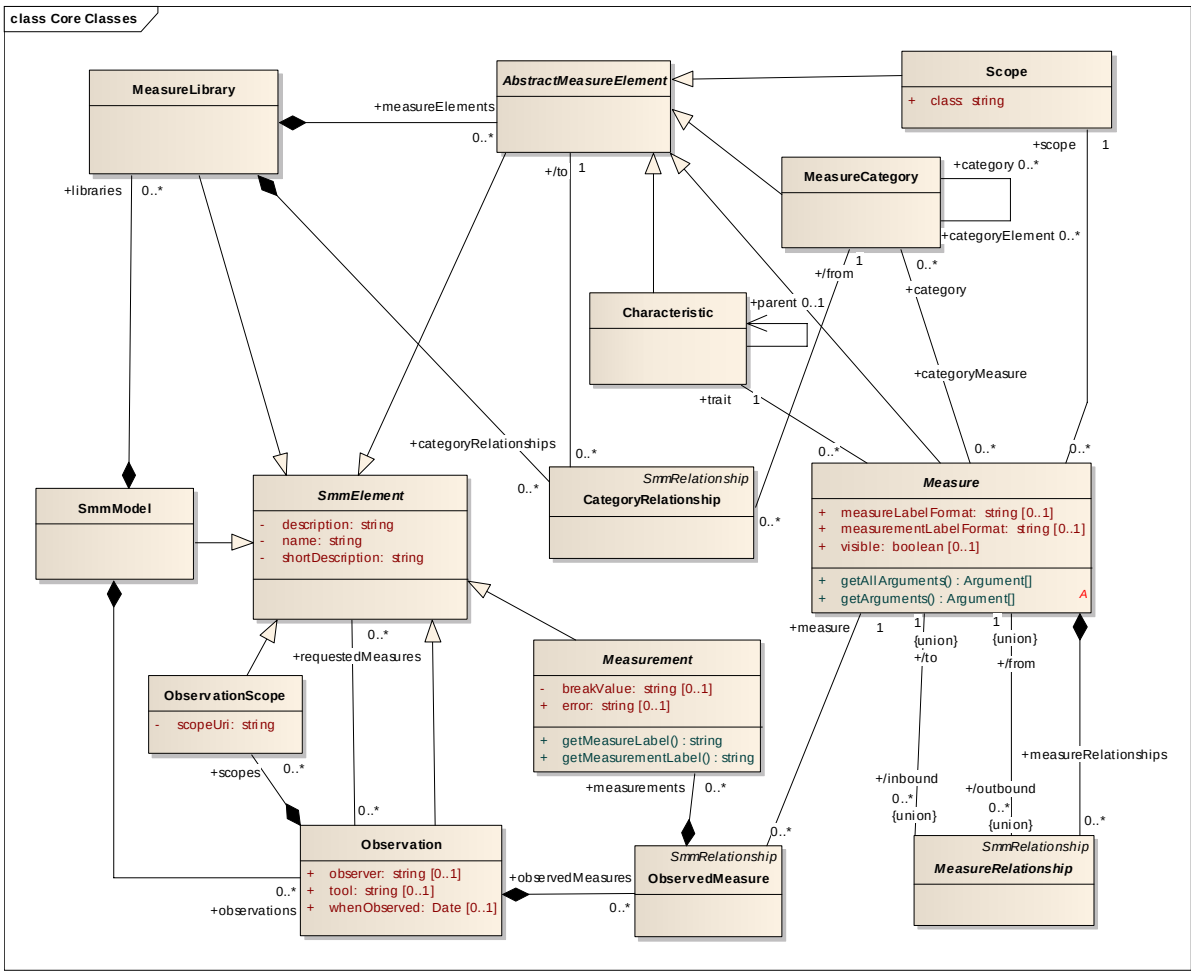

Figure 8.1 - Core Classes Diagram

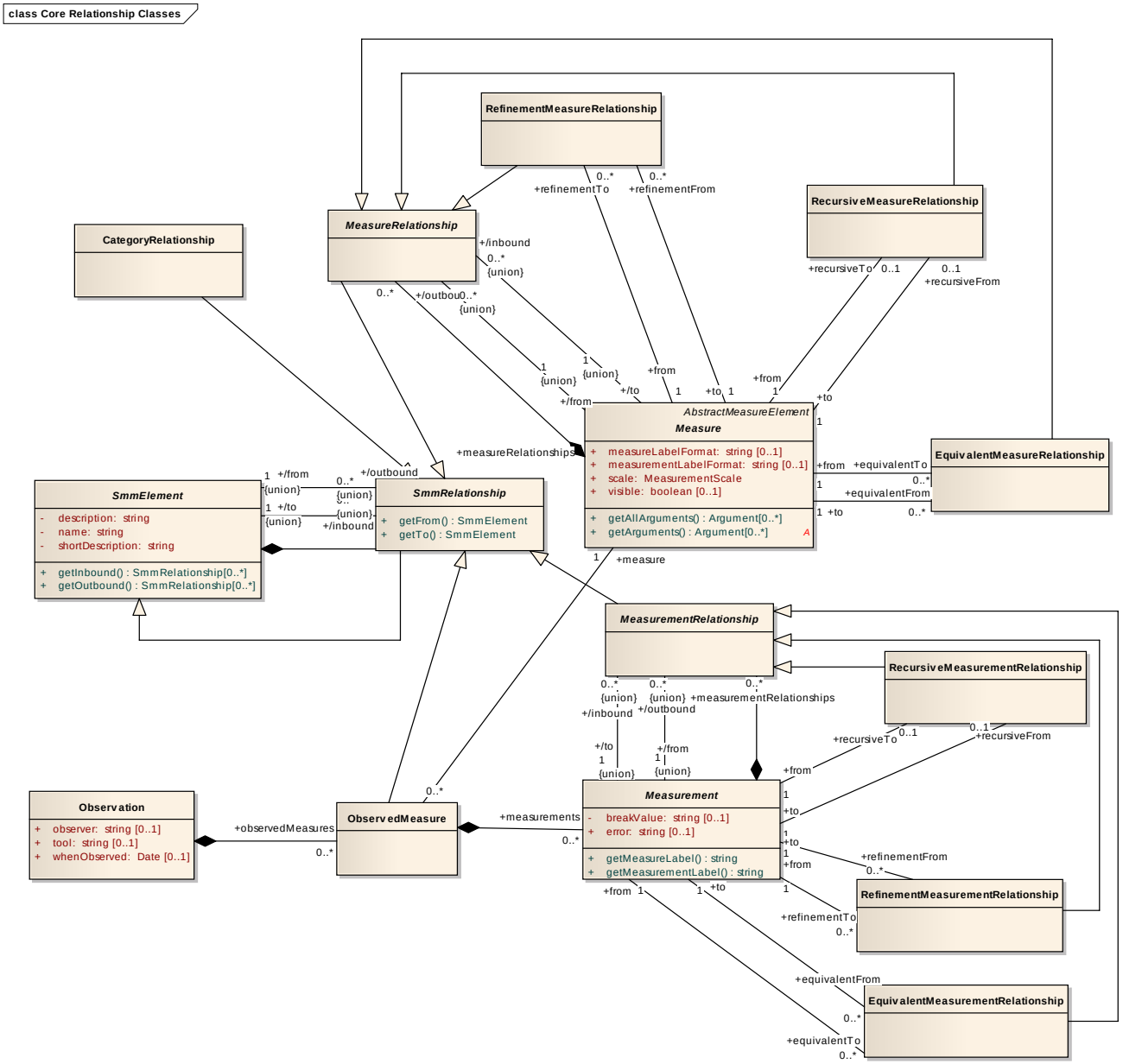

Figure 8.2 - Core Relationship Classes

8.2 SmmElement Class (Abstract)

An SmmElement constitutes an atomic constituent of a model. In the meta-model, SmmElement is the top class in the hierarchy. SmmElement is an abstract class.

Attributes

name: String	Specifies the name of the SMM element (optional)
shortDescription: String	A short description for the element (optional).
description: String	A detailed description for the element (optional).

Associations

inbound:SmmRelationship[0..*]	The set of relationship such that the current SmmElement is the to-endpoint of these relations. This property is a derived union.
outbound:SmmRelationship[0..*]	The set of relationship such that the current SmmElement is the from-endpoint of these relations. This property is a derived union.

Operations

getInbound:SmmRelationship[0..*]	This operation returns the set of relations represented by the derived union inbound relation.
getOutbound:SmmRelationship[0..*]	This operation returns the set of relations represented by the derived union outbound relation.

8.3 SmmModel Class

This class represents the entry point into the SMM model and provides the top-level container for all the elements of the SMM.

SuperClass

SmmElement

Associations

libraries:MeasureLibrary [0..*]	The set of all MeasureLibrary owned by the model.
observations:Observation[0..*]	The set of all Observation owned by the model.

8.4 SmmRelationship Class (abstract)

This class is a model element that represents semantic association between SMM elements.

SuperClass

SmmElement

Associations

from:SmmElement[1]	The *origin* element (also referred to as the from-endpoint of the relationship). This property is a derived union.
to:SmmElement[1]	The *target* element (also referred to as the to-endpoint of the relationship). This property is a derived union.

Operations

getFrom:SmmElement [1]	This operation returns the SmmElement that is the to-endpoint (the target) of the current relationship.
getTo:SmmElement[1]	This operation returns the SmmElement that is the from-endpoint (the origin) of the current relationship.

8.5 MeasureLibrary Class

This class represents libraries of measures. A library represents the top container for all measure artifacts. The library of measures defines a reference set of measures that can be applied over and over in a way that is independent and decoupled from the models under observation. Therefore it shall be possible to pre-define library of metrics and to pass those libraries to a builder so that the metrics can be applied to specified models, without affecting the measures in the library.

SuperClass

SmmElement

Associations

measureElements:AbstractMeasureElement [0..*]	The set of all AbstractMeasureElement owned by the measure library.
categoryRelationships:CategoryRelationship [0..*]	The set of all CategoryRelationship owned by the measure library.

Semantics

Measure elements can be related across libraries and need not be restricted to their own library.

8.6 MeasureCategory Class

This class represents categories of measures. A category has measures and other categories as its elements.

A category represents the measures directly associated with an ‘element’ and the measures of each sub-category likewise associated with an ‘element.’

A measure may appear in multiple categories. A category can be a subcategory of other categories indicating only that its measures also are measures of these other categories.

This class may be used to represent a family of similar measures that apply to different scopes such as lines of code in a file, lines of code in a method, and lines of code in program. It may also represent a category of measures that are associated with a given field or engineering task. For instance we speak often of Quality Assurance Metrics and Software Maintainability Metrics. The category of a metric may indicate the kind of purpose for which the metric is used.

- Environmental Metrics (e.g., number of screens, programs, lines of code, etc.)
- Data Definition Metrics (e.g., number of data groups, overlapping data groups, unused data elements, etc.)
- Program Process Metrics (e.g., Halstead, McCabe, etc.)
- Architecture Metrics (e.g., average call nesting level, deepest call nesting level, etc.)
- Functional Metrics (e.g., functions defined in system, business data as a percentage of all data, functions in current system that map to functions in target architecture, etc.)
- Quality Metrics (e.g., failures per day, meantime to failure, meantime to repair, etc.)
- Performance Metrics (e.g., average batch window clock time, average online response time, etc.)
- Software Assurance Metrics

Metric categorization has other uses as well. For example, measures may be categorized by tool support.

SuperClass

AbstractMeasureElement

Associations

category:MeasureCategory[0..*]	Represents the parent endpoint of the category hierarchy relationship.
categoryElement:MeasureCategory[0..*]	Represents the children endpoint of the category hierarchy relationship.
categoryMeasure:Measure[0..*]	Represents that measure is in this category.

8.7 CategoryRelationship

This class is a model element that represents semantic or named association between Measure categories and other Measure elements. For example, a modeler may choose to create a “gold standard” measure for a selected category. To do so, the modeler can use a category relationship named “gold standard” to associate the measure to the category. See Figure 18.1.

SuperClass

SmmRelationship

Associations

from:MeasureCategory[1]	Indicates the measure category that has relation.
to:AbstractMeasureElement[1]	Indicates the Category or Measure element related to the category. A constraint is used to limit the type of SmmElement that can be used.

Semantics

CategoryRelationship represents a named association between a measure category and a measure element (AbstractMeasureElement) such as a measure.

Constraints

```
context CategoryRelationship inv:

to.oclIsTypeOf(MeasureCategory) or

measures.oclIsTypeOf(Measure)
```

8.8 Date

This represents dates. In a language binding it should be mapped to a type that allows ordered comparison. For XMI it is mapped to the XML Schema **date** type.

8.9 Timestamp

This represents a point in time: for example, a combination of a date and a time within the day. For XMI it is mapped to the XML **dateTime** type.

9 Extensions

9.1 General

The SMM model provides for a set of simple extension mechanisms that provide a uniform meta-model pattern for extending the SMM model.

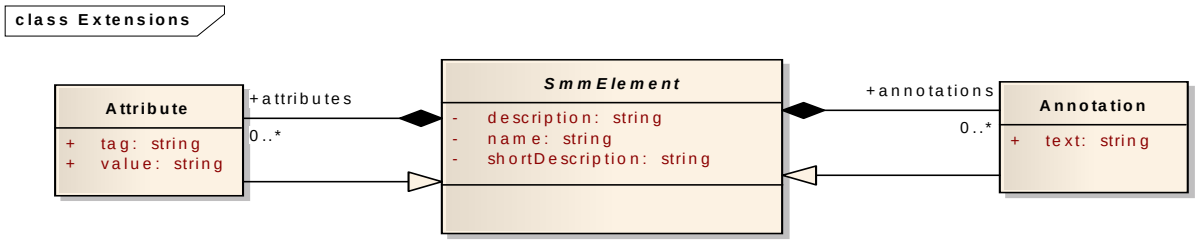

Figure 9.1 - SMM Extensions

This diagram defines meta-model elements that allow ad hoc user-defined attributes and annotations to instances of SMM elements. The mechanism of ad hoc user-defined attributes provides a capability to add pairs of <tag, value> to an individual element instance. An ad hoc user-defined attribute is owned by an individual element instance. This means that different instances of the same meta-model element may own completely different user-defined attributes (and some may have none at all).

An Annotation is an ad hoc note owned by an individual element instance. Annotations and attributes are applied to the elements of SMM instances. They may be used by implementer to add specific information to an individual element. They may also be used by an analyst, annotating a given SMM instance.

9.2 Attribute Class

An attribute allows information to be attached to any model element in the form of a “tagged value” pair (i.e., name=value). Attribute add information to the instances of model elements.

SuperClass

SmmElement

Attributes

tag: String	Contains the name of the attribute. This name determines the semantics that are applicable to the contents of the value attribute.
value: String	Contains the current value of the attribute.

Constraints

Attribute cannot have further annotations or attributes.

Semantics

The interpretation of attribute semantics is outside the scope of SMM. It must be determined by the user or the implementer conventions. It is expected that some tools will provide capability to add arbitrary attributes to the instances of the model to supply information needed for their operations beyond the basic semantics of SMM. Such information could support analysis of SMM models by analysis, etc.

An attribute element is not related to a particular meta-model element. It does not define a “virtual” attribute to an extended meta-model element that is instantiated with every instantiation of the new element. Instead, an attribute element can be added to any SMM element. It defines a property of a particular instance, not a property of a class of instances.

9.3 Annotation Class

Annotations allow textual descriptions to be attached to any instance of a model element.

SuperClass

SmmElement

Attributes

text: String	Contains the text of the annotation to the target model element.

Constraints

Annotations cannot have further annotations or attributes.

Semantics

Annotation allows associating a human-readable text with an instance of any Element.

10 Measures

10.1 General

Measures are evaluation processes that assign comparable numeric or symbolic values to entities in order to characterize selected qualities or traits of the entities. Counting the lines of program code in a software application is one such evaluation.

There may be many measures that characterize a trait with differing dimensions, resolutions, accuracy, and so forth. Moreover, trait or characteristic may be generalized or specialized. For example, line length is a specialization of length that is a specialization of size.

Each measure has a scope, the set of entities to which it is applicable; a range, the set of possible measurement results; and the measurable property or trait that the measure characterizes. For example, the aforementioned line counting has software applications as one of its scope with line length as one of its measurable trait. Explicitly representing the scope and the measurable trait allows for the consideration of different measures, which characterize the same attribute for the same set of entities. Each measurable trait may have multiple, identifiably distinct measures.

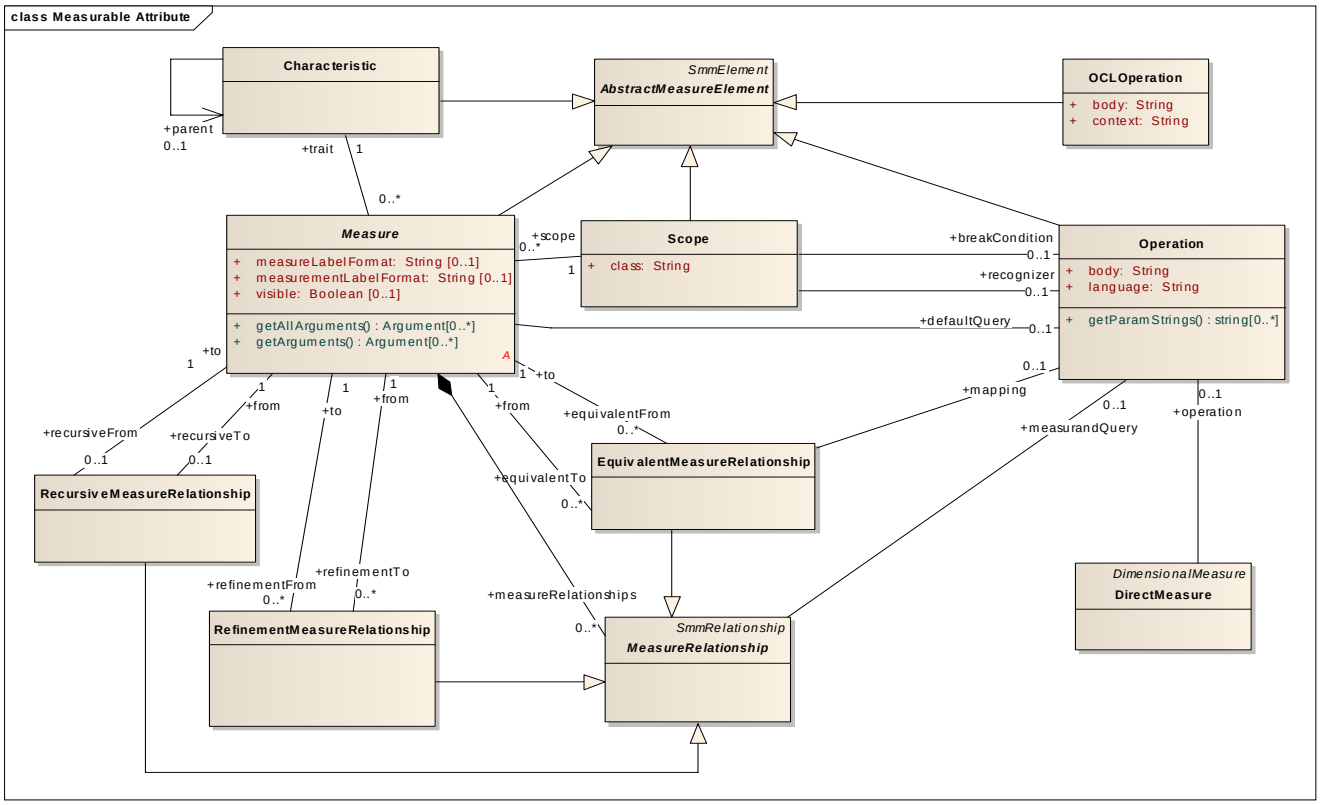

Figure 10.1 - Measurable Characteristic and Scope

The evaluation process may assign numeric values that can be ordered by magnitude relative to one another. These measures are modeled by the DimensionalMeasure class.

The evaluation process may alternatively assign numeric values that are percentages or, more generically, ratios of two base measurements. These measures are modeled by the Ratio class. The percentage of comment lines in an application exemplifies this type of measure.

The evaluation process may also assign symbolic values demonstrating a ranking that preserve the ordering of underlying base measures. These measures are modeled by the Ranking class. Cyclomatic reliable/unreliable criterion illustrates one such ranking. Reliable is comparably better than unreliable. Comparability is essential here because ranking is not intended to model every possible assignment of measurands.

The documentations of measures, accomplished with measure libraries, should stand by themselves so that an interchange of measurements may simply reference such documentation and not duplicate it. The documentation of measures should also be precise and complete enough to provide for an unambiguous specification that can be executed on a referenced model, with the exception of the NamedMeasure when used for simple result interchange. The actual ability to execute a model is not part of the compliance to this specification and neither is the method to provide execution defined within this specification. These are left to the implementers.

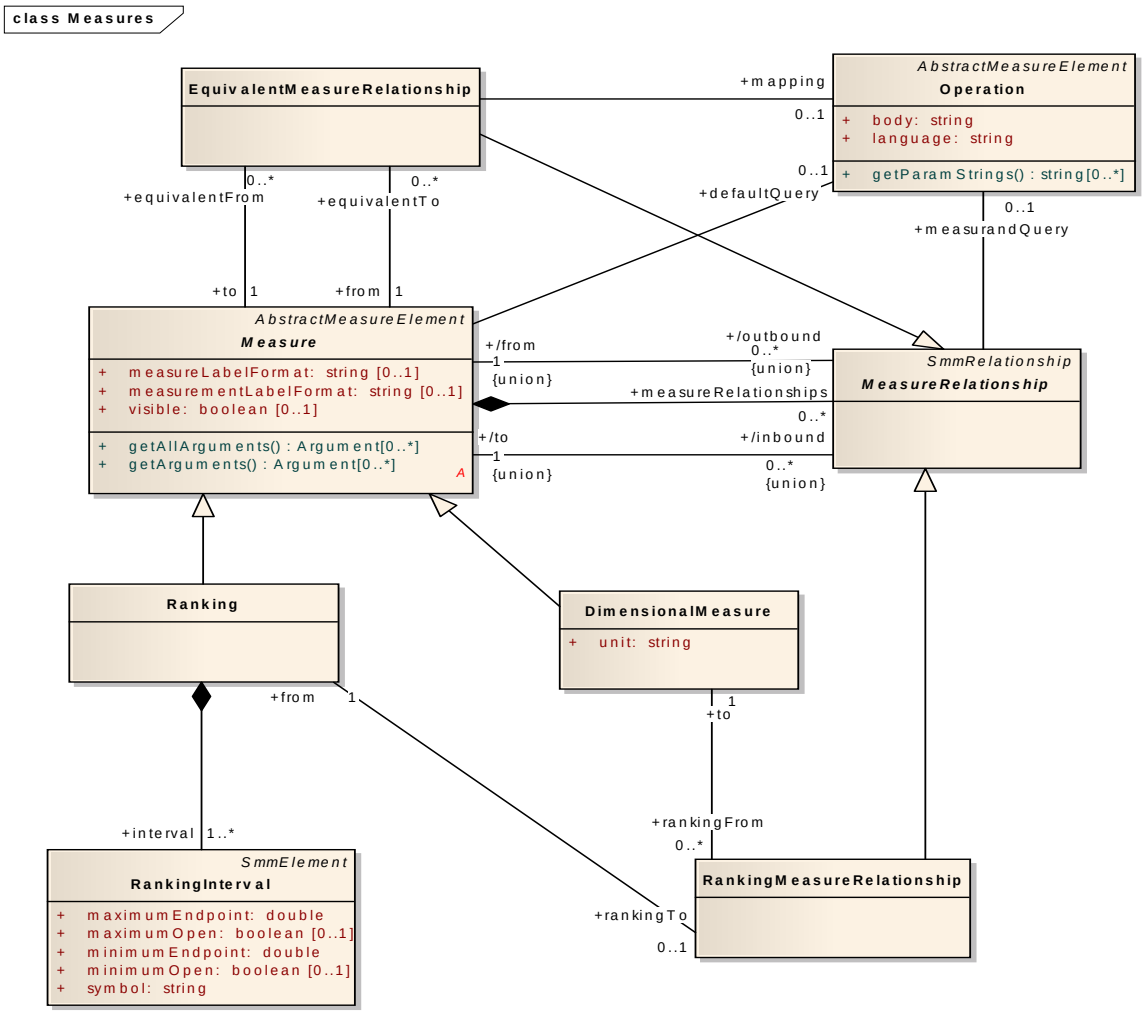

Figure 10.2 - Measure Class Diagram

The AbstractMeasureElement is the abstract parent class for all measure entities.

SuperClass

SmmElement

Associations

None

10.2 Characteristic Class

This class represents a property or trait of the members in its scope, a set of MOF Elements, which may be characterized by applying a measure to those members. By specifying a characteristic a modeler is indicating what aspect, trait, or property the measure purports to measure.

Note that Characteristic provides for a representation of a hierarchy of measures based upon the abstraction of measured trait. For example, a length characteristic may be the parent of the fileLength and programLength characteristics. programLength could be the parent of programLinesOfCodeLength.

SuperClass

AbstractMeasureElement

Attributes

name: String	Specifies the name of the SMM element. (inherited)

Associations

parent:Characteristic[0..1]	Specifies the generalization of this characterization.

10.3 Scope Class

This class represents sets of MOF::Elements as domains for measures. The domain is a subset instances of a class specified by the class attribute. If the subset does not include all instances of the given class, then a restriction is specified by specifying a recognizer for the subset elements.

The scope of a measure identifies a set of objects as the domain of the measure. The objects all exhibit to varying degrees the trait or property characterized by a measurement. SMM requires that the objects be instances of a single class. The set of objects may be further restricted by a recognizer operation. The recognizer is optional.

The recognizer, if given, is a boolean operation applicable to instances of the named class. The measure's scope is restricted to those instances for which the recognizer returns true.

SuperClass

AbstractMeasureElement

Attributes

class: String[1]	Specifies the class for elements of the set. See semantics for format rules (required).

Associations

recognizer:Operation[0..1]	If given, provides a boolean operation applicable to instances of the class that returns true if, and only if, the instance is an element of the set.
breakCondition: Operation[0:1]	If given, provides for an operation that returns a string describing a break condition to allow for dynamically grouping instances of the class in scope by a certain value. For example, this can be used to group elements by language name in KDM SourceItem or by folder name in Inventory Items, without having to know all of the possible conditions in advance.

Semantics

The class attribute may name a class within any MOF model. The entities associated as elements of a Scope are restricted to members of the specified class.

The class attribute should be able to provide an unambiguous way to specify a class name. In order to achieve this goal, the string should be formatted according to the following pattern, with all 3 elements being required:

Namespace:Package::ClassName

This usage of package pathnames is transitive and can also be used for packages within packages:

Packagename1::Packagename2::ClassName

Where:

- Namespace represents the model where the class is defined. Namespace can be either one of the pre-defined values (“kdm,” “astm,” or “smm” at the moment) or be a namespace defined in the XMI where this measure is located. For example a namespace value of “mykdm” would be valid if the SMM model contains the following XMI namespace definition in its header: “xmlns:mykdm=http://kdm.somecompany.com/spec/KDM/1.4”. XMI based namespace definition can also be used with the standard namespace to point the class name definition to a specific version of those model specification. Without such a namespace entry, the pre-defined values would point to a “current” unspecified version.
- Package represents the package name within the model
- ClassName represents the base class name within the specified package.

The breakCondition attribute is defined as an OCL operation that evaluates to a string representing the group or break value of the class instance.

- Examples:
    1. this.language
        1. This would represent a break on the attribute language, as seen in the KDM inventory model SourceFile class. Applicable as long as the measurand class is the same as the scope class, SourceFile in this example.

10.4 Measure Class (abstract)

The Measure class (see Figure 7.1) models the specification of measures either by name, by representing derivations of base measures, or by representing method operations directly applied to the measured object. The essential requirement for the measure class is that it meaningfully identifies the measure applied to produce a given measurement. For example, McCabe's cyclomatic complexity could be specified by its name, McCabe's cyclomatic complexity, by a direct measurement operation or by rescaling counts of either independent paths or choice points. A measure may alternatively be identified by citing a library of measure which includes the measure by name.

The scope of a measure identifies a set of objects as the domain of the measure. The objects all exhibit to varying degrees the trait or property characterized by a measurement. SMM requires that the objects be instances of a single class. The set of objects may be further restricted by a recognizer function. The recognizer is optional.

Scope need not be specified if the library and name are given. In that case, the scope can be found in the library.

A measure may be a refinement of another measure. The scope of the first measure is a subset of the second measure's scope. The characteristic of both measures must be identical.

SuperClass

AbstractMeasureElement

Attributes

name: String[1]	Specifies the unique name of the measure. (inherited)
measureLabelFormat:String[0:1]	Specifies a label format string to use when rendering this measure. See semantics for detailed content format.
measurementLabelFormat:String[0:1]	Specifies a label format string to use when rendering measurements of this measure. See semantics for detailed content format.
visible:boolean[1:1]	Specifies if rendering tools should display this measure or not. Some measures whose role is only to help produce other measures will often be marked as non-visible. Defaults to true.

Associations

scope:Scope[1]	Specifies a set of elements measurable by this measure.
defaultQuery:Operation[0..1]	Specifies a query that is used to determine a default value for the measure in case we are dealing with a non-direct measure (i.e., a measure that depends on another for its value) where its base measure returns no children. This is a normal situation that can happen when certain optional “children” don’t exist.
equivalentFrom:EquivalentMeasureRelationship[0..*]	Specifies the relationship instance that defines the equivalency of this measure.
equivalentTo: EquivalentMeasureRelationship[0..*]	Specifies the relationship instance that defines the equivalency of this measure.
refinementFrom:RefinementMeasureRelationship[0..*]	Specifies the relationship instance that defines the refinement of this measure.
refinementTo:RefinementMeasureRelationship[0..*]	Specifies the relationship instance that defines the refinement of this measure.
recursiveFrom:RecursiveMeasureRelationship[0..*]	Specifies the relationship instance that defines the recursivity of this measure.
recursiveTo:RecursiveMeasureRelationship[0..*]	Specifies the relationship instance that defines the recursivity of this measure.
category:MeasureCategory[0..*]	Specifies categories to which this measure belongs.
trait:Characteristic[1]	Specifies the trait characterized by this measure.
inbound:MeasureRelationship[0..*]	The set of relationship such that the current Measure is the to-endpoint of these relations. This property is a derived union.
outbound:MeasureRelationship[0..*]	The set of relationship such that the current Measure is the to-endpoint of these relations. This property is a derived union.
measureRelationships:MeasureRelationship[0..*]	The set of all MeasureRelationship owned by the measure.

Operations

getArguments:Argument[0..*]	This operation returns the set of arguments that the different operations of the measure have defined and got returned by getParamStrings().
getAllArguments:Argument[0..*]	This operation returns the set of arguments for this measure and any child measure required for the execution of the measure. It should call getArguments() on itself and every one of its child measures.

Semantics

The labelFormat is based on the concept of format string used in many languages to assemble string content for rendering. Although beyond the scope of this specification to cover implementation details, this format also supports the use of external resource to provide i18N internationalization.

Just like format strings, the labelFormat is defined as a text portion with possible replacement expressed as argument index surrounded by French braces “{}”, where the zero-based index is matched with its corresponding replacement argument, which follow the text portion.

Label format specification:

“Template Text”, Context Object: OperationName, ContextObject.attribute,…

Examples of the label String Template could be:

“This is a label”	A fixed string, in which case no arguments are necessary.
“This {1} of {0}	A label with replaceable arguments that will come from evaluating the corresponding argument from the list supplied (in numerical order, starting at 0).
$Resource:resource_text_constant	Here resource_text_constant would be replaced with a constant that will be matched in some resource location and for the proper locale (not defined here). The content returned by this resource resolution can be any valid label string template.

The arguments of the label format are defined in a comma separated list. Each of those arguments must follow a specific pattern. There is a standard syntax and also a shorthand version for some common cases.

The standard syntax, which is always valid, starts by specifying a context object, followed by a literal colon “:”, then an operation whose name must be the name of a valid instance in the Operation class,

- ContextObject: It is the first part and it represents the Object that we are interested in collecting information from. This object is related or associated with the measurement such as the Scope or the measure or the measurand …etc. It is important to understand here that the labelFormat is defined as part of the measure, but it is accessed normally from within the context of a measurement.
- OperationName: Defines the name of a valid instance of the Operation class that is designed to return a string.

The shorthand syntax is valid to get the value of attributes from the current instance of measurement, measure, and scope based on the current context of the initial measurement. This syntax calls for the use of a dotted notation being ContextObject.attributeName. For example you could get “Measure.name” or “Scope.class” directly.

The defaultQuery is designed to provide a way to specify a default value in the specific case where a non-direct measure (i.e., a measure that depends on another for its value) happens not to have any available value from what should have been its “base measure.” In those case, the query should be executed to provide for the value instead of returning null or failing the measurement, as this is a normal situation that can happen when certain optional “children” don’t exist.

10.5 Operation Class

Operation is a subclass of AbstractMeasureElement that defines an operation to execute.

SuperClass

AbstractMeasureElement

Attributes

language:String	Specifies the language of the operation. Valid values are currently “OCL” and “XQuery.”
body:String	Specifies the measurement operation expressed in the selected language.

Operations

getParamStrings:String[0..*]	This operation returns the set of String that defines the parameter in use by an operation.

Semantics

The operation body supports the use of replaceable parameters in order to support parameterized measures. This is accomplished by defining placeholders for incoming arguments that will be replaced at runtime with a specific value, like when dealing with date ranges for example.

The implementer is responsible, when using the measure library in an executable fashion, to determine base on the requested measures of his observation, what are all of the arguments that should be passed in with the observation in order to properly perform the measurements. The getArguments and getAllArguments operation of the Measure class are designed to help in this regard.

When parameters are used they must adhere to the following specification: '{' [typeName] parameterName [' =”' defaultValue '” '] '}' where:

- typeName represents the type of the parameter. The typeName must be one of the types supported by the “type” attribute of the Argument class.
- parameterName represents the name of the parameter (required).
- defaultValue represents a default value to offer (on getArguments()) or to use if not supplied as Argument to an observation. defaultValue is optional.

10.6 OCLOperationClass

OCLOperation is a subclass of AbstractMeasureElement that defines OCL helper methods.

SuperClass

AbstractMeasureElement

Attributes

context:String	Specifies the classifier for which this helper is being defined. OCL inheritance rules applies to resolve applicability of operation, based on the passed in context.
body:String	Specifies the body of the OCL helper method.

Semantics

The OCLOperation class allows for the definition and registration of OCL helper methods in the context of specific classifiers. These operations allow for the definition and reuse of often lengthy and complex OCL methods. It is the implementer’s

responsibility to determine how to best provide for the parsing or execution environment of those methods. Any helper method that is defined with an OCLOperation then becomes available for OCL based operations applied to the proper classifier.

10.7 MeasureRelationship Class (abstract)

MeasureRelationship is an abstract class representing any relationship between two measures. See Figure 10-2.

SuperClass

SmmRelationship

Attributes

name:String	Specifies the name of this measure relationship. (inherited)

Associations

from:Measure [1]	The origin element (also referred to as the from-endpoint of the relationship). This property is a derived union.
to:Measure [1]	The target element (also referred to as the to-endpoint of the relationship). This property is a derived union.
measurandQuery:Operation[0..1]	Specifies a query that is used to determine the measurands that satisfy the relation between two measures. It is most often used to specify the measurands that match a specific non-containment refinement relation between measures.

Semantics

By default, relationships between measures have their meaning implied by their concrete subtype. The measurandQuery defines an optional way to describe this relationship by allowing the specification of a query operation that will return the specific measure instance that satisfies the query condition. It is mostly designed to be used with RefinementMeasureRelationship in order to provide a navigation that is different than the default containment mode.

10.8 EquivalentMeasureRelationship Class

EquivalentMeasureRelationship is a class representing any relationship of equivalency between two measures. See Figure 10.2.

SuperClass

MeasureRelationship

Associations

from:Measure[1]	Specifies the equivalent measure at the from endpoint of the relationship.
to:Measure[1]	Specifies the equivalent measure at the to-endpoint of the relationship.
mapping:Operation[0..1]	Specifies the mapping operation query that retrieves the “to” measure between a pair of equivalent measures, when each measure is represented by a different scope.

Semantics

Defining a measure as being equivalent to another measure states that two measures are semantically indistinguishable. Any measurement result by one on a given entity under a given observation should equal a measurement by the other on the same or different entity as long as they are part of the same observation.

The semantics of this association is symmetric, but only one direction needs to be defined in a way that is resolvable, i.e., in a way that provides a path all of the way to base measures assigned against outside measurand. If a measure can’t resolve to base measurements but is defined as equivalent to another measure, then it can use this equivalency to derive its own measurement result.

This means that when establishing the dependency graph for calculation, a measure can find its base measure not only through direct lineage, but also through measure equivalency. For example, calculating LOC at various levels in code can be defined against ASTM. Then we define that the ASTM CompilationUnit level LOC measure is equivalent to the KDM SourceFile LOC measure. This then allows for the SourceFile LOC measure to find its result through its equivalency relationship.

10.9 RefinementMeasureRelationship Class

Refinement MeasureRelationship is a class representing any relationship of refinement between two measures.

SuperClass

MeasureRelationship

Associations

from:Measure[1]	Specifies the measure at the from endpoint of the relationship.
to:Measure[1]	Specifies the measure at the to-endpoint of the relationship.

Semantics

Throughout the remainder of this document we will say that a measure is a refinement of another measure if and only if the first is associated to the second as a refinement directly or transitively.

When this association is defined without a measurandQuery (from MeasureRelationship superclass), then it implies that the from and to measure of the refinement are related through a containment relation where the from measure is the container and the to measure represents the content of the container.

When the refinement relation between the two measure classes is not a direct containment, then a measurandQuery should be used to provide the appropriate query to retrieve the related children in the scope of the ‘to’ measure.

10.10 RecursiveMeasureRelationship Class

RecursiveMeasureRelationship is a class representing any relationship of recursivity on a measure upon itself.

SuperClass

MeasureRelationship

Associations

from:Measure[1]	Specifies the measure at the from endpoint of the relationship.
to:Measure[1]	Specifies the measure at the to-endpoint of the relationship.

Semantics

Defining a measure as being recursive to itself states that measure can recursively refine itself and that we intend to apply this recursive refinement to our measure.

Constraint

```
context RecursiveMeasureRelationship inv:
from = to.
```

10.11 DimensionalMeasure Class

This class models the specification of measures which assign numeric values that can be placed in order by magnitude. Dimensional measures have units of measures and their values span a dimension. See Figure 10.1.

The unit of measure is an archetypal or prototype element of the dimension. Every element of the dimension can be stated by a numerical multiple of the ‘unit of measure’ element.

The unit of measure does not distinguish between measures which share the same range. That distinction would be entirely within the purview of the measure identification. For examples, a height measure and a width measure may share the same unit of measure. That is to say, a measurement is not just a number and a unit of measure. The measured artifact must be indicated, the measure identified and contextual information retained as the observation.

SuperClass

Measure

Attributes

unit:String	Identifies the unit of measure.

Associations

rankingFrom:RankingMeasureRelationship[0..*]	Specifies the relationship instance that defines the rankings for this measure.
baseMeasureFrom:BaseMeasureRelationship[0..*]	Specifies the relationship instance that defines the accumulation for this measure.
baseMeasure1From:Base1MeasureRelationship[0..*]	Specifies the relationship instance that defines the 1st part of the binary comparator for this measure.
baseMeasure2From:Base2MeasureRelationship[0..*]	Specifies the relationship instance that defines the 2nd part of the binary comparator for this measure.
rescaleTo:RescaledMeasureRelationship[0..*]	Specifies the relationship instance that defines the measure rescaling this measure.

10.12 Ranking Class

This class represents simple range-based grading or classifications based upon already defined dimensional measures. See Figure 10.2.

Examples are:

- Small, medium, large
- Cold, warm, hot
- A, B, C, D or F
- Reliable / Unreliable

Collectively the ranking intervals may completely cover the base dimension or may leave gaps. A base measurement in such a gap is considered unranked and is not representable as a measurement of the ranking measure.

The intervals may overlap. A ranking resulting in a particular symbol means and only means that the base measure resulted in a value occurring a ranking's interval which mapped to that symbol. This does not exclude the possibility that the value might occur in another interval.

Ranking consists of mapping intervals to symbols where the intervals are parts of the underlying measure's dimension. For example, 100 to 90 points maps to "A," 80 up to 90 maps to "B," 70 up to 80 maps to "C," 60 up to 70 maps to "D," and below 60 maps to "F." The underlying dimension consists of grade points. The result is the usual A,B,C,D, and F style grade.

Ranking measure may represent a purely qualitative evaluation with no quantitative base measure. For example we could measure the non-standardness of the source language and evaluate it without quantification. It is identified as "2GL," "Unacceptable 3GL or 4GL," "Acceptable 3GL or 4GL," or "Ideal Strategic Language." The first two are judged equivalently non-standard. The third is more nearly standard and the last is standard.

SuperClass

Measure

Associations

rankingTo:RankingMeasureRelationship[0..1]	Specifies the relationship instance that defines the measure ranked by this ranking.
interval:RankingInterval[1..*]	Identifies intervals within the dimension of the base measure and the symbol to which each interval is mapped.

10.13 RankingMeasureRelationship

RankingMeasureRelationship is a class representing any relationship of ranking between a ranking measure and a dimensional measure.

SuperClass

MeasureRelationship

Associations

from:Ranking [1]	Specifies the ranking measure at the from endpoint of the relationship.
to:DimensionalMeasure[1]	Specifies the dimensional measure at the to-endpoint of the relationship.

10.14 RankingInterval Class

This class represents the mapping of an interval to a symbol that serves as a rank. See Figure 10.2.

SuperClass

SmmElement

Attributes

maximumOpen:Boolean	True if and only if interval include maximum endpoint. Default = false.
minimumOpen:Boolean	True if and only if interval include minimum endpoint. Default = false.
maximum:Number	Identifies interval's maximum endpoint.
minimum:Number	Identifies interval's minimum endpoint.
symbol:String	Base measurements within this interval are mapped by symbol.

Constraints

```
context RankingInterval inv:

maximum ≥ minimum and (maximumOpen or minimumOpen → maximum > minimum)
```

11 Collective Measures

11.1 General

This diagram represents measures that assess container entities by accumulating assessments of contained entities which are found by the base measure. See demonstration given in Figure 11.2.

Most engineering measures are collective. We count up lines of code for each program block and sum these values to measure routines, programs and eventually applications. A similar process is followed to count operators, operands, operator and operand occurrences, independent paths, and branching points.

Other frequently used container measures are based upon finding the maximum measurement of the container's elements. Nesting depth in a program and class inheritance depth exemplify these collective measures.

The collective measure specifies the following measurement process:

1. Apply the base measure to each contained element to obtain a set of base measurements.
2. Apply the n-ary accumulator to the set of base measurements to obtain the measurement of the container.

Figure 11.2 demonstrates this process, with simplified associations.

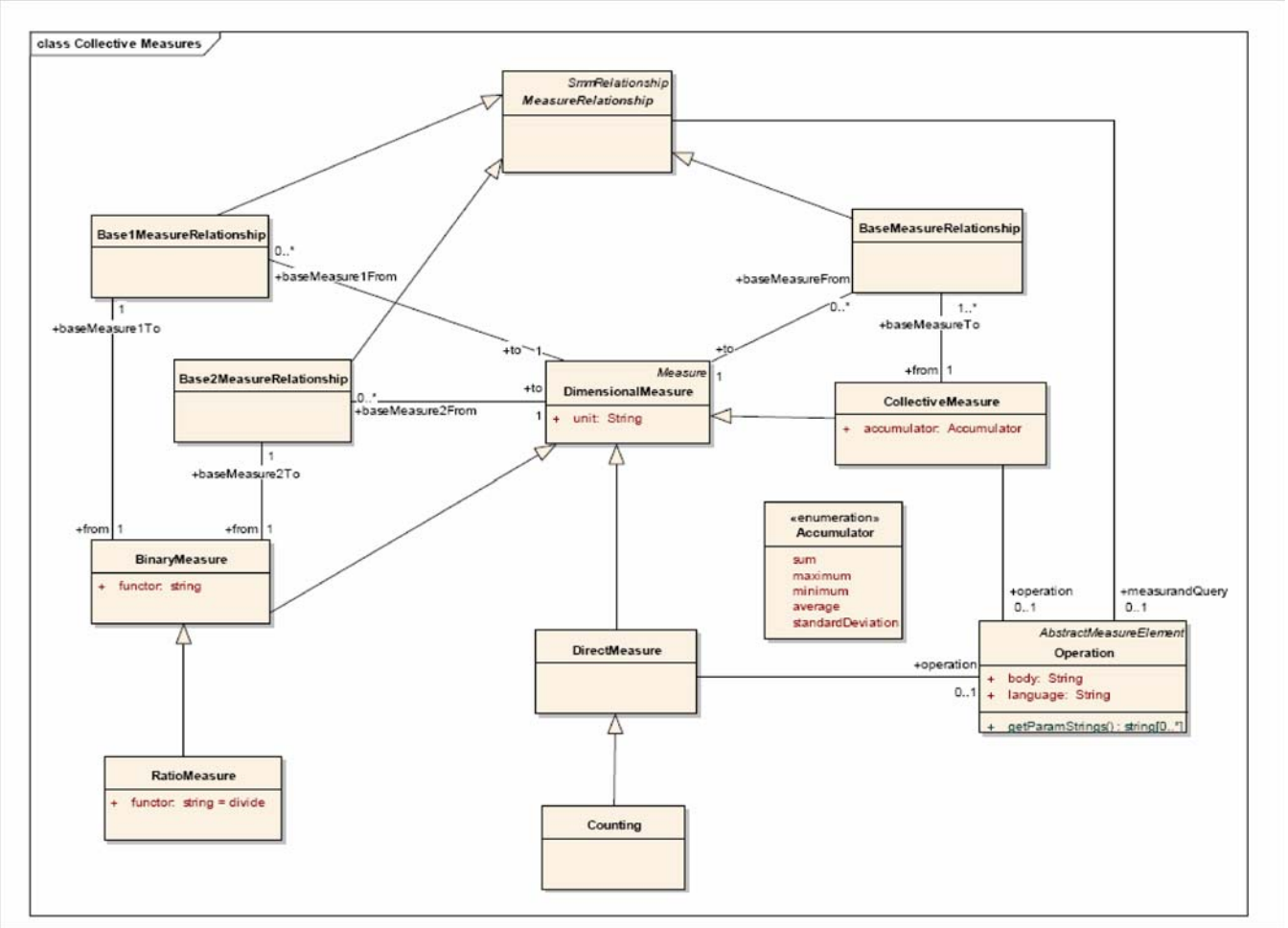

Figure 11.1 - Collective Measures

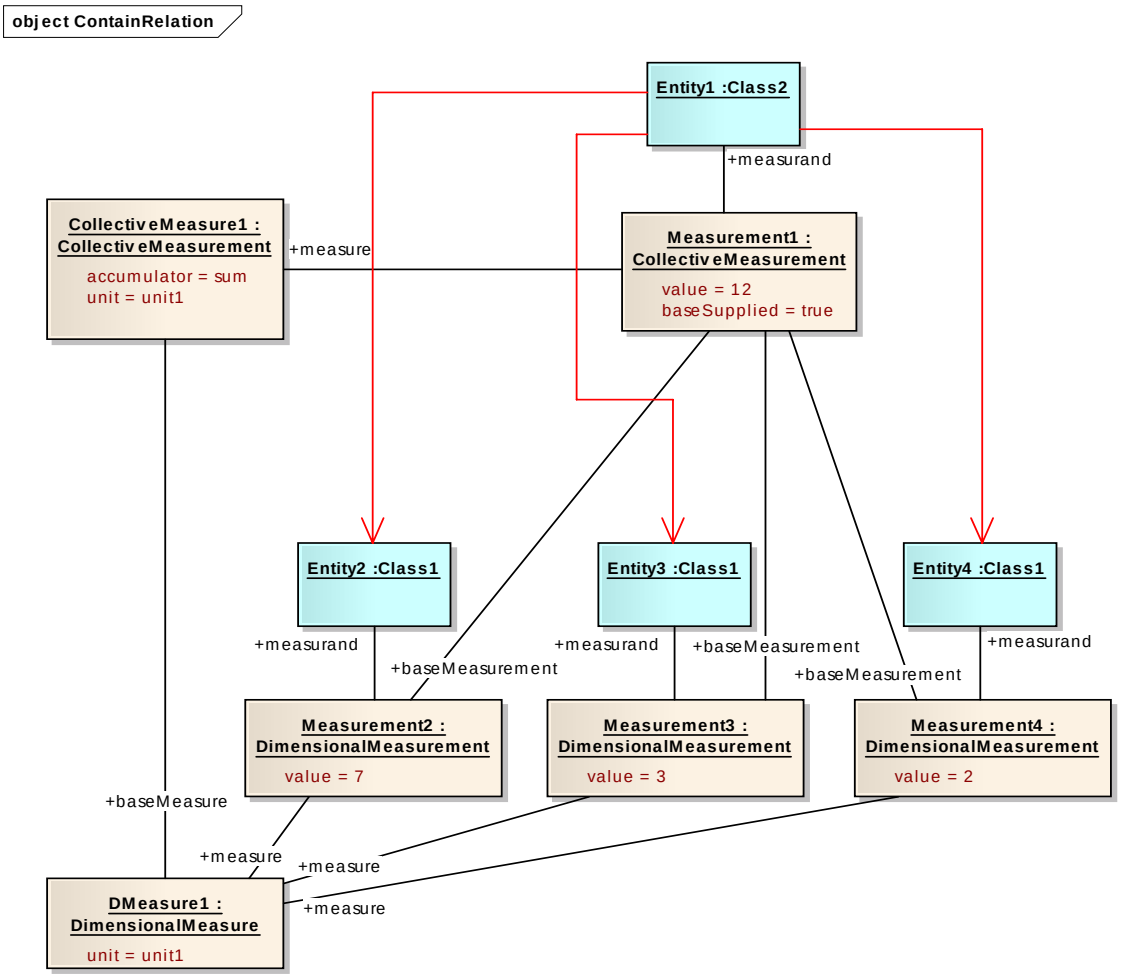

Figure 11.2 - Collective Measure Demonstration

11.2 CollectiveMeasure Class

The CollectiveMeasure class represents measures which when applied to a given entity accumulates measurements of entities similarly related to the given entity. See Figure 11.1. For example, counts for container entities are often found by accumulating (adding) counts of the containers' contained entities. In fact, sizing measures generally accumulate to containers by adding the results of applying the appropriate size measure to the contained entities.

Maximum is another frequent accumulator.

The measurands of the base measurements need not be the same of the measurand of the collective measurement. Within SMM, the measurands are just arbitrary MOF::Elements declared in another MOF model.

The SEI Maintainability Index is one such aggregation that does not change the unit of measure.

SuperClass

DimensionalMeasure

Attributes

accumulator:Accumulator	Identifies the n-ary or custom function that accumulates the base measurements.

Associations

baseMeasureTo:BaseMeasure Relationship[1..*]	Specifies the relationship instance that defines the measure accumulated by this collective measure.
operation:Operation[0..1]	Specifies the measurement operation of this measure.

Constraints

Context CollectiveMeasure inv:

accumulator->isEmpty or operation->iEmpty

11.3 Accumulator data type (enumeration)

The Accumulator enumeration defines DirectMeasure - a subclass of DimensionalMeasure which applies a given operation to the measured entity. See Figure 11.1.

Literal Values

- Sum
- Minimum
- Maximum
- Average
- standardDeviation

11.4 DirectMeasure Class

DirectMeasure - a subclass of DimensionalMeasure which applies a given operation to the measured entity. See Figure 11.1.

SuperClass

DimensionalMeasure

Associations

operation:Operation[0..1]	Specifies the measurement operation of this measure.

11.5 Counting Class

Counting is a subclass of DirectMeasure where the given operation returns 0 or 1 based upon recognizing the measured entity. See Figure 11.1.

SuperClass

DirectMeasure

Constraints

context Counting::self.operation(…):int

```
post: result = 0 or result = 1
```

The operation is a recognizer that selects some subset of the elements of the measure's scope found by self.scope. The recognizers returns 1 for the elements of the subset and returns 0 otherwise. self.unit need not be an element of the subset.

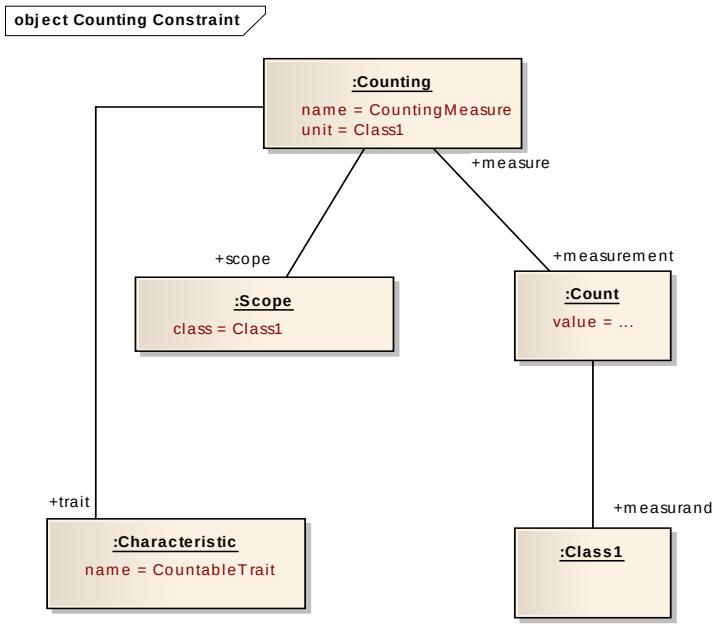

Figure 11.3 - Counting Unit of Measure Constraint

11.6 BinaryMeasure Class

The BinaryMeasure class represents measures which when applied to a given entity accumulates measurements of two entities related to the given entity. See Figure 11.1. For example, areas for two dimensional entities are often found by accumulating (multiplying) lengths.

The measurands of the base measurements need not be the same as the measurand of the collective measurement.

SuperClass

DimensionalMeasure

Attributes

functor:String	Identifies the binary function that combines two base measurements.

Associations

baseMeasure1:DimensionalMeasure	The first base measurement is derived by applying the specified measure or a refinement of it.
baseMeasure2:DimensionalMeasure	The second base measurement is derived by applying the specified measure or a refinement of it.

Semantics

The usual semantics of algebra would require that the unit of a binary measure equals applying the accumulator to the units of the base measures. While conforming to this requirement would ensure more easily understood models, SMM does not enforce this requirement.

11.7 Ratio Class

This class represents those measures that are ratios of two base measures. See Figure 11.1. Examples include:

- Average lines of code per module,
- Failures per day,
- Uptime percentage – Uptime divided by total time,
- Business data percentage of all data,
- Halstead level = Halstead volume divided by potential volume,
- Halstead effort = Halstead level divided by volume.

A ratio measure and its two base measures frequently characterize three different traits of the same entity. If the dividend characterized the total code length of an application and the divisor characterized the number of program in the application then the ratio characterizes the average code length per program.

Ratios may also characterize traits of distinct entities. For example, a ratio may contrast the code length between a pair of programs.

SuperClass

DimensionalMeasure

Constraints

```
context MaximalMeasure inv:

functor = 'divide'
```

11.8 BaseMeasureRelationship Class

BaseMeasureRelationship is a class representing relationship of hierarchy between a collective measure and a dimensional measure.

SuperClass

MeasureRelationship

Associations

from:CollectiveMeasure[1]	Specifies the collective measure at the from endpoint of the relationship.
to: DimensionalMeasure [1]	Specifies the dimensional measure at the to-endpoint of the relationship.

11.9 Base1MeasureRelationship Class

Base1MeasureRelationship is a class representing relationship of hierarchy between a binary measure and a dimensional measure.

SuperClass

MeasureRelationship

Associations

from:BinaryMeasure[1]	Specifies the binary measure at the from endpoint of the relationship.
to: DimensionalMeasure [1]	Specifies the dimensional measure at the to-endpoint of the relationship.

11.10 Base2MeasureRelationship Class

Base2MeasureRelationship is a class representing relationship of hierarchy between a binary measure and a dimensional measure.

SuperClass

MeasureRelationship

Associations

from:BinaryMeasure[1]	Specifies the binary measure at the from endpoint of the relationship.
to: DimensionalMeasure [1]	Specifies the dimensional measure at the to-endpoint of the relationship.

12 Other Measures

12.1 General

The following diagram presents three additional measures.

- Direct applications of named measurements. (One such named measure is Cyclomatic Complexity.)
- Simple algebraic change of scales of already defined numeric measures (e.g., the translation to ‘choice points’ from Cyclomatic complexity).

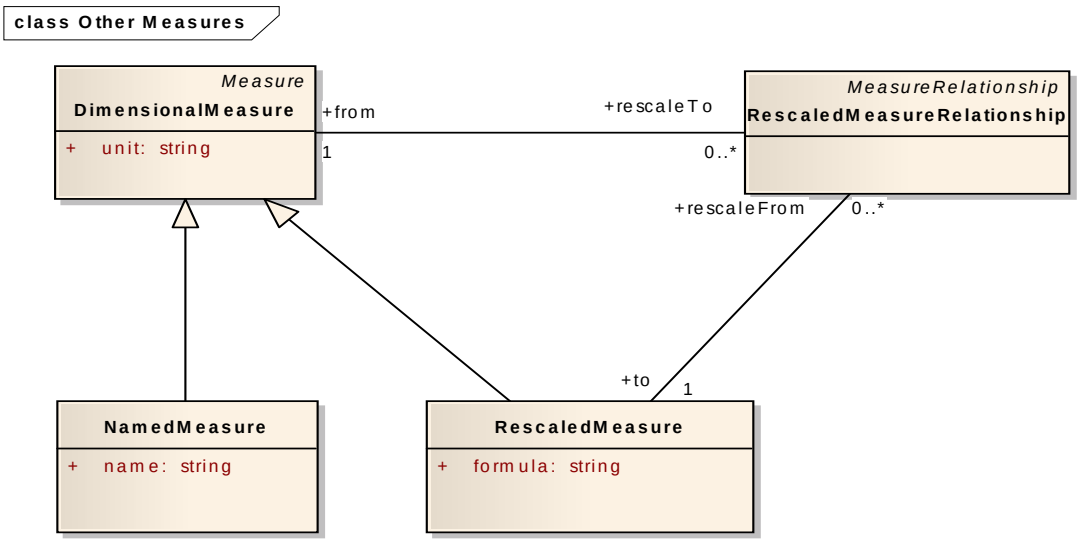

Figure 12.1 - Other Measures

12.2 NamedMeasure Class

The class allows for specifying measures which are well-known and can be specify simply by name. See Figure 12.1. For example, McCabe’s cyclomatic complexity. The meaning of applying the named measure should be generally accepted.

SMM is for the exchange of measurement results. To convey such results for well known measures, it suffices to identify the measure solely by name.

SuperClass

DimensionalMeasure

Associations

name: String	Specifies the name of the SMM element. This attribute is inherited from the SmmElement class where it is optional. Here it is required.

Constraints

```
context NamedMeasure inv:

not self.name->isEmpty
```

12.3 RescaledMeasure Class

The measure specifies a process that re-scales a measurement on an entity with one unit of measure to obtain a second measurement of the same entity with an different unit of measure. See Figure 12.1.

SuperClass

DimensionalMeasure

Attributes

formula:String	Specifies the algebraic formula that re-scales a result from the base measure's dimension to obtain a value expressed in a different unit of measure with respect to this measure's unit of measure

Associations

baseMeasure:DimensionalMeasure	Identifies the measure applied to each "contained" entity to determine base measurements.
rescaleFrom:RescaledMeasureRelationship[0..*]	Specifies the relationship instance that defines the measure rescaled by this rescaled measure.

12.4 RescaledMeasureRelationship Class

RescaledMeasureRelationship is a class representing relationship of measure rescaling between a rescaled measure and a dimensional measure.

SuperClass

MeasureRelationship

Associations

from: DimensionalMeasure[1]	Specifies the dimensional measure at the from endpoint of the relationship.
to: RescaledMeasure [1]	Specifies the rescaled measure at the to-endpoint of the relationship.

13 Measurements

13.1 General

Measurement results are values from ordered sets. Such a set may be nominal (e.g., Poor, Fair, Good, Excellent) as long as there is an underlying order. A set may instead define a dimension where its values may be stated in orders of magnitude with respect to archetypal member. SMM allows for dimensional measurements. The magnitude is the measure's unit of measure.

SMM also allows for dimensionless measurements derived by ratios and ranking schemes. In the former the ratio is derived from two measurements of the same dimension; whereas, in the latter measurements from a dimension are mapped to symbolic representations (e.g., 100-90 becomes "A", 89-80 becomes "B").

The modeling of measurements mirrors the modeling of measure.

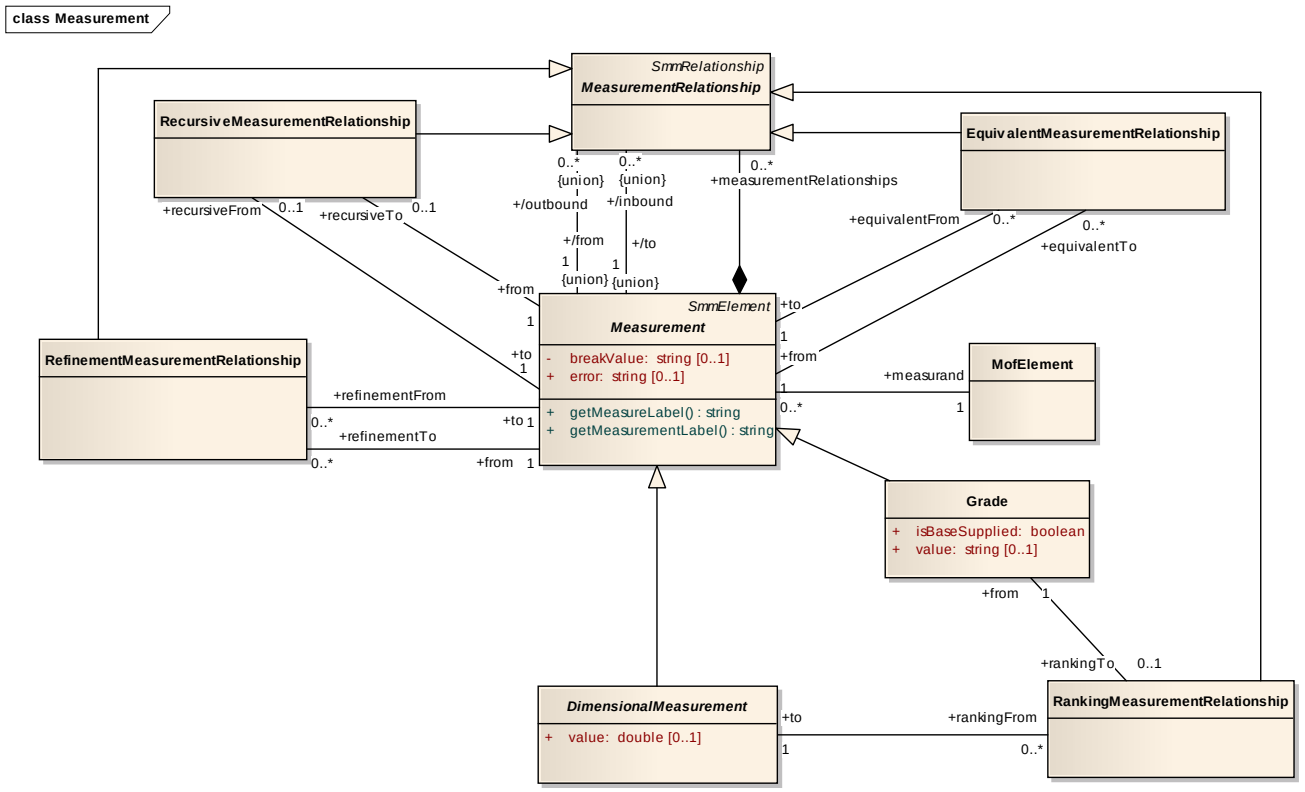

Figure 13.1 - Measurements

13.2 Measurement Class (abstract)

The Measurement class represents the results of applying the associated Measure to the associated Measurand. See Figure 13.1. Two measurements of the same measurand by the same measure can be distinguished by observation information provided by the associated Observation. Measurand is in the scope of the measure.

The value of a measurement is an element of an ordered set. It may be a number where the ordering is the usual standard. The DimensionalMeasurement and Percentage subclasses of Measurement defined below have numeric values. The value may also be a symbol that we can map to a numeric interval. The Grade subclass has a symbolic value.

Measure is a process and, hence, may fail. The error attribute of measurement allows such failures to be noted. A measurement either has a value or an error is recorded.

SuperClass

SmmElement

Attributes

error:String[0..1]	If an error occurred in the measurement process, this field contains a code representing the error
breakValue:String[0:1]	If the scope specifies a break condition, this field contains the instance value associated with the break condition.

Associations

measurand:MOF::Element[1]	Identifies the object measured.
equivalentFrom:EquivalentMeasurementRelationship[0..*]	Specifies the relationship instance that defines the equivalency of this measurement.
equivalentTo: EquivalentMeasurementRelationship[0..*]	Specifies the relationship instance that defines the equivalency of this measurement.
refinementFrom:RefinementMeasurementRelationship[0..*]	Specifies the relationship instance that defines the refinement of this measurement.
refinementTo:RefinementMeasurementRelationship[0..*]	Specifies the relationship instance that defines the refinement of this measurement.
recursiveFrom:RecursiveMeasurementRelationship[0..*]	Specifies the relationship instance that defines the recursivity of this measurement.
recursiveTo:RecursiveMeasurementRelationship[0..*]	Specifies the relationship instance that defines the recursivity of this measurement
inbound:MeasurementRelationship[0..*]	The set of relationship such that the current Measurement is the to-endpoint of these relations. This property is a derived union.
outbound:MeasurementRelationship[0..*]	The set of relationship such that the current Measurement is the to-endpoint of these relations. This property is a derived union.
measurementRelationships:MeasurementRelationship[0..*]	The set of all MeasurementRelationship owned by the measure.

Operations

getMeasureLabel:String[1]	This operation returns the label describing the measure of this measurement according to the rule specified in measureLabelFormat in the Measure class.
getMeasurementLabel:String[1]	This operation returns the label describing this measurement and measurand according to the rule specified in measurementLabelFormat in the Measure class.

Constraints

context Measurement inv:
scope.breakCondition->isEmpty == breakValue->isEmpty

Semantics

Measurand must be in the scope of measure. Specifically, measurand must be an instance of the class named in measure. scope.class. If measure. scope.recognizers is given then the recognizer applied to the measurand must return true.

13.3 MeasurementRelationship Class (abstract)

MeasurementRelationship is an abstract class representing any relationship between two measurements. See .

SuperClass

SmmRelationship

13.4 EquivalentMeasurementRelationship

EquivalentMeasurementRelationship is a class representing any relationship of equivalency between two measurements.

SuperClass

MeasurementRelationship

Associations

from:Measurement [1]	Specifies the equivalent measurement at the from endpoint of the relationship.
to:Measurement[1]	Specifies the equivalent measurement at the to-endpoint of the relationship.

13.5 RefinementMeasurementRelationship Class

Refinement MeasurementRelationship is a class representing any relationship of refinement between two measurements.

SuperClass

MeasurementRelationship

Associations

from:Measurement [1]	Specifies the measurement at the from endpoint of the relationship.
to:Measurement[1]	Specifies the measurement at the to-endpoint of the relationship.

13.6 RecursiveMeasurementRelationship Class

RecursiveMeasurementRelationship is a class representing any relationship of recursivity on a measurement upon itself.

SuperClass

MeasurementRelationship

Associations

from:Measurement[1]	Specifies the measurement at the from endpoint of the relationship.
to:Measurement[1]	Specifies the measurement at the to-endpoint of the relationship.

13.7 DimensionalMeasurement Class

The DimensionalMeasurement class represents the results of applying a dimensional measure to an entity. The result is given in terms of the measure's unit. See Figure 13.1.

SuperClass

Measurement

Attributes

value:Number[0..1]	Represents the measurement result as a magnitude with respect to the unit of measure.

Associations

rankingFrom:RankingMeasurementRelationship[0..*]	Specifies the relationship instance that defines the rankings for this measurement.
baseMeasurementFrom:BaseMeasurementRelationship[0..*]	Specifies the relationship instance that defines the accumulation for this measurement.
baseMeasurement1From:Base1MeasurementRelationship[0..*]	Specifies the relationship instance that defines the 1st part of the binary comparator for this measurement.
baseMeasurement2From:Base2MeasurementRelationship[0..*]	Specifies the relationship instance that defines the 2nd part of the binary comparator for this measurement.
rescaleTo:RescaledMeasurementRelationship[0..*]	Specifies the relationship instance that defines the measurement rescaling this measurement.

Constraints

```
context DimensionalMeasurement inv:

measure.oclIsTypeOf(DimensionalMeasure) and

error->isEmpty <> value->isEmpty
```

13.8 Grade Class

The Grade class represents the grade found by Ranking measure. Its ranking scheme mapped the grade's underlying base measurement to the grade's symbol. Once again, the base measurements share its measurand with this derived grading. See Figure 13.1.

Super Class

Measurement

Attributes

value: String[0..1]	Identifies rank as a measurement derived from the base measurement.
isBaseSupplied:Boolean	True if baseMeasurement is supplied.

Associations

rankingTo:RankingMeasurementRelationship[0..1]	Specifies the relationship instance that defines the measurement graded by this grade.

Constraints

```
context Grade inv:
measure.oclIsTypeOf(Ranking) and
error->isEmpty <> value->isEmpty and
isBaseSupplied →(measurand = baseMeasurement.measurand and baseMeasurement.measure =
measure.baseMeasure)
```

Semantics

If isBaseSupplied holds, then value is one of the symbols found by measure.interval where baseMeasurement.value is in the interval. A numeric value is in the interval if and only if the it is less than the maximumEndPoint when maximumOpen is false, less than or equal to maximumEndPoint when maximumOpen is true, greater than minimumEndPoint when minimumOpen is false, and greater than or equal to minimumEndPoint when minimumOpen is true.

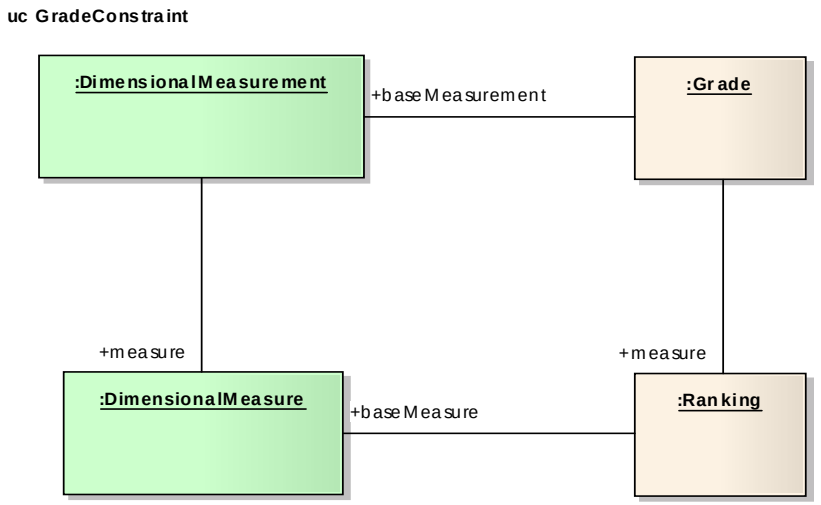

Figure 13.2 - Grade Constraint

13.9 RankingMeasurementRelationship Class

RankingMeasurementRelationship is a class representing any relationship of grading between a grade measurement and a dimensional measurement.

SuperClass

MeasurementRelationship

Associations

from:Grade [1]	Specifies the grade measurement at the from endpoint of the relationship.
to:DimensionalMeasurement[1]	Specifies the dimensional measurement at the to-endpoint of the relationship.

14 Collective Measurements

14.1 General

This class represents measurements found by accumulating a set of base measurements. For example, the number lines of code in application can be determines by accumulating the number lines in its programs.

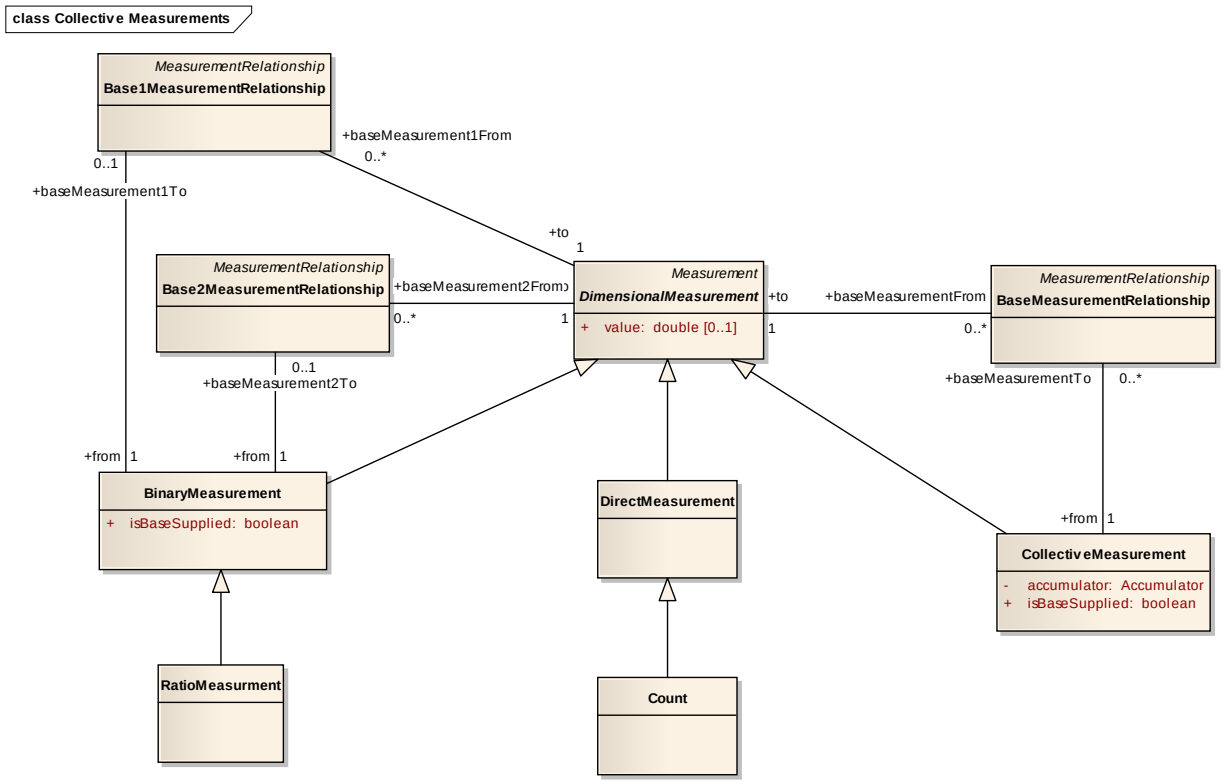

Figure 14.1 - Collective Measurements

14.2 CollectiveMeasurement Class

The CollectiveMeasurement class represents the results of applying its CollectiveMeasure measure to an entity. See Figure 14-1. In this case, applying the measure is as follows:

1. Apply the base measure to each contained element to obtain a set of base measurements.
2. Apply the n-ary accumulator to the set of base measurements to obtain the measurement of the container.

The results of step 1 are the DimensionalMeasurements associated by base measurement.

SuperClass

DimensionalMeasurement

Attributes

isBaseSupplied:Boolean	True if baseMeasurements are supplied. All are supplied or none is assumed.
accumulator: Accumulator	Enumerated value indicating the type collective measure.

Associations

baseMeasurement:DimensionalMeasurement[0..*]	Identifies the measurements from which this collective measurement was derived.

Constraints

```
context CollectiveMeasurement inv:
measure.oclIsTypeOf(CollectiveMeasure) and
```
isBaseSupplied →
```
(not baseMeasurement->isEmpty and baseMeasurement.measure=measure.baseMeasure)
```

Semantics

If isBaseSupplied holds, then value equals the result of applying measure.accumulator the set of values given by baseMeasurement.value.

14.3 DirectMeasurement Class

The DirectMeasurement class represents the measurement results found by of applying the measure's specified operation directly to the measurand. See Figure 14.1.

SuperClass

DimensionalMeasurement

Constraints

```
context DirectMeasurement inv:
measure.oclIsTypeOf (DirectMeasure)
```

14.4 Count Class

Counting forms the basis for multiple metrics. This class consists of a particular subclass of directMeasurement that is very useful in counting. See Figure 14.1. Its associated measure is a CountingMeasure where the specified operation is a recognizer operation. Therefore, the value of any instance of this class is 1 or 0 depending upon whether or not the measurand is recognized.

SuperClass

DirectMeasurement

Constraints

```
context Count inv:
measure.oclIsTypeOf (CountingMeasure)
```

14.5 BinaryMeasurement Class

SuperClass

DimensionalMeasurement

Attributes

isBaseSupplied:Boolean	True if both base measurements are supplied.

Associations

baseMeasurement1:DimensionalMeasurement[0..1]	Identifies the first base measurement.
baseMeasurement2:DimensionalMeasurement[0..1]	Identifies the second measurement.

Constraints

```
 context RatioMeasurement inv:
measure.oclIsTypeOf(BinaryMeasure) and
isBaseSupplied →
(not baseMeasurement1.isEmpty and not baseMeasurement2.isEmpty) and
not baseMeasurement1.isEmpty →
(baseMeasurement1.measure = measure. baseMeasurement1) and
not baseMeasurement2.isEmpty →
(baseMeasurement2.measure = measure. baseMeasure2)
```

Semantics

If isBaseSupplied holds, then value equals the result of applying measure.functor to baseMeasurement1.value and baseMeasurement2.value.

14.6 RatioMeasurement Class

The RatioMeasurement class affords evaluations of a ratio measure of two evaluations of different dimensional measures. See Figure 14.1. The measure associated with the dividend has its unit of measure in common with the measure associated with the divisor.

SuperClass

BinaryMeasurement

Constraints

```
context RatioMeasurement inv:
measure.oclIsTypeOf(RatioMeasure) and
isBaseSupplied → (value = baseMeasurement1.value / baseMeasurement2.value)
```

14.7 BaseMeasurementRelationship Class

BaseMeasurementRelationship is a class representing relationship of hierarchy between a collective measurement and a dimensional measurement.

SuperClass

MeasurementRelationship

Associations

from:CollectiveMeasurement[1]	Specifies the collective measurement at the from endpoint of the relationship.
to: DimensionalMeasurement [1]	Specifies the dimensional measurement at the to-endpoint of the relationship.

14.8 Base1MeasurementRelationship Class

Base1MeasurementRelationship is a class representing relationship of hierarchy between a binary measurement and a dimensional measurement.

SuperClass

MeasurementRelationship

Associations

from:BinaryMeasurement[1]	Specifies the binary measurement at the from endpoint of the relationship.
to: DimensionalMeasurement [1]	Specifies the dimensional measurement at the to-endpoint of the relationship.

14.9 Base2MeasurementRelationship Class

Base2MeasurementRelationship is a class representing relationship of hierarchy between a binary measurement and a dimensional measurement.

SuperClass

MeasurementRelationship

Associations

from:BinaryMeasurement[1]	Specifies the binary measurement at the from endpoint of the relationship.
to: DimensionalMeasurement [1]	Specifies the dimensional measurement at the to-endpoint of the relationship.

15 Named and Rescaled Measurements

15.1 General

Measurement is in terms of its unit of measure as specified under its associated DimensionalMeasure. That is, the measurement is a multiple of its unit of measure where value determines the multiple.

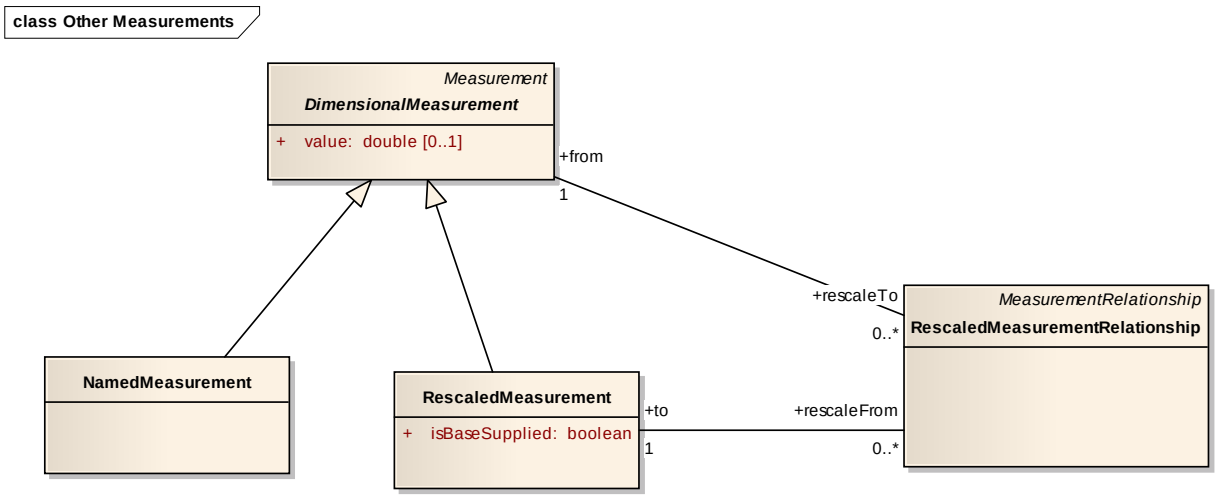

Figure 15.1 - Named and Rescaled Measurements

15.2 NamedMeasurement Class

The NamedMeasurement class represents the measurement results of applying to the Measurand measurement processes that are generally known and identifiable by name. See Figure 15.1.

SuperClass

DimensionalMeasure

Constraints

```
context NamedMeasurement inv:
measure.oclIsTypeOf(NamedMeasure).
```

15.3 RescaledMeasurement Class

The RescaledMeasurement class represents the measurement results of applying to the base measurement the operation specified by the Measure to rescale the measurement. That is, given a one measurement of the measurand with respect to one unit of measure, we obtain a second measurement of the measurand with respect to a different unit of measure. See Figure 15.1.

Measure is a RescaledMeasure.

SuperClass

DimensionalMeasure

Attributes

isBaseSupplied:Boolean	True if the base measurement is supplied.

Associations

rescaleFrom:RescaledMeasurementRelationship[0..*]	Specifies the relationship instance that defines the measurement rescaled by this rescaled measurement.

Constraints

```
context RescaledMeasurement inv:
measure.oclIsTypeOf(RescaledMeasure) and
```
isBaseSupplied →
```
not baseMeasurement->isEmpty and baseMeasurement.measure = measure.baseMeasure
```

Semantics

If isBaseSupplied is true then value equals result of applying measure.operation to the baseMeasurements' values.

15.4 RescaledMeasurementRelationship Class

RescaledMeasurementRelationship is a class representing relationship of measurement rescaling between a rescaled measurement and a dimensional measurement.

SuperClass

MeasurementRelationship

Associations

from: DimensionalMeasurement [1]	Specifies the dimensional measurement at the from endpoint of the relationship.
to:RescaledMeasurement [1]	Specifies the rescaled measurement at the to-endpoint of the relationship.

16 Observations

16.1 General

Measurements are sometimes repeated. An old carpentry rule is measure twice, cut once.

To distinguish these multiple measurements, the observation and scope class can represent contextual information such as the time of the measurement and the identification of the measurement tool and the artifacts that are under measurement.

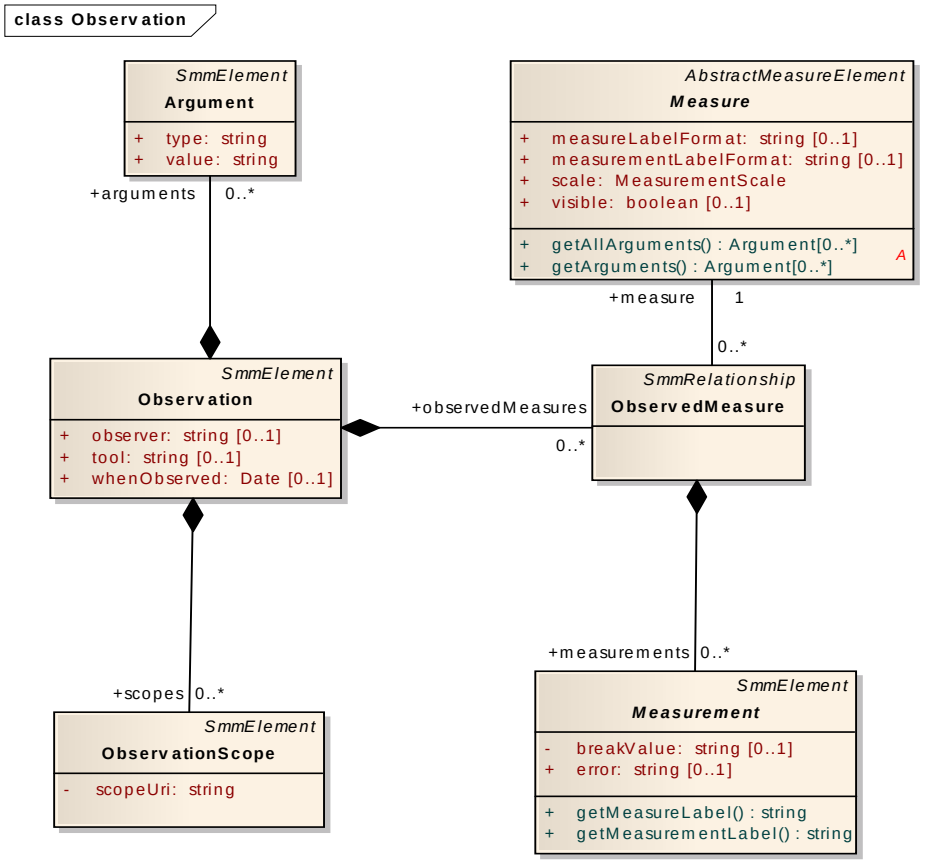

Figure 16.1 - Observations

16.2 Observation Class

This class represents some of the contextual information which may be unique to this measurement such as date, measurer and tool used. See Figure 16.1.

SuperClass

SmmElement

Attributes

whenObserved:date[0..1]	Identifies the “moment” when the measurement was taken.
observer:String[0..1]	Identifies measurer
tool:String[0..1]	Identifies tool used in measurement.

Associations

observedMeasures:ObservedMeasure[0..*]	The set of all ObservedMeasure owned by the observation.
requestedMeasures:SmmElement[0..*]	Specifies the measures or their category that are part of the observation request. This association is optional and can be used by a builder to know what to include in a specific observation.
scopes:ObservationScope[0..*]	Specifies the scopes of the observation, i.e., the models or model portions that are the subject of the Observation

Constraints

```
context Observation inv:
requestedMeasures.oclIsTypeOf(MeasureCategory) or
requestedMeasures.oclIsTypeOf(CategoryRelationship) or
requestedMeasures.oclIsTypeOf(Measure)
```

16.3 ObservationScope Class

This class represents the model(s) or sub model that are the subject of the related observation. This information can be used initially by builders to understand which model to gather measurements from, later by anyone wishing to recreate a new observation of the same artifacts. See Figure 16.1.

SuperClass

SmmElement

Attributes

scopeUri:String[1]	Uri that identifies model(s) or model fragment.

Semantics

The scopeUri represents specific schemes following the *RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax*. As a hierarchical URI, the scopeUri supports all features associated with such URI, including both absolute and relative addressing. The starting point for the resolution of relative addressing should match generally accepted rules, but this specification doesn’t dictate any such details.

To quote the URI syntax:

At the highest level a URI reference (hereinafter simply “URI”) in string form has the syntax

[scheme:]scheme-specific-part[#fragment]

The scopeUri should inherently accept and understand the following 2 schemes: *mof* and *ecore*, respectively representing models expressed as MOF and Ecore (Eclipse EMF model variant of MOF).

Our scheme-specific-part complies with the definition of hierarchical URI and as such it has the following syntax:

[//authority][path][?query]

The general form of a scope uri is then:

mof://kdm.example.com/projectName/kdmName	Uri for a specific MOF KDM model.
ecore://astm.example.com/pathToWherever/longPath/modelName	Uri for a specific Ecore ASTM model

A more advanced form of the URI for our schemes is made to support the query part of the URI in order to specify portion of models and also to specify models in paths that represent folders or collections.

The query part of the scopeUri follows the general form of key=value separated by ampersand (&). The following keys are defined by our schemes:

Model	Regex based pattern representing the name of model or models that should be matched in the path.
Recursive	True if the search for models matching the model pattern should also recursively descend the hierarchical path structure rooted at the path specified in the URI. Default is false.
queryType	Type of query to use in select. “OCL” (default) or “XQuery.”
Select	Query into selected model(s) that represent a selection of a subset or portion of the entire model that will be used as the scope of performing measurements. For example this could represent a segment in a KDM that is related to a specific application.

The general form of a scope uri is then:

mof://kdm.example.com/projectName?model=a?rt*&recursive=true	Uri for all MOF models with name matching a?rt* located in projectName or under.
ecore://kdm.example.com/path/ ?queryType=Xquery&select=/Segment[@name=”default”]/ Segment[@name=”myApp”]	Uri for a specific Ecore KDM model segment representing a particular application segment.

16.4 ObservedMeasure Class

This class represents association between observations and the measures that make up such observation. This class also serves to hold the list of measurements characterized by the related measure that are part of a given observation.

SuperClass

SmmRelationship

Associations

Measurements:Measurement[0..*]	The set of all Measurement owned by the observed measure.
measure:Measure[1]	The measure that is being observed.

16.5 Argument Class

This class represents some of the variable arguments or parameters that are being passed to the measures that have Operations that make use of replaceable parameters.

SuperClass

SmmElement

Attributes

name: String[1..1]	Specifies the name of the argument. (inherited)
type:String[1..1]	Specifies the type of the argument. See semantic section for detailed information.
value:String[1..1]	The value of the argument, expressed in a “typesafe” fashion.

Associations

None

Semantics

The type attribute represents the type of the argument being passed. The accepted types are the basic types that are defined in OCL, as this is the main operation language supported. Those types are, as defined in section 7.1 of the OCL 2.1 specification: Boolean, Integer, Real, and String.

The above supported types are very limited. For example there is no direct support for Date or DateTime. The implementation of additional types is left to the implementers. As a suggestion (not normative), implementers should try to use OCLOperation helper functions in order to facilitate hiding the implementation and make their implementation shareable and portable.

For all accepted types, the value attribute is a String whose content directly matches what is expected by the Operation language, so that it can be transferred verbatim into the Operation body during the parameter replacement. Implementer specific types can define their own value format if needed.

17 Historic and Trend Data (Non-normative)

17.1 General

SMM does not model tracking or trend data directly. Linking versions of objects through a software evolution poses a concern in modeling software evolution even if measures are never taken. When the measurand's model provides the linkage (e.g., an "EvolvesTo" relationship), then a measurement of an original artifact could be traced to its newer versions and to their measurements if available. Figure 17.1 is overly simplistic, but hopefully conveys the gist of such tracing. The beige filled instances indicate the metric representations augmenting the base model (green). The central point is that the evolves path is between instances of the base model. The measures of the evolving artifacts can be gathered or compared only if the linkage between the artifacts is captured and maintained through the modeling of the system development and modification.

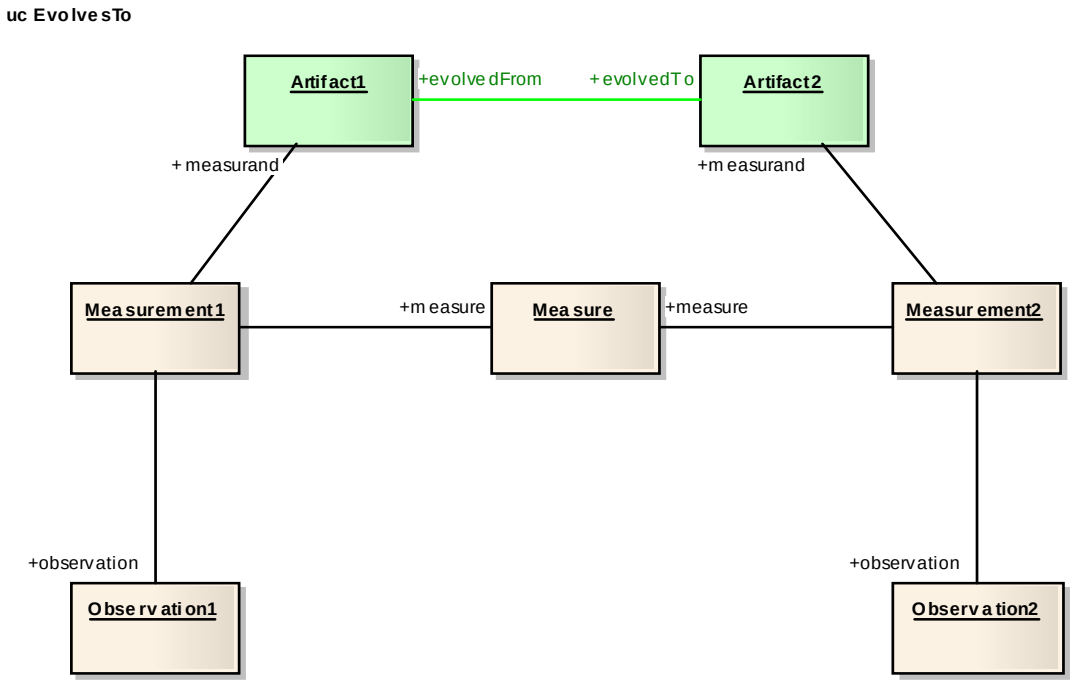

Figure 17.1 - Tracking Measurements across Versions

18 Inaccuracy (Non-normative)

18.1 General

Inaccuracy of a measurement is the amount by which the measurement is in error. That is, we may model inaccuracy as measure if we first model a measure which is assumed to be true. Inaccuracy of a measurement is then just the difference between the measurement and a “true” measurement of the same entity.

In SMM inaccuracy is representable by measures that characterize inaccuracy. The measures are comparable elevation of measurements evaluated by the difference between the measurement and the truest (at least accepted as such) measurement of that entity for that trait.

Given two measures which characterize the same trait and share the same scope, then inaccuracy can be modeled as a binary measure expressing the difference taken over the two measures.

In the demonstration below (), a category collects measures that are applicable to ExampleClass1 and characterize ExampleTrait. The category identifies the “truest” measure by the goldStandard relationship and identifies an appropriate inaccuracy measure for Measure1 by the InaccuracyMeasure relationship.

A Characteristic may have a measure that is designated as the best or truest measure of the attribute. That measure may be associated as the attribute’s gold standard. Such a designation allows for the representation of inaccuracy for each of the attribute’s measures as the difference between the measure and the gold standard.

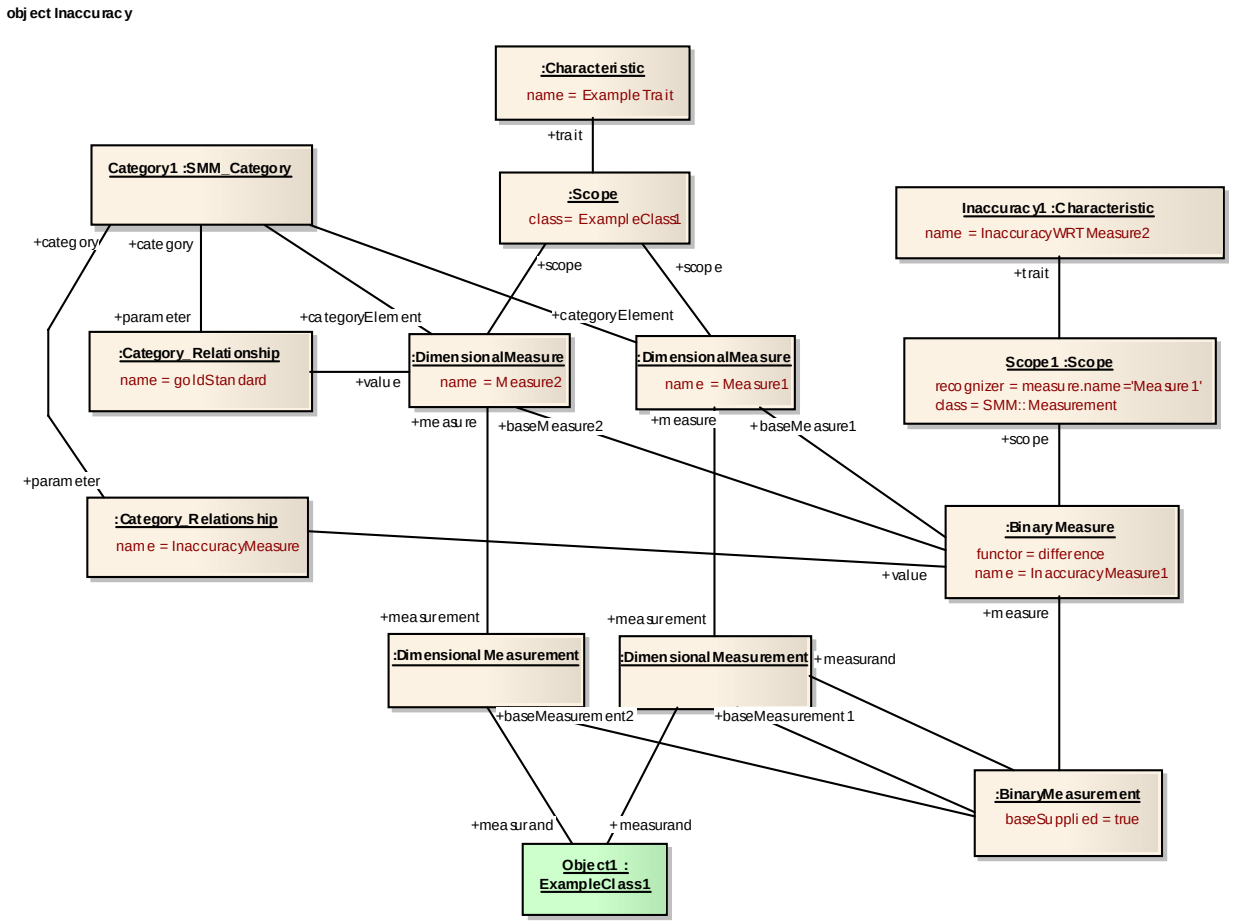

Figure 18.1 - Inaccuracy Demonstration

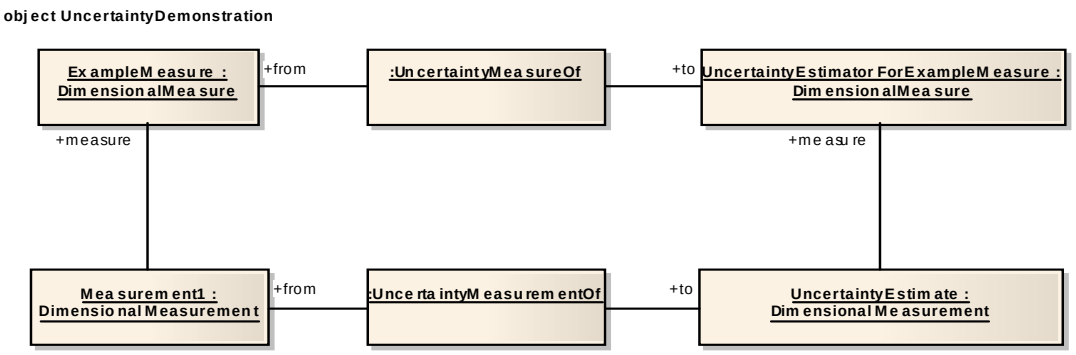

Figure 18.2 - Uncertainty Demonstration

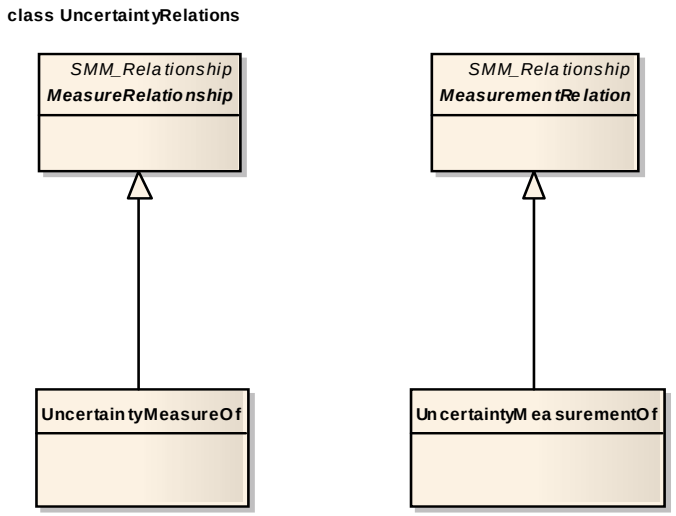

Figure 18.3 - SMM Extension for Uncertainty

19 Library of Measures (Non-normative)

19.1 General

The following is a suggestive list of measurement classes along with their measure classes and measurand classes. Sources include:

- Comsys Systems Redevelopment Methodology: www.comsysprojects.com/SystemTransformation/TMethodology.htm
- “A Survey of Software Metrics” by F. Riguzzi, DEIS Technical Report no. DEIS-LIA-96-010, July 1996, Università degli Studi di Bologna.

Each measure is defined using the classes of the SMM. The referenced software artifacts are modeled using the Knowledge Discovery Metamodel (KDM) unless otherwise noted.

19.2 Various Counts

19.2.1 Module Count[1]

Module Count ≡ A count of the number of modules in a system.

Assume that the system is modeled by a KDM model. The KDM:AbstractCodeElement serves as a container of code parts as well as modeling the code parts themselves. The KDM:Module is an AbstractCodeElement subclass that models modules. See Figure 19.1.

Counting the modules in the code model requires summing the results of a recognizer for module across the model. The unit of measure is module. See Figure 19.2 for the library entry and see Figure ??? for a brief description.

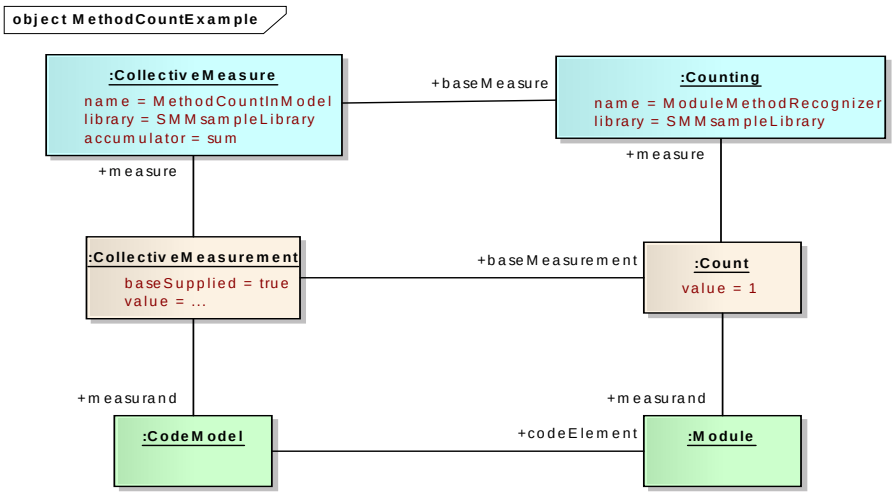

Figure 19.1 -

1. See GAM 003 in Comsys Systems Redevelopment Methodology

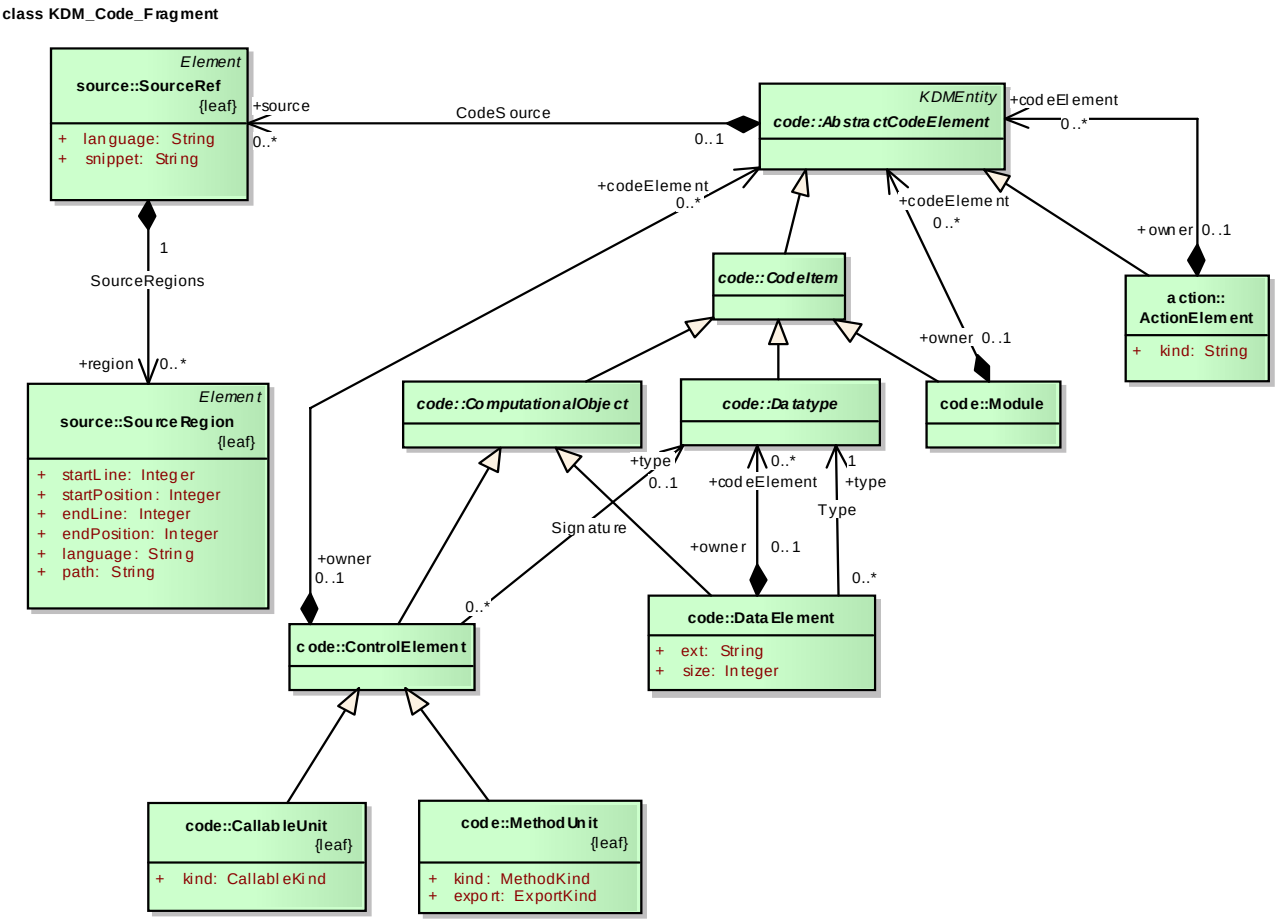

Figure 19.2 - KDM Code Package Fragment

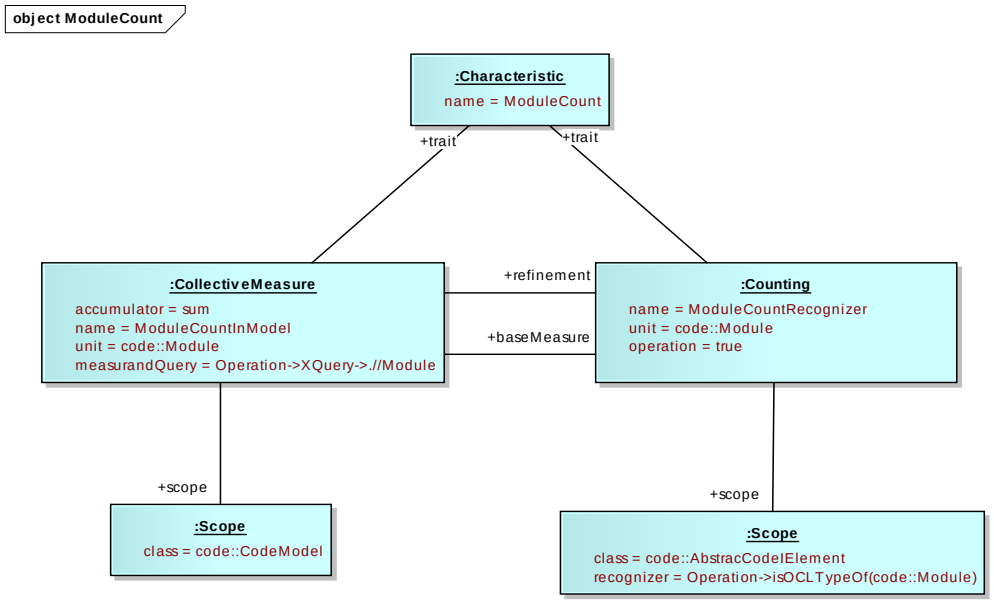

Figure 19.3 - Library Entry for Module Count in Code Model

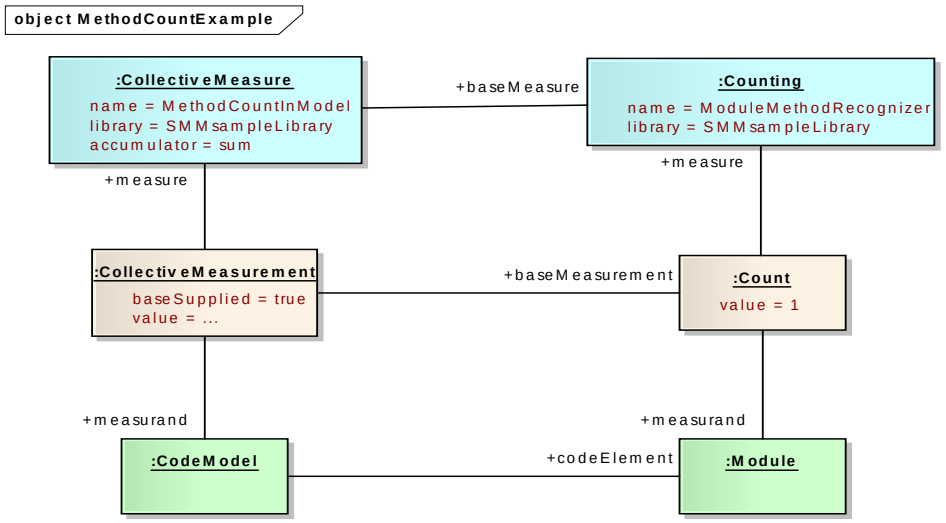

Figure 19.4 - Module Count in Model Demonstration

For an entire system, we identify each CodeModel instance in the KDM (or a specific subset depending on the ObservationScope). Then for each code::CodeModel, its baseMeasure elements are identified. In this example the default containment association relation is overridden by a measurand query expressed as the XQuery operation of '..//Module' which states that we want all Module children of our CodeModel recursively. Next we move to apply the scope recognizer, which filters out any elements that are not of class code::Module, which here is just a safety test as the measurand query already

provides this level of filtering. This leaves us with only instances of code::Module, on which we apply a Counting measure with a default operation of true so that it always returns 1.

All of the Counting measurement with a value of 1 representing here the code:Module are then summed up into a Collective measurement for each code::CodeModel according to the accumulator defined in the Collective measure.

Another possible approach would be to move the recognizer to the Counting class instead of the scope as shown in Figure 7.1.

The difference between these two approaches is subtle but very interesting. In the first case, the recognizer is applied to determine if a class instance is in scope or not. In the second approach, the recognizer is used to determine if the counting class will return 0 or 1 for the measurement of the class instance. The 1st approach would normally be preferred as it avoids creating measurements with a value of 0 for any non-matching class instance, whereas the second approach will have measurement for every AbstractCodeElement in the CodeModel. Obviously, the sum applied by the collective measure will produce the same final result.

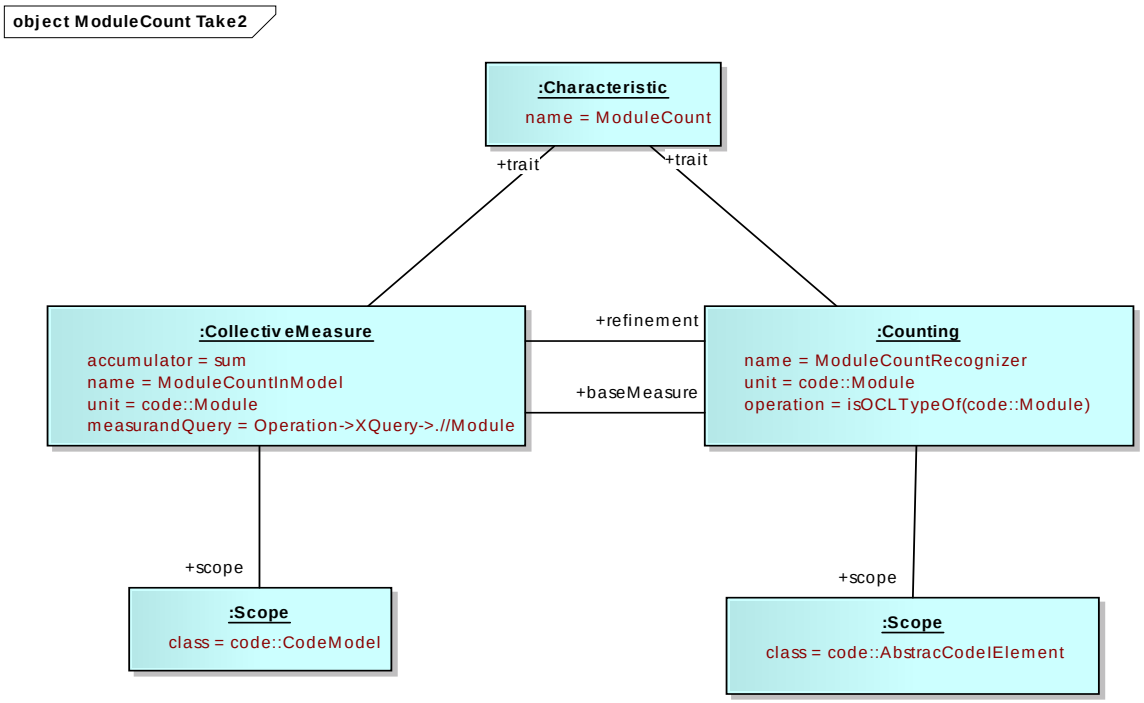

Figure 19.5 - Module Count in Model (take 2)

19.2.2 Screen Count[1]

Screen Count ≡ A count of the number of screens in a system.

1. See TEM 153 in Comsys Systems Redevelopment Methodology

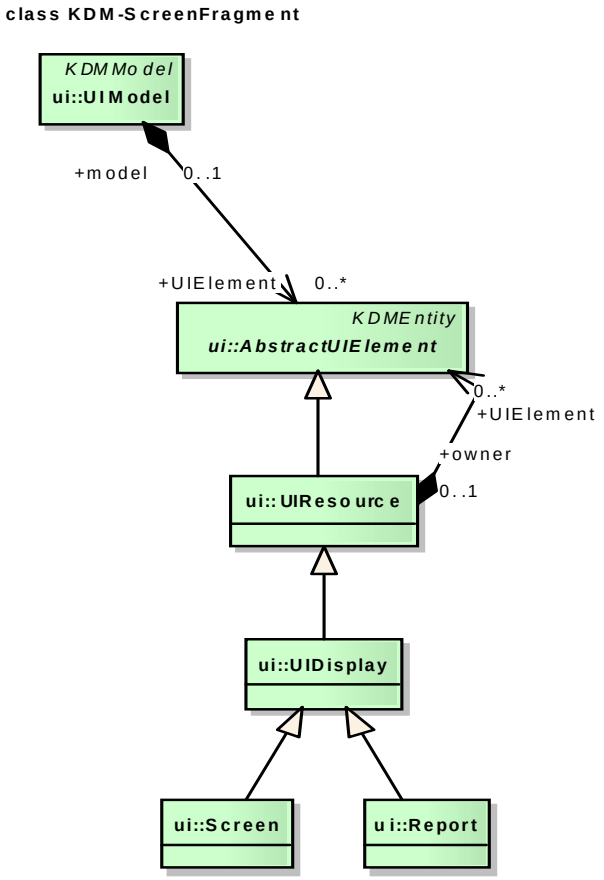

Figure 19.6 - KDM Action Package Fragment

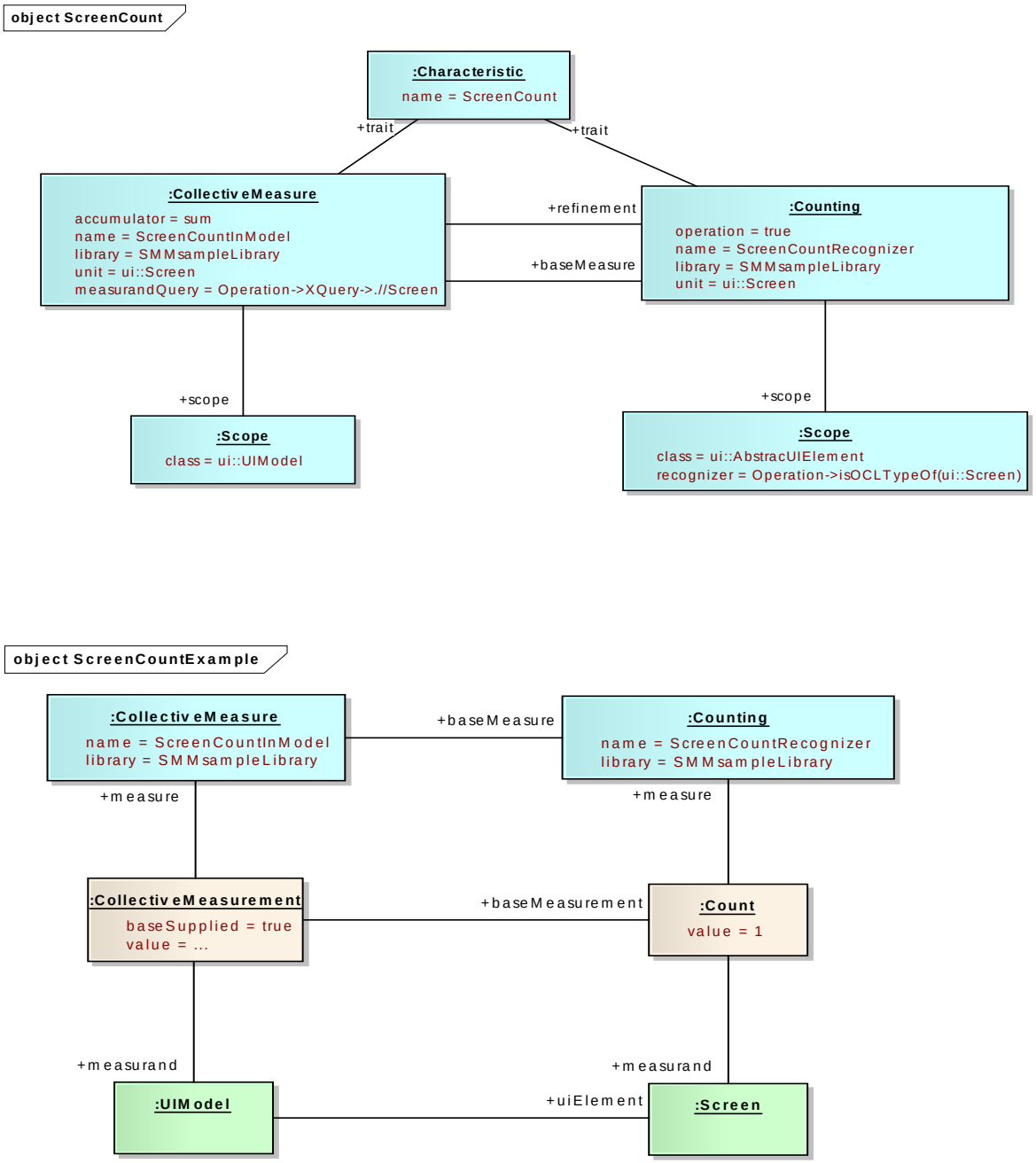

Figure 19.7 - Screen Count Demonstration

Assume that the system is modeled by a KDM model. The KDM:UIElement serves as a container of user interface parts as well as modeling the user interface parts themselves. The KDM:Screen is a UIElement subclass that models screens.

For an entire system, we identify each UIModel instance in the KDM (or a specific subset depending on the ObservationScope). Then for each ui::UIModel, its baseMeasure elements are identified. In this example the default containment association relation is overridden by a measurand query expressed as the XQuery operation of '..//Screen' which states that we want all Screen children of our UIModel recursively. Next we move to apply the scope recognizer, which filters out any elements that are not of class ui::Screen, which here is just a safety test as the measurand query already provides this level of filtering. This leaves us with only instances of ui::Screen, on which we apply a Counting measure with a default operation of true so that it always returns 1.

All of the Counting measurement with a value of 1 representing here the ui::Screen are then summed up into a Collective measurement for each ui::UIModel according to the accumulator defined in the Collective measure.

19.2.3 Method Count

Method Count ≡ A count of the number of methods in a system.

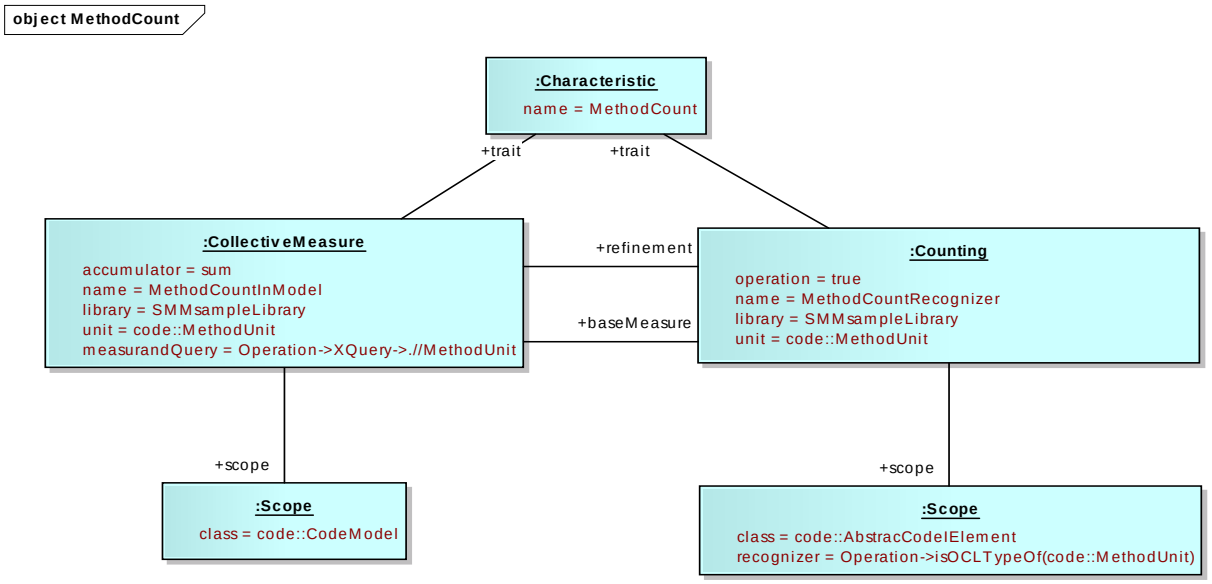

Figure 19.8 - Method Count Library Entry

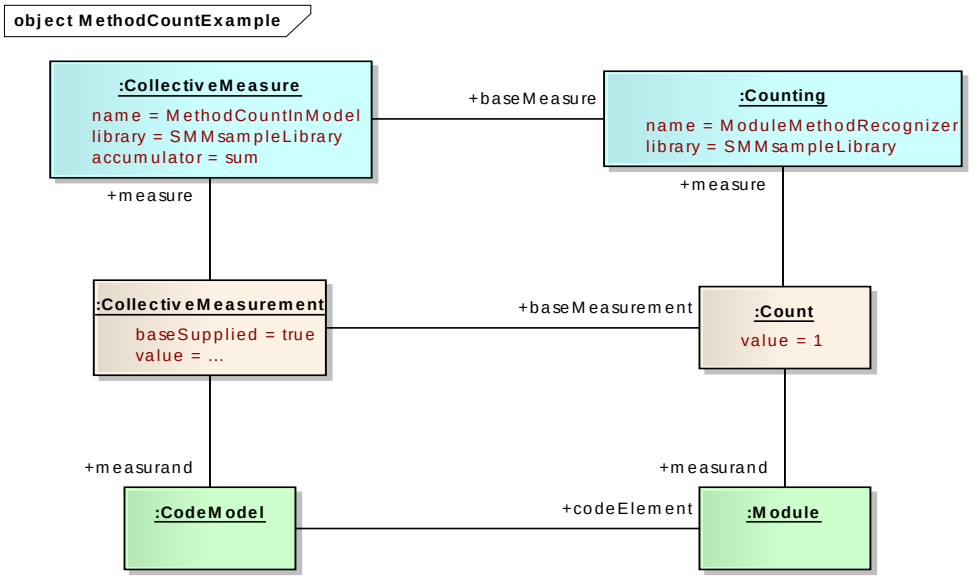

Figure 19.9 - Method Count Demonstration

Assume that the system is modeled by a KDM model. The KDM:MethodUnit is a CodeElement subclass which models methods. The counting of methods then is very similar to the counting of modules given above.

For an entire system, we identify each CodeModel instance in the KDM (or a specific subset depending on the ObservationScope). Then for each code::CodeModel, its baseMeasure elements are identified. In this example the default containment association relation is overridden by a measurand query expressed as the XQuery operation of '..//MethodUnit' which states that we want all MethodUnit children of our CodeModel recursively. Next we move to apply the scope recognizer, which filters out any elements that are not of class code::MethodUnit, which here is just a safety test as the measurand query already provides this level of filtering. This leaves us with only instances of code::MethodUnit, on which we apply a Counting measure with a default operation of true so that it always returns 1.

All of the Counting measurement with a value of 1 representing here the code::MethodUnit are then summed up into a Collective measurement for each code::CodeModel according to the accumulator defined in the Collective measure.

19.2.4 Lines of Code[1]

A line of code is any line of program text that is not a comment or a blank line, regardless of the number of statements or fragments of statements on the line. This specifically includes all lines containing program headers, declarations, and executable and non-executable statements"[2] Lines of code here means fully expanded lines of code including copy books, includes and comments.

KDM does not directly model lines of source, code or otherwise. As a demonstration, let us assume that blank lines may be included. This allows us to use the KDM SourceRegion to measure lines of code. We will further assume source region do not overlap or even having one start on the line that another ends on. The problem here is that code snippets are the smallest pieces of source modeled in KDM. Lines by themselves are not modeled, which means that counting them is indirect. We will sum of the line size of code snippets and call that counting lines of code.

1. See ERP 001 in Comsys Systems Redevelopment Methodology.
2. See S. Conte, H. Dunsmore, V. Shen, *Software Engineering Metrics and Models*, Benjamin/Cummings, Menlo Park, CA.

Lines of SourceRegion and SourceRef

KDM specifies a code snippet with a SourceRegion element that has two attributes, startLine and endLine, that interest us here. The number of lines in the SourceRegion is endLine – StartLine + 1.

Our representation is a DirectMeasure with a class of SourceRegion and a function of endLine – startLine + 1.

SourceRef consists of multiple SourceRegions. Assuming no overlap as stated above, the determination of lines of code in a SourceRef is a sum accumulator CollectiveMeasure with the previous lines of SourceRegion as its base measure.

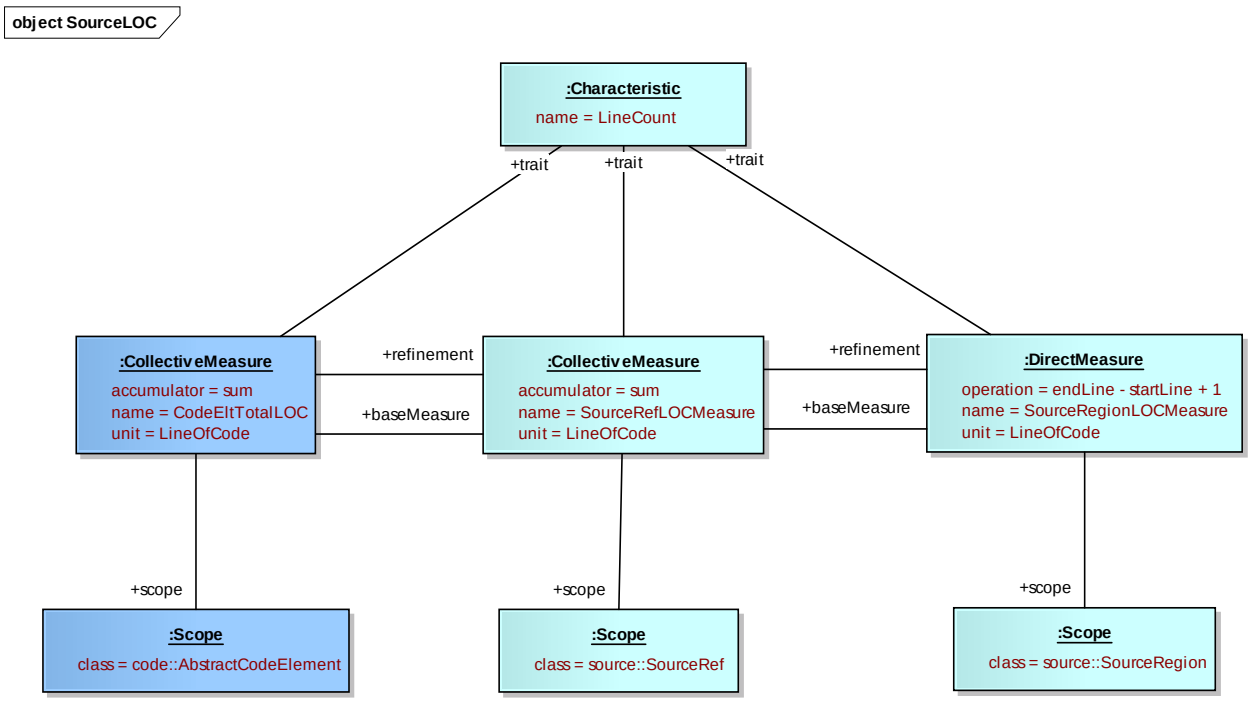

Figure 19.10 - Lines of Code Measures

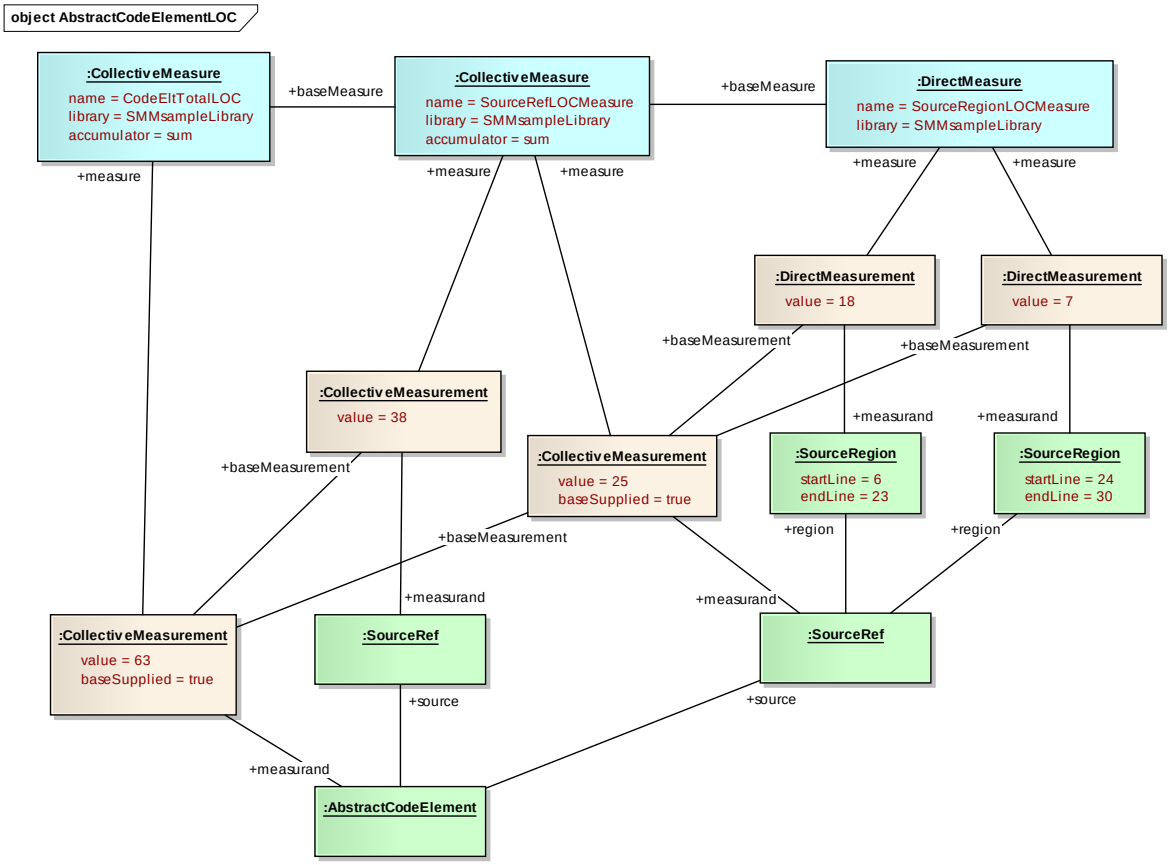

Figure 19.11 - Lines of Code Demonstration

Refinement of Lines of ControlElement, CodeElement and Module

The source role for these elements is SourceRef. Determining the lines of code in each is a sum accumulator CollectiveMeasure where the base measure is the lines of SourceRef given above (the one in darker blue).

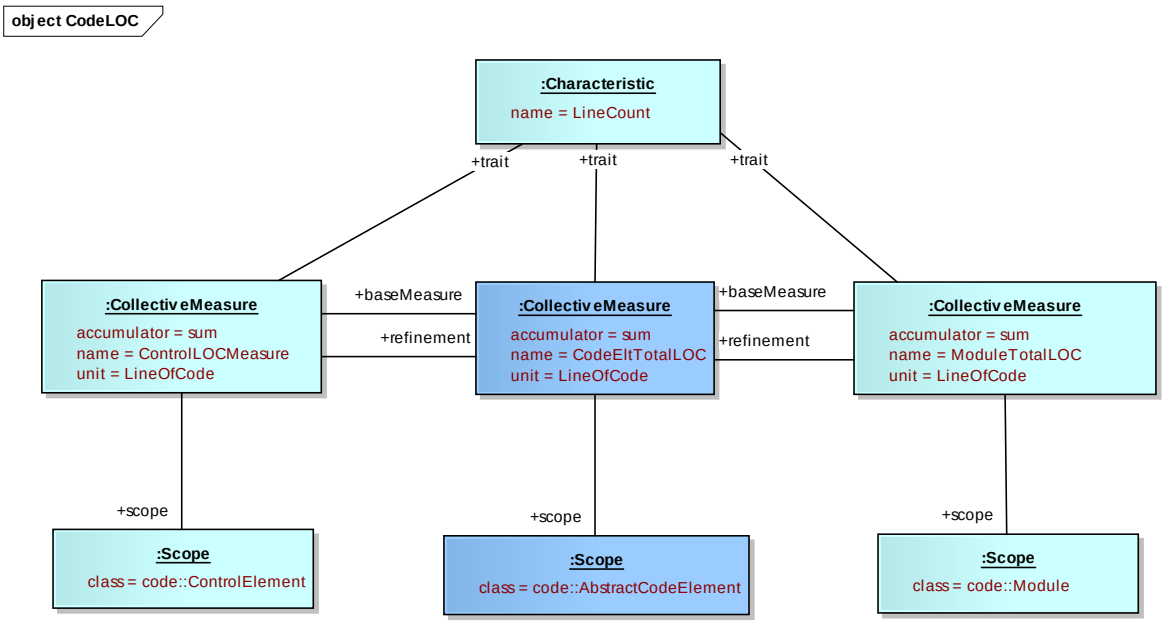

Figure 19.12 - Additional Lines of Code Measures

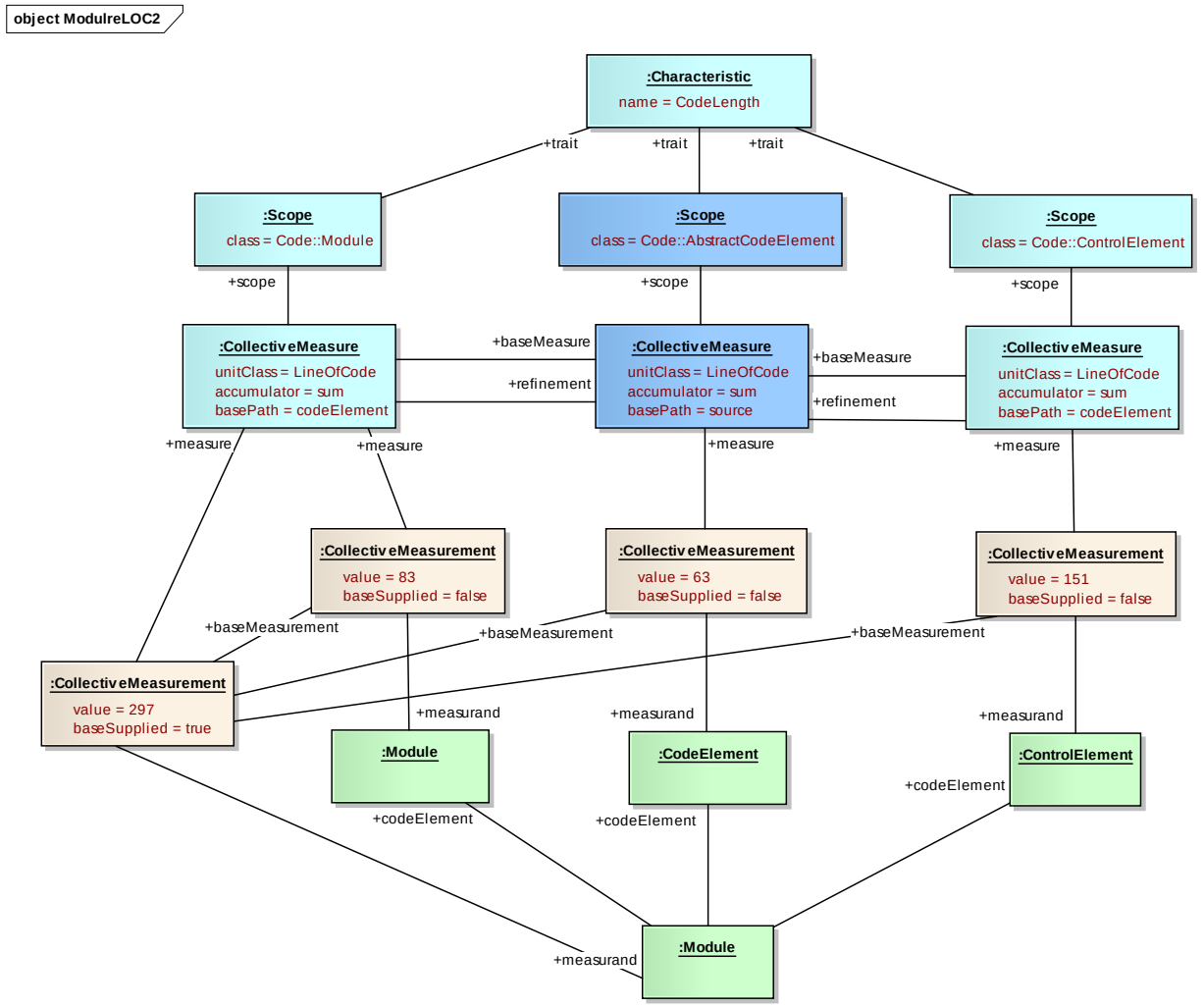

Figure 19.13 - Module and Control Element LOC Demonstration

19.2.5 Lines of Code for ASTM

The Abstract Syntax Tree Metamodel (ASTM) facilitates the interchange of programming language constructs parsed as abstract syntax trees. The Generic Abstract Tree Metamodel establishes a common core for modeling across a wide variety of programming languages. Each of these constructs may, of course, be measured by their lines of code.

GASTM does not directly model lines of source, code, or otherwise. We will, consequently, make the same assumptions we made above for KDM. Blank lines are included and overlaps are ignored.

Figure 19.14 shows a fragment of the proposed ASTM covering the core syntax object, source location and source file. Figure 19.15 shows a possible SMM library entry to represent lines of code measure of GASTM syntax objects.

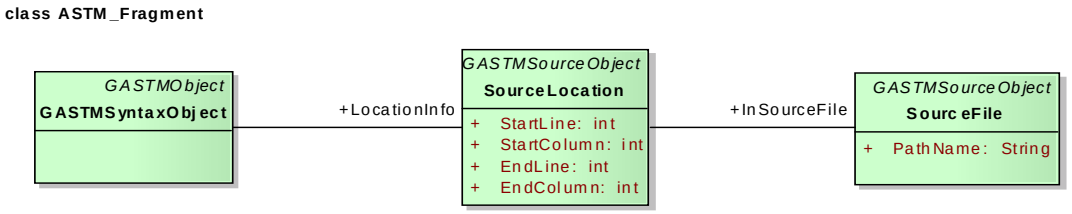

Figure 19.14 - GASTM Fragment

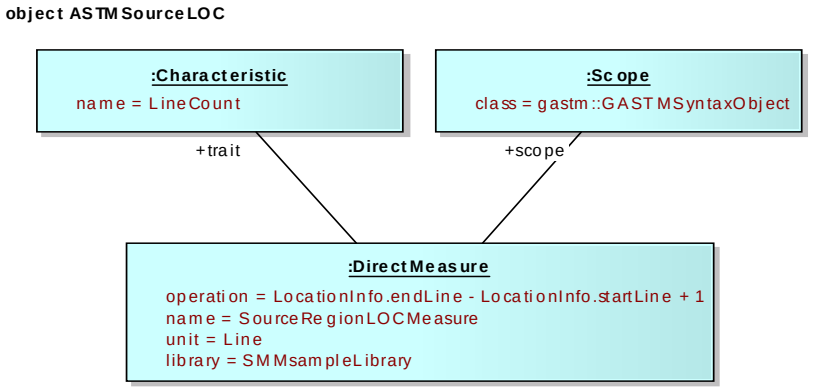

Figure 19.15 - LOC Library Entry for GASTM

19.3 McCabe

McCabe's cycolmatic complexity could be modeled in different ways. It could be a RescaledMeasure from count of independent paths found by adding 2. Another representation would be as aRescaledMeasure from count of branching points found by adding 1. Each of these representations represents equivalent measures. We demonstrate below cyclomatic as a NamedMeasure and as a RescaledMeasure from branching factor.

19.3.1 Branching Factor of ActionElements and Modules

Branching Factor is simply the difference between the number of nodes and edges in a module's control flow graph. KDM models the nodes as ActionElements, the edges as ControlFlow. Branching factor is then measured by subtracting the count of ControlFlow instances from the count of ActionElements.

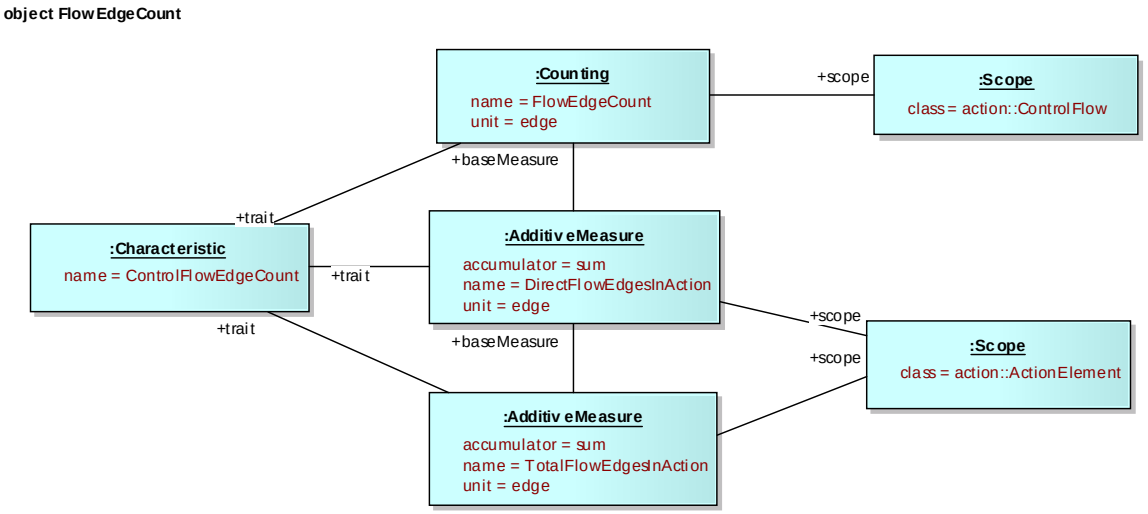

Figure 19.16 - Control Flow Edge Count Library Entry

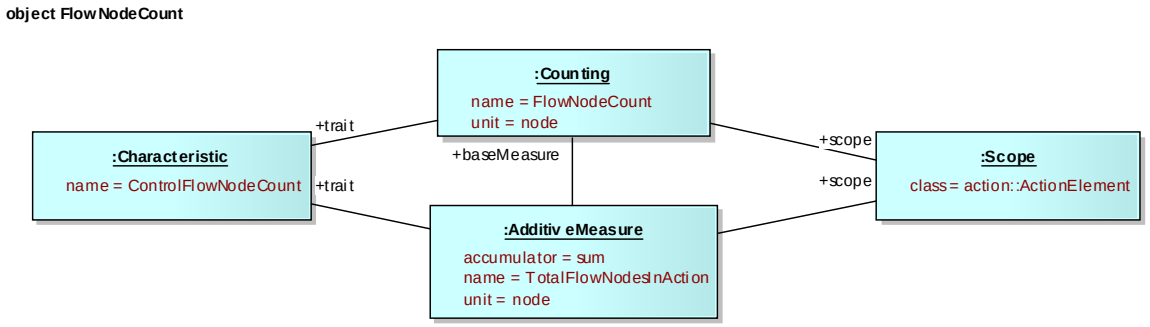

Figure 19.17 - Control Flow Node Count Library Entry

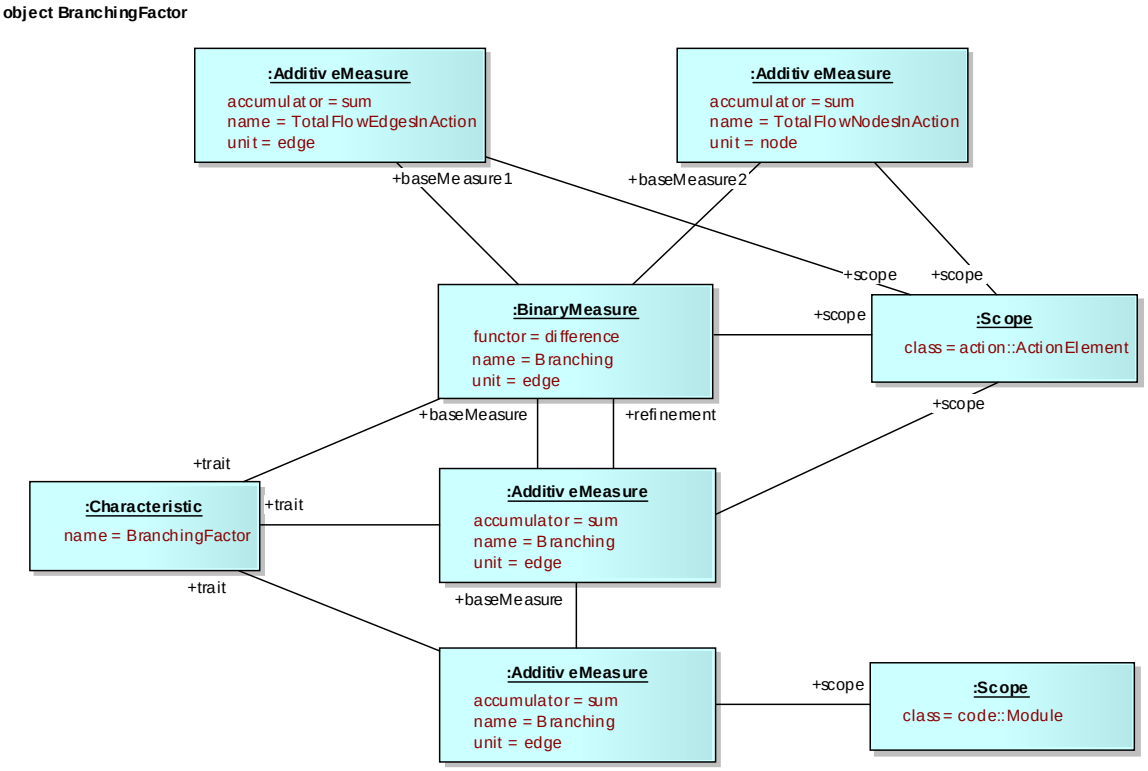

Figure 19.18 - Control Flow Branching Factor Library Entry

19.3.2 Cyclomatic Complexity of a Module[1]

Cyclomatic complexity (CC) = $E - N + p$ where E is the number of edges of the flow graph, N is the number of nodes of the flow graph and p is the number of connected components.

In this demonstration we assume that the control graph of each module is entirely connected. That is, p is always 1. Cyclomatic is then simply the branching factor of a module plus one.

1. See TPM 065 in Comsys Systems Redevelopment Methodology.

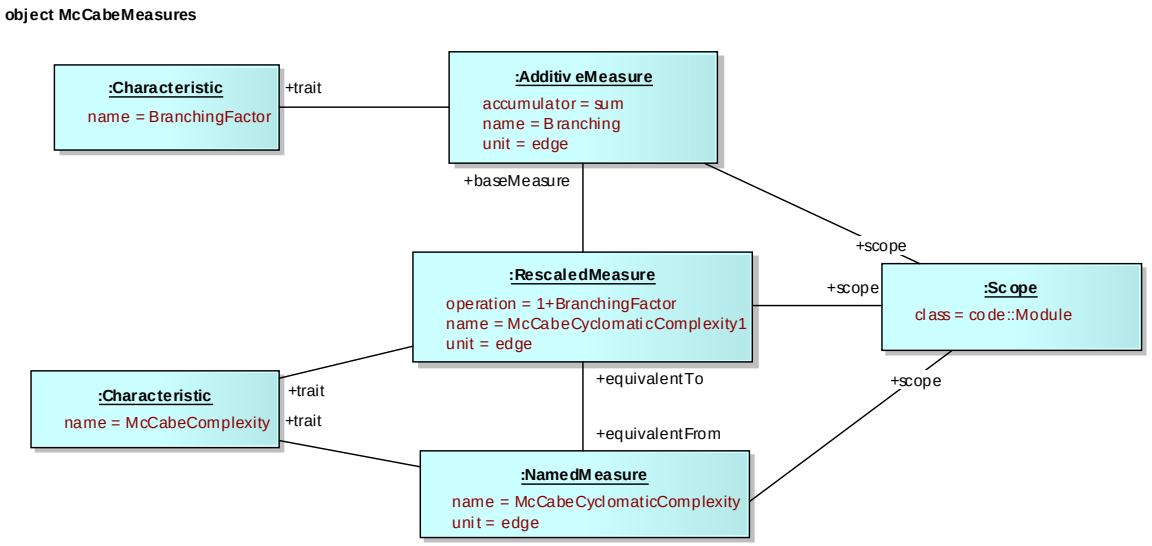

Figure 19.19 - McCabe Cyclomatic Complexity Library Entry

19.3.3 Extended Cyclomatic Complexity of a Module[1]

Extended cyclomatic is the count of predicates or atomic formula in the condition of branching statements. We demonstrate this count based upon ASTM modeling of an “if” statement. The condition of the “if” is an expression that can be navigated to find its atomic formulas.

19.3.4 Average Extended Cyclomatic Complexity of Modules in the System

19.4 Ratio of Additive ECC over Additive Counting of modules. Counts of Operating Systems

The Application Management and System Monitoring for CMS Systems (ASMS) specification provides a PIM based upon commercial enterprise management called the DMTF Common Information Model (CIM). “CIM models a software or hardware system as a collection of component models connected via associations. A specific instance of a system is modeled as a collection of instances of component models and associations.”[2]

We demonstrate the counting of operating systems installed and running on computer systems.

1. See ”An extension to the Cyclomatic measure of Program Complexity”, Glenford Myers, SIGPLAN Notices, vol 12 no 10, 1977.
2. See dtc/07-05-02.

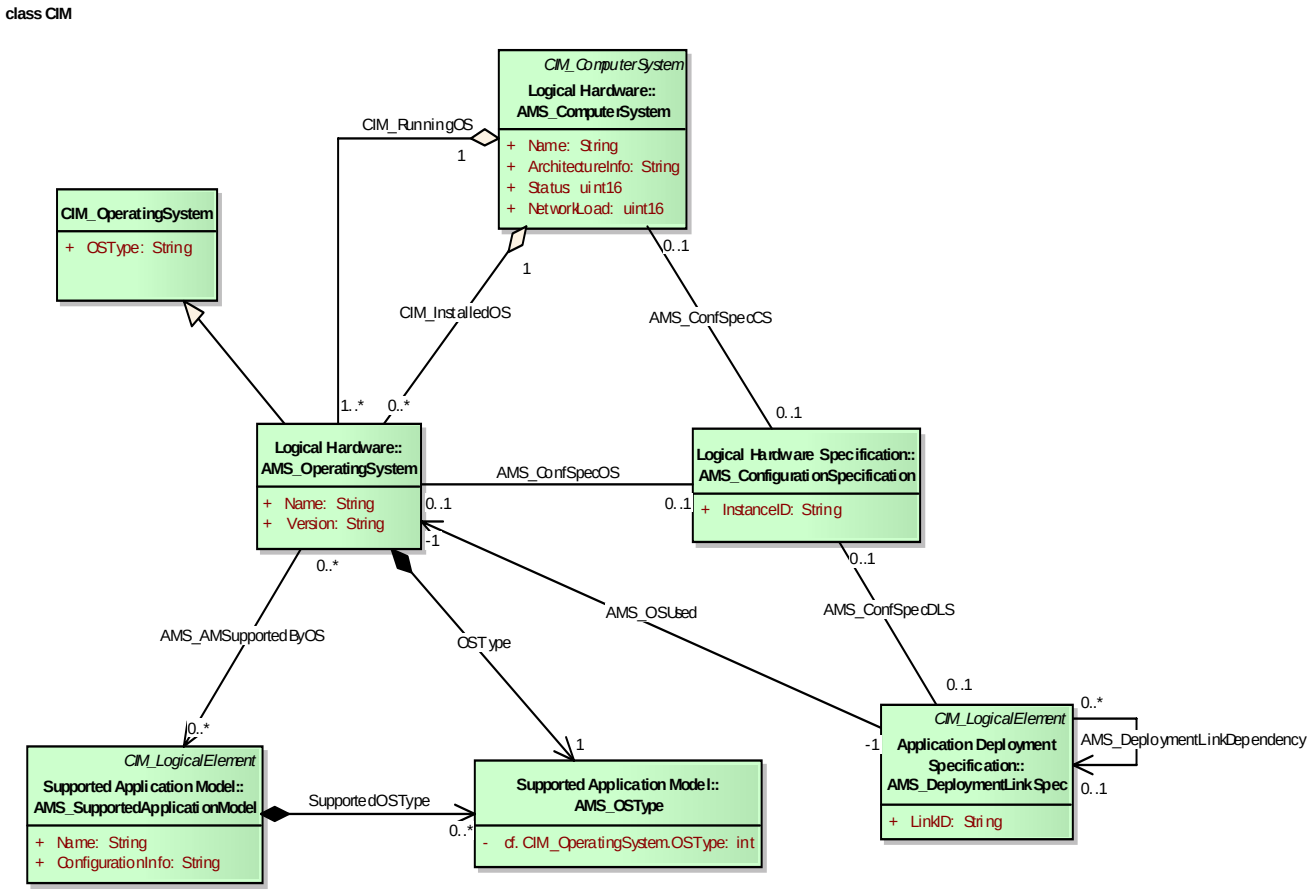

Figure 19.20 - ASMS Fragment

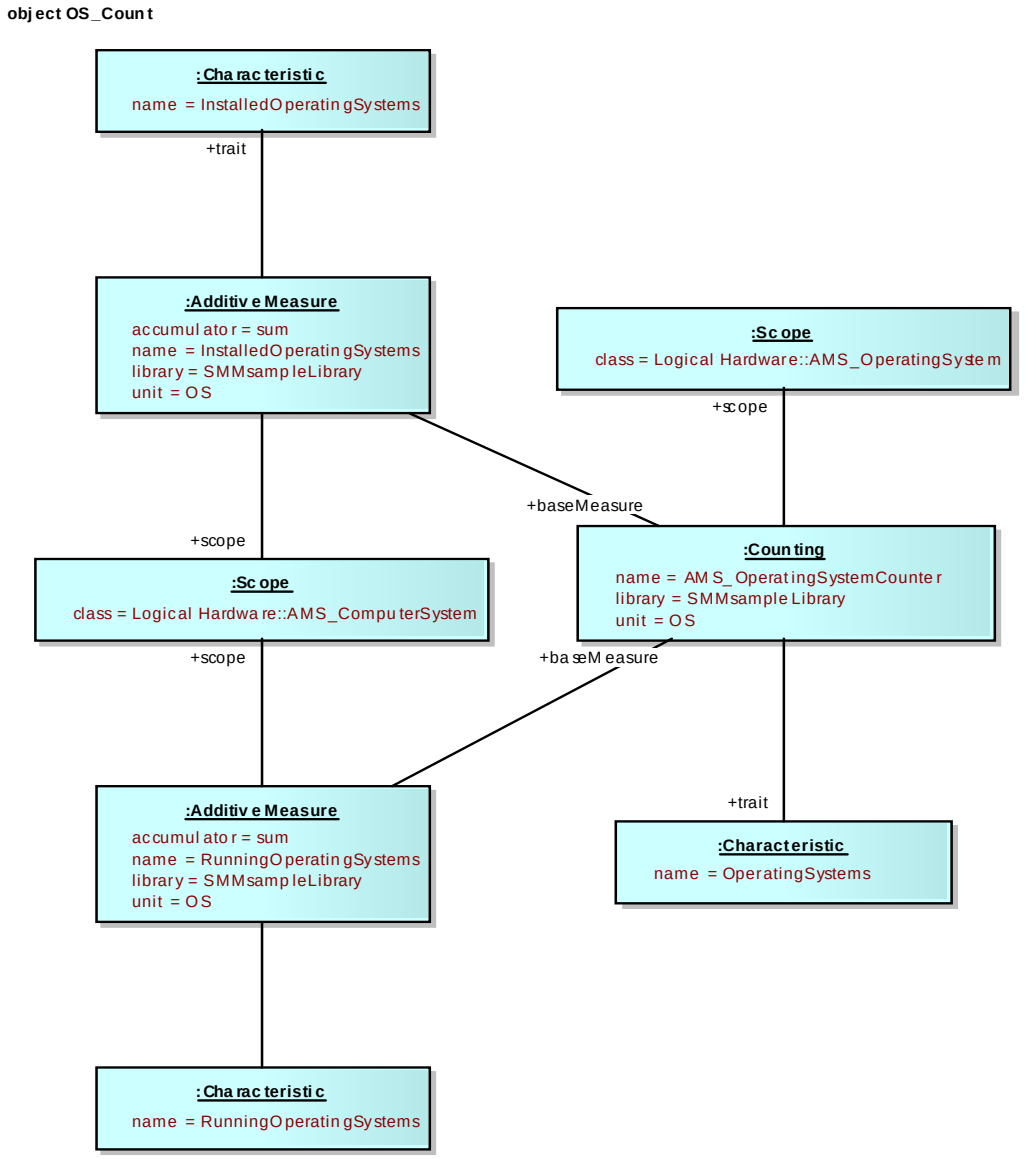

Figure 19.21 - OS Counting Demonstration

19.5 Halstead

19.5.1 Distinct Operator Count of a Module

$\acute{\eta}_1 \equiv$ A count of the number of distinct operators in a module.

Distinguishing operators invocations from calls to externally defined routines is not the type of higher level architectural concerns represented in the KDM. Counting the number of called, but not defined elements would get us close to this metric.

19.5.2 Distinct Operand Count of a Module

$\acute{\eta}_2 \equiv$ A count of the number of distinct operands in a module.

This is the data count shown above.

19.5.3 Operator Occurrence Count of a Module

$N_1 \equiv$ A count of the number of operator occurrences in a module.

This is a count of the calls to elements identified as operators.

19.5.4 Operand Occurrence Count of a Module

$N_2 \equiv$ A count of the number of operand occurrences in a module.

For KDM, this is a count StorableElements owned by ActionElements.

19.5.5 Halstead Length of a Module

$N=N_1+N_2$

This is an CollectiveMeasure where the aggregator is addition and the base measures are the occurrence counts given above.

19.5.6 Halstead Vocabulary of a Module

$\acute{\eta} = \acute{\eta}_1+\acute{\eta}_2$

This is an CollectiveMeasure where the aggregator is addition and the base measures are the counts given above.

19.5.7 Halstead Volume of a Module

$V=N \log_2 \acute{\eta}$

First $\log_2 \acute{\eta}$ is a ReScaledMeasure based upon the vocabulary metric given above. The volume is then an CollectiveMeasure of the length given above and the rescaled vocabulary with multiplication as the aggregator. The unit of measure for the rescaled vocabulary and for the volume is “required bits of representation.”

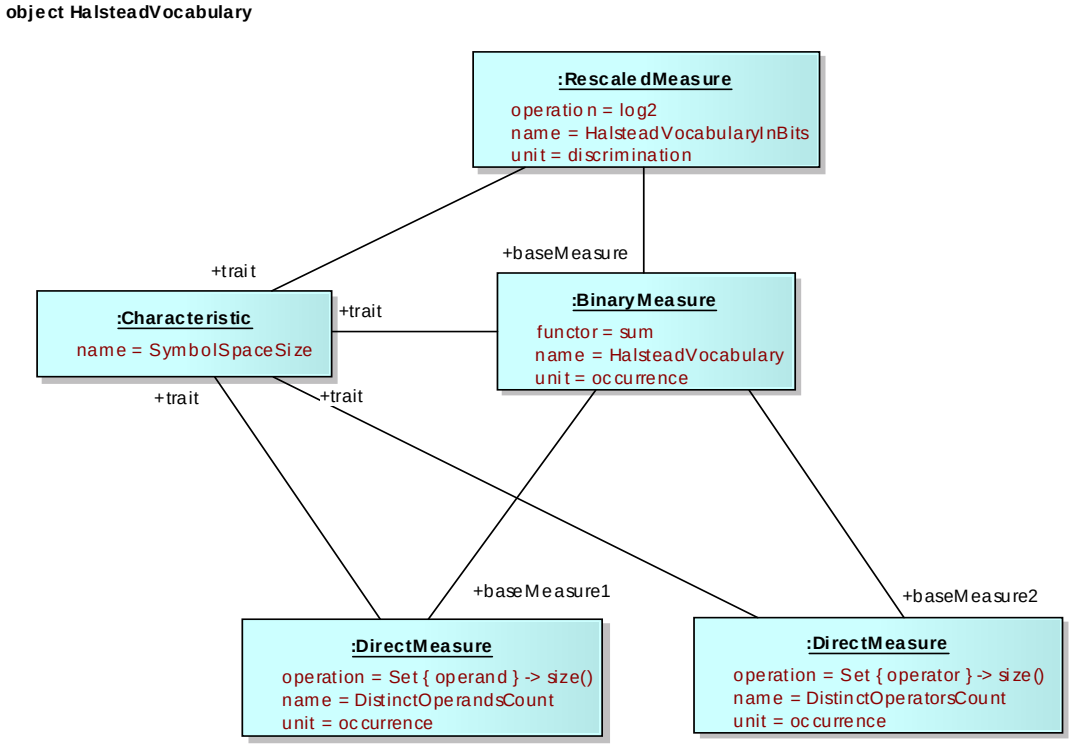

Figure 19.22 - Halstead Vocabulary Library Entry

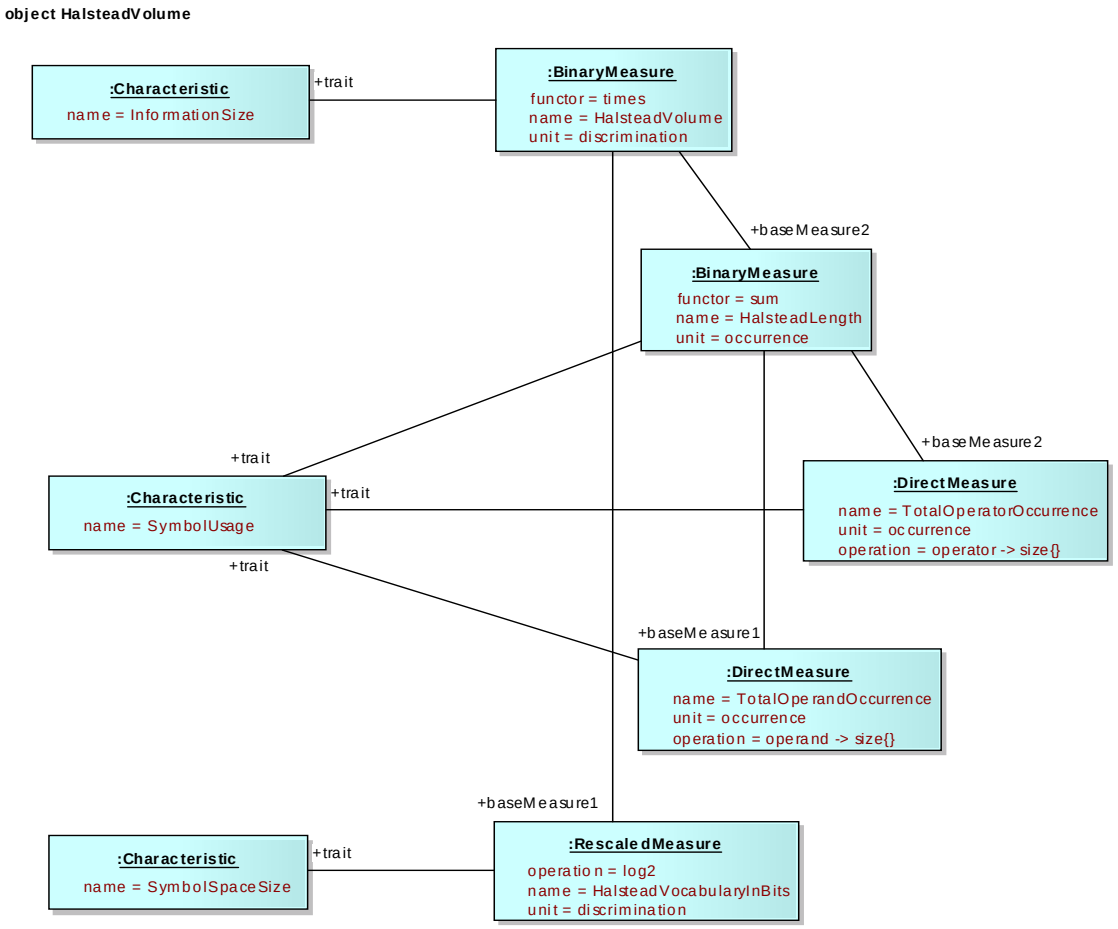

Figure 19.23 - Halstead Volume Library Entry

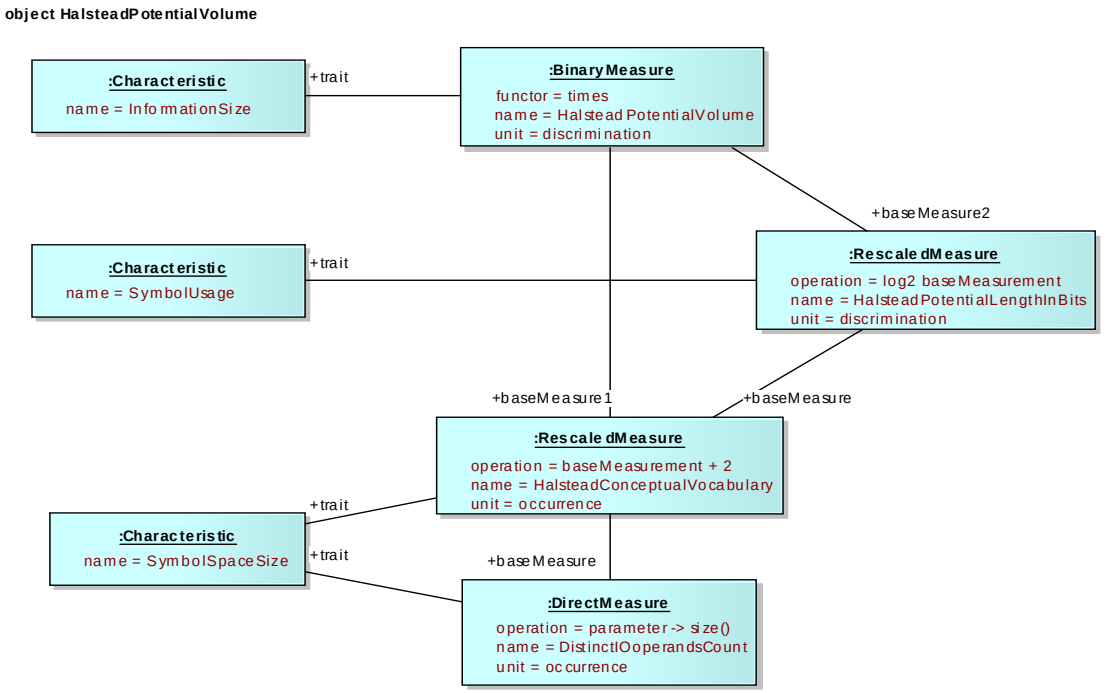

Figure 19.24 - Halstead Potential Library Entry

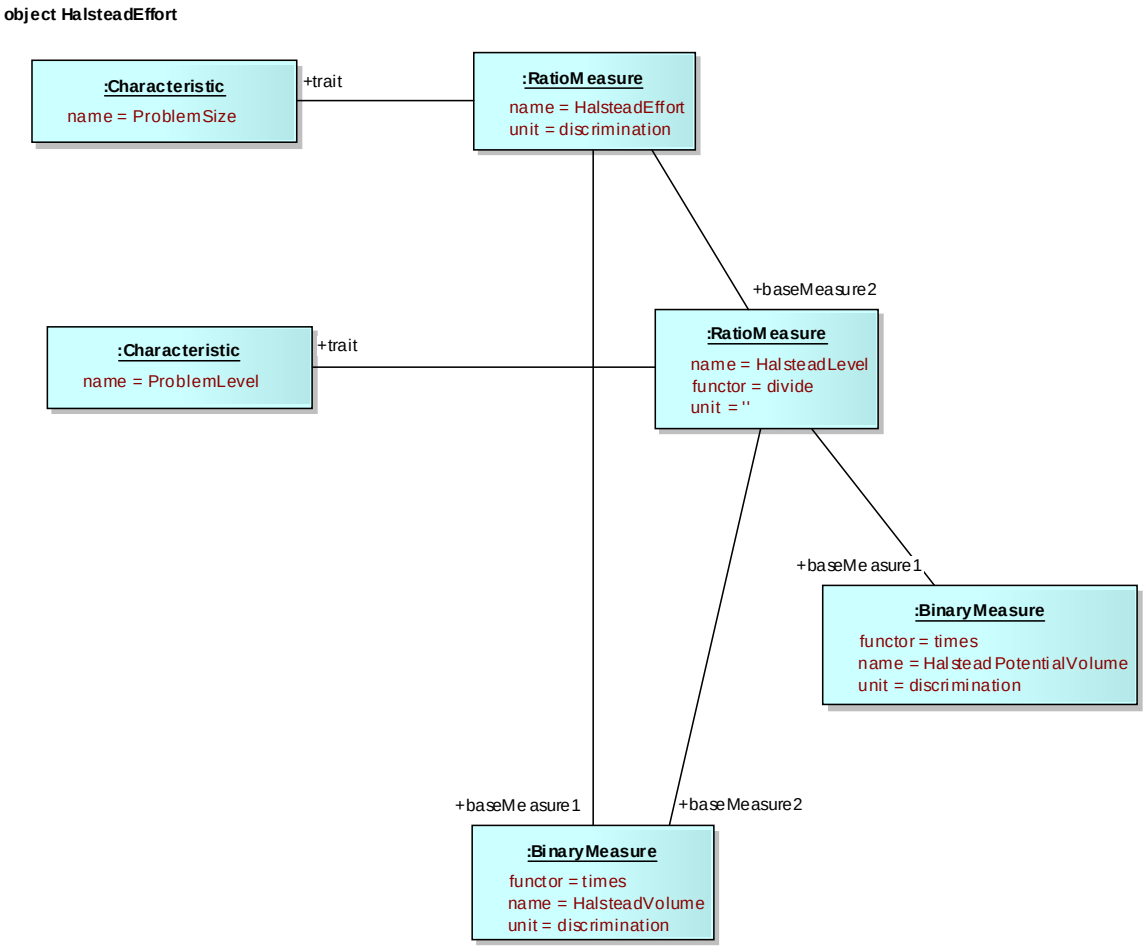

Figure 19.25 - Halstead Effort Library Entry

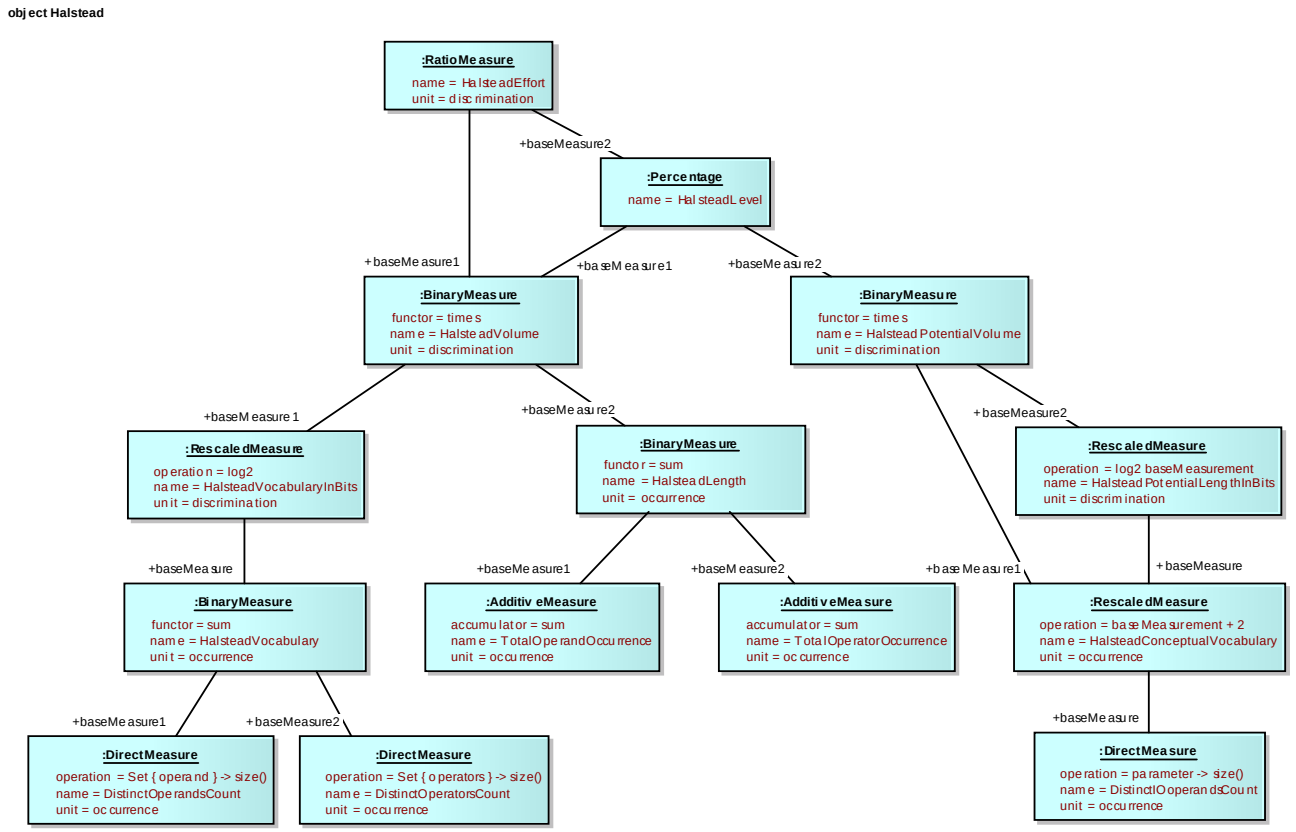

Figure 19.26 - Halstead Measures Demonstration

19.6 Software Engineering Institute (SEI) Maintainability Index

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

Each of the averages are RatioMeasures of their respective metric (V for Halstead volume, V(g') for extended Cyclomatic complexity and LOC of line of code) for modules over the count of modules. perCM, the percentage of comments in a module, is a PercentageMeasure of line count of comments over the total line count of a module.

Each resulting metric is rescaled to share the same unit of measure, namely maintainability index points.

aveV rescaled	50 – 5.2(ln(aveV)
aveV(g') rescaled	50 – 0.23(aveV(g'))
aveLOC rescaled	21 – ln(aveLOC)
perCM rescaled	50(sin (sqrt(2.4(perCM))))

The SEI index is then a CollectiveMeasure for a module of the above four rescaling with addition as the aggregator.

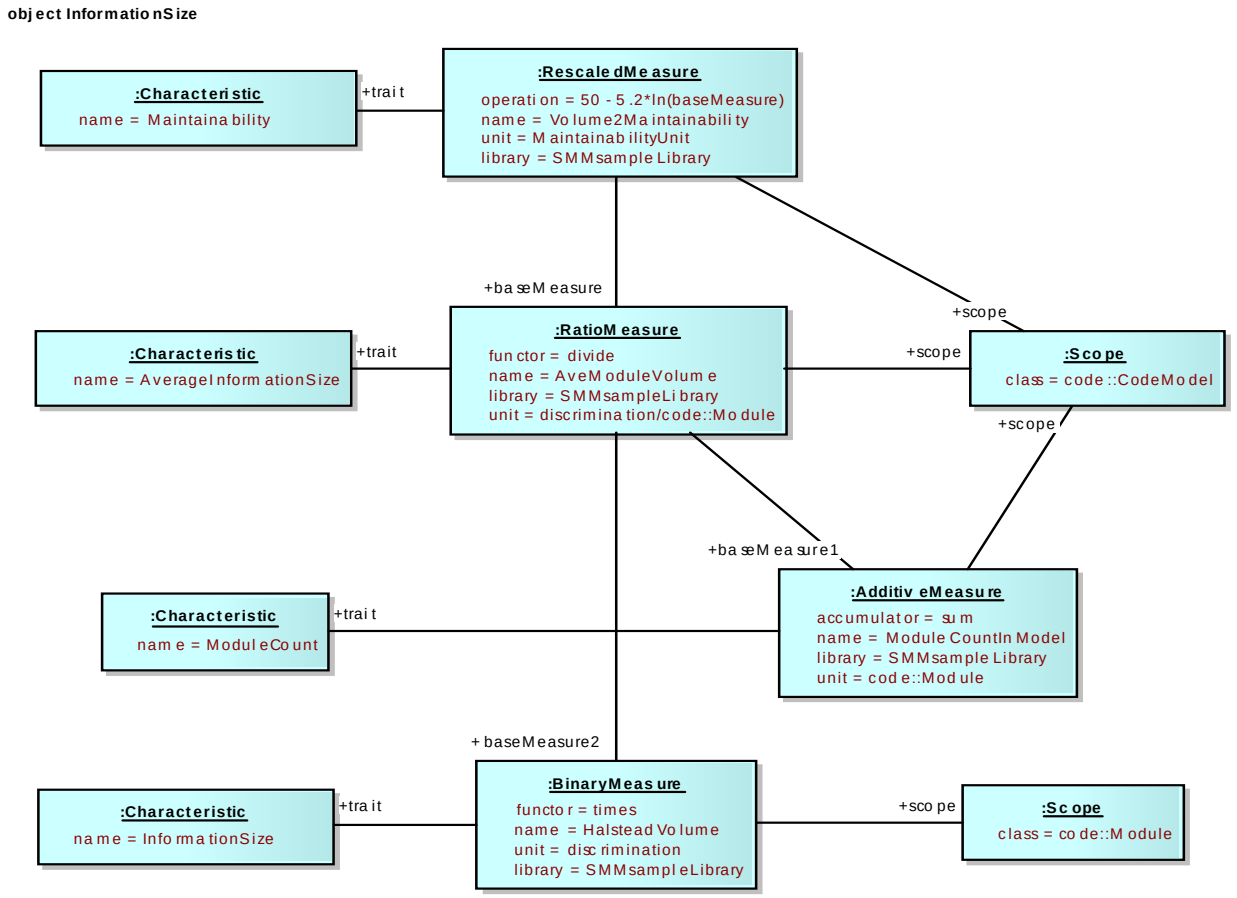

Figure 19.27 - Conversion of Information Size to Maintainability

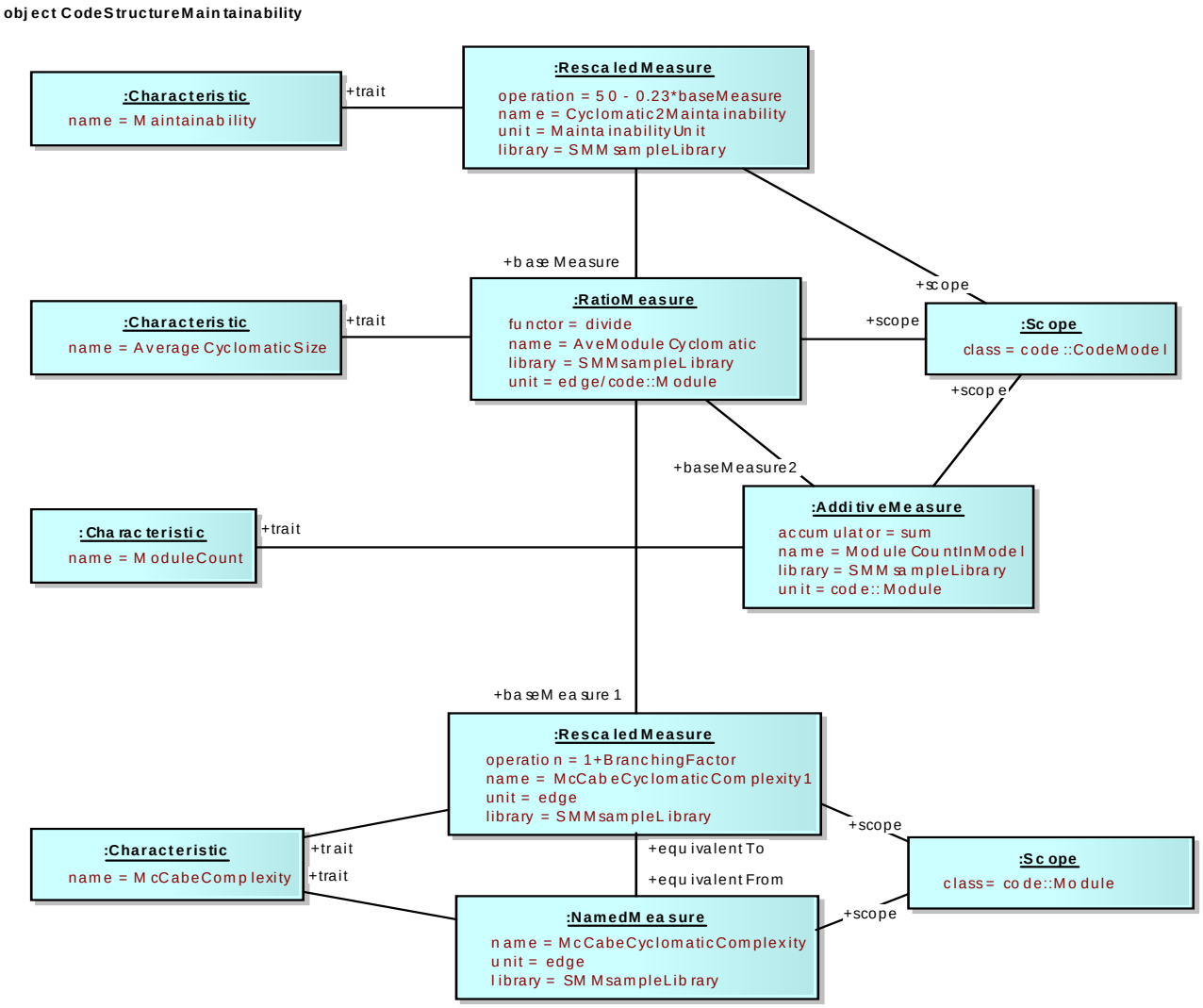

Figure 19.28 - Conversion of McCabe Cyclomatic to Maintainability

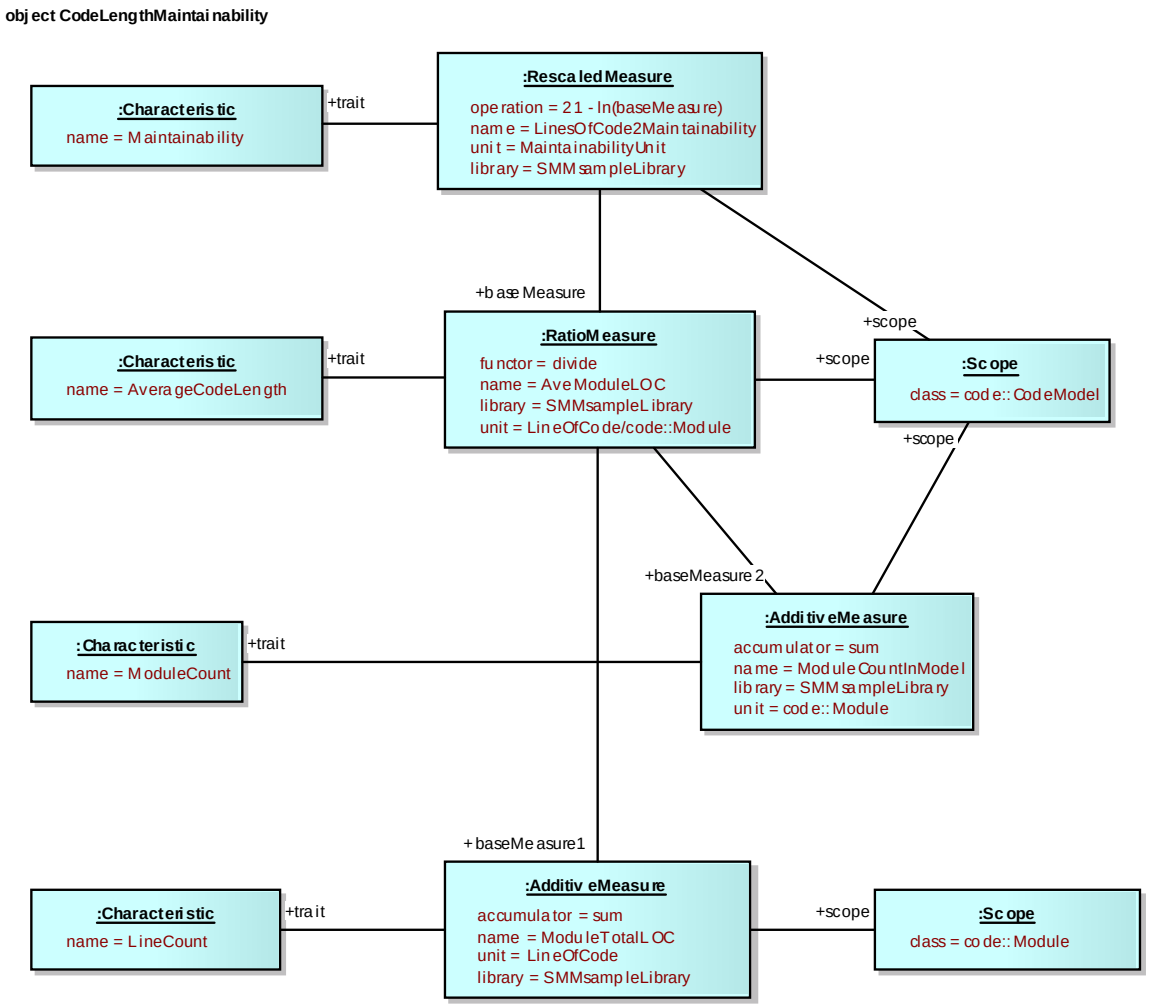

Figure 19.29 - Conversion of LOC to Maintainability

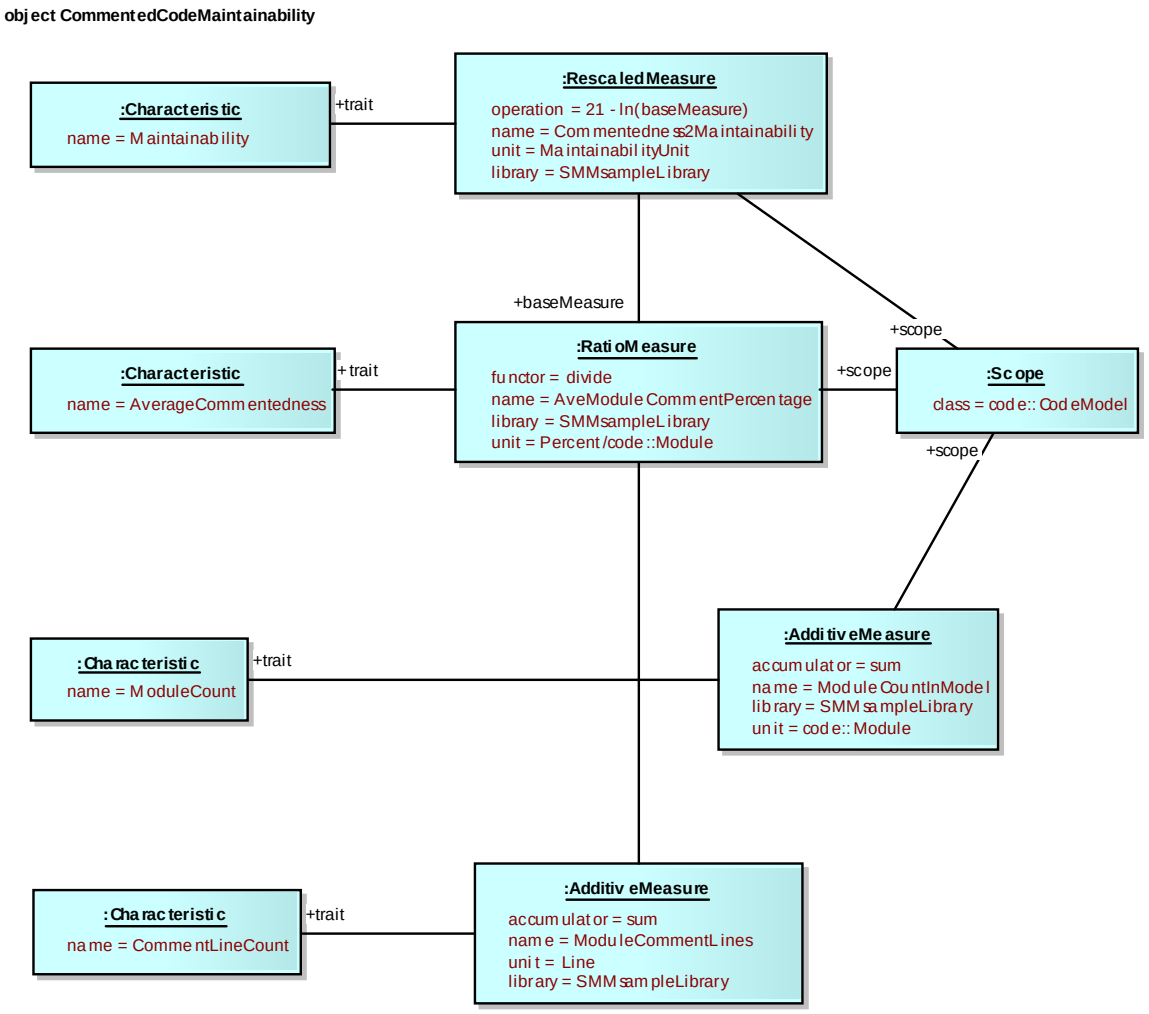

Figure 19.30 - Conversion of Comment Count to Maintainability

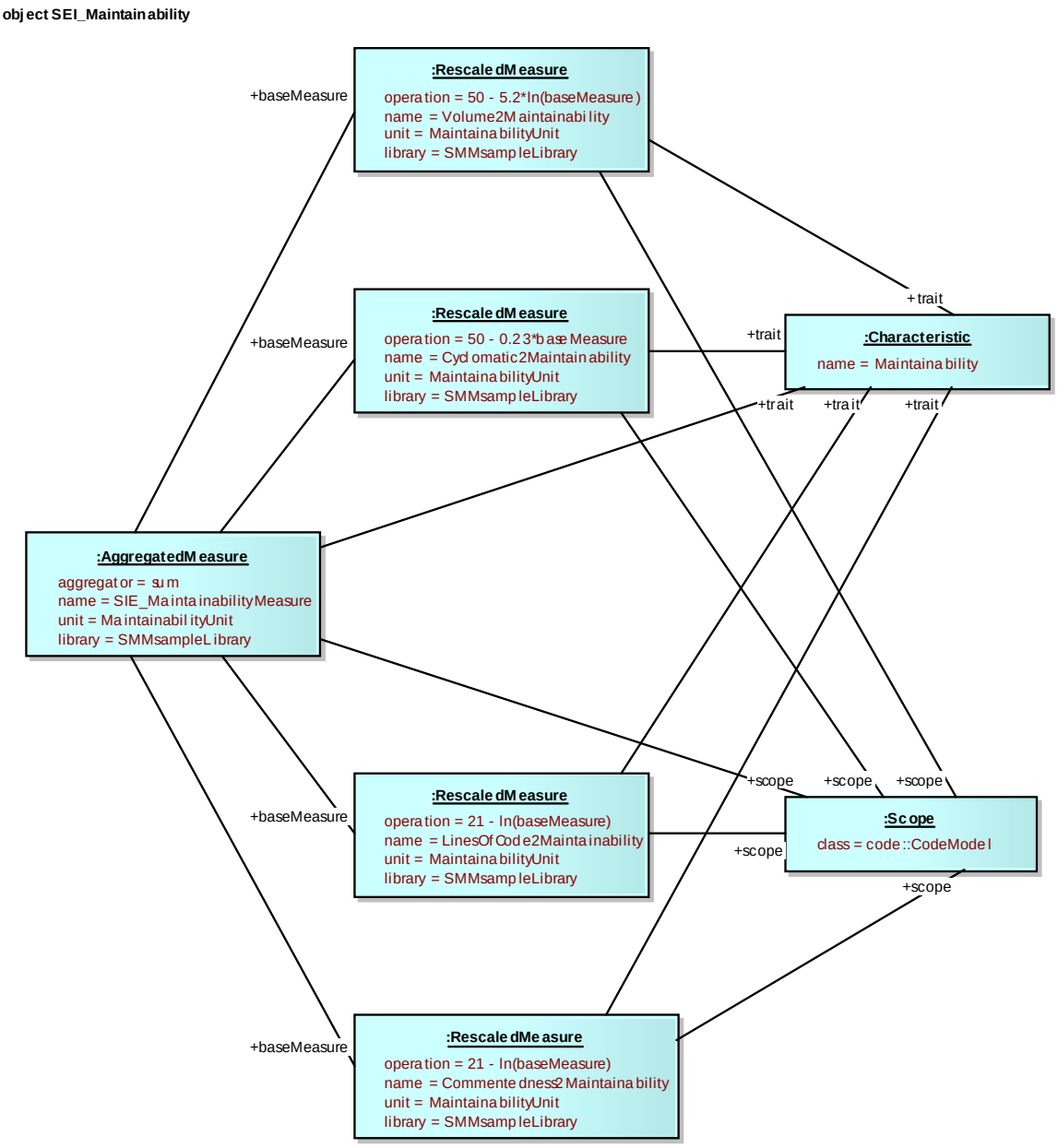

Figure 19.31 - SEI Maintainability Demonstration

19.7 Qualitative Example

19.7.1 Non-standard language usage score

Non-standard languages are defined by an organization's accepted technology standards. Assign the following scores where a 1 or 2 is low, a 3 is medium and a 5 is high:

1. 2GL or unacceptable 4GL assign 1 or 2
2. Acceptable 3GL or 4GL assign 3 or 4
3. Ideal strategic language assign 5

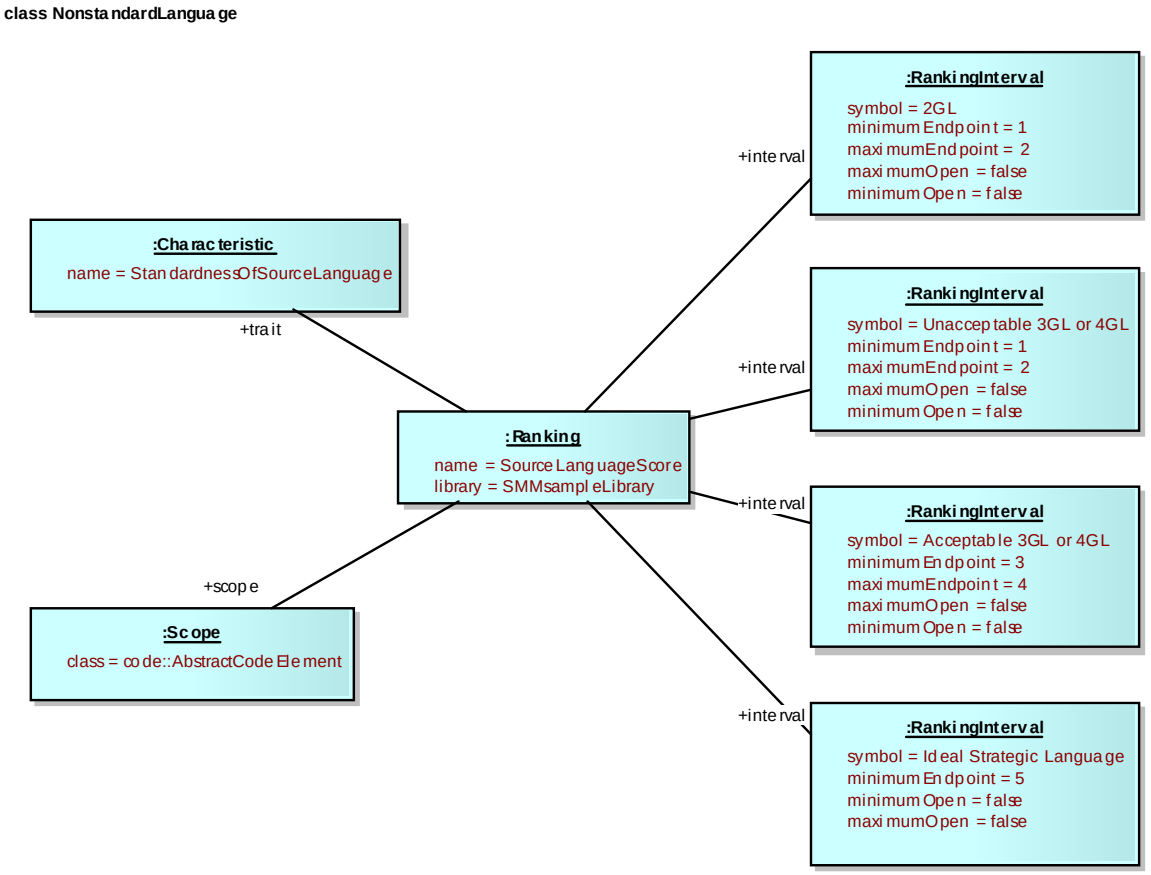

Figure 19.32 - Qualitative Measure Demonstration

20 Library of Categories (Software example)

20.1 General

SMM does not establish a standard set of measurement categories that presents an organization of measures applicable to every environment or every engineering activity. SMM minimally establishes a demonstration library of metric categories. The library does not assert that the given categories are standards. These metric categories reflect a high-level summary of industry metrics that support some engineering processes.

20.2 Environmental Metrics

Number of screens, programs, lines of code, etc.

20.3 Data Definition Metrics

Number of data groups, overlapping data groups, unused data elements, etc.

20.4 Program Process Metrics

Halstead, McCabe, etc.

20.5 Architecture Metrics

Average call nesting level, deepest call nesting level, etc.

20.6 Functional Metrics

Functions defined in system, business data as a percentage of all data, functions in current system that map to functions in target architecture, etc.

20.7 Quality / Reliability Metrics

Failures per day, meantime to failure, meantime to repair, etc.

20.8 Performance Metrics

Average batch window clock time, average online response time, etc.

20.9 Security / Vulnerability

Breaches per day, vulnerability points, etc.