Model states and therefore a suitable modeling language rather than a natural language can be used to express it. This modeling in fact allows accuracy, reduces ambiguity and gains the possibility of expressing the structure of information. The ideal modeling language used to represent metrics must be generic enough so that any conceivable metric can be expressed but, at the same time, concrete enough so that all the metric concepts can be treated with specific semantics. Also, the language must allow the modeling from systematic adaptation of metrics in a very changing environment and at the same way must keep most of the commonalities unmodified.

For example, *Unified Modeling Language (UML)* [Obj10] can be used as a language to express metrics. UML is a standardized general-purpose modeling language in the field of Software Engineering described by a number of class diagrams. By the very nature of object orientation, UML is multi-purpose and at the same time limited to the set of concepts its authors choose to include. This means that the representation of metrics is possible and the usage of UML would facilitate model driven development from a tool that could be supporting the tailoring of metrics.

Another alternative, besides UML, is a domain-specific language (DSL) [Dun94]. This specification language can be used to fit the representation of metrics as well. However for the sake of applicability, which can be limited in comparison to learn the language, the UML is selected as the language included in this work for representation purposes.

2.2. Metamodeling

A way to avoid misunderstanding regarding the Measurement Information Model is to establish a shared, agreed-upon set of concepts and terms and use them systematically for every communication. It is worth emphasizing that not only metric concepts and terms must be standardized across the organization, but also the mapping between them. This means that there is small value in establishing the terms to be used and the concepts in play if the relationships between them are not equally defined.

The term *metric* is probably the best example, as it is commonly misused to describe many different measurement concepts. Without concise and consistent terminology, effective communication among stakeholders is impossible. Because decision makers need to fulfill their measurement information needs, consistent measurement terminology is mandatory. This is achieved by the use of *meta-models*, where meta- indicates something *further* or *beyond*. Metamodeling, its characterization and needs are introduced in the next section.

2.2.1. Meta-models

The term *meta-model* is a qualified variant of *model*. This recommends that metamodeling is a specific kind of modeling. In fact, metamodeling is the analysis, construction and development of meta-models, which are a qualified variant of models. The differ-

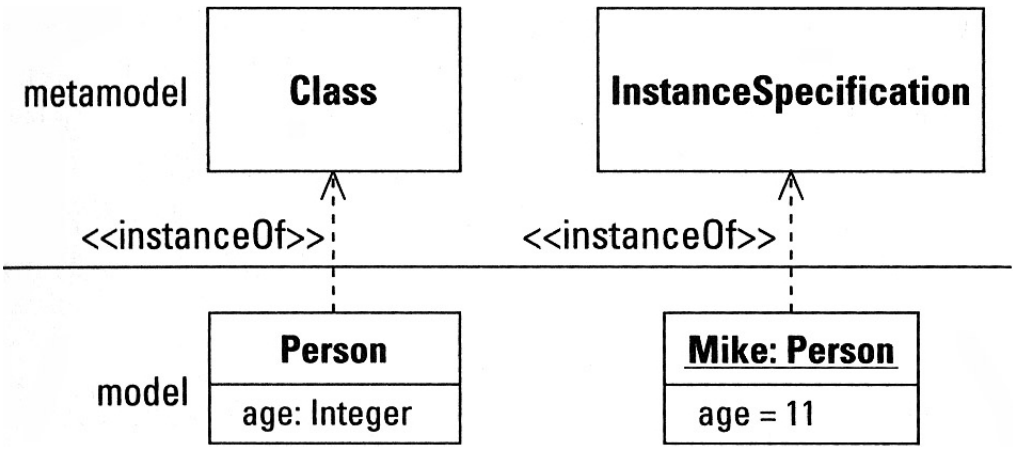


Figure 2.4.: Type-Instance relationship

ence between a regular model, one that is not a meta-model and a meta-model is that the information represented by a meta-model is a model itself. In epistemology and other branches of philosophy, the prefix meta- is often used to refer to its own category; on this concrete case, meta- means that the metric model that is being built represents other models, e.g. a metric meta-model is a model of the different project models within a certain organization. This relationship is often described as paralleling the Type-Instance relationship. In other words, the meta-model (Type) and each of the models (Instances) are composed of *different content* and are described using different languages (natural of software engineering languages). An example of such Type-Instance relationship is depicted in Figure 2.4.

There is one aspect that makes metamodeling a special kind of modeling: the subjects of metamodeling are models; in other words, the input and output artifacts are *made of the same stuff*, e.g. they are of the same type. This gives metamodeling a recursive nature that makes it much more complex than other modeling areas in which the subject being modeled is of a different nature.

The description of metamodeling and meta-model, provided above, is not especially related to metrics; in fact, any model that provides the language to represent models is a meta-model. The boundary to metrics refers to the fact that metrics are based in a measurement information model. The next statement may sound odd at first, but it is included to get this idea of metamodeling. If the metric is a model, creating that metric is modeling, whereas creating the language concepts used to describe the metric is metamodeling. Consequently, the language used to express a metric is a meta-model.

As it has been said before, the concepts of meta-model and metamodeling are not necessarily tied to metrics, but the current work is tied to metrics; therefore the focus on this particular application of metamodeling is also tied to metrics. According to this, it can be mentioned that:

"Metric meta-model is a domain-specific language oriented towards the

> *representation of a set of metrics within an organization."*

And also:

> *"Metric metamodeling is the analysis, construction and development of the models applicable and useful for modeling metrics."*

The metrics are expressed as a structure binding the relevant entities and attributes of concern with the information needs in the organization. These boundaries or relationships between entities and attributes are rich and complex. The meta-model helps to address these relationships. This structure needs a way to communicate these information needs and for that reason, it is also important to include them in the metamodeling as a mean of representation. The Measurement Information Model (see Section 2.1.2) can be used for reference during metamodeling. Traditionally four aspects for the metric metamodeling have been identified: metrics and their approaches; entity of measure and type of scale; information needs and interpretation aids and; reporting. These aspects are described below.

Metric and measurement approach. The core element from the meta-model is the metric. According to the Standard ISO/IEC 15939 [Sta07], the metrics are classified in two types, either base or derived. A base metric is a metric defined by a specific metric method, explained later on this section. Executing the method produces a direct value for the metric. An example of such metrics is *Earned Value* which describes how much of the resources have been consumed from the team members at a certain point of time. A derived metric is a metric that is defined as a function, covered here later, of two or more base and/or derived metric. An example of the last type of metric is the *productivity* that is derived by dividing *hours of effort* by the *lines of code*.

The measurement approach describes the specific implementation of a metric method or function within a given context. A method can be expressed in a formal language or simple textual description, while the function is typically represented in a mathematical form.

Entity of measurement and type of scale. Every metric requires two important aspects. The first is the entity of measurement, which binds it to a portfolio of metrics belonging to the domain model. This can ensure that metrics are collected only from the entities for which they are defined.

The second aspect that every metric requires is a scale. A scale is a set of values, continuous or discrete, or a set of categories to which an attribute is mapped. Every scale has a type, which specifies the values and also restricts the operations allowed for that type.

Five types of scale[2] are commonly defined:

[2] Standard ISO/IEC 15939 [Sta07] manages 4 types of scale, whereas Lichter uses an additional type of scale: absolute [Lic07]

Interval - numeric data for which equal distances correspond to equal quantities of the attribute without the use of 0 values. Example: week calendar.

Ordinal - discrete rankings. Example: excellent, very good, good and bad.

Nominal - categorical data. Example: HR, Logistic and Personnel.

Ratio - numeric data which is additive. Example: cost from development.

Absolute - data cannot be compute in another scale. Example: number of found errors during testing. Unit of measurement: error.

Information needs and interpretation aids. Information needs are the starting point to measure a product or process. Information needs are textually described. To satisfy an information need, the measurement results need to be interpreted in context of these information needs. The interpretation aids provide an evaluation of specified attributes derived from an analysis model with respect to defined information needs. Interpretation aids are the basis for measurement analysis and decision making. Indicators are formal interpretation aids. Therefore, value indicators are what should be presented to stakeholders.

Reporting. The communication of the measurement results within the organizations is equally important as the collection of the metrics. For this purpose, a report is specified and with it the measuring results are displayed in a common document.

Regarding the modeling language, metamodeling must provide a language (generic or more likely metric domain-specific) that offers the necessary expressiveness so that any conceivable metric can be appropriately specified. It should be made as few assumptions as possible about the organization and projects needs when defining this language.

Finally, the modeling language used to represent the metrics should be able to express measurement and reporting concepts such as derived metric, indicators and report. From a cognitive perspective, this means categorizing into well defined groups or sub-models, which in turn, involves having a clear notion of the concepts and their surrounding environment. For example, if the modeling language lacks of the concept *report visualization*, how could a metric be communicated? Again, from a cognitive perspective, the only way to implement a categorization of a concept is based on that concept.

2.2.2. Metrics at the Meta Level

Metamodeling is used to provide the degree of formality that refers to metrics. This representation can involve inheritance or association relations. For example, base and derived metrics have a common ancestor. Although they both belong to the metric root, it is not possible to completely join them. It is necessarily to make the differentiation since they differ on their measurement approach.

The goal from the resulting metric meta-model is a generic content. That is, it looks for the presentation of aspects related to measurement that are common to all the intended audience. It is not expected to introduce terms hard to grasp. For this purpose, there are a number of standards to introduce key concepts in the context of metric definitions.

Standard ISO/IEC 15939 [Sta07] defines a model for measurement process by specifying the steps that are necessary to measure products and processes. The standard introduces the Measurement Information Model (see Section 2.1.2). This describes how information needs are linked to the measured entities.

IEEE Standard 1061 [Sta05] defines a concept model for establishing quality factors and identify, implement, analyze and validate products and processes based on quality indicators. The standard results on this so-called quality factors that will be represented by direct quantitative metrics. This standard can be used to create a common definition of metrics in heterogeneous environments or develop a domain-specific language for metrics.

2.2.3. MeDIC Tool

The MeDIC (Measure Documentation, Integration and Calculation) Tool [VHLN08] was developed to improve the measurement portfolio of a full-service IT provider of one of the biggest financial services groups in Germany. The MeDIC Tool addresses three key aspects. First, the current collection of metrics from the organization must be reviewed and from there, it should be possible to define a collection of metrics for the organization and also their respective reporting. Second, the generated documentation should be role-specific. Finally, there should be a series of reports oriented to management.

A two steps approach was followed to get to the final product. The approach started with the definition of metrics, which was formalized by using metamodeling. Afterwards, the development of the tool was started. The content from both steps is explained below:

Metric meta-model definition. The definition of the metric meta-model aims to reach two goals, which are:

1. It should be complete enough, that it works as the basis for the metrics definition and their respective approaches and, the documentation from the metric results should be possible to define.

2. It should be technically as detailed and consistent as possible with the metric terms, so it can be used as a basis for a model-driven development tool.

An illustration from the metric meta-model is shown in Figure 2.5. This metric meta-model can be split in two major parts: the measurement information model (represented by all the entities, with exception of the entity *Report*) and the means of visualization.

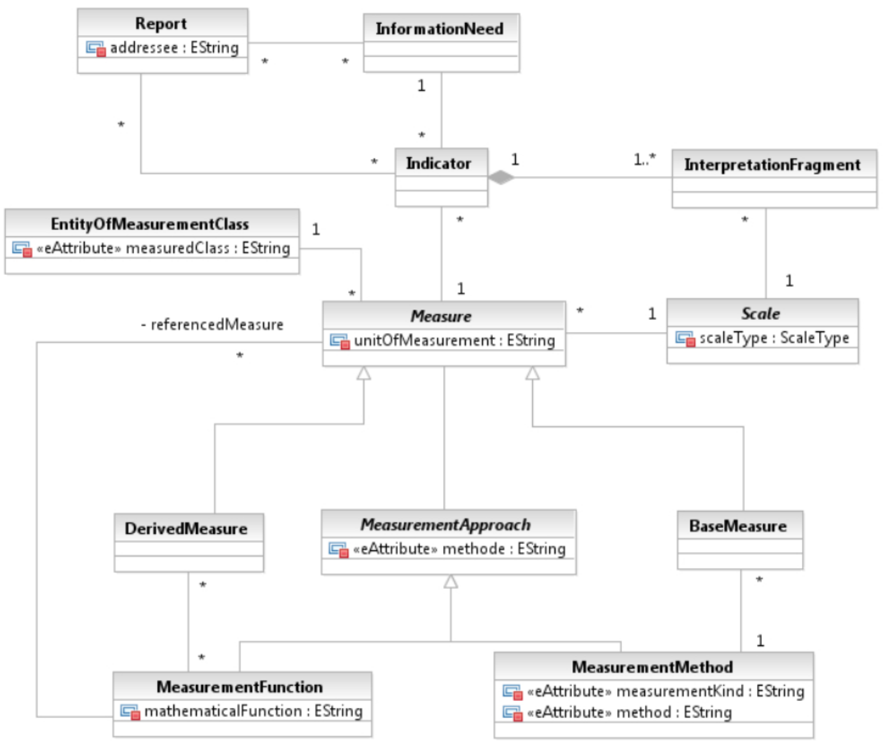


Figure 2.5.: The core from the metric meta-model

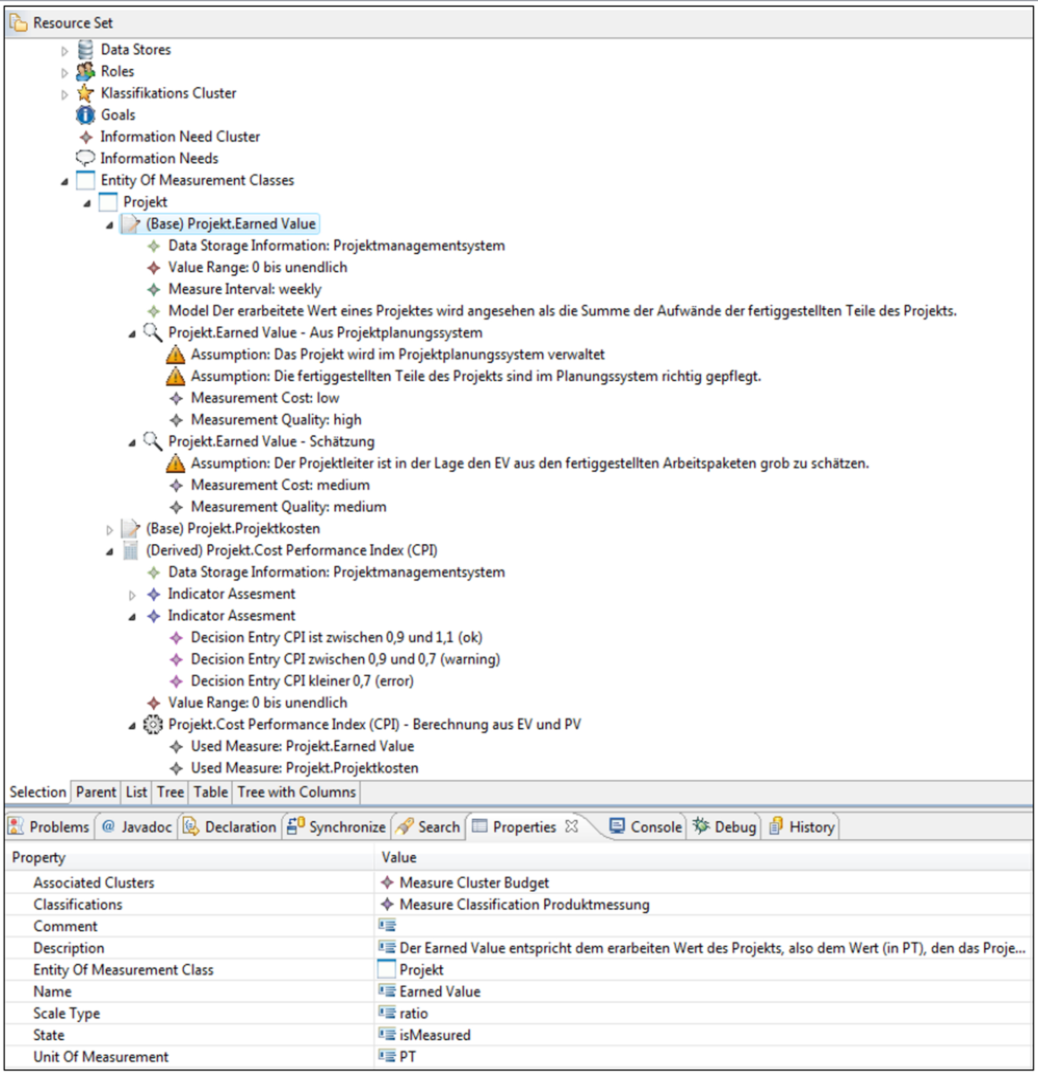
Tool support. The MeDIC tool consists of three parts:

Metric-Editor. A tree based editor, developed with *EMF-Technology*[3], which was generated out of the metric meta-model. The editor allows the definition of metrics, their approaches and reports for visualization. The validation of these definitions is covered with assistance of OCL instructions (completeness and consistence is verified).

Ongoing maintenance and improvement on the metrics is supported. Finally, project specific metric definitions can be created. These project specific metrics are frequently adjusted from the organizational wide defined metrics. Figure 2.6 illustrates a screenshot from the MeDIC-Editor. Each element from the tree represents an entity from the metric meta-model. Every element has associated a set of properties-values, shown below the tree structure.

Documentation generator. The documentation is produced in the form of HTML-Documentation, which can be stored in the organization repository. Thus, making it available across the organization. The generator is base on a model-to-text transformation and can be adjusted to fit individual needs.

[3]Eclipse-based modeling framework and code generation facility for building tools and other applications based on a structured data model [Ecl10].


Resource Set
Selection | Parent | List | Tree | Table | Tree with Columns
Problems | Javadoc | Declaration | Synchronize | Search | Properties | Console | Debug | History

```
Data Stores
Roles
Klassifikations Cluster
Goals
Information Need Cluster
Information Needs
Entity Of Measurement Classes
  Projekt
    (Base) Projekt.Earned Value
      Data Storage Information: Projektmanagementsystem
      Value Range: 0 bis unendlich
      Measure Interval: weekly
      Model Der erarbeitete Wert eines Projektes wird angesehen als die Summe der Aufwände der fertiggestellten Teile des Projekts.
      Projekt.Earned Value - Aus Projektplanungssystem
        Assumption: Das Projekt wird im Projektplanungssystem verwaltet
        Assumption: Die fertiggestellten Teile des Projekts sind im Planungssystem richtig gepflegt.
        Measurement Cost: low
        Measurement Quality: high
      Projekt.Earned Value - Schätzung
        Assumption: Der Projektleiter ist in der Lage den EV aus den fertiggestellten Arbeitspaketen grob zu schätzen.
        Measurement Cost: medium
        Measurement Quality: medium
    (Base) Projekt.Projektkosten
    (Derived) Projekt.Cost Performance Index (CPI)
      Data Storage Information: Projektmanagementsystem
      Indicator Assesment
      Indicator Assesment
        Decision Entry CPI ist zwischen 0,9 und 1,1 (ok)
        Decision Entry CPI zwischen 0,9 und 0,7 (warning)
        Decision Entry CPI kleiner 0,7 (error)
      Value Range: 0 bis unendlich
      Projekt.Cost Performance Index (CPI) - Berechnung aus EV und PV
        Used Measure: Projekt.Earned Value
        Used Measure: Projekt.Projektkosten
```

Property	Value
Associated Clusters	Measure Cluster Budget
Classifications	Measure Classification Produktmessung
Comment	
Description	Der Earned Value entspricht dem erarbeiten Wert des Projekts, also dem Wert (in PT), den das Proje...
Entity Of Measurement Class	Projekt
Name	Earned Value
Scale Type	ratio
State	isMeasured
Unit Of Measurement	PT

Figure 2.6.: Screenshot from the MeDIC Graphical User Interface (GUI)

Calculation tool prototype. In order to perform any calculation, the measurement methods and function have to be specified as OCL instructions. In order to effectuate the calculation, the tool iterates over all the required elements of the metric meta-model and the results are provided as reports.

The MeDIC Tool allows storing existing metric documentation in a single, integrated and consistent way. It also allows metrics to be available as a reference to any who requires it. Since the core of the tool is the metric meta-model, the interpretation of the results has been improved and the measurement processes are more transparent.

The tool support establishes a process to define and maintain the metrics, which are managed by the metric expert team. This approach also improves the communication of new metrics that have been used successfully in a project, to the rest of the company.

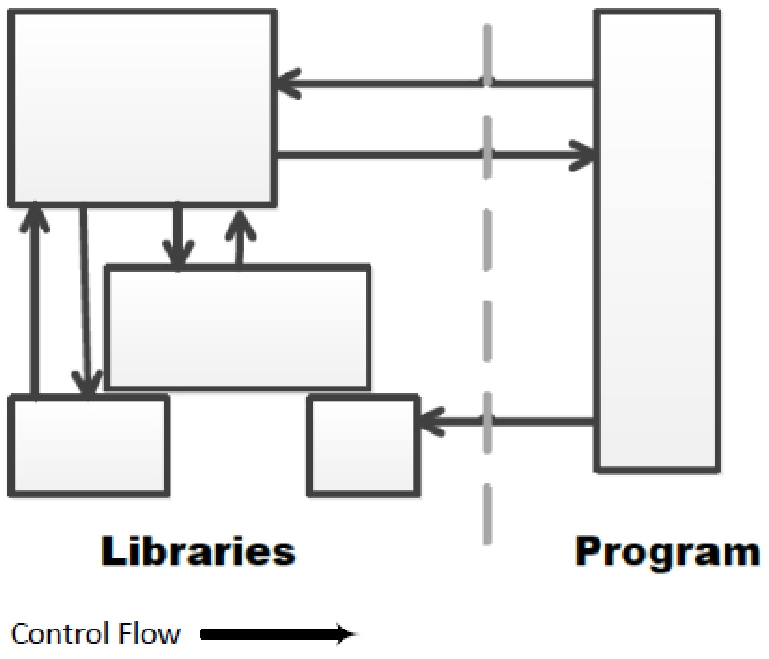


Figure 2.7.: An application making use of many software libraries

2.3. Reusable Software Component Models

In this part of the chapter, two software component models are provided: software library and object oriented framework. First, a *software component* is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties [Bos98]. A library is a component made available to develop software and an object oriented framework is a special case of software libraries, which is reusable abstraction of code wrapped in a well-defined Application Programming Interface (API) [Lic08].

2.3.1. Library

A library is a collection of reusable components, usually classes, used to develop software. The content of the library is independent to the application's context and provides with certain functionality. That is, it provides access to the code that performs a programming task.

The usage of the libraries can increase productivity by reusing code and Know-how and focus in the real problem; improve the quality since the libraries were developed by specialists; decrease maintenance effort because there exist *standard components* used in the different applications. A common usage of libraries is depicted in Figure 2.7.