2009 Ninth International Conference on Quality Software

Model-Driven Software Measurement Framework: a case study

Beatriz Mora
Indra Software Labs
Information Technology Company
Ciudad Real, Spain
bmorar@indra.es

Felix Garcia[1], Francisco Ruiz[1, 2], Mario Piattini[1]
[1] Department of Information Technologies and Systems
University of Castilla-La Mancha
Ciudad Real, Spain
[2]Dep. of Mathematics, Statistic and Computing
{felix.garcia;francisco.ruizg;mario.piattini}@uclm.es

***Abstract*—Measurement is an important factor in the process life cycle since it controls issues and deficiencies during software maintenance and development projects. The model-driven engineering (MDE) paradigm may be extremely useful in a software measurement process owing to the considerable amount of entity types and models involved, and a model-driven Software Measurement Framework (SMF) has previously been developed with this goal in mind. This framework facilitates the measurement of any type of software entity. This paper presents the use of SMF in an IT company, in order to demonstrate the utility and importance of the application of model-driven software measurement in the real world.**

Software Measurement, Model-driven Engineering, MDA.

I. INTRODUCTION

Measurement is an engineering activity that enables us to obtain quantitative information with regard to the engineering process or the systems which are being developed, and is a key factor in the process life cycle since it provides a support for the planning, monitoring, control and evaluation of a software process. Measurement has, in fact, become a fundamental aspect of Software Engineering [1].

Software process improvement (SPI) programs make it necessary for companies to measure a highly heterogeneous set of software entities (processes, projects, products, resources), and this diversity highlights the importance of providing the means to define the measurement models in companies in an integrated and consistent manner [2].

Software measurement can therefore benefit from the MDE (Model-Driven Engineering) paradigm [3]. This paradigm is becoming increasingly more important due to the complex nature of business landscapes in which there is a growing diversity of systems and platforms. The underlying motivation for MDE is to improve the productivity of software companies, as new software artefacts can be generated from (mostly technical) models, thereby supporting developers in their productivity. MDE is based on the usage of models as primary artefacts, from which validation, code, test and documentation are derived. The significant role that models have recently taken on in this scenario reinforces the importance of measuring models as a previous and fundamental step towards their later improvement. The Software Measurement Framework (SMF) has been developed to provide a reference framework for the measurement of any kind of model [4]. SMF follows the MDE principles in which software measurement models (SMM) and domain models (models of the entities to be measured) are the core artefacts of the measurement process. The MDE paradigm is applied to support the measurement of heterogeneous software entities, which implies: a) the definition of measurement models which conform to a Software Measurement Metamodel [2]; b) the definition of reusable generic measurement methods which must be applicable to any model; and c) supporting the computation and storage of the defined measures and later decision making processes. The main goal of SMF is to ensure that the measurement process is carried out in a consistent and more productive manner by providing companies with the necessary infrastructures and methodology.

This paper describes the application of SMF in a real-world IT company. The potential benefits of the synergic combination of MDE and measurement are also illustrated. The company's measurement process was supported by SMF through the provision of a homogenized framework into which the measurements of the different kinds of entities considered (requirements and databases among others) were integrated. The paper also presents an enhanced version of SMF functionality through a list of parameterized measurement methods which increase support to the definition of new generic base measures. The remainder of the paper is organized as follows. Section 2 provides an overview of related works, and Section 3 summarizes the main characteristics of SMF. Section 4 describes the parameterized methods. The use case is presented in Section 5 and conclusions and future works are outlined in Section 6.

1550-6002/09 $26.00 © 2009 IEEE
DOI 10.1109/QSIC.2009.39

II. Related Works

Literature contains numerous publications dealing with tools which are important success factors in software measurement efforts [5], which supply work environments and general approximations [6], or which provide architectures with more specific solutions [7]. The work of [8] includes a list of tools which support the creation, control and analysis of software measurements, and that of [9] examines various software measurement tools, such as MetricFlame, MetricCenter, Estimate Professional, CostXPert and ProjectConsole, in heterogeneous environments. Other proposals through which to tackle software measurement, which are more integrated and less specific than in the aforementioned cases, also exist. These include [10] which proposes the MMR tool, based on the CMMI model for the evaluation of software processes. Other tools can be found in [11-13]. These proposals are, however, restricted to concrete domains or to evaluation models of a specific quality characteristic.

[2, 14] present the FMESP framework with the aim of providing a more generic environment. FMESP proposes the original idea of using metamodels to manage the software measurement of any kind of models.

SMF is an adaptation of FMESP to the MDE paradigm by using MDA technology [4]. This idea has been replicated in other technological environments [15]. However, certain important aspects characterize SMF as a complete environment with which to manage software measurement in an MDA context, particularly, among others, a robust Software Measurement Metamodel (SMM) [2] for the definition of software measurement models and a textual and graphical concrete syntax to do so (the Software Measurement Modeling Language, SMML [16]). A further added value of SMF is that the Software Measurement Ontology (SMO) [17, 18] was used as the basis for the development of the SMM. The SMO states the elements involved (concepts and relationships) in the software measurement domain, and was built by analysing the most relevant sources from both the existing international standards (ISO, IEEE) and the research proposals dealing with software measurement concepts and terminology. The SMO therefore provides a common vocabulary which has been used to resolve the problems of completeness and consistency identified in the aforementioned sources. The SMML language also facilitates definition in a more usable and intuitive variety of software measurement models, which is the starting point of the generic software measurement processes.

III. Software Measurement Framework

The Software Measurement Framework (SMF) [4] facilitates the measurement of any type of software entity. In this framework, any software entity in any domain which has a metamodel associated with it can be measured with a common metamodel (SMM) and by using QVT [19] transformations. SMF has three fundamental elements: a conceptual architecture, a technological environment and a methodology. These elements have all been adapted to the MDE paradigm and to MDA technology, taking advantage of their benefits within the field of software measurement.

The following subsections briefly explain the conceptual, technological and methodological elements which are part of SMF. A more detailed description of SMF can be found in [4].

A. Conceptual architecture

The need for a generic and homogeneous environment for software measurement has led to the inclusion of a conceptual architecture and a tool with which to integrate software measurement into SMF. The conceptual framework used to manage model-driven software measurement is presented in Fig. 1. Two new elements, namely the QVT Relations metamodel and model, have been added in order to fully adapt the FMESP conceptual architecture [20] to MDA.

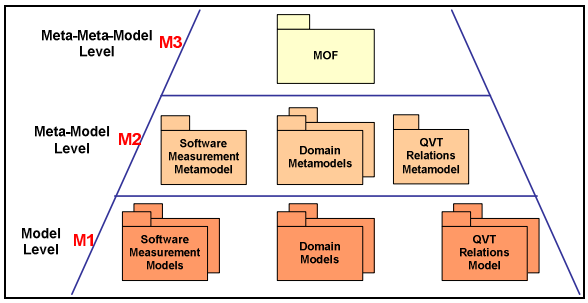

Figure 1. Conceptual framework with which to manage model-driven software measurement.

The aim of this architecture is to achieve the integrated management of modeling and measurement through the representation of the elements which are related in different abstraction levels. As Fig. 1 shows, the architecture has been organized into the following conceptual MOF-based metadata levels: Meta-Metamodel Level (M3), Metamodel Level (M2) and Model Level (M1).

B. Method

The steps necessary to carry out the software measurement by using the SMF are explained below (see Fig. 2):

1) Incorporation of domain metamodel: the measurement is made in a specific domain. This

domain must be defined according to its metamodel. For instance, if the aim is to measure UML diagrams, then the UML metamodel must be included in the SMF repository.

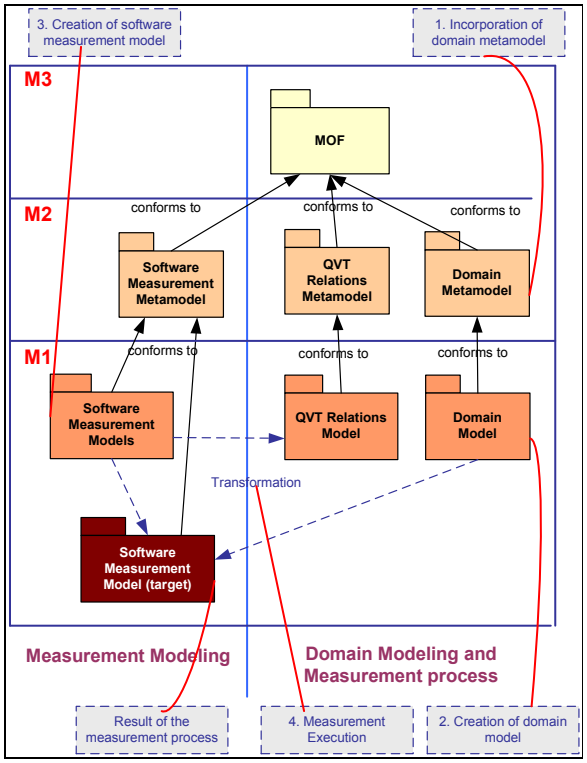

Figure 2. Elements of SMF.

2) Creation of domain model: this is defined according to its corresponding domain metamodel (created in the first step). The domain models are the entities whose attributes are measured by calculating the measurements defined in the corresponding measurement models. For example, a UML model which represents the analysis and design of a bank application is a domain model.

3) Creation of measurement model: the measurement model is created according to the SMM (a key integrated part of SMF). This constitutes the source model. The results of the measurement are stored as an instance of the "Measurement Action" package. The packages which are included in the SMM are: Software Measurement Characterization and Objectives, Software Measures, Measurement Approaches and Measurement Action (a full description of these can be found in [2]).

4) Measurement execution: the automatic measurement execution is carried out through a QVT transformation in which the target measurement model is obtained from the two source models (the measurement model and the domain model) in which the results are defined, i.e. the "Measurement Action" package is instantiated. The target measurement model is obtained by extending the source measurement model with the results of the measurements. These results are calculated by running OCL queries on the domain model.

The QVT Relations model, denominated as the extended or final QVT Relations model (see Fig. 2), is the transformation which is needed to perform the measurement. Since the proposal concerns generic measurement, it is very important that this QVT model be obtained in a generic manner, i.e., that it can be used in several domains. The MDE paradigm and MDA technology are applied for this reason. The final QVT Relations model, in which there are two source models, is obtained from a QVT transformation. These source models are: the basic or initial QVT Relations model (which conforms to the QVT Relations metamodel) and the Software Measurement model (previously defined) [4].

C. *Technological Environment*

This section provides a brief presentation of the technological elements of SMF:

- **Adaptation to MDA:** in Fig. 2 the SMF elements which are necessary to carry out a software measurement process are presented according to MOF levels. The QVT Relations Model is obtained automatically through a transformation from a Measurement model. It contains all the information that is necessary to carry out the QVT transformation of the SMF proposal, i.e. transformations for the measurement of software entities.
- **Software Measurement Model Definition tool:** the tool selected to define the software measurement model is MetaEdit+ [21] from MetaCase. MetaEdit+ offers significantly different approaches towards the definition of Domain Specific Modelling (DSM) support. It provides the support necessary to develop graphic editors such as SMML. The software measurement models defined by MetaEdit+, can be exported to ECORE by means of the MetaEdit+ Generator System [22]. The MetaEdit+ Generator System uses MERL (a domain-specific language for creating generators) to produce code/text from models and integrate them into Ecore. The software measurement model exported in ECORE can be used in SMF.
- **QVT support tools:** two tools have been selected to support the transformation definition and execution in SMF: MEDINI QVT [23] and MOMENT (MOdel

manageMENT) [24]. These model management environments implement the OMG's QVT Relations specification in a powerful QVT engine and are integrated into the Eclipse platform. The Ecore language has been selected from these environments because it is a common language based on EMOF (the part of the MOF 2.0 specification that is used to define simple metamodels by using simple concepts).

IV. PARAMETRIZED MEASUREMENT METHODS

Until very recently a large quantity of base measures (a measure of an attribute that does not depend upon any other measure, and whose measurement approach is a measurement method) defined in the Software Measurement Process in SMF were obtained by applying the count measurement method. This measurement method permits us to define a large quantity of base measures [25] in multiple software measurement domains, and these base measures can be used to define a considerable amount of derived measures and indicators in a software measurement process. A selection of base measures obtained by applying the count measurement method in different domains is shown in the Table I.

TABLE I. A SELECTION OF MEASURES OBTAINED WITH THE COUNT MEASUREMENT METHOD

Measure	Measurement Approach
Base measure: NOM (UML Class Diagrams)	*Measurement Method:* to **count** the number of local methods.
Base measure: NOP (Java Code)	*Measurement Method:* to **count** the number of packages
Base measure: NT (Relational Databases schema)	*Measurement Method:* to **count** the tables in the schema
Base measure: NE (Entity Relationship)	*Measurement Method:* to **count** the entities in the model

Since our objective is to obtain new base measures in the Software measurement process, it is of interest to consider the use of parameterized measurement methods. The definition of parameterized methods increases the power of the definition of base measures by adding restrictions or conditions that the measurement method must satisfy at the domain metamodel level to obtain the measures. These conditions are included in the measurement method by using a declarative language, such as OCL.

TABLE II. A SELECTION OF PROPERTIES WITH WHICH TO DEFINE THE PARAMETRIZED MEASUREMENT METHOD

Metamodel Element	Properties
Class1	`attribute11 operation value1 operator attribute12 operation value2 operator … attribute1N operation valueN operator`
OCL Constraint	`MetamodelElement.allInstances()-> select(m:MetamodelElement| m.attribute1 operation value1 operator m.attribute2 operation value2 operator. m.attributeN operation valueN`
references of Class1	`Attribute21 operation value1 operator Attribute22 operation value2 operator … Attribute2N operation valueN operator`
OCL Constraint	`MetamodelElement.allInstances().referenceName-> select(r:RefMetamodelElement| r.attribute1 operation value1 operator r.attribute2 operation value2 operator r.attributeN operation valueN`
Logical operators: `and, or, xor, not`	**Operations:** `=, <>, >, < <= >=`

An illustration of how the conditions and restrictions are specified at the metamodel domain level can be provided by considering the basic elements of a metamodel which has been defined as an instance of MOF or Ecore: a metamodel is built with classes which have attributes and operations and binary associations (implemented with references between classes). This basic schema is illustrated in Fig. 3a. In order to define parameterized measurement methods, it is necessary to specify conditions on the classes, their attributes and/or references. One example of a metamodel element that permits us to parameterize the values of the attributes is the Enumeration element, in which the parameterization is restricted to values that are bounded by enumeration. Table II shows the general conditions that can be added to a measurement method. These conditions or properties, which are added to the measurement method, are composed of a set of operations between the class attributes, and the values for these attributes and these operations are joined through logical operators. The declarative expression has also been defined with OCL to provide greater understanding.

The benefits of the parameterized measures can be illustrated by considering the example of the relational database measurement. The domain metamodel and the domain model have been defined as is shown in Fig. 3b.

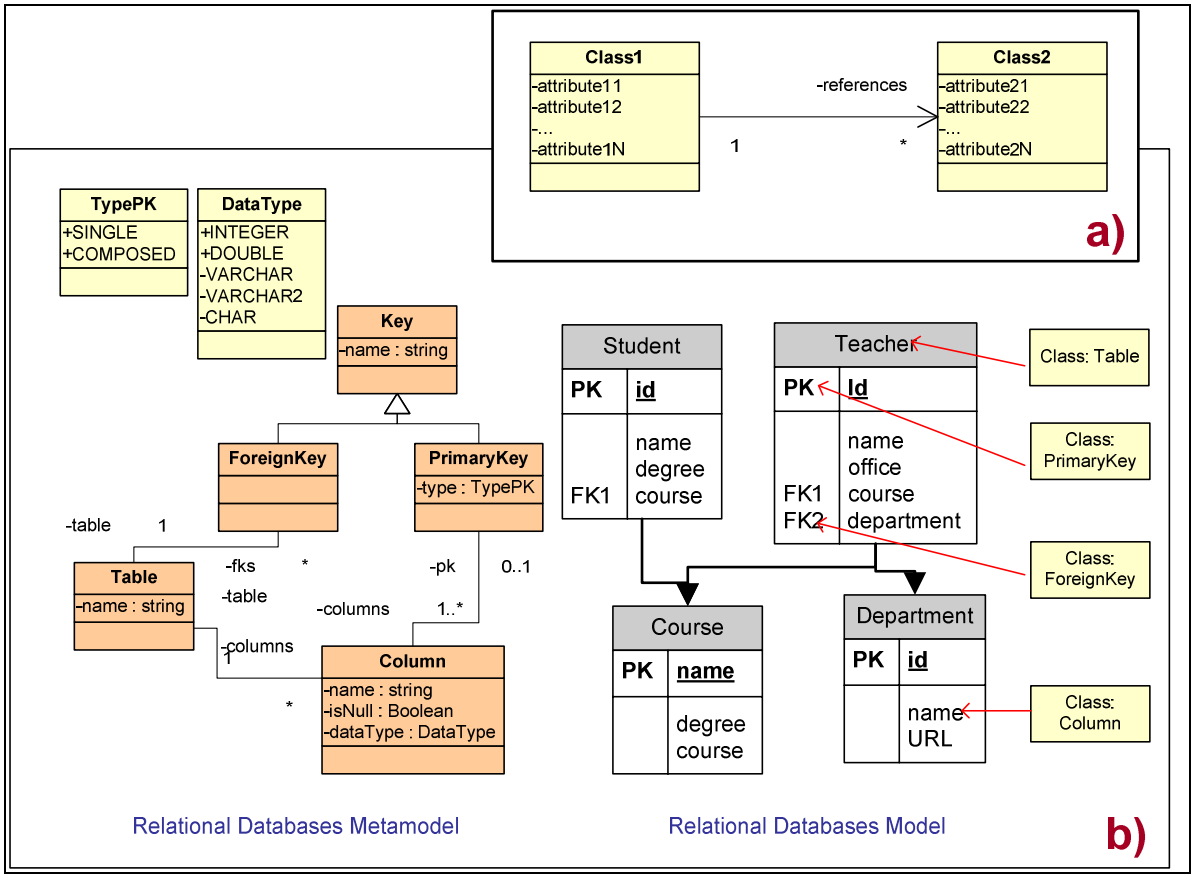

Figure 3. Relational Databases Metamodel and Model.

Table III shows the parameterized measurement methods defined on the domain relational database. These methods have been defined by adding parameters to the basic COUNT measurement method. According to our general approach, the COUNT measurement method allows us to obtain the number of instances of a specific Class. With the parameterized methods, the number of instances returned can be restricted solely to those which satisfy certain conditions specified by means of a declarative expression.

As Table III shows, if a measurement engineer wishes to define a measurement method s/he must specify or select the following elements:

- **Method:** the kind of measurement approach which is applied in the software measurement, for instance "count".
- **Metamodel element:** the element to which the measurement method is applied (primary key, column, table, etc.).
- **Attributes:** the attributes used to define the condition for the base measure. The values of the attributes must be defined in the metamodel by means of Enumerations, such as TypePK and DataType (Fig. 3b).
- **Condition:** the condition necessary to parameterize the base measure.

The measurement results are obtained by executing the measurement method with the conditions from the metamodel element. For example, in the relational schema in Fig. 3b, there are 4 "*primary keys*" whose "*type*" is single.

TABLE III. A SELECTION OF COUNT MEASUREMENT METHODS

	Method	Metamodel Element	Attribute/s	Condition	Result
1	Count	PrimaryKey	Type	Type = SINGLE	4
	OCL Expression	`PrimaryKey.allInstances()->select(PK:PrimaryKey | PK.type= 'SINGLE')->size()`			
2	Count	Column	isNull, dataType	isNull=false and dataType=Integer	3
	OCL Expression	`Column.allInstances()->select(C:Column | C.isNull= false and C.dataType=Integer)->size()`			
3	Count	Table	No parameterized attributes to measure	Without condition	4
	OCL Expression	`--This is not a parameterized measurement method Table.allInstances()->size()`			
4	Count	columns on Table	isNull, dataType	isNull=false or dataType<>Integer	15
	OCL Expression	`Table.allInstances().columns->select(c:Column | c.isNull=false or c.dataType<>Integer)->size()`			

V. CASE STUDY

In order to illustrate the benefits of using SMF, let us consider a case study in the INDRA IT company, in a real project denominated as Health 2.0. Indra is the premier IT company in Spain and is a leading IT multinational in Europe and Latin America. It is ranked as the second European company in its sector according to stock market capitalisation, and it is one of the three Spanish companies with the highest investment in R&D. INDRA has received formal level-3 CMMi certification from the Software Engineering Institute, which implies, among other things, that INDRA projects are carried out in the context of quality support processes in which software measurement is a fundamental area. At present, the company has specific methods with which to carry out software measurement in the various domains involved. Moreover, due to the necessities of technological improvement, the tools used are diverse and are in a continuous state of change. It is, therefore, in the company's interest to homogenize the software measurement process and to use a framework based on a common software measurement metamodel which permits the definition of software measurement models and the execution of a software measurement process in any domain.

The satisfaction of Indra's necessities with regard to quality led us to consider the use of the MDA compliant SMF in order to carry out automatic software measurements of any software entity types or domains. We therefore decided to apply the proposal to the Health 2.0 project, which advocates a global, innovative version of healthcare. This requires both the creation of new information systems, designed to be used by healthcare professionals, managers and citizens, and a rapid, ubiquitous and simple interaction. This proposal can be illustrated by considering two of the software domains to which SMF was applied: requirements and database.

A. Measurement of Requirements Stability

The first software domain evaluated by our measurement framework was in the "Requirements analysis" within "Software development activities".

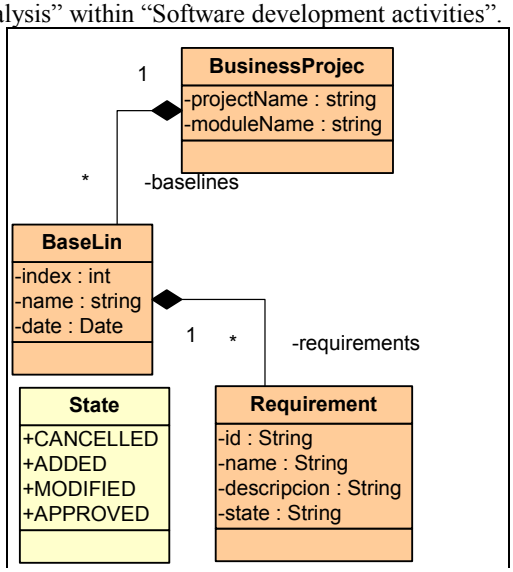

Figure 4. Requirements metamodel.

The requirements analysis is a critical activity in the creation of a software product, and it is for this reason that a good quality requirements process must be carried out. The idea is to determine the stability of one base line's requirements with regard to another previous base line, i.e. how stable the requirements are upon passing from one version to another. This necessitates discovering the percentage of requirements that have been cancelled, approved, added and modified in one version with regard to the previous one.

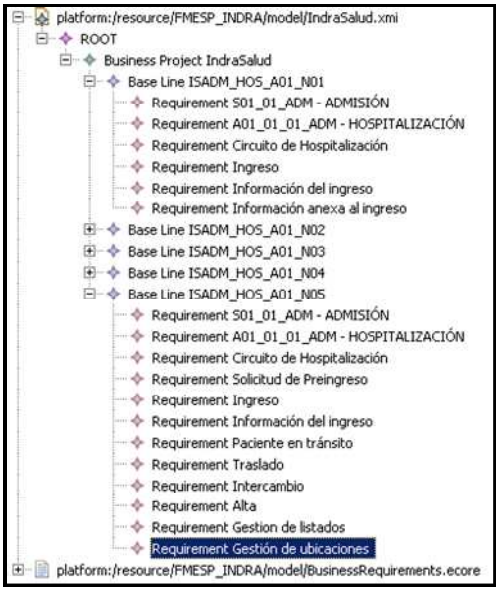

Figure 5. Requirements model.

Fig. 4 shows the Requirements Metamodel, which is the metamodel that characterizes the type of entity to be measured. An example of an instance of this metamodel (a specific requirements document from the Health 2.0 project) is shown in Fig. 5. Parameterized count methods have been used to define the base measures in the Requirements Metamodel, since it is necessary to evaluate the state of a requirement with regard to its previous version.

TABLE IV. INSTANCES OF "CHARACTERIZATION AND OBJECTIVES" PACKAGE

Element	Measurement model (instances M2)
Attribute	Size
Entity class	Project requirement (Baseline 1 to 5)
Measurable concept	Stability
Quality model	ISO 9126
Information need	To discover the project requirements stability

In accordance with the SMF method, the following four steps were followed to carry out the measurement:

1) Incorporation of Requirements metamodel (Fig. 4).

2) Creation of model conforms to Requirements metamodel. In this case, the model is a set of project requirements, from baseline one to baseline five (see Fig. 5). The model is stored in XMI.

3) Creation of measurement model which conforms to SMM. In order to evaluate the requirements stability, it was necessary to define a software measurement model, which is detailed in Table IV and Table V and Table VI. As Table V shows, parameterized count methods have been defined with the exception of the TOR base measure.

TABLE V. INSTANCES OF "SOFTWARE MEASURES AND SOFTWARE APPROACHES" PACKAGE FOR BASE MEASURES

Base Measure	Measurement method
NCR (Number of cancelled requirements)	To count the cancelled requirements (state=cancelled)
NAR (Number of approved requirements)	To count the approved requirements (state=approved)
NADR (Number of added requirements)	To count the added requirements (state=added)
NMR (Number of modified requirements)	To count the modified requirements (state=modified)
TOR (Total of requirements)	To count the requirements

In Table V the value of scale and unit for every base measure is the same, i.e., the value of Scale is "Ratio: integers from zero to infinite" and the value for the Unit is "Requirement".

TABLE VI. INSTANCES OF "SOFTWARE MEASURES AND SOFTWARE APPROACHES" PACKAGE FOR INDICATORS

Indicator Analysis Model Scale & Unit	Decision Criteria
RCI: Requirements Cancellation Index **Analysis Model:** RCI=NCR/TOR **Scale:** Ratio: Real from zero to infinite. **Unit:** Percentage	If RCI > 75 → 'Very High' If 50 < RCI ≤ 75 → 'High' If 25 < RCI ≤50→ 'Medium' If 0 ≤ RCI ≤ 25 → 'Low'
RAI: Requirements Approbation Index **Analysis Model :** RAI=NAR/TOR **Scale:** Ratio: Real from zero to infinite. **Unit:** Percentage	If RAI > 75 → 'Very High' If 50 < RAI ≤ 75 → 'High' If 25 < RAI ≤ 50 → 'Medium' If 0 ≤ RAI ≤ 25 → 'Low'
RADI: Requirements Addition Index **Analysis Model :** RADI=NADR/TOR **Scale:** Ratio: Real from zero to infinite. **Unit:** Percentage	If RADI > 75 → RADI = 'Very High' If 50 < RADI ≤ 75 → RADI = 'High' If 25 <RADI≤50→'Medium' If 0 ≤ RADI ≤ 25 → 'Low'
RMI: Requirements Modification Index **Analysis Model :** RMI=NMR/TOR **Scale:** Ratio: Real from zero to infinite. **Unit:** Percentage	If RMI > 75 → 'Very High' If 50 < RMI ≤ 75 → 'High' If 25 < RMI ≤50→'Medium' If 0 ≤ RMI ≤ 25 → 'Low'

One important feature of SMF is the Software Measurement Modeling Language, which supports the metamodel with a graphical notation. This helps to build unambiguous and more comprehensive software measurement models than, for example, the use of tables or text. To illustrate this feature, the measurement model represented with the above tables

has been represented by using the SMML language Fig. 6). The NR, NMR and PMR measures have been omitted due to space limitations. The models are stored in XMI (XML Metadata Interchange) which facilitates their automatic management by the tools in the SMF framework.

4) Measurement execution: the source models used to carry out the measurement are: the measurement model (2nd step), the domain model (3rd step) and the extended QVT Relations model. The target model obtained is the measurement model with the defined Measurement Result. This process is completely transparent to the user. The results obtained are presented in the following table:

TABLE VII. RESULTS OF THE SOFTWARE MEASUREMENT PROCESS (REQUIREMENTS STABILITY)

Stability of baseline 5 with regard to previous baseline		
Cancelled	0	0.0%
Added	29	55.77%
Modified	23	44.23%
Approved	0	0.0%
Total	52	

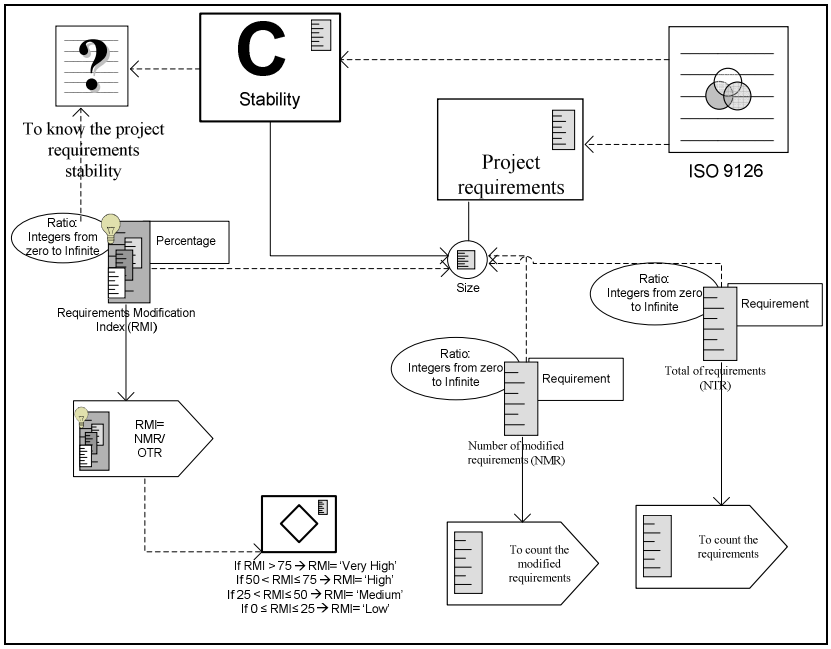

Figure 6. Software Measurement Model by using SMML (Requirements Stability).

B. Measurement of Database Complexity

The second software domain evaluated in the Health 2.0 project was that of "database development". One of the INDRA company's outstanding business goals is to support the evolution of their databases by providing the necessary means to facilitate the improvement (and consequently maintenance) of conceptual and logical database models. To achieve this, INDRA needs to know the maintainability (ease of maintenance) of its database conceptual schemas, represented with E/R notation, and the maintainability of its relational schemas. The measures proposed in [26] have therefore been applied. The database selected was ISHOS, which is used to manage the following aspects: information with regard to the patients admitted; the events and movements associated with the admitted patients; the situation with regard to hospital beds; the nurses' shifts; and the patients' records. As illustrated in the previous subsection, the following four steps were followed to carry out the measurement:

1) Incorporation of Relational Databases metamodel.

2) Creation of model conforms to Relational Database metamodel. In this case, the model (relational schema) is the hospital management domain which is composed of 29 tables with their corresponding primary keys, foreign keys and attributes (the model has not been represented due to space limitations).

3) Creation of measurement model conforms to SMM. A software measurement model, which includes the measures defined in [26] to evaluate the

maintainability of relational schemas, has been represented by using SMML language. This model is not illustrated in this paper owing to space limitations.

TABLE VIII. RESULT OF THE SOFTWARE MEASUREMENT PROCESS (DATABASE MAINTAINABILITY)

Maintainability of databases (ISHOS of Health 2.0 Project)	
NT (Number of tables)	29
NA (Number of attributes)	290
TMI (Tables Maintenance Index)	290/29 = 10% (Medium)
NFK (Number of Foreign Key)	40
SCI (Scheme Connectivity Index)	40/29 = 1,37 % (Medium)
RFK (Ratio of Foreign Key)	40/290 = 0,14

4) Measurement execution: The results obtained in order to execute the software measurement process are presented in Table VIII. This step is automatic and therefore completely transparent to the user.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, a generic framework for the definition of measurement models based on a common metamodel, denominated as SMF, has been applied to carry out the software measurement process in the INDRA Company. The benefits of the SMF (which follows the MDE paradigm to support model measurement) discovered have been confirmed in two main aspects:

- The software measurement process was homogenized by using a common reference framework. The company has perceived the benefits of using a flexible framework to measure any kind of model, regardless of the type of entity it characterizes. At present, diverse software tools are used to support measurement activities in the company. These tools are restricted to specific entities, are rigid (no new measures can be defined) and store results in highly heterogeneous formats which make the decision making process more difficult. With SMF, the measurement process is completely user-transparent, and the user task consists of selecting the domain metamodel (the domain to be measured) and defining the source models.
- Measurements can be modeled in the same way as that in which the company builds models for its software products (UML, databases, etc.). SMF supports this key aspect by means of a consistent metamodel and the SMML graphical notation The measurement models contain all the information necessary to support measurement activities and therefore facilitate the decision making process. These models are automatically managed by the framework in order to calculate the measures in the corresponding attributes of the entities.

Among related future works, one important effort is the realization of an Eclipse plug-in which will guide the user in the application of the SMF methods and will provide a smoother integration of the tools of which the SMF technological environment is composed. This plug-in will enable users to instantiate measurement models in an easy and intuitive manner. A further future work will be to align our metamodel with the Software Metrics Meta-Model (SMM) OMG proposal [27], which is at present in its development phase. Finally, we shall apply SMF with parameterized measurement methods to other real contexts to obtain further refinements and validation.

ACKNOWLEDGEMENT

This work has been partially financed by the following projects: INGENIO (Junta de Comunidades de Castilla la Mancha, PAC08-0154-9262), ESFINGE (Ministerio de Educación y Ciencia, TIN2006-15175-C05-05), ALTAMIRA (Junta de Comunidades de Castilla-La Mancha, Fondo Social Europeo, PII2I09-0106-2463) and MEDUSAS (Centro para el Desarrollo Tecnológico Industrial. Ministerio de Ciencia e Innovación, IDI-20090557).

REFERENCES

[1] N. Fenton and S. L. Pfleeger, "Software Metrics: A Rigorous & Practical Approach, Second Edition": PWS Publishing Company, 1997.

[2] F. García, M. Serrano, J. Cruz-Lemus, F. Ruiz, and M. Piattini, "Managing Software Process Measurement: A Metamodel-Based Approach", Information Sciences, vol. 177, pp. 2570–2586, 2007.

[3] J. Bézivin, F. Jouault, and D. Touzet, "Principles, standards and tools for model engineering", in Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'2005), pp. 28-29, 2005.

[4] B. Mora, F. García, F. Ruiz, M. Piattini, A. Boronat, A. Gómez, J. Á. Carsí, and I. Ramos, "Software Measurement by using QVT Transformation in an MDA context", in Proceedings of the 10th International Conference on Enterprise Information Systems - ICEIS 2008, vol. DISI, pp. 117-124, Barcelona (Spain), 2008.

[5] S. Komi-Sirviö, P. Parviainen, and J. Ronkainen, "Measurement Automation: Methodological Background and Practical Solutions-A Multiple Case Study", in Proceedings of the Seventh International Software Metrics Symposium (METRICS'01), London, 2001.

[6] R. Kempkens, P. Rösch, L. Scott, and J. Zettel, "Instrumenting Measurement Programs with Tools", in Proceedings of the PROFES 2000, Oulu, Finland, 2000.

[7] T. Jokikyyny and C. Lassenius, "Using the internet to comunicate software metrics in a large organization", in Proceedings of the Proceedings of GlobeCom'99, 1999.

[8] M. Brown and G. Dennis, "Measurement and Analysis: What Can and Does Go Wrong?" 10th IEEE International Symposium on Software Metrics (METRICS'04), pp. 131-138, 2004.

[9] M. Auer, B. Graser, and S. Biffl, "A Survey on the Fitness of Commercial Software Metric Tools for Service in Heterogeneous Environments: Common Pitfalls ", in Proceedings of the Ninth International Software Metrics Symposium. (Metrics '03), pp. 144, 2003.

[10] E. Palza, C. Fuhrman, and A. Abran, "Establishing a Generic and Multidimensional Measurement Repository in CMMI context ", in Proceedings of the 28th Annual NASA Goddard Software Engineering Workshop (SEW'03), pp. 12-22, Greenbelt (Maryland, USA), 2003.

[11] W. Harrison, "A flexible method for maintaining software metrics data: a universal metrics repository", Journal of Systems and Software vol. 72, pp. 225-234 2004.

[12] L. Lavazza and A. Agostini, "Automated Measurement of UML Models: an open toolset approach", Object Technology, vol. 4(4), pp. 115-134, 2005.

[13] M. Scotto, A. Sillitti, G. Succi, and T. Vernazza, "A relational approach to software metrics", in Proceedings of the Proceedings of the 2004 ACM symposium on Applied computing (SAC'2004), pp. 1536-1540, Nicosia, Cyprus, 2004.

[14] F. García, F. Ruiz, J. Cruz, and M. Piattini, "Integrated measurement for the evaluation and improvement of software processes", in Proceedings of the Proceedings of the 9th European Workshop on Software Process Technology (EWSPT'9), Lecture Notes in Computer Science, vol. 2786, pp. 94-111, 2003.

[15] M. Monperrus, J.-M. Jéezéquel, J. Champeau, Brigitte, and Hoeltzener, "A Model-Driven Measurement Approach", in Proceedings of the Proceedings of the ACM/IEEE 11th International Conference on Model Driven Engineering Languages and Systems (MODELS'2008), vol. 5301/2008, pp. 505-519, Nantes (France), 2008.

[16] B. Mora, F. García, F. Ruiz, and M. Piattini, "SMML: Software Measurement Modeling Language", in Proceedings of the The 8th OOPSLA Workshop on Domain-Specific Modeling, Nashville (Tennessee) EU, 2008.

[17] F. García, M. F. Bertoa, C. Calero, A. Vallecillo, F. Ruíz, M. Piattini, and M. Genero, "Towards a consistent terminology for software measurement", Information and Software Technology vol. 48 (8), pp. 631-644 2006.

[18] F. García, F. Ruiz, C. Calero, M. F. Bertoa, A. Vallecillo, B. Mora, and M. Piattini, "On the Effective Use of Ontologies in Software Measurement", The Knowledge Engineering Review (in press), vol. 0, pp. 1-24, 2008.

[19] OMG, "QVT Standard Specification", 2005.

[20] F. García, M. Piattini, F. Ruiz, G. Canfora, and C. A. Visaggio, "FMESP: Framework for the modeling and evaluation of software processes", Journal of Systems Architecture - Agile Methodologies for Software Production, vol. 52, pp. 627-639, 2006.

[21] K. Smolander, K. Lyytinen, V.-P. Tahvanainen, and P. Marttiin, "MetaEdit: A flexible graphical environment for methodology modelling", in Proceedings of the CAiSE'91, 3rd Intl. Conference on Advanced Information Systems Engineering, vol. 498/1991, pp. 168-193, 1991.

[22] J.-P. Tolvanen, R. Pohjonen, and S. Kelly, "Advanced Tooling for Domain-Specific Modeling: MetaEdit+", in Proceedings of the The 7th OOPSLA Workshop on Domain-Specific Modeling, 2007.

[23] "Medini QVT Home Page", 2009.

[24] A. Boronat and J. Meseguer, "Algebraic Semantics of EMOF/OCL Metamodels", CS Dept., University of Illinois at Urbana-Champaign Technical Report UIUCDCS-R-2007-2904, 2007.

[25] M. Genero, M. Piattini, and C. Calero, "A Survey of Metrics for UML Class Diagrams", Journal of Object Technology vol. 4(9,), pp. 59-92, 2005.

[26] C. Calero, M. Piattini, and M. Genero, "Empirical validation of referential integrity metrics", Information & Software Technology. Special Issue on Controlled Experiments in Software Technology, vol. 43(15), pp. 949-957, 2001.

[27] OMG, "Architecture-Driven Modernization (ADM): Software Metrics Meta-Model (SMM). OMG Document: dmtf/2007-03-02", Object Management Group 02-03-2007 2007.