Available online at www.sciencedirect.com
ScienceDirect
Information Sciences xxx (2007) xxx–xxx
INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL
www.elsevier.com/locate/ins

Managing software process measurement: A metamodel-based approach

F. García *, M. Serrano, J. Cruz-Lemus, F. Ruiz, M. Piattini,
ALARCOS Research Group

Technologies and Information Systems Department, UCLM-Soluziona Research and Development Institute, University of Castilla-La Mancha, Paseo de la Universidad, 4 – 13071 Ciudad Real, Spain

Received 15 March 2006; received in revised form 20 October 2006; accepted 20 January 2007

Abstract

The evaluation of software processes is nowadays a very important issue due to the growing interest of software companies in the improvement of the productivity and quality of delivered products. Software measurement plays a fundamental role here. Given the great diversity of entities which are candidates for measurement in the software process improvement context (process models, projects, resources, products) this measurement must be performed in a consistent and integrated way. This will facilitate the making of decisions in process improvement. In this paper, a proposal for the integrated management of the software measurement is presented. The goal is to provide companies with a generic and flexible environment for software measurement which facilitates and establishes the basis for a common and effective measurement process and which is not restricted to only one kind of software entity or to a single quality or evaluation model. In order to achieve this, the proposal adopts the Model Driven Engineering philosophy and provides: a metamodel for the definition of software measurement models; a flexible method to measure any kind of software entity represented by its corresponding metamodel and GenMETRIC, which is the software tool that supports the framework.

Keywords: Software process; Measurement; Metamodelling

1. Introduction

Software measurement plays a fundamental role in Software Engineering [8]. Measurement can help to address some critical issues in software development and maintenance by facilitating the making of decisions. Software measurement provides a support for planning, monitoring, controlling and evaluating the software process.

* Corresponding author. Tel.: +34 926295300; fax: +34 926295354.
E-mail addresses: Felix.Garcia@uclm.es (F. García), Manuel.Serrano@uclm.es (M. Serrano), JoseAntonio.Cruz@uclm.es (J. Cruz-Lemus), Francisco.RuizG@uclm.es (F. Ruiz), Mario.Piattini@uclm.es (M. Piattini).

0020-0255/$ - see front matter
doi:10.1016/j.ins.2007.01.018

It is, specifically, the evaluation of software processes that has become a very important issue recently, due to the growing interest of software companies in the improvement of the productivity and quality of delivered products. Moreover, the current competitive marketplace forces software organizations to improve their processes on a continual basis. To achieve this, successful management is necessary [9], and this involves the definition, measurement, control and improvement of the process.

Companies therefore require the carrying out of the software measurement process to be made in an effective and consistent way. The quantitative basis necessary for the identification of the areas which are candidates for improvement can thus be established. This implies the need for a disciplined approach to measurement and data analysis, if a software or systems engineering enterprise is to succeed [4].

In the context of software process measurement the following basic types of entities can be identified as candidates:

- *Software Process Models*. These models constitute the starting point for the understanding and carrying out of the software process through their enactment in concrete projects. Software process modelling has become a highly acceptable solution for treating the inherent complexity of software processes, and a great variety of modelling languages and formalities, known as "Process Modelling Languages" (PML), can be found in the relevant literature. With a software process model (SPM) the different elements related to a software process are represented precisely and without ambiguity.
- *Software Projects*. These are concrete enactments of software process models and their measurement is fundamental if we are to know how they perform, and are measured mainly through schedule and cost and resource-related metrics.
- *Software Products*. As a result of carrying out software projects, different products may be obtained and these are also candidates for measurement. The quality of the process has to be reflected in the products obtained and that is why software products themselves must be measured.

The great diversity in the kinds of entities which are candidates for measurement in the context of the software processes points to the importance of providing the means necessary to define measurement models in companies in an integrated and consistent way. This involves providing companies with a suitable and consistent reference for the definition of their software measurement models as well as the necessary technological support to integrate the measurement of the different kinds of entities.

In addition, we are currently witnessing a new focus in the development and maintenance of software systems: Model Driven Engineering. The Model Driven Architecture (MDA) proposal [25] and its related standards, such as MOF (Meta Object Facility) [26], XML Metadata Interchange (XMI) [27] and Unified Modeling Language (UML) [28], reinforce this new approach to specifying and building systems by giving special attention to models and metamodels. Given the growing complexity of software systems, this new approach seeks to isolate business logic from the implementation level by building platform-independent models in the context of a framework which organizes the necessary metamodels efficiently.

Software measurement integration can therefore be achieved by adopting the MDA approach. This implies the definition of the measurement models in a homogeneous and consistent way by using a suitable metamodel. It also involves the measurement of any software entity through the metamodel which defines them.

In this paper, we present a proposal which supports the consistent and integrated measurement of software. This is achieved by providing a generic measurement metamodel to represent the metadata related to the measurement process along with a set of generic metrics defined within the metamodel scope. The proposal is supported by GenMETRIC a tool for software measurement that is both generic and extensible. The paper is organised as follows. Section 2 provides an overview of the related works and in Section 3 the conceptual architecture of the proposal to manage software measurement is described in the context of MDA. Section 4 presents a measurement metamodel which manages the measurement process by means of a suitable language for defining concrete software measurement models. In Section 5, the measurement based on metamodels is illustrated with an example. Section 6 describes the GenMETRIC tool, developed to support generic and extensible software measurement. Finally, conclusions and future work are outlined in Section 7.

2. Related works

Before introducing improvement plans in companies, a quantitative basis for evaluating their software processes should be established. This fact is evidenced by the central role that measurement has in the current standards and models for process maturity and improvement such as CMMI [31], ISO 15504 [16] and the ISO/IEC 90003 [17]. By measuring software processes we can control them and, as stated by these recognised standards and models, we can improve the process maturity, which cannot be achieved without any measurement support. From the methodological perspective, software measurement is supported by a wide variety of proposals, with the Goal Question Metric (GQM) method [34], the Practical Software & Systems Measurement (PSM) methodology [24] and the ISO 15539 [18] and IEEE 1061-1998 [15] standards deserving special attention.

It is widely accepted that by using an automatic measurement tool to measure software processes and products, the measurement process can be improved by avoiding calculation errors thereby reducing effort in measurement and providing some enhanced analysis tools [22]. Other important improvements are to avoid bias in the measurement process (inaccurate or inflated data) which could be the result of human intervention. Moreover, the tool can store the results and allow people to automatically obtain reports and perform historical comparisons to facilitate decision making processes.

The importance of software measurement support in organizations has led to the development of a great diversity of tools. An exhaustive list of software measurement tools can be found in [7,35]. Most of the tools are focused on measuring one kind of entity (concrete systems, paradigms, programming languages, etc.). However, a wider support is required to integrate the measurement process in the context of organizations' process improvement and there are not many tools which support the measurement process as a whole. With regard to this issue we observe tools such as MMR [29] which is based on the CMMI model for the evaluation of software processes. Others tools are based on measuring several kinds of entities by performing SQL queries on repositories [23] [14,30], but this kind of user interface can decrease their usability for novice database users.

Nevertheless, from our point of view many of these tools are restricted to specific domains, measurement standards or evaluation and process quality models, which may reduce their generality and scope. Each tool works according to its own philosophy, collects its own data and calculates specific measures. Currently, companies have to measure highly heterogeneous entities whose results should be stored and processed from a common repository to facilitate the making of decisions, as they can not only depend on specific software artifacts.

The idea of flexible measurement frameworks has been applied to the domain of object oriented systems, like in the by Vaishanavi et al. proposal [33] which captures the generic structure of the object oriented product metrics space.

With the aim of providing a generic support for software measurement that is not restricted to only one kind of entity to be measured or to any single quality or evaluation model and which facilitates and provides the basis for a common measurement process in companies, we have developed a generic approach which is described in the following sections.

3. Conceptual architecture for integrating software measurement

In this section, the conceptual architecture to manage the software measurement is described, first by describing how the MDA has been applied, and then by explaining the elements of which it is made up.

3.1. Applying the model-driven paradigm to software measurement

In any engineering discipline the rigorous analysis of a design artifact takes place through the representation and manipulation of mathematical objects called models. These models are then used to develop the prototypes, and later the complete engineering system [32]. The new Model-Driven Engineering paradigm (MDE) [2] is an attempt to apply these ideas to the problem of designing and constructing software systems. Its main goal is to ensure that the core artifacts in the software engineering processes will be models instead of code, so

Please cite this article in press as: F. García et al., Managing software process measurement: ..., Informat. Sci. (2007), doi:10.1016/j.ins.2007.01.018

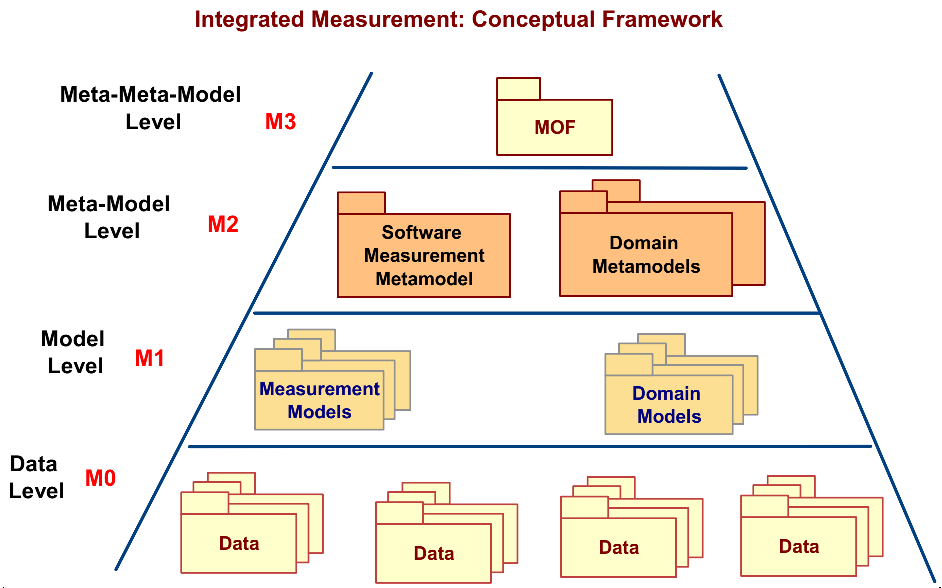

Fig. 1. Conceptual framework to manage software measurement.

that designs are expressed and managed in the manner of models with a much higher level of abstraction than the code. This has two main advantages: (i) to reduce complexity during the design phase when that complexity does not need to be born in mind due to the specific technology of implementation, and (ii) to obtain generic systems, because it is possible to create software with more general functionalities that give solutions to a greater quantity of cases or situations.

MDA [25] is the OMG proposal by which to carry out the MDE paradigm. The core of MDA is a set of standards, amongst which it is necessary to highlight the "Meta-Object Facility" (MOF) [26], a conceptual architecture with four levels of modeling (see Fig. 1) that allows us to define models (level M1, for instance, a UML class diagram for a concrete application) based on metamodels (level M2, for instance, UML), which in turn are all defined by means of a universal object-oriented and auto-defined meta-metamodel (level M3). A necessary complement of the MOF is the XML Metadata Interchange (XMI) standard [27], which defines an XML-based language for the representation and exchange of MOF metamodels and models.

The same ideas as the MDE paradigm have been used for the development of more powerful CASE tools. In this way, it has been possible to build "Software Engineering Meta-Environments" [32], which is to say, integrated sets of tools that can be adapted to different domains and languages.

In a similar way, in our work we apply the ideas of MDE and MDA in order to create a framework that facilitates the software measurement process. On one hand, we use metamodels (all of which are based on a common software measurement ontology) (Section 4) to represent all the types of measurable elements in a software project (design diagrams, source code, documentation, data of the running projects, etc...), and on the other hand we have developed a tool called GenMetric which allows us to define "Software Measurement Models" for any software entity. Thanks to this, it is possible to define any type of measurement for any property of any software element. This work is presented with more detail in the following sections.

3.2. Elements of the conceptual architecture

The proposal for integrating software measurement described in this paper is part of the FMESP framework [11], which provides the support necessary for the representation and management of knowledge related to software processes from the perspectives of modelling and measurement. We focus on the measurement support of the framework whose elements are described in great detail according to the three layers of abstraction of metadata that they belong to, in line with the MOF standard. In Fig. 1, the conceptual architecture for integrated measurement is represented:

Please cite this article in press as: F. García et al., Managing software process measurement: ..., Informat. Sci. (2007), doi:10.1016/j.ins.2007.01.018

As can be observed in Fig. 1, the architecture has been organised into the following conceptual levels of metadata:

- *Meta-MetaModel Level (M3)*. At the highest conceptual level (M3), an abstract language for the definition of metamodels is found. This is the MOF language, which is basically composed of two structural constructors: MOF class and MOF association. With these levels the integration of different domains is possible. In pursuit of our objective to support the software measurement of any software entity in a consistent and integrated way, we use the MOF for the definition of the elements in level M2. That is to say, we employ a metamodel for the definition of software measurement models as well as the metamodels which represent the measured entities, called *domain metamodels*.
- *Metamodel Level (M2)*. In the M2 level, generic metamodels which are useful for the creation of specific models should be included. In our framework, the generic metamodels required are:
- *Measurement Metamodel*, to define specific measurement models. This metamodel is described in greater detail in Section 4.
- *Domain Metamodels*, to represent the kinds of entities which are candidates for measurement in the context of the evaluation of the software processes. These kinds of entities can vary from the company's SPMs themselves, to the projects carried out and the resources necessary, as well as the products used, modified and produced. For example, if a company requires the measurement of conceptual databases which have been developed with the Entity Relationship notation or the measurement of its UML class diagrams, the corresponding metamodels which allow the definition and representation of these kinds of entities (UML and E/R) must be included in the M2 level of the architecture.
- *Model Level (M1)*. At this level specific models are included. These models may be of two types:
  - *Measurement Models*. These models are instances of the measurement metamodel of the M2 level and they are defined in such a way as to satisfy some of the company's information needs. For example, if a company needs to know the size of its E/R diagrams, a measurement model could be defined in which the kind of software entity measured would be the E/R models. In this way, by using the measurement metamodel, all the measurement models developed in the company would be consistently represented and managed independently of the kinds of entities that they evaluate.
  - *Domain Models*, which are defined according to their corresponding domain metamodels. Examples of domain models are: the UML models (use cases, class diagrams, etc.) of a software application for managing a banking service, or the E/R model of the database of this application which is defined with the UML and E/R metamodels of the M2 level. The domain models are the entities whose attributes are measured by calculating the measurements defined in the corresponding measurement models.

With the conceptual architecture proposed it is possible to include specific measurement models for the evaluation of different kinds of entities: products such as relational databases [5], object-relational, UML class or state transition diagrams [13]; software projects, by having adequate metamodels such as the example in [1]; and software process models [11]. The proposal provides companies with the conceptual support necessary to carry out and store the results of their measurement processes in an integrated and consistent way and also avoids the development of specific tools for the measurement of each new kind of entity required.

4. Software measurement metamodel

A fundamental element to take into consideration when establishing a process improvement initiative is the possibility of defining objective process indicators that will allow a software company to evaluate and improve its processes efficiently at any given moment. A measurement process framework must be established when this is taking place.

Human and technical factors are fundamental to the success of software measurement. The human aspect is vitally important in the identification of business needs, the corresponding measures which satisfy them and the building of knowledge and decisions from measurement results. Moreover, managers and software developers involved in the measurement process must perceive that their measurement programs are aligned with their expectations. The technical support, as stated in Section 2, can help by enabling people to collect and interpret the measurement results in the best way. From these perspectives, one significant problem in collecting data in a measurement process is mainly due to a poor definition of the software measures being applied [21]. Thus it is important not only to gather the values pertaining to the measurement process, but also to represent the metadata associated with this data appropriately, i.e., to develop a measurement metamodel which can be the reference for companies in the definition of their measurement models. In this way, all the data and metadata related to the measurement of the different kinds of relevant entities in companies would be represented homogeneously. This metamodel could also be the basis for the development of a measurement repository as required, to therefore improve process maturity according to CMMI or ISO 15504.

To establish and clarify the elements involved (concepts and relationships) in the software measurement domain before designing the metamodel, an ontology for software measurement was developed [10]. This ontology enabled us to identify all the concepts, provide precise definitions for all the terms, and clarify the relationships between them. Moreover, this common ontology has served as the basis for us to compare the different standards and proposals, thus helping to achieve the required harmonization and convergence process for all of the aforementioned. Based on the concepts and relationships stated in the ontology, the measurement metamodel was derived. Fig. 2 shows the UML diagram (MOF compliant) which displays the main elements of the Software Measurement Metamodel:

The Software Measurement Metamodel is organized around four main packages (see Fig. 2):

- *Software Measurement Characterization and Objectives*, which includes the concepts required to establish the scope and objectives of the software measurement process. The main goal of a software measurement

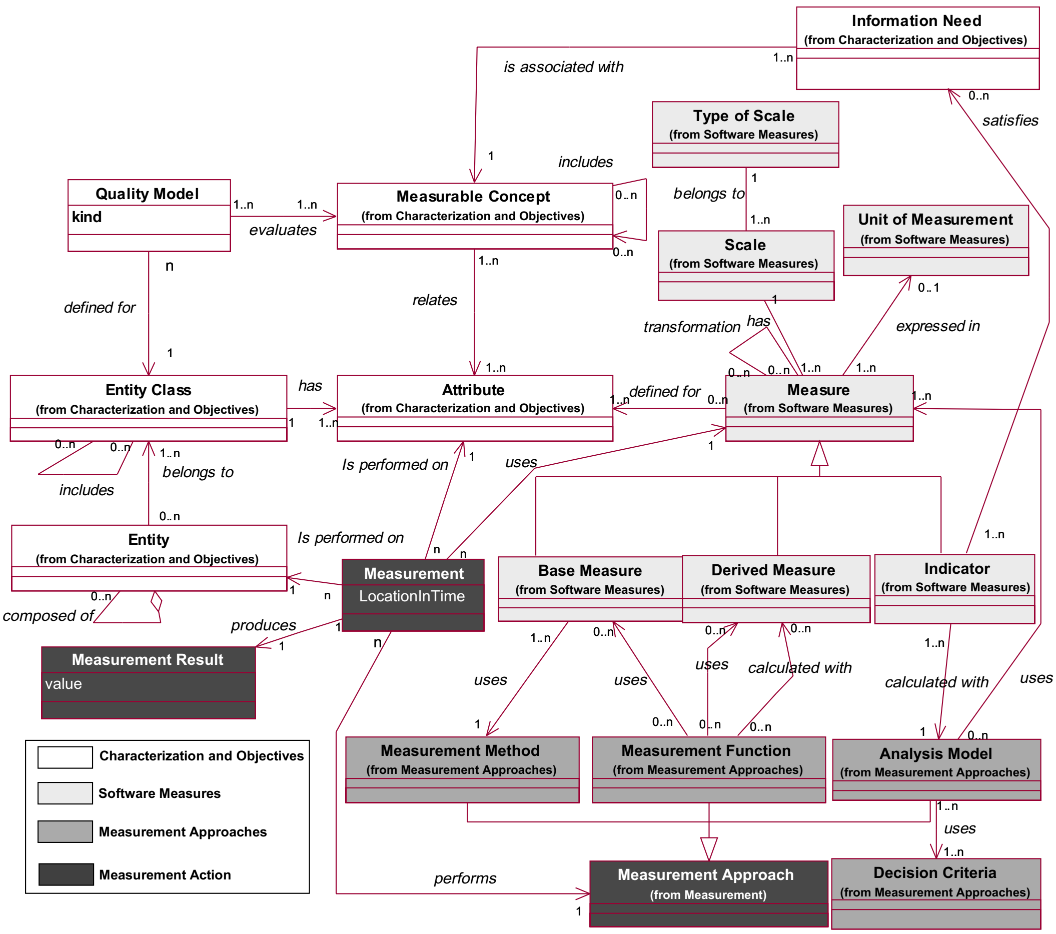

Fig. 2. Software measurement metamodel.

process is to satisfy certain *information needs* by identifying the *entities* (which belong to an *entity class*) and the *attributes* of these entities (which are the object of the measurement process). *Attributes* and *information needs* are related via *measurable concepts* (which belong to a *quality model*).

- *Software Measures*, which aim at establishing and clarifying the key elements in the definition of a software *measure*. A *measure* relates a defined *measurement approach* and a *measurement scale* (which belongs to a *type of scale*). A *measure* is expressed in a *unit of measurement*, and can be defined for more than one *attribute*. Three kinds of measures are defined: *base measures*, *derived measures*, and *indicators*.
- *Measurement Approaches*. This sub-ontology introduces the concept of *measurement approach* to generalize the different approaches used by the three kinds of *measures* for obtaining their respective *measurement results*. A *base measure* applies a *measurement method*. A derived measure uses a *measurement function* (which rests upon other *base* and/or *derived measures*). Finally, an indicator uses an *analysis model* (based on a *decision criteria*) to obtain a measurement result that satisfies an information need.
- *Measurement*. This establishes the terminology related to the act of measuring software. A *measurement* (which is an action) is a set of measurement results, for a given *attribute* of an *entity*, using a *measurement approach*. *Measurement results* are obtained as the result of performing *measurements* (actions).

5. Example of application

To illustrate the benefits of the proposal consider the following example: The main activity of a software company is the development and maintenance of database applications. According to what its main activities are, one relevant business goal of the company is to support the evolution of their databases by providing the necessary means to facilitate the improvement (and consequent maintenance) of the conceptual and logical models of its databases. To achieve this, the company needs to know the maintainability (easiness of maintenance) of its database conceptual schemas, represented with E/R notation, and the maintainability of its relational schemas. These are the information needs of the company. The aim is to build more maintainable databases to facilitate its evolution. The following figure illustrates how the current proposal can be applied to the management of the measurement process according to the information needs of the company:

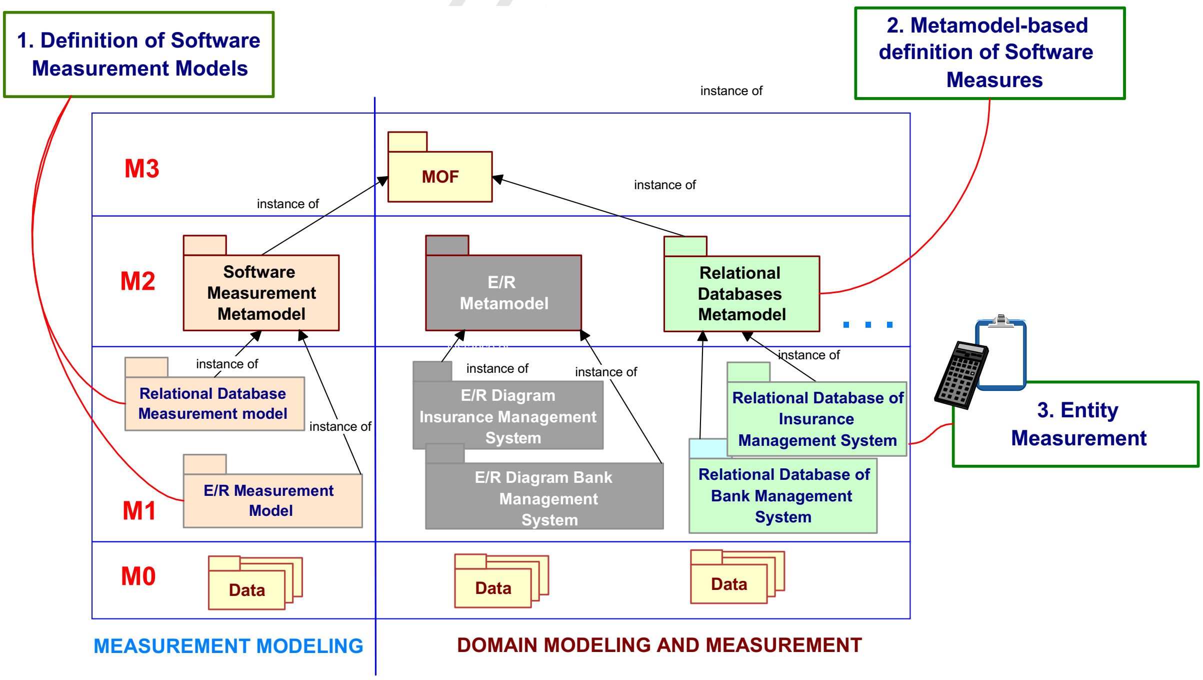

Fig. 3. Example of FMESP measurement application for database evaluation.

Please cite this article in press as: F. García et al., Managing software process measurement: ..., Informat. Sci. (2007), doi:10.1016/j.ins.2007.01.018

As can be observed in Fig. 3, the first step in the application of the proposal is to define the measurement models for the fulfilment of the company's information needs in order for it to attain its business goals. Based on these information needs, which drive the measurement process, two measurement models can be defined: A measurement model to evaluate the E/R diagrams' maintainability and a measurement model to evaluate relational schemas. To satisfy the information need of the first model, the kind of entity candidate which should be used for measurement is ''E/R Diagrams''. The attributes to evaluate could be ''size'' and ''complexity'' and based on these we can include in the model measures as proposed in [12]. More detailed information about this measurement model can be found in [6]. Once the measures have been included in the measurement model, they have to be defined according to the E/R metamodel elements and finally, the measures are automatically calculated in concrete E/R diagrams (level M1) represented as instances of the E/R metamodel. The same steps are applied to the measurement of relational schemas, which are described with more detail in the following subsections.

5.1. Definition of a relational schemas maintainability measurement model

The measurement model to evaluate maintainability of relational schemas is based on the proposal of measures of Calero et al. [5]. In Fig. 4, the object diagram which represents the measurement model of Relational Schemas Maintainability as an instance of the package ''Software Measurement Characterization and Objectives'' (see Section 4), is shown:

As we can observe in Fig. 4, the context of the measurement model for Relational Schemas Maintainability is established by identifying the information needs to be satisfied, the measurable concepts, the kind of entity candidate for measurement, attributes to be evaluated and the related quality model. In this model, we also include the concrete entities to be evaluated such as, for example, the relational schemas obtained as a result of a bank management system and an insurance management system. These entities are those which are used to carry the measurement process out (see Section 5.3).

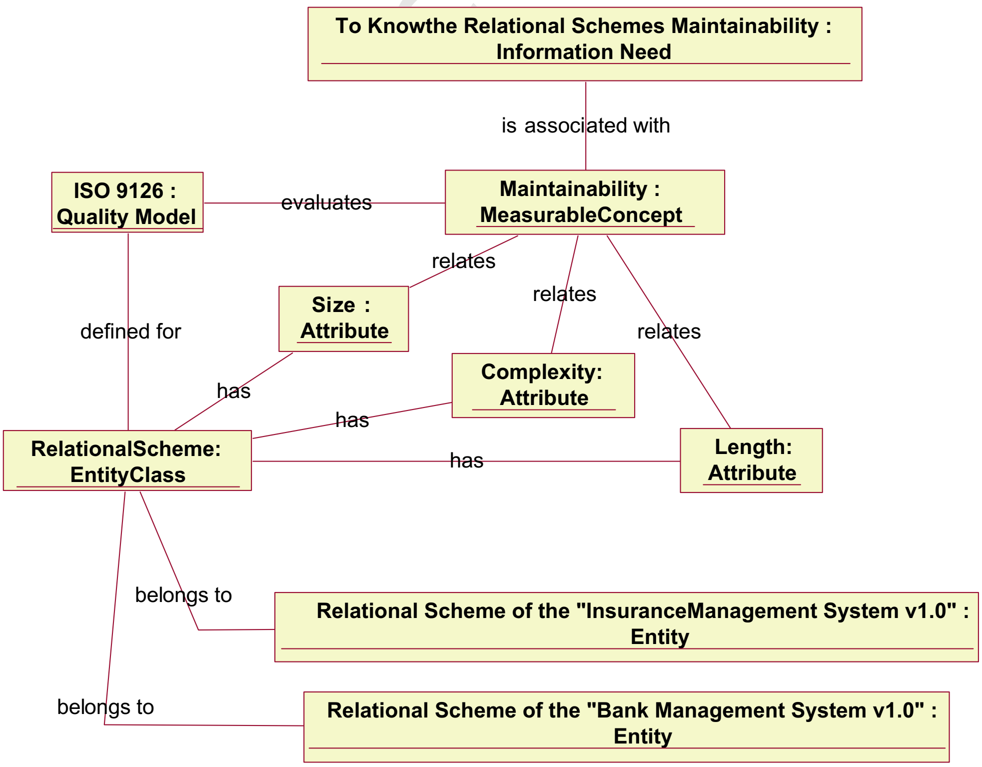

Fig. 4. Relational scheme measurement model: software measurement characterization and objectives package instance.

From the identified attributes to be evaluated, the next step is the definition of the software measures, which should clearly identify the attributes they evaluate, the scale they belong and the unit in which they are expressed. In Fig. 5, the object diagram which represents the part of the measurement model for Relational Schemas, in the instance of the "Software Measures Package", is shown:

As can be observed in Fig. 5, for the evaluation of the identified entity attributes, three kind of measures have to be defined. The first kind is that of "base measures", which are obtained directly by applying a measurement method to quantify the attribute of interest. The base measures of the measurement model for Relational Schemes have been adopted from the proposal in [5]. Table 1 shows the defined base measures and measurement methods in greater detail:

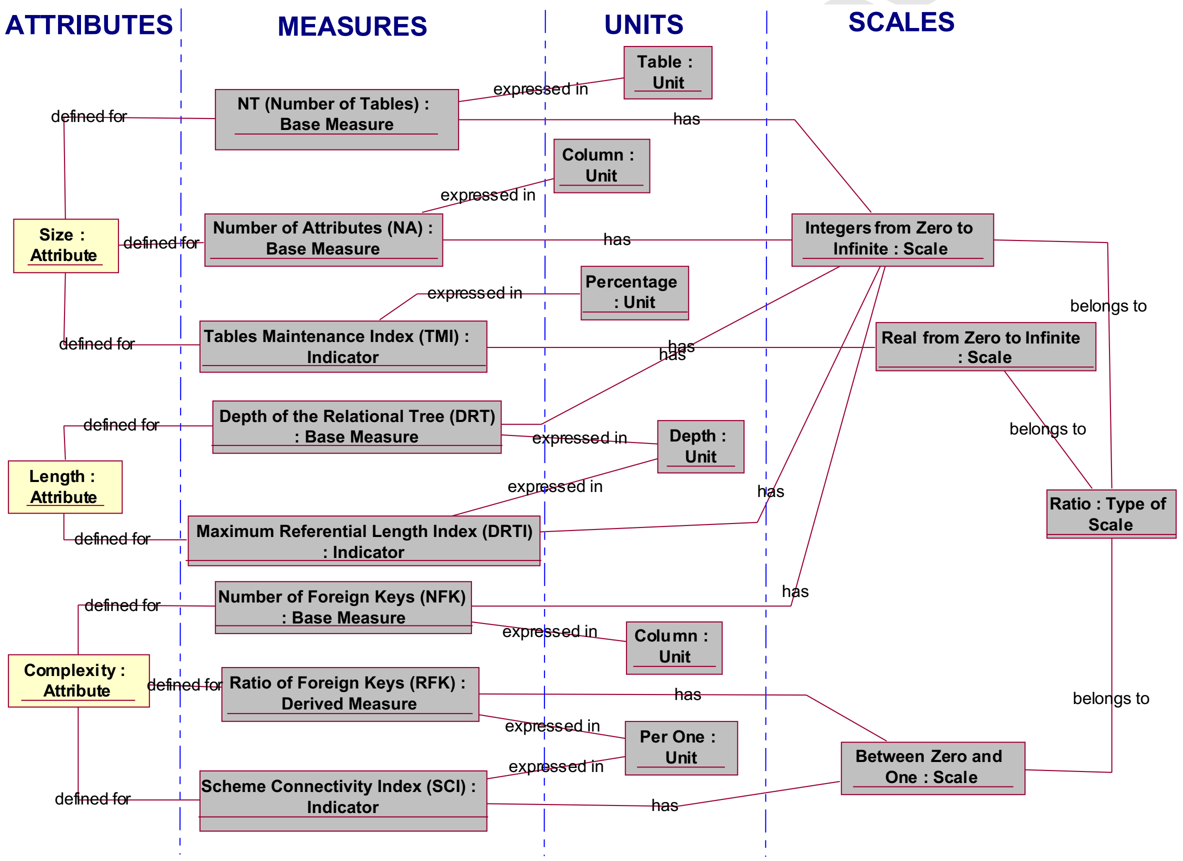

Fig. 5. Relational scheme measurement model: software measures package instance.

Table 1
Base measures and measurement methods of the relational schemas measurement model

Base measure	Description	Measurement method
NT	Number of *Tables* in the scheme	*To count* the *Tables* in the schema
NFK	Number of *Foreign Keys* in the scheme	*To count* the *Foreign Keys* in the schema
NA	Number of *Attributes* in the scheme	*To count* the *Attributes* in the schema
DRT	Depth of relational tree (DRT). The number of tables which are part of the longest path obtained by following the referential integrity relationships between tables	To calculate the maximum depth of the paths obtained by following all the possible foreign keys in the schema

Table 2
Indicators, analysis models and decision criteria of the relational schemas measurement model

Indicator	Description	Analysis model	Decision Criteria
TMI	Tables Maintenance Index. This is obtained by establishing the proportion of attributes and tables in the relational scheme. The higher this measure is the more difficult it is to maintain the tables in the scheme	TMI = NA/NT	If TMI > 18 → TMI = 'Very High' If 12 < TMI ⩽ 18 → TMI = 'High' If 6 < TMI ⩽ 12 → TMI = 'Medium' If 0 ⩽ TMI ⩽ 6 → TMI = 'Low'
SCI	Scheme Connectivity Index. This is the proportion of foreign keys and tablesin the scheme. The higher this measure is the more difficult it is to maintain the relational scheme.	SCI= NFK/NT	If SCI ⩾ 2 → SCI = 'Very High' If 1,5 ⩽ SCI < 2 → SCI = 'High' If 1 < SCI < 1,5 → SCI = 'Medium' If 0,5 ⩽ SCI ⩽ 1 → SCI = 'Low' If 0< SCI < 0,5 → SCI = 'Very Low'
DRTI	Maximum Referential Length Index. Indicator based on the DRTI measure. The higher this measure is the more difficult it is to maintain the relational scheme	DRTI = DRT	If DRTI > 15 → DRTI = 'Very High' If 8 < DRTI ⩽ 15 → DRTI = 'High' If 2 < DRTI ⩽ 8 → DRTI = 'Medium' If 0 ⩽ DRTI ⩽ 2 → DRTI = 'Low'

Once the base measures have been defined, the next step is the definition of the derived measures. These are calculated by applying a measurement function to other base/derived measures. In the measurement model for relational schemas maintainability the derived measure Ratio of Foreign Keys (RFK) has been included which is calculated by applying the measurement function: RFK = NFK/NA. Finally, the indicators are defined by using the other measures as a base, and these are the kinds of measures that can satisfy the information needs defined in the model. In Table 2, the indicators, analysis models and decision criteria defined to satisfy the information need "To know the maintainability of relational schemas" are shown:

To complete the measurement model, the instances of the packages "Measurement Approaches" and "Measurement Action" should be included. The part of the model corresponding to the package "Measurement Approaches" can be obtained from the measures definition as shown in Tables 1 and 2. Finally, according to the measurement model defined, the measurement process can be executed. This consists of evaluating concrete relational schemas (entities) by calculating the measures defined (see Section 5.3). The results will be represented as an instance of the package "Measurement Action".

5.2. *Metamodel-based definition of measures*

The proposal in this paper provides support for the measurement of any software entity and, to achieve this, the measures definition is performed by analysing the metamodels which represent these entities (domain metamodels). This analysis is based on the definition of the base measures based on the classes (constructors) and relationships included in the metamodel. In Fig. 6, the Relational Metamodel, included in level 2 of the conceptual architecture as a domain metamodel, is represented by using the graphical notation of UML:

The diagram in Fig. 6 includes the constructors necessary to define relational schemas. A relational scheme is composed of tables which include attributes. The attributes can be part of a key, whose types are: foreign keys which reference concrete tables or primary keys that identify the table rows.

Once the metamodel which represents the software entity to be measured is known, the next step in the definition of the measurement model is to define or adopt (from known proposals) the necessary measures by which to evaluate the attributes considered.

The base measures of the measurement model can be obtained by using the elements of the Relational Scheme metamodel (MOF-classes and MOF associations) as a base. The base measures defined in the measurement model described in Section 5.1 are calculated by applying two basic kinds of measurement methods to the elements of the metamodel (classes or associations):

–*Count*. To count the number of instances of a class or relationship in the metamodel. The measures "Number of Tables", "Number of Foreign Keys" and "Number of Attributes" are obtained by counting the instances of the classes "Tables", "Foreign Keys" and "Attributes".

Please cite this article in press as: F. García et al., Managing software process measurement: ..., Informat. Sci. (2007), doi:10.1016/j.ins.2007.01.018

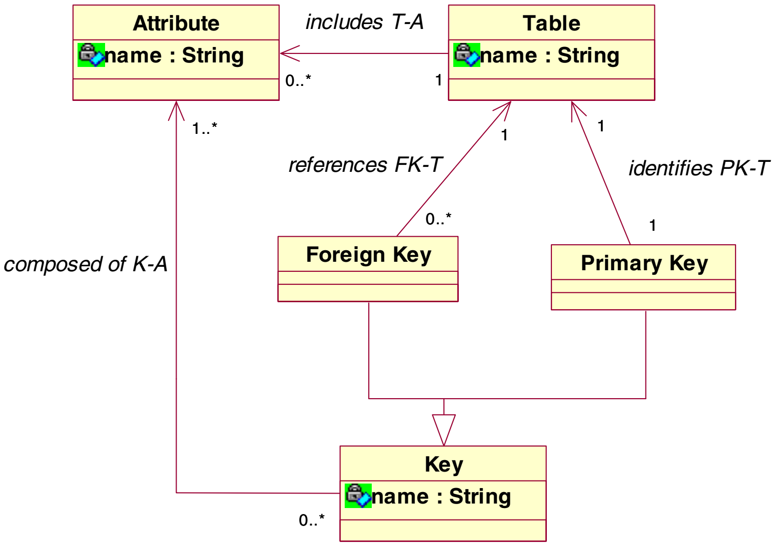

Fig. 6. Relational metamodel.

–*Graph Length*. To calculate the maximum path obtained by following a concrete relationship in the metamodel. This measurement method is applied by considering the metamodel as a graph in which the nodes are the objects and the edges are the relationships. The measure "Depth of the Relational Tree" is obtained by applying this measurement method.

The former measurement methods can be successfully applied to the definition of any base measure which is related to the structural complexity of the entity to be evaluated. For the measurement of other aspects, new measurement methods can be incorporated.

5.3. *Entity measurement*

The last step in the measurement process is to collect the values of the defined measures in order to satisfy the information needs. In order to achieve this, the entities candidate to be measured must be evaluated. In the example shown, the evaluation should be performed on relational schemas which must be defined as instances of the Relational Metamodel (on which the base measures have been defined).

Fig. 7 shows a relational schema defined as an instance of the Relational Metamodel:

The example in Fig. 7 illustrates a simple relational schema of the "Bank Management" system, and is composed of two tables: "Bank Account" and "Client". The "Client" table includes the columns (attributes) "Id_C" of which the primary key is "Name" and "Address". The "Bank Account" table includes the attributes: "number", "office", of which the primary key is both "balance" and "owner", which is the foreign key to the "Client" table.

The calculation of the base measures defined in the Relational Measurement model (Section 5.2) has to be performed in the following way:

–The measures "Number of Tables", "Number of Foreign Keys" and "Number of Attributes" are obtained by applying the measurement method "Count", i.e. by counting the instances of the "Tables", "Foreign Keys" and "Attributes" classes. In this example the values are, respectively, 2, 1 and 7.
–The measure "Depth of the Relational Tree" is obtained by applying the "Graph Length" measurement method to the "foreign key references table" metamodel association and its value is 1, so only this link needs to be computed.

Please cite this article in press as: F. García et al., Managing software process measurement: ..., Informat. Sci. (2007), doi:10.1016/j.ins.2007.01.018

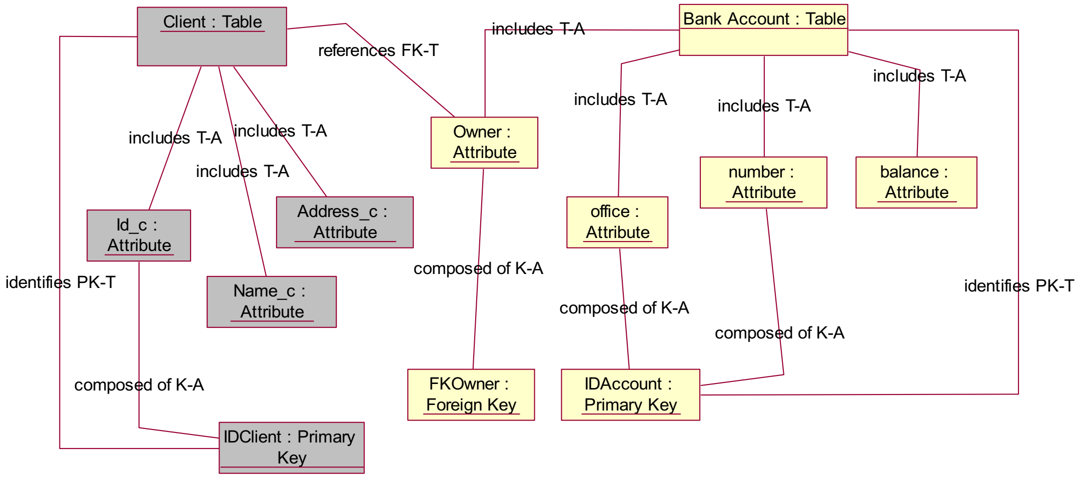

Fig. 7. Relational scheme of the "Bank Management System" entity.

On the other hand, derived measures and indicators are calculated according to their measurement functions. For example, the value of the measure Ratio of Foreign Keys (RFK) in the example is 0,142 (1/7).

6. GenMETRIC tool

With the aim of supporting the integrated measurement process commented on in previous sections, the GenMETRIC tool has been developed. GenMETRIC is a tool for the definition, calculation and visualisation of software metrics. This tool supports the management of the measurement process by supporting the definition of measurement models, the calculation of the measures defined in the measurement models and the presentation of the results in graphical and tabular ways. For the definition of measurement models the tool is based on the measurement metamodel presented in Section 3. Two key characteristics of GenMETRIC are that it is:

- *Generic*. With this tool it is possible to measure any software entity. The requirement necessary to achieve this is that the metamodel which represents the software entity (domain metamodel) must be included in the repository of the tool. As has been mentioned in previous sections, the measures are defined on the elements of the domain metamodels. This implies that in order to measure new entities it is not necessary to add a new code to GenMETRIC.
- *Extensible*. GenMETRIC supports the definition of any software measure. The base measures are defined on the domain metamodel elements (classes and associations) by using standard measurement methods such as "count" or "graph length". For the definition of derived measures and indicators the tool includes an evaluator of arithmetical and logical expressions (see Fig. 9).

To support generic and extensible software measurement the tool has been developed as part of a software engineering environment (SEE) which supports the conceptual architecture proposed to integrate measurement. This SEE is shown in Fig. 8:

As we can observe in Fig. 8, GenMETRIC is the key tool of the SEE developed for software measurement. The metadata of the SEE (metamodels and models) are stored in a repository as XMI documents. The necessary services for the definition of metadata according to MOF and its load and storage in the repository are provided by the components *MOFImplementation* and *RepManager*. These services are used by the SEE tools. To facilitate the management of metamodels and models of the repository the auxiliary tool METAMOD was developed. METAMOD is a tool for the definition of metamodels and models. In the context of the proposed

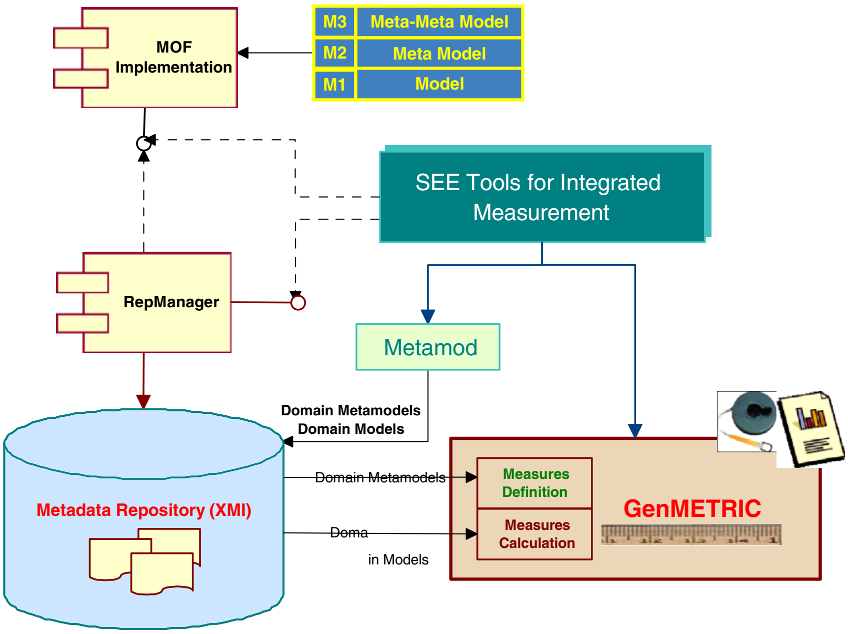

Fig. 8. Architectural view of GenMETRIC.

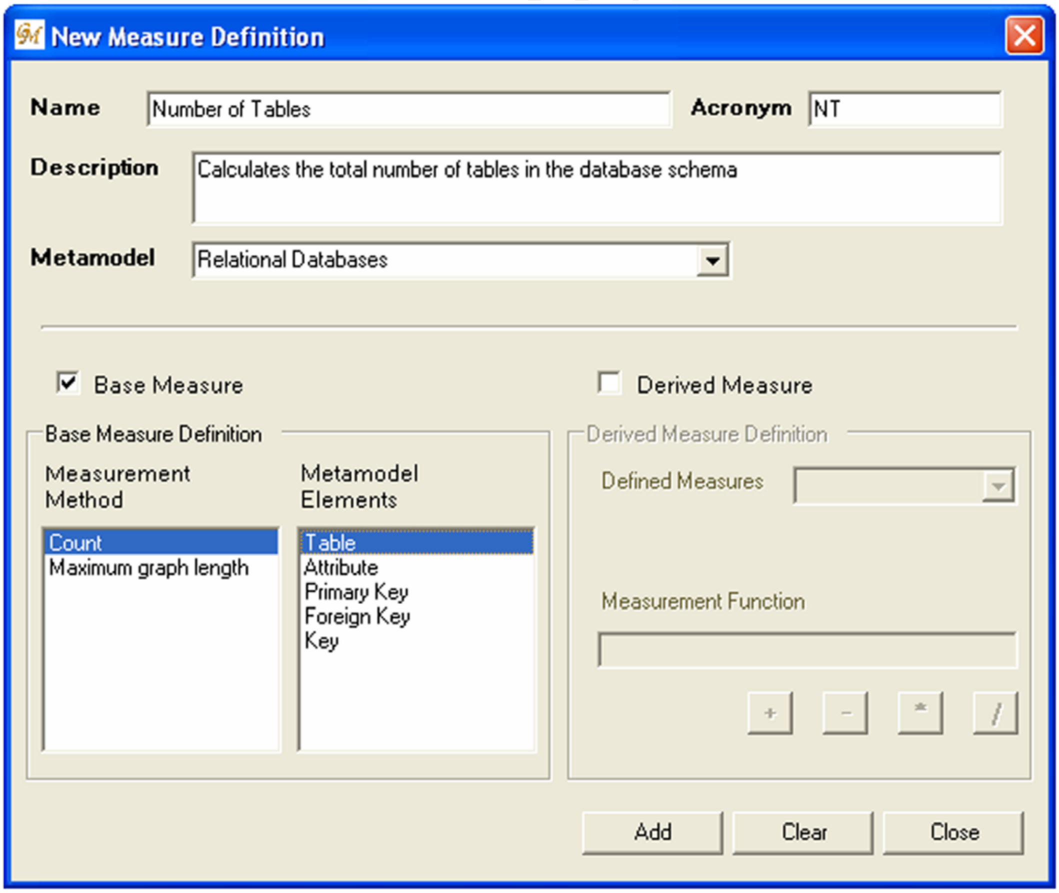

Fig. 9. Measure definition frame.

Please cite this article in press as: F. García et al., Managing software process measurement: ..., Informat. Sci. (2007), doi:10.1016/j.ins.2007.01.018

SEE, METAMOD provides the functionality for the definition of the domain metamodels and their corresponding measurement models which represent the kinds of entities and concrete entities which are candidates for measurement. GenMETRIC imports the domain metamodels for the definition of the measures and these measures are calculated on concrete entities (domain models).

GenMetric provides the user with a powerful interface for the definition of measurement models and for the calculation and visualisation of results. Two roles are defined: *Administrator*, who can interact with the complete functionality of the tool (definition and update of measurement models and calculation and presentation

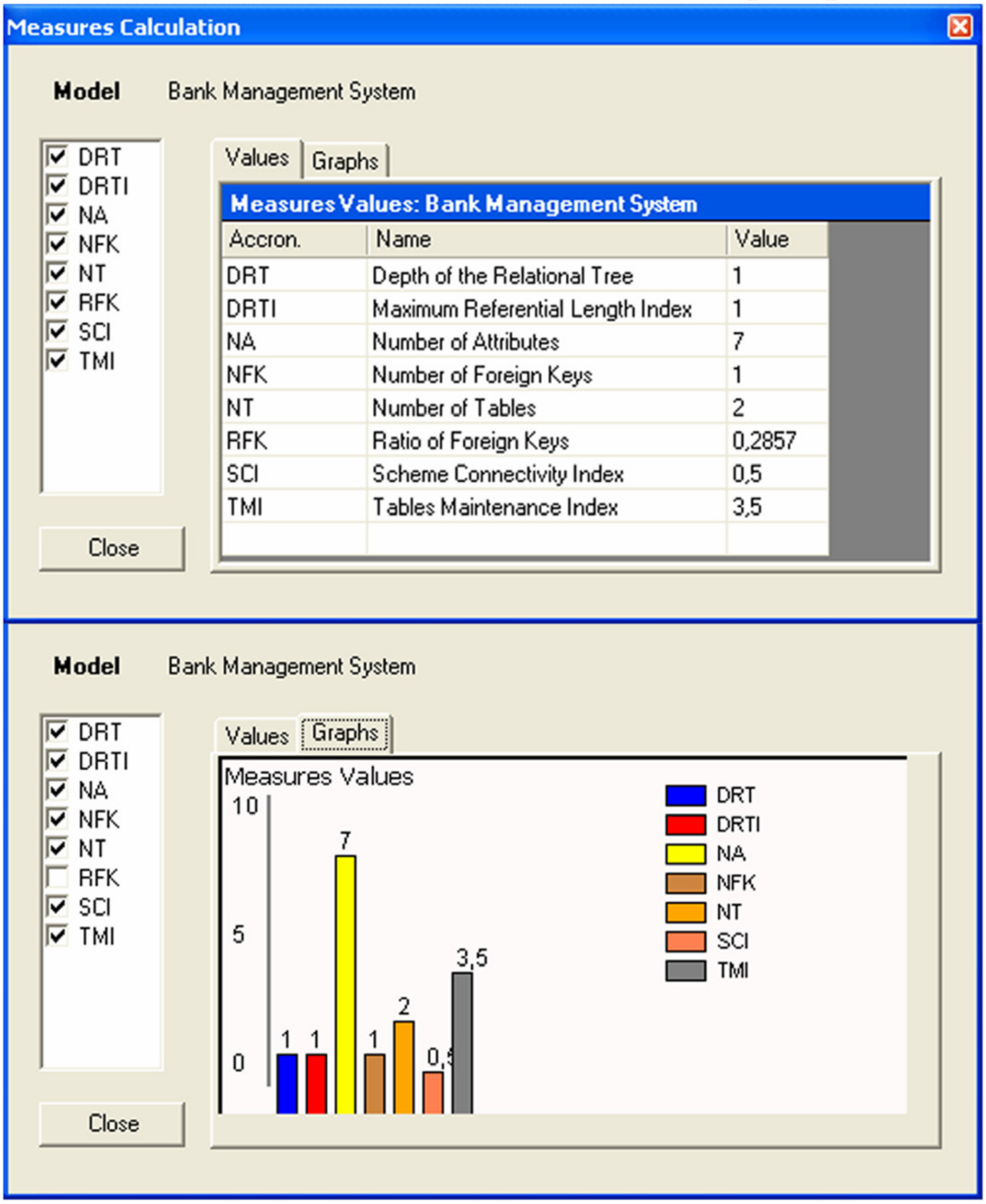

Fig. 10. Measurement results frame.

of the measurement results); and *User*, who can calculate the measures of existing measurement models and present the results. The form for the definition of software measures is shown in Fig. 9:

Fig. 9 shows an example of the definition of the base measure "Number of Tables" in the "Relational Metamodel" (Section 5.2). This measure is defined by applying the measurement method count on the "Table" metamodel element. In the same way, by using GenMETRIC we can define all the measures related to a specific measurement model (such as that which is presented in Section 5.1), by using the different elements from which it is composed.

Subsequently, we can use the measure definitions for the calculation of the metrics in real models. Fig. 10 summarizes the results of the automatic calculation of the measures presented in Section 5.1 when applied to the "Bank Management System" entity (shown in Section 5.3):

As we can observe in Fig. 10, GenMETRIC provides both the tabular and the graphical (using bar charts) representation to provide users with the necessary information for making decisions.

Therefore, an integrated and automatic environment for measurement is provided with the proposed tool. This being a generic tool, the definition of any new measure on the existing domain metamodels is possible, without having to code new modules. Furthermore, the tool is extensible, which eases the measurement of new entities by means of the incorporation of their domain metamodels. For example, a metamodel for defining web elements (formed by web pages, links between pages, etc.) could be included in the SEE and in this way it could be possible to measure web sites. Moreover, as it works with XMI documents, it eases communication and the possibility of openly importing new domain metamodels, or domain and measurement models stored in other MOF compliant repositories.

7. Conclusions and future work

In this paper, we have proposed an approach to enable the management of measurement of software processes. The evaluation of software processes involves the measurement of a great diversity of entities, from the models of the process to projects, resources and the products obtained. The proposal allows the integrated management of the measurement of these kinds of entities by means of:

– A measurement metamodel which includes the necessary constructors to define software measurement models. The metamodel is CMMI, ISO 15939 and PSM compliant [20], as the ontology has mainly used and adapted the concepts included in these proposals, and it provides companies with the initial support necessary to sustain the measurement process.
– A flexible method to measure any kind of artefacts within metamodel scope. The measures included in the measurement models are defined on the metamodel elements which represent the kinds of entities to be evaluated. This implies great flexibility in the ability to include new kinds of evaluative entities in the measurement programs of software companies without having to develop new tools to support them.

The framework allows the measurement of any software entity during the whole software lifecycle and especially in the early stages of software development where analysis and design models are produced. The identification of defects in these stages helps companies to reduce costs of late defect fixing and to improve the final quality [3,19].

The proposal is supported by the GenMETRIC tool, a generic tool for the definition of measurement models. The measures of the models are automatically calculated by examining the XMI documents of the entities to be evaluated. The metamodels which represent the different kinds of entities must be stored in the repository. The measurement capabilities of the tool increase owing to the inclusion of new kinds of entities to be measured which implies the inclusion of their metamodels in the repository.

The approach presented in this paper is MDA-compliant. The model management principles allow metamodel based tools to exchange compatible models, i.e. models conforming to metamodels which themselves conform to common metametamodels. This paradigm has been proven to be quite powerful in practice [1,32] and these advantages can also be applied to software measurement. In fact, with the MDA approach the increase of CASE tools with XMI import/export capabilities, as is currently happening with many UML CASE tools, will provide this proposal with significant power in the support of measurement. In this

Please cite this article in press as: F. García et al., Managing software process measurement: ..., Informat. Sci. (2007), doi:10.1016/j.ins.2007.01.018

way, companies will easily be able to import all their software entity models in the repository of the SEE (see Fig. 8).

So far, this proposal has been successfully applied in a software company to improve its processes by providing the necessary evaluation support [6] by means of a consistent and unique terminology and a complete measurement template for the collection of all the data and metadata related to the measurement process. The company's former measurement system was based on the calculation of various isolated project indicators. With the measurement support, the measurement process was enriched with the overall information and tools for computing and interpreting the indicators. Furthermore, a flexible environment was provided, initially composed of measurement models to evaluate their database models, but extensibly to support the measurement of any software entity, such as object-oriented systems or data-warehouses, especially given the current adaptation of the company's processes to object-oriented technologies.

Among related future works, the following deserve special attention:

–The development of a graphic notation for the representation of measurement models according to the FMESP measurement meta-model. As a result, a software tool which extends the functionality of GenMETRIC should be incorporated into the SEE.
–The development of new case studies in software companies to integrate the measurement of their relevant software entities in order to promote the improvement of their software processes.
–The incorporation of estimation as well as measurement capacities within the framework, by extending the current software measurement ontology and corresponding metamodel with the necessary constructors, and by including the domain metamodels and necessary technical support. In this context, relevant measurement techniques such as function point analysis could also be supported by the proposal.

Acknowledgements

This work has been partially financed by the projects: MECENAS (Junta de Comunidades de Castilla-La Mancha, Consejería de Educación y Ciencia, reference PBI06-0024), ENIGMAS (Junta de Comunidades de Castilla-La Mancha, Consejería de Educación y Ciencia, reference PBI-05-058) and CALIA (Universidad de Castilla-La Mancha). We acknowledge anonymous reviewers for their comments and suggestions that have helped us to improve this work.

References

[1] J. Bezivin, E. Breton, Applying the basic principles of model engineering to the field of process engineering, UPGRADE: European Journal for the Informatics Professional. (<http://www.upgrade-cepis.org/issues/2004/5/upgrade-vol-V-5.html>), (2004) V: 27–33.
[2] J. Bezivin, F. Jouault, 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.
[3] L. Briand, S. Morasca, V. Basili, Defining and validating measures for object-based high-level design, IEEE Transactions on Software Engineering 5 (5) (1999) 722–743.
[4] M. Brown, D. Goldenson, Measurement and Analysis: What Can and Does Go Wrong?, in: Proceedings of the 10th International Symposium on Software Metrics (METRICS'04), 2004, pp. 131–138.
[5] C. Calero, M. Piattini, M. Genero, Empirical validation of referential integrity metrics, Information Software and Technology, Special Issue on Controlled Experiments in Software Technology 43 (15) (2001) 949–957.
[6] G. Canfora, F. García, M. Piattini, F. Ruiz, C. Visaggio, Applying a framework for the improvement of software process maturity, Software: Practice and Experience 36 (3) (2006) 283–304.
[7] R. Dumke, R. Winkler, CAME Tools for an Efficient Software Maintenance, in: Proceedings of the 1st Euromicro Working Conference on Software Maintenance and Reengineering (CSMR'97), Berlin (Germany), March 17–19, 1997, pp. 74–81.
[8] N. Fenton, S. Pfleeger, Software Metrics: A Rigorous & Practical Approach, 2nd ed., PWS Publishing Company, 1997.
[9] W.A. Florac, A.D. Carleton, Measuring the Software Process. Statistical Process Control for Software Process Improvement, Addison Wesley, New York, 1999.
[10] F. García, M. Bertoa, C. Calero, A. Vallecillo, F. Ruiz, M. Piattini, M. Genero, Towards a consistent terminology for software measurement, Information and Software Technology 48 (8) (2006) 631–644.
[11] F. García, F. Ruiz, M. Piattini, G. Canfora, C.A. Visaggio, FMESP: framework for the modeling and evaluation of software processes, Journal of Systems Architecture 52 (2006) 627–639.

[12] M. Genero, L. Jiménez, M. Piattini, Measuring the quality of entity relationship diagrams, in: Proceedings of the 19th International Conference on Conceptual Modeling (ER 2000), Salt Lake City, UT, 2000, pp. 513–526.
[13] M. Genero, M. Piattini, C. Calero (Eds.), Metrics for Software Conceptual Models, Imperial College Press, 2005.
[14] W. Harrison, A flexible method for maintaining software metrics data: a universal metrics repository, Journal of Systems and Software 72 (2004) 225–234.
[15] IEEE Std 1061-1998 IEEE Standard for a Software Quality Metrics Methodology. <http://standards.ieee.org/reading/ieee/std_public/description/se/1061-998_desc.html>.
[16] SO/IEC 15504-2:2003, Information technology – Process assessment – Part 2: Performing an assessment, International Standards Organization, Geneva, Switzerland, 2004.
[17] ISO/IEC 90003, Software and Systems Engineering – Guidelines for the Application of ISO/IEC 9001:2000 to Computer Software, International Standards Organization, Geneva, Switzerland, 2004.
[18] ISO/IEC 15939, Software Engineering – Software Measurement Process, Organization for Standardization, Geneva, 2002.
[19] A. Janes, M. Scotto, W. Pedrycz, B. Russo, M. Stefanovic, G. Succi, Identification of defect-prone classes in telecommunication software systems using design metrics, Information Sciences 176 (2006) 3711–3734.
[20] C. Jones, Making Measurement Work, CROSSTALK The Journal of Defense Soft-ware Engineering 16 (1) (2003) 15–19.
[21] B.A. Kitchenham, R.T. Hughes, S.G. Linkman, Modeling software measurement data, IEEE Transactions on Software Engineering 27 (9) (2001) 788–804.
[22] L. Lavazza, Providing Automated Support for the GQM Measurement Process, IEEE Software 17 (3) (2000) 56–62.
[23] L. Lavazza, A. Agostini, Automated measurement of UML models: an open toolset approach, Journal of Object Technology (JOT) 4 (4) (2005) 115–134.
[24] J. McGarry, D. Card, C. Jones, B. Layman, E. Clark, J. Dean, F. Hall, Practical Software Measurement. Objective Information for Decision Makers, Addison-Wesley, New York, 2002.
[25] Object Management Group (OMG), MDA Guide, Version 1.0.1, June 2003. <http://www.omg.org/mda/specs.htm>.
[26] Object Management Group (OMG), Meta Object Facility (MOF). Core Specification Version 2.0. October 2003. <http://www.omg.org/docs/formal/00-04-03.pdf>.
[27] Object Management Group, MOF 2: XMI – Mapping Specification, version 2.1, 2005. <http://www.omg.org/docs/formal/05-09-01.pdf>.
[28] Object Management Group (OMG), Unified Modeling Language: Superstructure Specification, October 8, 2004. <http://www.uml.org>.
[29] E. Palza, C. Furhman, 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), Greenbelt (Maryland, USA), December 3–4, 2003, pp. 12–22.
[30] M. Scotto, A. Sillitti, G. Succi, T. Vernazza, A relational approach to software metrics, in: Proceedings of the Software Applied Computing (SAC'2004), Nicosia, Cyprus, March 14–17, 2004, pp. 1536–1540.
[31] Software Engineering Institute (SEI), Capability Maturity Model Integration (CMMI), version 1.1. <http://www.sei.cmu.edu/cmmi/>.
[32] J.M. Sprinkle, A. Ledeczi, G. Karsai, G. Nordstrom, The New Metamodeling Generation, in: Proceedings of the Eighth Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, April 2001, pp. 275–279.
[33] V.K. Vaishnavi, S. Purao and J. Liegle. Object-oriented product metrics: A generic framework. Information Sciences, in press.
[34] R. Van Solingen, E. Berghout, The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development, McGraw-Hill, New York, 1999.
[35] <http://irb.cs.uni-magdeburg.de/sw-eng/us/bibliography/bib_main.shtml>.

Please cite this article in press as: F. García et al., Managing software process measurement: ..., Informat. Sci. (2007), doi:10.1016/j.ins.2007.01.018