# *Graphical versus textual software measurement modelling: an empirical study*

**Software Quality Journal**

ISSN 0963-9314
Volume 19
Number 1

Software Qual J (2010)
19:201-233
DOI 10.1007/s11219-010-9111-x


Springer

**Your article is protected by copyright and all rights are held exclusively by Springer Science+Business Media, LLC. This e-offprint is for personal use only and shall not be self-archived in electronic repositories. If you wish to self-archive your work, please use the accepted author's version for posting to your own website or your institution's repository. You may further deposit the accepted author's version on a funder's repository at a funder's request, provided it is not made publicly available until 12 months after publication.**

Springer

Author's personal copy

Software Qual J (2011) 19:201–233
DOI 10.1007/s11219-010-9111-x

ORIGINAL PAPER

# Graphical versus textual software measurement modelling: an empirical study

**B. Mora · F. García · F. Ruiz · M. Piattini**

Published online: 22 September 2010
© Springer Science+Business Media, LLC 2010

**Abstract** Model-driven Engineering (MDE) has attained great importance in both the Software Engineering industry and the research community, where it is now widely used to provide a suitable approach with which to improve productivity when developing software artefacts. In this scenario, measurement models (software artefacts) have become a fundamental point in improvement of productivity, where MDE and Software Measurement can reap mutual benefits. MDE principles and techniques can be used in software measurement to build more automatic and generic solutions, and to achieve this, it is fundamental to be able to develop software measurement models. To facilitate this task, a domain-specific language named "Software Measurement Modelling Language" (SMML) has been developed. This paper tackles the question of whether the use of SMML can assist in the definition of software measurement models. An empirical study was conducted, with the aim of verifying whether SMML makes it easier to construct measurement models which are more usable and maintainable as regards textual notation. The results show that models which do not use the language are more difficult—in terms of effort, correctness and efficiency—to understand and modify than those represented with SMML. Additional feedback was also obtained, to verify the suitability of the graphical representation of each symbol (element or relationship) of SMML.

**Keywords** Software measurement · DSL · Empirical validation

B. Mora (✉)
Indra Software Labs, Information Technology Company, Ciudad Real, Spain
e-mail: bmorar@indra.es

F. García · F. Ruiz · M. Piattini
ALARCOS Research Group, Department of Information Technologies and Systems,
University of Castilla-La Mancha, Paseo de la Universidad, 4, 13071 Ciudad Real, Spain
e-mail: Felix.Garcia@uclm.es

F. Ruiz
e-mail: Francisco.RuizG@uclm.es

M. Piattini
e-mail: Mario.Piattini@uclm.es

Author's personal copy



## 1 Introduction

Measurement is a key aspect in the software life cycle, since it provides a support for the planning, monitoring, control and evaluation of the quality of software processes (how) and products (what) (Fenton and Pfleeger 1997). It is therefore highly advisable and useful to define and carry out a measurement process in order to obtain quantitative information.

Furthermore, in recent years, the new Model-driven Engineering (MDE) paradigm has become widely used, achieving great importance in the research field and the Software Engineering industry. This paradigm can be used by software companies to improve productivity, because new software artefacts are (semi)automatically generated from models (Bézivin 2004). The measurement of models in this scenario has become a fundamental step in making it easier to improve them later on, and MDE and Software Measurement can consequently benefit each other mutually, as has been analysed by several authors (see Sect. 6). MDE can therefore take advantage of the benefits of software measurement, because the importance of measuring models is greater when applying MDE as a prior, fundamental step towards later improvement of the models. In the opposite respect, the principles and techniques of MDE can be used in software measurement, with the objective of building more generic and automatic solutions (García et al. 2003; Bézivin et al. 2005; García et al. 2007; Mora et al. 2009). Additionally, a current problem when modelling software measurement is the absence of an appropriate notation which makes it possible to have a more intuitive and easier definition of software measurement models, which are the most relevant type of artefact in this context. To achieve this, it is fundamental to provide practitioners and researchers with a suitable language, and MDE has two alternative means by which to define or to develop languages: (i) General Purpose Languages (GPL) and (ii) Domain Specific Languages (DSL). A GPL is a language that can be used to solve a wide variety of problems such as, for example, UML (OMG 2005b). On the other hand, "a DSL is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain" (Deursen et al. 2000).

When designing DSLs, we must identify the source for the languages, that is, we need to find what the DSL should express, and identify what the relevant abstraction and notations in the DSL are (Völter 2009). The source for a language is often an existing framework, library, architecture or architectural pattern. Besides, patterns exist to aid the DSL developer in the decision, analysis, design and implementation phases of DSL development (Mernik et al. 2005). In this sense, the use of a Domain Definition Metamodel (DDMM), which is a specification of the domain's conceptualization (Kurtev et al. 2006), can be considered of interest in obtaining the abstraction language. Once the knowledge has been identified, building the DSL is mainly about formalizing the knowledge: defining a notation (graphical or textual), putting it into a formal language and building generators/editors to generate parts of the implementation code (Völter 2009).

Both types of language commented on previously (GPL and DSL) can be used to apply MDE in the context of software measurement, but the option of a DSL seems to be a more appropriate choice for the following main reasons:

(a) A DSL has a clearly identified concrete problem domain (in contrast to a GPL which covers multiple domains).

Author's personal copy



(b) Thanks to the existence of the "Software Measurement Ontology" (SMO) (García et al. 2006) we have the conceptual and semantic foundations for the DSL.[1]
(c) The previously developed "Software Measurement Metamodel" (SMM) (García et al. 2007), derived from the SMO, can be used as the DDMM.

One important aspect which was raised in developing the Software Measurement DSL was whether it was better to use a graphical notation or a textual representation to facilitate the understanding and ease of maintenance of the models. The answer is not clear, as a representation can perform better than the alternative, depending on the type of information to be managed. To represent information which deals with relationships between entities, for the timing/sequence of events or for some kind of signal/data flow, graphical notation is better (Völter 2009). However, rendering expressions graphically is a bad solution, so textual representation can perform better in this context. On the other hand, in this context, it is interesting to consider the visual programming area, which tackles two important issues in the Software Engineering field (Ahmad 1999): first, it allows users to master the inherent complexity of the programming activity by means of a graphical notation. The high level of visual abstraction makes program comprehension easier by visualizing the semantic relationships between the program entities and it facilitates an increase in the productivity of developers and users, due to the fact that the latter can generally understand the beginning of the system design process.

The "Software Measurement Modelling Language" (SMML) (Mora et al. 2008b, c) has therefore been developed for the reasons outlined earlier. SMML is a graphical DSL, whose goal is to make it easier to write and read software measurement models. As stated previously, a graphical notation can be a good option for representing domains in which entities and relationships between them are involved. SMML is a core element of the "Software Measurement Framework" (SMF[2]) (Mora et al. 2008a) which includes all the conceptual, architectural, methodological and technical tools to support the main aspects and activities of the software measurement process in an MDE-based manner.

The aim of the work presented in this paper was to test the usability and maintainability of the language by means of an empirical study, which focused on the understandability and changeability sub-characteristics according to the ISO 9126 quality model (ISO/IEC 2001). Following the principle of replicability, basic to any empirical scientific research, we provide detailed information on the empirical validation and how this was done, including the planning, material, method and analysis and interpretation results (Basili et al. 1999; Shull et al. 2008). The study was carried out with the collaboration of a group of Software Engineering experts who worked with software measurement models defined with both SMML and TEXTUAL notations, and with data in tables organized in a predetermined format, but with free text in cells. The intention of the empirical study was therefore to test which type of representation (graphical or textual) can be most suitable for representing software measurement models according to the SMM.

The remainder of the paper is organized as follows. Section 2 puts the SMML language into context by summarizing the main characteristics and components of the Software Measurement Framework (SMF). Section 3 then presents the main aspects of the SMML, including the abstract syntax (based on the software measurement metamodel, SMM), the concrete syntax (collection of icons that constitutes the graphical language), the constraints

[1] In http://alarcos.inf-cr.uclm.es/ontologies/smo/—The Web Ontology Language (OWL) is a family of knowledge representation languages for authoring ontologies endorsed by the World Wide Web Consortium.

[2] In http://alarcos.esi.uclm.es/smf/.

Author's personal copy



on the abstract syntax and the semantics. Sections 4 and 5 explain the empirical study, with the empirical validation and a discussion of the experimental validity, respectively. In Sect. 6, an overview of related work is provided, and conclusions and future work are outlined in the final section. Appendix A includes an excerpt of the experimental material.

## 2 The Software measurement framework (SMF)

The SMF provides a reference framework for the measurement of any kind of MOF-compliant model (Mora et al. 2008a, 2009). 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. As a result, SMF ensures that the measurement process is carried out in a consistent and more productive manner, by providing companies with the necessary infrastructures and methodology. Figure 1 illustrates how the measurement of any kind of MOF-compliant model is supported by the SMF framework. The architecture has been organized into the following conceptual Meta-Object Facility (MOF)-based metadata levels: Meta-Metamodel Level (M3), Metamodel Level (M2) and Model Level (M1).

As can be observed in Fig. 1, the measurement of models is carried out by following the steps set out below:

1. *Creation or incorporation of the domain metamodel:* the measurement is done in a specific domain and this domain must be defined according to its MOF-compliant metamodel. For instance, if the model to be measured is a UML model, the UML metamodel must be incorporated into the repository.
2. *Creation or incorporation of the domain model:* this must be defined according to its corresponding domain metamodel which is MOF-compliant (created or included in the first step).
3. *Creation of measurement model:* the measurement model is created according to the Software Measurement Metamodel (SMM), a key integrated part of the SMF (see Sect. 3) and which is MOF-compliant. This constitutes the source model and contains all the information about the software measurement process.
4. *Measurement execution:* the automatic measurement execution is carried out through a model transformation, in which the target measurement model is obtained from the two source models (the measurement model and the domain model). The target measurement model is obtained by extending the source measurement model with the results of the measurements. As can observed in Fig. 1, the metamodel applied to define the transformation is Query/View/Transformation (QVT) (OMG 2005a), whose metamodel is MOF compliant too.

   In addition to all said above, we should comment that by means of this framework any software entity in any domain which has a MOF-compliant metamodel associated with it (for example UML, E/R models, Relational schemas, Requirements models, etc.) can be measured. This is achieved by using a common metamodel (SMM) to represent software measurement models and by employing model transformations.

One of the main objectives of the SMF framework is to make it possible for the work of the potential users (measurement engineers, measurement experts, etc.) to be done with greater ease when they are aiming to define measurement models. The aim is also to apply them to the measurement of software entity attributes. This is achieved through the application of the Software Engineering *black box* principle, which implies that users are

Author's personal copy



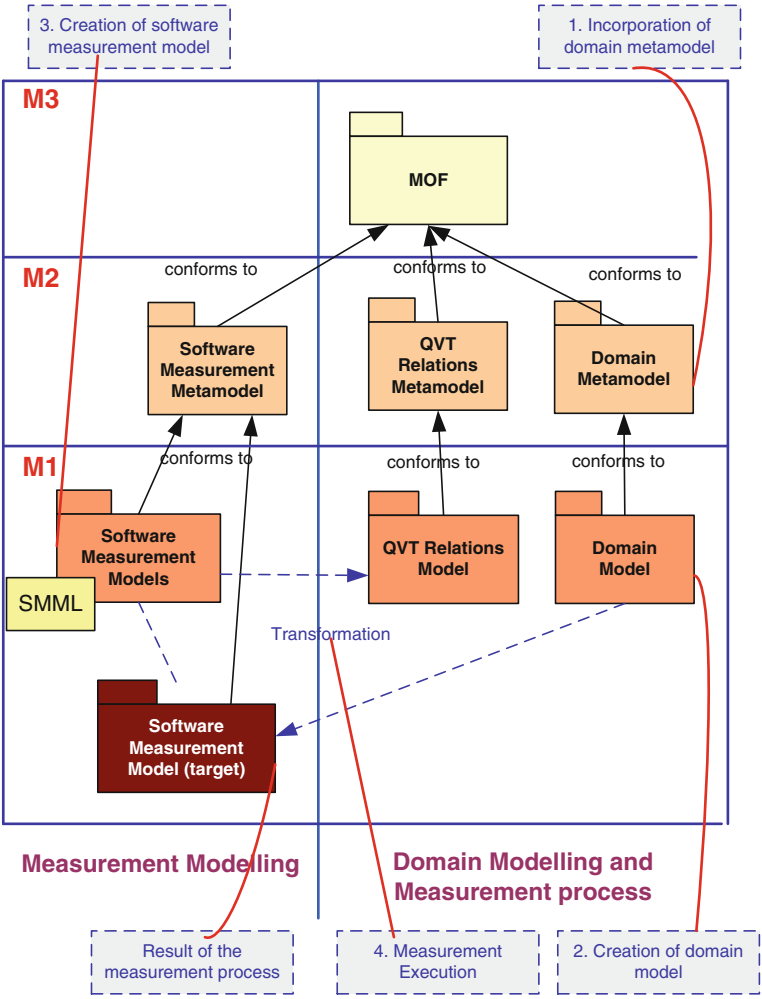



**Fig. 1** Overview of SMF

not required to have any knowledge of the internal workings of the SMF, owing to the automatic model transformations of the MDE paradigm which are automated in the framework.

Therefore, as is illustrated earlier, one of the key steps is the definition of software measurement models. The Software Measurement Modelling Language has been developed to support this task as an alternative to textual notation, and the comparison between these, by means of an empirical study, is the main focus of this paper. This language is intended to play a fundamental role in the SMF, as it provides the user with the definition of the measurement models by using a graphical notation, a task which, until now, had to be done by directly instantiating the software measurement metamodel in a textual manner. SMML incorporates all the constructors needed to represent software measurement models of any type of entity, i.e. measurements of processes, projects and products.

Author's personal copy



SMF is supported by SMTool,[3] which automates the generic measurements of any software domain. The tool is composed of two main components which have been developed by using the Eclipse framework:

- *A Model Transformation Engine*, which supports the management and transformation of the models by means of QVT (Mora et al. 2009). The models and metamodels managed by the engine must be ECORE compliant.
- A *Graphical Editor*, which supports the SMML (see Sect. 3.4).

Due to the fact that the SMTool has been developed using the Eclipse framework, the domain metamodel defined must be represented with ECORE, which is MOF compliant.

The following sections show details of SMML and the empirical research which was conducted to provide a preliminary insight into its potential usefulness.

## 3 The Software measurement modelling language (SMML)

The SMML (Mora et al. 2008b) plays a fundamental role in the SMF, as it allows users to define the measurement models which are the input for the software measurement process. The visual representation of the measurement models aims to make SMF a more usable and intuitive framework for the user. The set of icons which form a part of the language have been created and/or selected in order to make it easier for software engineers to define software measurement models. The use of general purpose languages to define domain measurement models is thus avoided.

SMML is a DSL with a clear syntactic and semantic definitions, thanks to its ontological foundation (Mernik et al. 2005). Moreover, the SMML has been created with the objective of fulfilling the general requirements of a DSL: usability, conformability, orthogonality, supportability and simplicity (Kolovos et al. 2006). The use of the SMML is illustrated throughout the rest of the paper by means of examples. The difference between the use of the SMML and a tabular textual representation is also shown (see Figs. 11, 12 and 13 in Appendix A). The example provided is about a measurement model which aims to evaluate the maintainability of relational database schemas. More examples of SMML usage can be found in http://alarcos.esi.uclm.es/smf/smml.

The next subsections summarize how the SMML has been developed by following the tasks defined by Feilkas (2006).

### 3.1 Definition of an abstract syntax (domain definition metamodel)

The SMML has been developed using the Software Measurement Metamodel (SMM). This metamodel is derived from the Software Measurement Ontology (SMO) (García et al. 2006) and defines the abstract syntax of SMML. Figure 2 shows the UML representation of the SMO.

The SMM is aligned with the SMO sub-ontologies which tackle the definition phase of software measures. As a result, the SMM is composed of the following packages (Fig. 3):

- *Basic Package*: this Basic Package identifies and establishes the general constructors which are necessary to define any measurement model.

[3] In http://alarcos.esi.uclm.es/smf/smtool/.

Author's personal copy



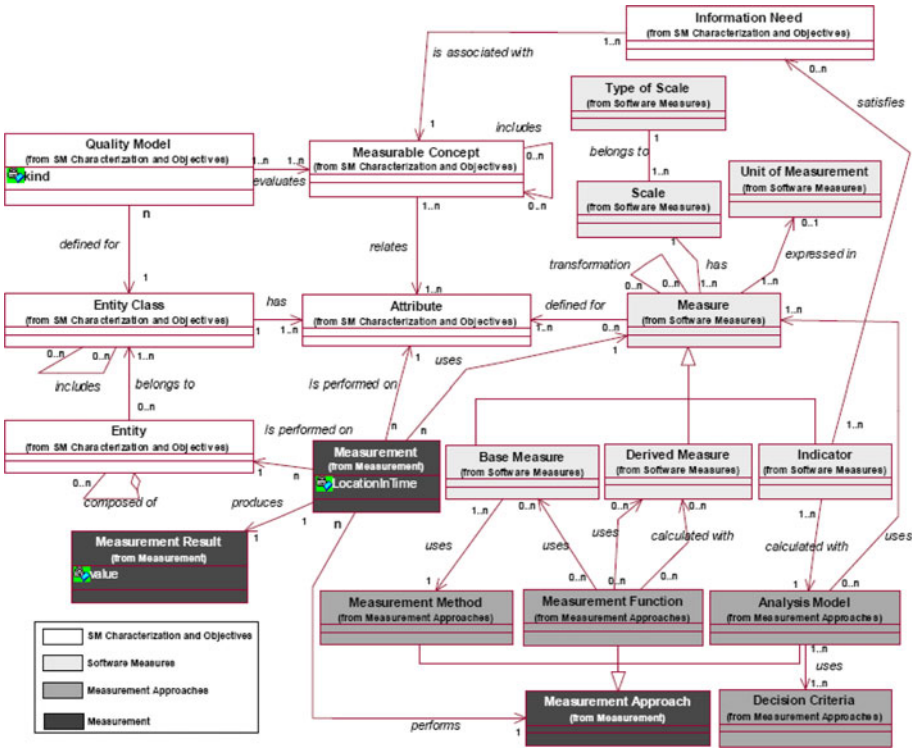



**Fig. 2** Software measurement ontology (García et al. 2006)

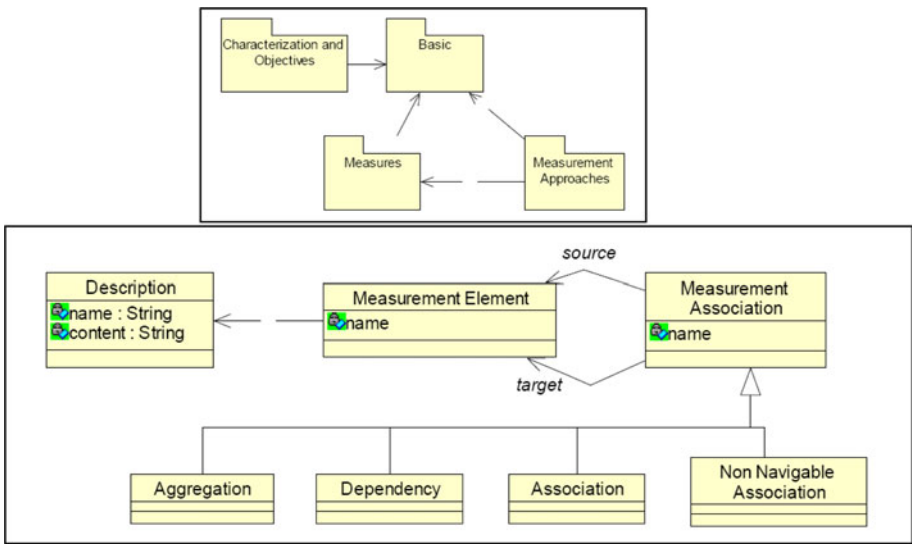



**Fig. 3** Software measurement metamodel: package structure and basic package

Author's personal copy



- *Characterization and Objectives Package*: this package includes the constructors required to establish the scope and objectives of the software measurement process.
- *Software Measures Package*: this package includes the constructors needed to establish and to clarify the key elements in the definition of a software measure.
- *Measurement Approaches Package*: this package includes the constructors needed to generalize the different 'approaches' used by the three kinds of measures (base, derived and indicators) to obtain their respective measurement results.

The measurement action sub-ontology (see Fig. 2) is excluded from the SMM, as this sub-ontology includes the terminology related to the act of measuring software by means of obtaining measurements, which is a set of measurement results, for a given attribute of an entity, using a measurement approach. This sub-ontology is therefore beyond the scope of the software measurement definition step, which is intended to be supported by means of the SMML.

According to the SMM basic constructors (Fig. 3), the SMML is composed of two types of elements: measurement elements and measurement associations. The measurement elements in the SMML are the elements which are defined in the SMM (Measure, Information need, Measurable concept, etc.) on which the SMML language is based. With regard to the relationships between measurement elements, four types of measurement associations have been identified: association, non-navigable association, aggregation and dependency. These relationships have been defined in the Basic Package (for further details see (Mora et al. 2008c)).

All of the SMM packages therefore maintain the original definition of (García et al. 2007), with the exception of the Basic Package, which was accordingly adapted so as to satisfy the requirements of the SMML for representing the relationships between the modelling constructors.

### 3.2 Definition of a concrete syntax

A concrete syntax was defined in order to make the language usable. All of the elements are defined in the Basic Package: measurement elements and association elements.

Each of these elements of the language must be associated with a graphical icon which represents the element of the abstract model, and these icons were designed bearing in mind their usability when applied by users. As a result, some of the proposed icons were extracted and/or adapted from well-known modelling languages such as Entity-Relationship Diagrams to facilitate their use by software engineers. When no suitable source with which to represent a constructor was found, it was built from scratch. Table 1 includes a representative sample of the measurement elements. The complete concrete syntax can be found in (Mora et al. 2008c).

Table 2 shows the relationships included in the "Software Measurement Characterization and Objectives" package. Note that each relationship in the SMO has been classified under one of the four types of associations of the SMML, which are defined in the *Basic* Package. The complete set of associations is included in Mora et al. (2008c).

### 3.3 Abstract syntax constraints

Another key aspect of a DSL specification is the definition of constraints on the abstract syntax, which describes in precise terms the way to structure the constructors of the language.

These constraints establish the cardinality and the participant elements of the associations, and they have been defined by using OCL (OMG 2003). An example of a constraint

Author's personal copy



**Table 1** A selection of the SMML elements

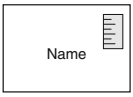
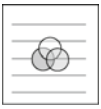
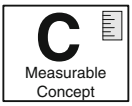
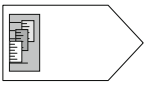
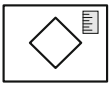
| Information need | Entity | Base Measure | Quality Model |
|---|---|---|---|
|  Information need |  Name |  |  Quality Model |
| **Measurable Concept** | **Attribute** | **Measurement Function** | **Decision Criteria** |
|  C Measurable Concept |  Attribute |  |  |

**Table 2** A selection of the SMML associations

| Relationships | Description |
|---|---|
| Includes  | An *entity class* may include several other *entity classes*. An *entity class* may be included in several other *entity classes* |
| Defined for  | A *quality model* is defined for a certain *entity class*. An *entity class* may have several *quality models* associated |
| Relates  | A *Measurable concept* relates one or more *attributes*. An *Attribute* is related to one or more *measurable concepts* |
| Has  | An *entity class* has one or more *attributes*. An *attribute* can only belong to one *entity class* |

in OCL which verifies whether the Measurement Elements involved in the "*Dependency*" Association (source and target) are correct is shown in Table 3. All the SMML constraints defined in OCL can be found in (Mora et al. 2008c).

### 3.4 Definition of semantics

The SMO and its derived metamodel (SMM) express the semantics of the SMML concepts, whose syntax is defined by the metamodel. The static semantics in the SMML is composed of the OCL constraints which have been on the abstract syntax or metamodel (to see Sect. 3.3). Finally, the editor of the SMML (to see Sect. 3.5) and the QVT transformations which have been defined to execute the model measurements (to see Sect. 2) provide the operational semantics of the language.

### 3.5 SMTool: the SMML editor

The SMML editor is part of SMTool and has been created by using the Eclipse Graphical Modelling Framework (GMF), which provides a generative component and runtime

Author's personal copy



**Table 3** OCL dependency constraints

| Measurement association: dependency |
|---|
| `(self.src.oclIsTypeOf (QualityModel) and self.tgt.oclIsTypeOf (EntityClass)) or` |
| `(self.src.oclIsTypeOf (QualityModel) and self.tgt.oclIsTypeOf (MeasurableConcept)) or` |
| `(self.src.oclIsTypeOf (MeasurableConcept) and self.tgt.oclIsTypeOf (InformationNeed))` |
| `or (self.src.oclIsTypeOf (AnalysisModel) and self.tgt.oclIsTypeOf (DecisionCriteria))` |
| `or(self.src.oclIsTypeOf (Indicator) and self.tgt.oclIsTypeOf (InformationNeed)) or` |
| `(self.src.oclIsTypeOf (Measure) and self.tgt.oclIsTypeOf (Attribute))` |

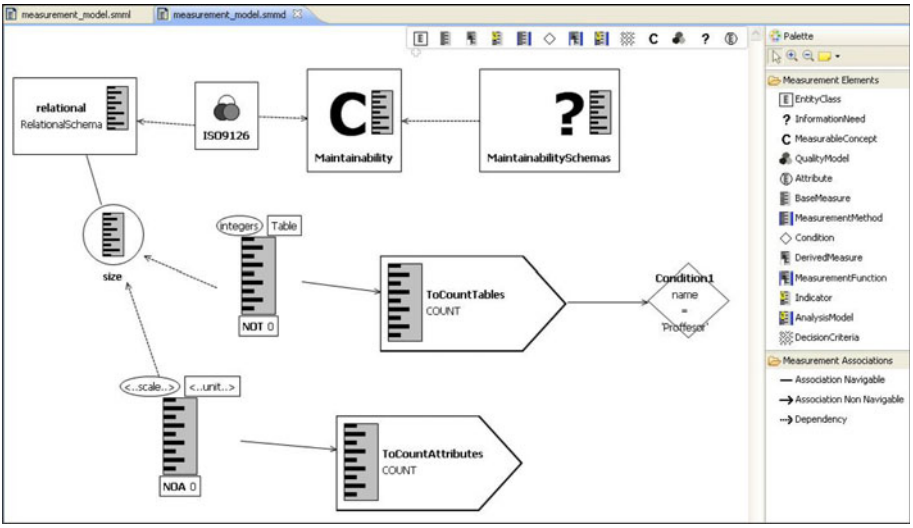



**Fig. 4** SMML editor

infrastructure for developing graphical editors based on EMF (Eclipse Modelling Framework) and GEF (Graphical Editing Framework). This has facilitated the creation of the SMML editor from the Software Measurement Metamodel (SMM) which was previously defined in EMF. Figure 4 shows the environment of the SMML editor. Thanks to the SMML editor, it is possible to define the software measurement MOF-compliant models that will be used to execute the measurement process in a more friendly way. Otherwise, these models would have to be textually defined by using EMF to directly instantiate the SMM, which is a more laborious and error-prone task.

## 4 Empirical validation of the SMML

This section describes the experiment carried out to provide empirical evidence with regard to the potential usefulness of the SMML in measurement modelling. The main goal of this

Author's personal copy



experiment was to ascertain whether models represented with the SMML are easier to understand and modify than models represented with a textual notation. Our research question can be stated as: "*Is the SMML more usable and does it facilitate the construction of more maintainable software measurement models?*"

## 4.1 Experiment planning

The general goal of our empirical study is derived as follows:

> *To analyse* measurement models represented with and without the SMML *with the purpose of* comparing them, *with regard to* the usability and maintainability of the models, *from the point of view of* software measurement analysts *within the context* of Software Engineering modelling experts.

In order to test the proposed hypothesis, an experiment was designed and conducted. This was achieved by following the templates and recommendations presented in Wohlin et al. (2000; Juristo and Moreno (2001); Kitchenham et al. (2002); Jedlitschka and Ciolkowski (2005). Figure 5 shows an overview of the experimental plan. Each of the steps of this experimental plan is explained in greater detail in the following sections.

### *4.1.1 Subjects*

The subjects of the experiment were 20 members of the Department of Information Technologies and Systems from the University of Castilla-La Mancha (UCLM, Ciudad Real, Spain). The group of subjects was chosen for their strict compliance with the following characteristics:

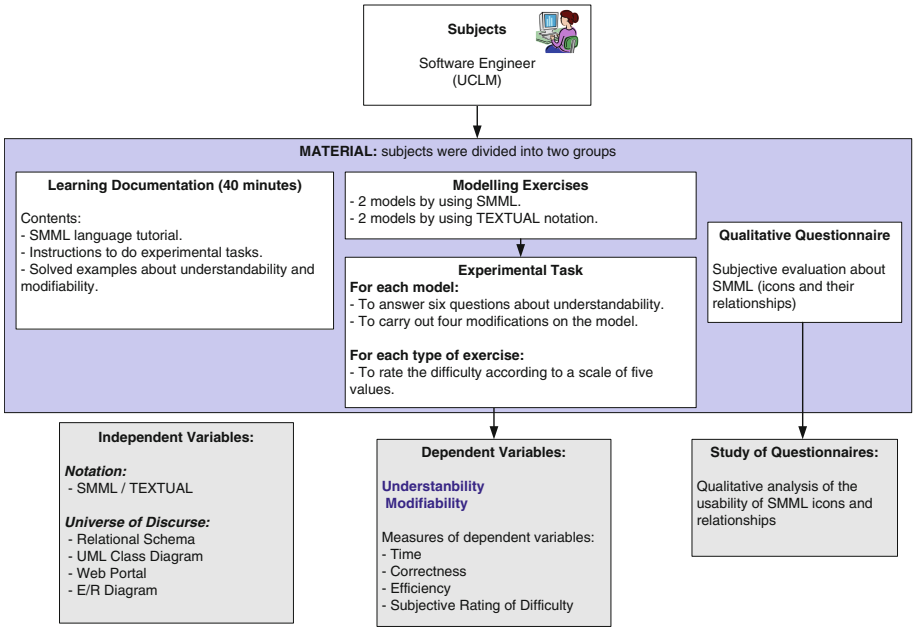



**Fig. 5** Overview of the experimental plan

Author's personal copy



- lecturers or M.Sc (Master of Science) in computer science.
- their experience in modelling.
- their knowledge of software measurement (software measurement metamodel and ontology).
- their lack of knowledge of the SMML.

#### *4.1.2 Material*

The material given to the subjects was composed of three parts, which are detailed in the following subsections:

(a) *Learning Documentation*: this part of the material was prepared to provide subjects with the information needed to learn about the SMML and to carry out the experimental tasks. This documentation was planned to be read in about 40 min on average and it was composed of the following parts:

- Introduction to the SMML language: in which the objective of the language and the ontology from which it was defined were explained. The reason for including this part was to balance the knowledge needed by all subjects to carry out the experimental tasks, thereby avoiding differences due to different levels of knowledge about the ontology.
- Structure of the SMML: A brief explanation of the 3 packages of which this language is formed was given, and an example of a diagram of each package used in the SMML was provided.
- Instructions, which had to be followed to carry out the tasks in the experiment.
- A solved example of the understandability exercise.
- A solved example of the modifiability exercise.

(b) *Modelling exercises*: the material given to each subject consisted of two types of exercises: an exercise concerning the understandability of the model and an exercise concerning its modifiability. Each type of exercise was related to one of four Universes of Discourse (UoD), respectively. Special care was taken in the choice of the UoD selected, and only those which would be familiar to Software Engineers were chosen. These UoD were Relational Schema, UML Class Diagram, Web Portal and E/R Diagram.
Two semantically equivalent software measurement models were prepared for each UoD: one by using the SMML language, and the other with TEXTUAL notation (free texts organized in tables). Hence, a total of eight software measurement models were prepared for each subject. Table 4 shows the distribution of material per subject.
Each understandability exercise included a questionnaire composed of six questions (with yes/no answers) about the measurement model, and at the end of the exercise, a section was included in which subjects had to rate the level of difficulty of the model according to a scale composed of five values (from very simple to very complex) (see Fig. 11). Each modifiability exercise included four new requirements which involved making certain modifications to the models, and a final part with the subjective evaluation of the level of difficulty of the model (see Fig. 12 in Appendix A).

(c) *Qualitative Questionnaire*: another key aspect to be assessed was the subjects' perception of the suitability of each symbol of the graphical language used to represent each element of the metamodel. This was considered by including a final questionnaire (see Fig. 6) in which the subjects rated each icon of the SMML on a

Author's personal copy



**Table 4** Description of software measurement models

| N | UoD | Notation |
|---|---|---|
| 1 | Relational schema | SMML |
| 2 | UML class diagram | TEXTUAL |
| 3 | Web portal | SMML |
| 4 | E/R diagram | TEXTUAL |
| 5 | Relational schema | TEXTUAL |
| 6 | UML class diagram | SMML |
| 7 | Web portal | TEXTUAL |
| 8 | E/R diagram | SMML |

scale from 1 (very suitable) to 5 (not very suitable) in accordance with their perception of the suitability of the symbol assigned to the element. They could also add comments, which would be taken into account to improve the language.

#### *4.1.3 Variables*

According to the standard ISO/IEC 9126, usability is a characteristic of software defined by "a set of attributes that bear on the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users". Usability is structured in the sub-characteristics of learnability, understandability and operability. Following this same norm, the other software characteristic is maintainability, which is defined as "a set of attributes that bear on the effort needed to make specified modifications" and whose sub-characteristics are stability, analyzability, changeability (also named modifiability) and testability. The *dependent variables* of our empirical study were the following sub-characteristics of usability and maintainability, respectively:

- *Understandability*, which we measured by means of the time spent by the subjects in answering the six questions, the number of correct answers (*correctness*), the *efficiency*, i.e., the relationship between the number of correct answers and the time spent answering them ([number of correct answers]/[task completion time]), and the subjective evaluation of the models' level of difficulty (*valuation*).
- *Modifiability*, which was measured by the *time* the subjects spent on carrying out the necessary modifications (four new requirements), the score of the modifications (*correctness*), the *efficiency* of those modifications and the subjective evaluation of the models' level of difficulty (*valuation*).

Our main independent variable was the use or non-use of the SMML (*notation*) to represent the software measurement models. The *UoD* was also considered as an independent variable in order to ensure a complete analysis of the experimental data.

#### *4.1.4 Hypotheses*

The experiment was planned with the purpose of testing the hypotheses stated in Table 5 which, for each set of hypothesis, details the dependent and independent variables, the measures of the dependent variables and the null and alternative hypothesis formulation.

Author's personal copy



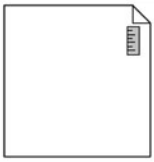
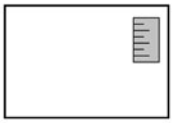
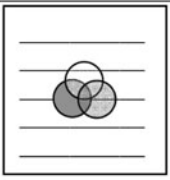
| Element | Icon | Definition | Evaluation | Comments |
|---|---|---|---|---|
| **Description** |  | Description of the *measurement element*. | | |
| **Information need** |  | Insight necessary to manage objectives, goals, risks, and problems. | | |
| **Measurable Concept** |  | Abstract relationship between *attributes of entities* and *information needs*. | | |
| **Entity Class** |  | The collection of all *entities* that satisfy a given predicate | | |
| **Attribute** |  | A measurable physical or abstract property of an *entity* that is shared by all the *entities* of an *entity class*. | | |
| **Quality Model** |  | The set of *measurable concepts* and the relationships between them which provide the basis for specifying quality requirements and evaluating the quality of the *entities* in a given *entity class*. | | |

**Fig. 6** Excerpt of final questionnaire concerning the suitability of the icons

### *4.1.5 Experimental tasks and treatment*

In accordance with the nature of the planned experiment, we applied an experimental design of confounded factorial with interaction (see Table 6). The subjects were randomly divided into two groups: Group A, whose members received material set A, and Group B, who were given material set B. As can be observed in Table 6, the intention was that a subject working with a UoD in the understandably exercise and using SMML, would work with the same UoD in the modifiability exercise with TEXTUAL notation, so that all the subjects would eventually have a set of 8 models distributed throughout the two types of exercise (understandability and modifiability). Each subject was required to answer the questionnaires, to carry out the modifications on each model and to rate the difficulty of

Author's personal copy



**Table 5** Sets of hypotheses

| ID | Dependent variable | Measures of dependent variables | Independent variables | Null hypothesis—$H_0$ | Alternative hypothesis—$H_1$ |
|---|---|---|---|---|---|
| *SU* | Understandability | Time<br>Correctness<br>Efficiency<br>Valuation | Notation | $H_{0Su}$<br>The use of SMML does not improve the understandability of the software measurement models | $H_{1Su}$<br>The use of SMML improves the understandability of the software measurement models |
| *SM* | Modifiability | Time<br>Correctness<br>Efficiency<br>Valuation | Notation | $H_{0Sm}$<br>The use of SMML does not improve the modifiability of the software measurement models | $H_{1Sm}$<br>The use of SMML improves the modifiability of the software measurement models |
| *UoD* | Understandability<br>Modifiability | Time<br>Correctness<br>Efficiency<br>Valuation | UoD | $H_{0UoD}$<br>There are no differences in understandability or modifiability owing to the universe of discourse of the model | $H_{1UoD}$<br>There are differences in understandability or modifiability owing to the universe of discourse of the model |
| *SxUoD* | Understandability<br>Modifiability | Time<br>Correctness<br>Efficiency<br>Valuation | Notation<br>UoD | $H_{0SxUoD}$<br>There are no differences in understandability or modifiability as a result of the combined effect of SMML usage and universe of discourse of the model | $H_{1SxUoD}$<br>There are differences in understandability or modifiability as a result of the combined effect of SMML usage and universe of discourse of the model |

**Table 6** Experimental design

| Type of exercise | Notation | Universe of discourse | | | |
|---|---|---|---|---|---|
| | | Relational schema | UML class diagram | Web portal | E/R diagram |
| Understandability | SMML | Group A | Group A | Group B | Group B |
| | TEXTUAL | Group B | Group B | Group A | Group A |
| Modifiability | SMML | Group B | Group B | Group A | Group A |
| | TEXTUAL | Group A | Group A | Group B | Group B |

each model according to their opinion. Finally, each subject had to answer a qualitative questionnaire related to the usability of the SMML. Table 7 shows the specific models each subject received according to his/her group and the type of exercise.

To develop the understandability questionnaires, the same template was followed for the SMML and textual representation. This was true for the case of modification requests (see Table 7) too. As a consequence, the complexity of assignments was balanced in both

Author's personal copy



**Table 7** Specific material for each subject classified by groups and types of exercises

| Group | Type of exercise | Models | | | |
|---|---|---|---|---|---|
| A | Understandability (6 questions) | 1 | 2 | 3 | 4 |
| A | Modifiability (4 requirements) | 5 | 6 | 7 | 8 |
| B | Understandability (6 questions) | 5 | 6 | 7 | 8 |
| B | Modifiability (4 requirements) | 1 | 2 | 3 | 4 |

the textual representation and graphical representation. To achieve this, a template was designed and followed, to derive the questions and modification requests for inclusion in each model, thus avoiding their having a possible influence on the results.

## 4.2 Execution of the experiment

The authors checked the material several times in order to guarantee that it contained no mistakes. A pilot experiment was also run beforehand by a software engineer with a similar background to that of the subjects who would be participating in the experiment and with ten models. Once the pilot experiment had been completed, we verified that the average time was appropriate if fatigue effects were to be avoided and that the learning material, experimental models and tasks were understandable. The obtained results were as expected, so no significant design changes were necessary.

The experimental run consisted of giving the material to the subjects, who had a strict deadline by which the completed material had to be returned. The experiment was not, therefore, directly supervised by experimenters. This was not considered necessary in the context of this experiment, as the subjects had previous experience in experimentation and they were committed to collaborating with this research by strictly following the instructions they received.

## 4.3 Data analysis and interpretation

This section describes the data collection and further analysis and interpretation.

### *4.3.1 Descriptive statistics*

Once the task had been carried out, we collected the forms filled in by the subjects, ensuring that they were complete. As all the forms were indeed complete, the whole set of subject results was considered. When the results were obtained, we first performed a descriptive analysis of the data. Tables 8 and 9 show the main descriptive statistics for the understandability and modifiability measures, respectively. Since the valuation measure belongs to an ordinal scale, the median has been calculated for it. Graphical representation of descriptive data by means of boxplots is shown in Figs. 7 and 8 for efficiency and time, respectively (see Appendix B for the rest of the dependent variable measure boxplots).

The results of the experiment show that the *time* the subjects spent on understanding and modifying the questions about the models which were defined with the SMML was shorter than the results for the models which were not defined with the SMML (textual notation). Subjects working with the SMML spent on average 12.525 fewer seconds (6.7% better) on understanding tasks which was slightly shorter, and 89.77 s (1 m 29 s) less than on the textual method when they worked on modifications (36.5% better). Moreover, the *correctness* of

Author's personal copy



**Table 8** Understandability results: descriptive statistics

| Understandability measures (dependent variables) | SMML | | TEXTUAL | |
|---|---|---|---|---|
| | Mean | Std dev | Mean | Std dev |
| Time (m, s) | 3 m 4 s | 1 m 35 s | 3 m 16 s | 1 m 23 s |
| Correctness (%) | 93.33 | 9.091 | 88.33 | 16.537 |
| Efficiency | 0.639 | 0.32 | 0.535 | 0.24 |
| Valuation (Median) | Median = 2 | 0.868 | Median = 3 | 0.992 |

Valuation: 1 = very simple, 2 = quite simple, 3 = normal, 4 = quite complex, 5 = very complex

**Table 9** Modifiability results: descriptive statistics

| Modifiability measures | SMML | | TEXTUAL | |
|---|---|---|---|---|
| | Mean | Std dev | Mean | Std dev |
| Time (m, s) | 4 m 5 s | 1 m 36 s | 5 m 35 s | 2 m 44 s |
| Efficiency | 0.482 | 0.230 | 0.286 | 0.200 |
| Correctness (%) | 99.37 | 3.953 | 67.50 | 22.072 |
| Valuation (Median) | Median = 2 | 0.944 | Median = 4 | 0.987 |

Valuation: 1 = very simple, 2 = quite simple, 3 = normal, 4 = quite complex, 5 = very complex

understandability answers was 5.66% better when using the SMML. Moreover, the correctness of modifiability answers was 47.21% better with use of the SMML in diagrams. The *efficiency* was consequently improved using the SMML, the difference of efficiency in understandability was on average 0.104 better (19.4%) in favour of the SMML, and this positive difference in modifiability was 0.196 (an improvement of 68.5%). Finally, the *subjective valuation* of the difficulty of the diagrams which were defined with the SMML was better than the valuations of the models which were not defined with the SMML.

#### *4.3.2 Hypotheses contrast and graphical interpretation*

Once the descriptive data had been analysed, the following step was to obtain an insight into whether these differences were statistically significant. We performed an ANOVA statistical test (significance level $\alpha = 0.05$), in order to analyse the interaction between the independent variables under study when the measurement of the dependent variables was repeated. The statistical results for the hypotheses testing in relation to the understandability measure and modifiability measure of the dependent variables are summarized in Table 10.

As Table 10 shows, non-conclusive results were obtained to explain the differences in understandability values, as statistical evidence was not found except in the subjective valuation variable ($p = 0.01 < \alpha$); $H_{0VT}$ can be rejected and the use of the SMML is therefore considered by the subjects to be easier than the use of textual notation in understandability.

With regard to modifiability we obtained evidence that

- Non-SMML models were more difficult to modify than the SMML versions. Significant differences were found with regard to the time, correctness, efficiency ($p = 0.000 < \alpha$) and subjective valuation ($p = 0.02 < \alpha$), and $H_0S_m$ can therefore be rejected.

Author's personal copy



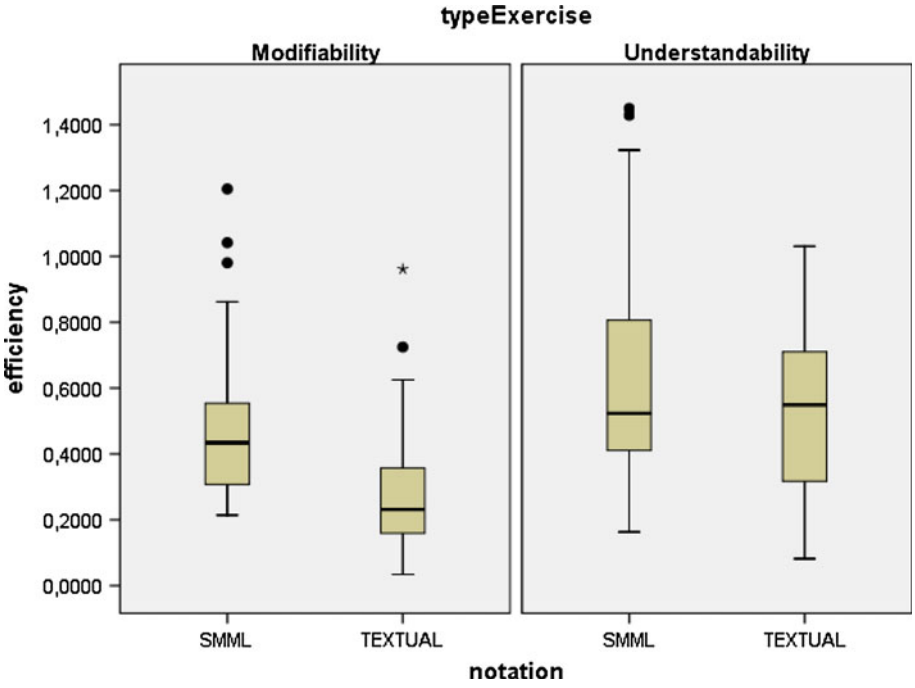



**Fig. 7** Boxplots of modifiability/understandability efficiency

- The UoD also had an influence on the modifiability time, efficiency, and subjective valuation, i.e., subjects obtained different modifiability performances and opinions when they worked with different UoDs ($p = 0.000 < \alpha$, $H_0$ UoD can be rejected).
- The combined use of notation and UoD did not influence the results obtained.

On the other hand, as can be observed in column Group of Table 10, there were no significant differences in any measure of the dependent variable between the subjects in the two groups A and B. The group to which the subjects belonged consequently did not affect the ease with which they understood or modified the software measurement models.

In addition to the numeric analysis, a graphical analysis was performed by means of profile plots, which show whether there are interactions between the variables in the study. Specifically, plots were obtained to show whether there are interactions between the UoD (body of the plot) and the use or non-use of SMML (X axis) with regard to the understandability and modifiability time/efficiency/valuation/correctness (Y axis), respectively.

Figures 7, 8 and 9 show that the time and efficiency involved in carrying out the modifiability exercises is better when using the SMML language than when not using it, regardless of the UoD.

In relation to understandability, as stated earlier, the differences found were not statistically significant. Nevertheless, Fig. 10 shows that efficiency in understandability is better when the SMML language is used than when it is not used, regardless of the UoD to which it is applied. This profile shows that there is no interaction effect between the usage or non-usage of the SMML and the UoD, and these plots demonstrate that the use of the SMML produced better scores, irrespective of the UoD.

Author's personal copy



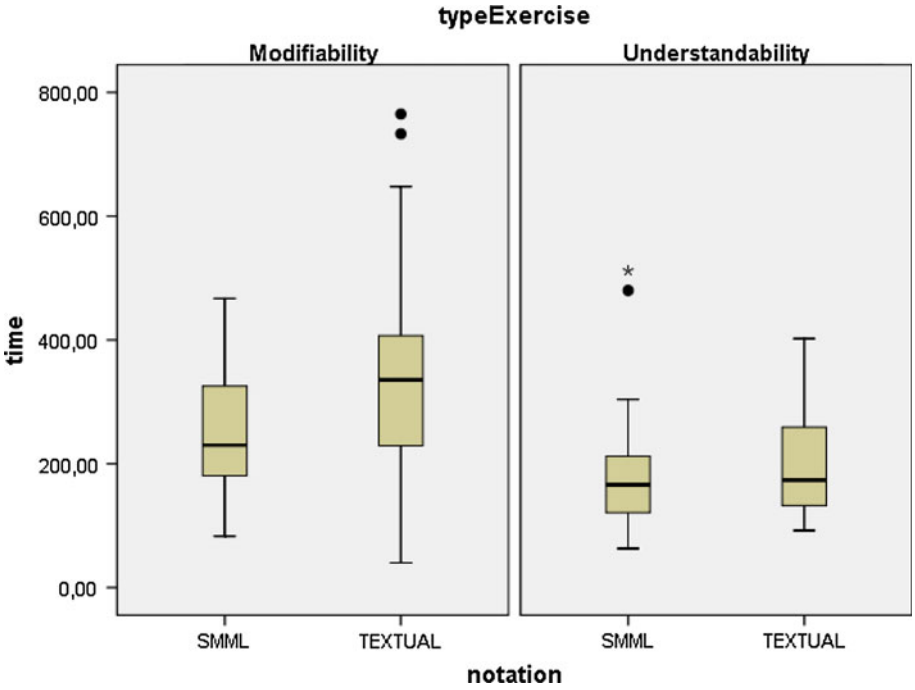



**Fig. 8** Boxplots of modifiability/understandability time

**Table 10** ANOVA results (significance levels)

| Variable dependent | Type of exercise | Notation | UoD | Notation × UoD | Group |
|---|---|---|---|---|---|
| Time | Understandability | 0.539 | 0.366 | 0.830 | 0.254 |
| | Modifiability | 0.002 | 0.000 | 0.423 | 0.212 |
| Efficiency | Understandability | 0.110 | 0.100 | 0.987 | 0.338 |
| | Modifiability | 0.000 | 0.000 | 0.747 | 0.324 |
| Correctness | Understandability | 0.093 | 0.161 | 0.390 | 0.158 |
| | Modifiability | 0.000 | 0.26 | 0.051 | 0.202 |
| Valuation | Understandability | 0.01 | 0.235 | 0.348 | 0.236 |
| | Modifiability | 0.02 | 0.000 | 0.096 | 0.110 |

In short, graphical analysis provided more insights into the potential usefulness of the SMML for modifiability for all UoDs. It can also be deduced that different scores were obtained for each UoD, which furthermore indicates that the application domain may also affect the ease of modification. Moreover, as can be observed in the profile plots, there is no interaction effect between the usage or non-usage of the SMML and the UoD as lines are near to be parallel. On the other hand, although statistical evidence was not found, there is good insight into the usefulness of the SMML with regard to understandability for all UoDs. As different scores were obtained for each UoD, the application domain also seems to affect the ease of understanding of the model.

Author's personal copy



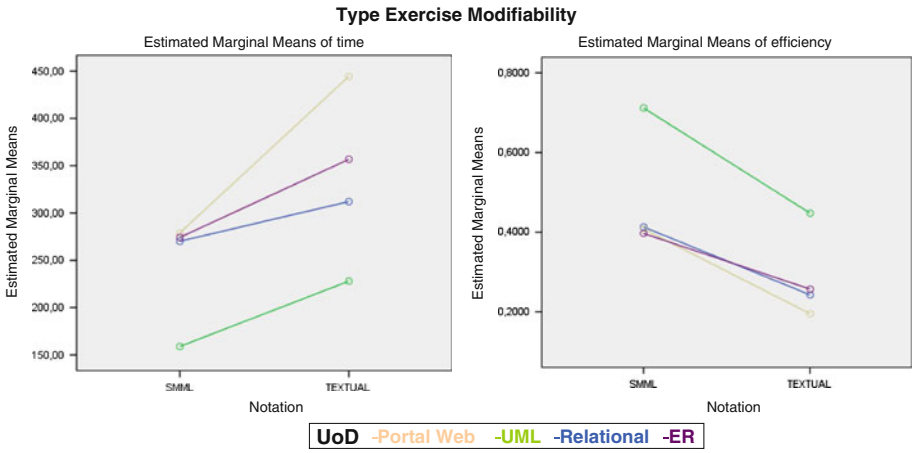



**Fig. 9** Profile plot diagram of the interaction of UoD × usage of SMML in the experiment for modifiability time and efficiency

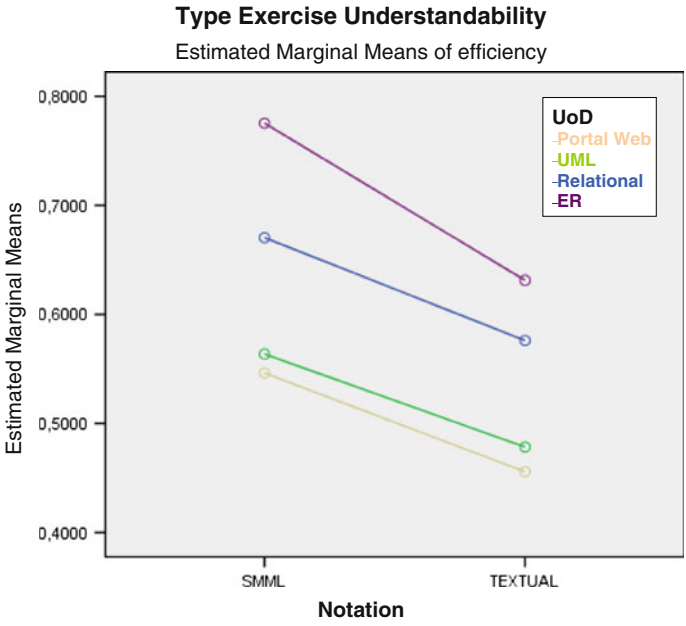



**Fig. 10** Profile plot diagram of the interaction of UoD × usage of SMML in the experiment for understandability efficiency

### *4.3.3 Valuation of the SMML elements*

Finally, a descriptive analysis was made with the data obtained from the qualitative questionnaire in order to detect whether any improvements could be made to the SMML

Author's personal copy



**Table 11** Valuations of language entities

| Element | Median | Element | Median | Element | Median |
|---|---|---|---|---|---|
| Information need | 1 | Description | 2 | Measurement function | 2 |
| Decision criteria | 1 | Unit of measurement | 2 | Measurement method | 2 |
| Base measure | 1 | Indicator | 2 | Quality model | 2 |
| Attribute | 2 | Analysis model | 2 | Scale | 2 |
| Derived measure | 2 | Measurable concept | 2 | Entity class | 3 |

Median: 1 = very suitable, 2 = quite suitable, 3 = normal, 4 = quite unsuitable, 5 = very unsuitable

**Table 12** Valuations of language associations

| Element | Median | Element | Median |
|---|---|---|---|
| Measurable concept (*relates*) attribute | 1 | Indicator (*satisfies*) information need | 2 |
| Measurable concept (*includes*) measurable concept | 1 | Base measure (*uses*) measurement method | 2 |
| Derived measure (*calculated with*) measurement function | 1 | Measurable concept (*is associated with*) information need | 2 |
| Entity class (*includes*) entity class | 1 | Quality model (*evaluates*) measurable concept | 2 |
| Indicator (*calculated with*) analysis model | 1 | Measure (*defined for*) attribute | 2 |
| Entity class (*has*) attribute | 2 | Quality model (*defined for*) entity class | 2 |
| Analysis model (*uses*) decision criteria | 2 | | |

Median: 1 = very suitable, 2 = quite suitable, 3 = normal, 4 = quite unsuitable, 5 = very unsuitable

language. Tables 11 and 12 show, in ascending order, the median of the scores assigned by the subjects to each entity and association of the SMML, respectively.

As Table 11 shows, the icons of eleven entities were considered as being suitable on average, three as quite suitable and only one as normal. Table 12 shows that the graphical interpretation of eight types of associations are considered as suitable and five as quite suitable. These results are therefore considered to be encouraging and provide preliminary feedback with regard to the usability of the concrete syntax of the SMML, according to the opinion of software engineers.

## 5 Experimental validity: discussion

The various issues which might have threatened the validity of the results were analysed during the planning process of the experiment.

*Conclusion Validity.* One issue which could affect the conclusion validity of this study is the size of the sample data (160 values, 8 models per subject × 20 subjects). We are aware of this, so we will consider carrying out replications of this study with a larger sample size, to reinforce the conclusion validity of the findings obtained.

*Construct Validity.* With regard to this aspect, the measures of the dependent variables (time, correctness, efficiency) are commonly used in studies which involve cognitive tasks

Author's personal copy



(Eysenck and Keane 2005), which is the nature of the research presented here, and are usually applied in empirical studies in which these dependent variables are evaluated (Genero et al. 2005; Patig 2008). Since these measures were obtained from the data collected from the forms filled in by the subjects, the measurements may have lacked accuracy, i.e., the subjects may have failed to record the exact time at which they completed the task. To moderate this threat, subjects received detailed instructions about how to use and fill in the questionnaires. Additionally, as stated in Sect. 4.2, the subjects were committed to this empirical study, and this therefore facilitated the process of ensuring that the values that they filled in were more accurate.

*Internal Validity.* The aspects that could have threatened the internal validity were tackled in the following manner.

- *Persistence Effects.* The experiment was carried out with subjects who had never done a similar experiment before, thus avoiding persistence effects.
- *Learning Effects.* The universe of discourse of each of the models included in the material for each subject was different, and the subjects were given the models in random order to alleviate this threat.
- *Knowledge of the Universe of Discourse*. The knowledge of the domain did not affect internal validity, since the measurement models were from different UoDs (ER, Relational, UML and Web), which were familiar to the subjects (software engineers).
- *Fatigue Effects.* The average duration of the experiments was one and half hours (including learning about the documentation), so fatigue effects did not appear.
- *Subject Motivation.* The subjects were highly committed to this research, and the results could be potentially beneficial to them, since they research and work on related software engineering topics.
- *Plagiarism and Influence Among and Between Subjects* were not controlled, but we do not believe that this aspect was decisive, given the profile of the subjects and their commitment to empirical studies such as this.
- *External Validity.* The material and tasks used were designed bearing in mind the time restrictions and the fact that the subjects were not professionals, as they belong to the academic setting and research environment. Replication of this empirical research with models representative of projects and with professionals as subjects will be considered in the future. The complete study can be accessed from http://alarcos.esi.uclm.es/smf/smml.

## 6 Related work

The nature of the present study means that it is of interest first of all to analyse existing papers which have compared graphical versus other language representations.

As can be observed in the following related papers, the comparison between graphical and textual languages is a problem which has always been present. Even today, we find a great variety of graphic languages "versus" their corresponding textual languages. Some examples are the following:

- In the end-user automation area, Automator (Apple 2010a) vs. AppleScript (Apple 2010b) on Mac OS X.
- In the scientific workflow area, Graphical languages such as Taverna (Hull et al. 2006; Oinn et al. 2006) vs. scripting-based language such as Swift (Zhao et al. 2007).

Author's personal copy



An early study performed by Shneiderman et al. (1977) describes previous research on flowcharts and a series of controlled experiments to test the utility of detailed flowcharts as an aid to program composition, comprehension, debugging and modification. No statistically significant difference between flowchart and non-flowchart groups has been shown.

Cunnif and Taylor have carried out several studies comparing visual and textual programming languages and studying the graphical representation of programming. An example of these works is the comparison of comprehension of Pascal and of an informationally equivalent visual notation, FPL ("First programming language"), both of which were in use as teaching languages for novices. They found an advantage for FPL in compression of micro-structure (Cunniff and Taylor 1987).

In the work by (Green et al. 1991), they analysed if graphical representations are inherently superior to textual representations. The graphical representation selected was LabView. LabView[4] (together with Simulink[5]) is an example of the commercially successful graphical programming (and modelling) tools in the end-user automation area. In the study, they found that the LabView representations consistently required much longer times for the subject to answer questions, whatever the structure involved was (circumstantial vs. sequential) or whatever the direction of the question was (forward vs. backward). Inspired by this work, Moher et al. (1993) conducted a study where three forms of petri net representations were tested against two textual program representations for comprehensibility. In general, the results indicated that the efficacy of a graphical program representation is not only task-specific but also highly sensitive to seemingly ancillary issues such as layout and the degree of factoring.

Pandey and Burnett (1993), in their work, compared time, ease and errors in the construction of code using visual and textual languages. They found that matrix and vector manipulation programs constructed using visual programming had fewer errors.

Yoder and Black (2006) describe the preliminary developments of comparing the use of LabVIEW (a graphical programming language) to MATLAB (a text-based language) in teaching discrete-time signal processing (DSP). In this work, the authors do not expect this study to answer the "which is better?" question. Rather, it will give experience in assessing what the tradeoffs are in choosing between two very different types of programming languages for teaching DSP.

On the other hand, as far as DSL development in Software Engineering is concerned, diverse papers exist. A number of these publications present methodologies, proposals, tools and patterns with which to facilitate the development of DSLs (Deursen et al. 2000; Cook 2004; Mernik et al. 2005; Kolovos et al. 2006; Pelechano et al. 2006; Özgür 2007). Other interesting papers can be useful in designing domain-specific languages thanks to the guidelines presented (Karsai et al. 2009), as well as for avoiding the worst practices for domain specific modelling (Kelly and Pohjonen 2009).

Numerous papers presenting DSL also exist: a QVT-like model transformation language (OMG 2005a) and its execution environment which is based on the Eclipse framework;

[4] LabVIEW (short for Laboratory Virtual Instrumentation Engineering Workbench) is a platform and development environment for a visual programming language from National Instruments.

[5] Simulink is an environment for multidomain simulation and Model-Based Design for dynamic and embedded systems. It provides an interactive graphical environment and a customizable set of block libraries that let you design, simulate, implement, and test a variety of time-varying systems, including communications, controls, signal processing, video processing, and image processing.

Author's personal copy



ATL (ATLAS Transformation Language) (Jouault et al. 2006); KM3 (Kernel MetaMeta-Model) (Jouault and Bézivin 2006), which is a DSL for describing metamodels.

Finally, as regards existing DSLs for Software Measurement, Guerra et al. (2008) present a framework for the creation of domain specific visual languages (DSVL), in which a language called SLAMMER was developed as a case study to tackle measurement model representation. This language is part of the suite of model management tools that Guerra et al. have defined using graph grammars and graph transformations. Another interesting work in this area (measurement) is a Measurement-Domain Specific Language (MDSL) (Arpaia et al. 2009). This language tests procedure definition, measurement tasks synchronization, and instrument configuration is proposed. MDSL is a formal language specially designed for a specific domain of measurement and test, which has been applied to the specifications of superconducting magnet tests of the Large Hadron Collider at CERN.

The main contributions of the SMML are that it

- was developed specifically to support measurement modelling.
- is based on the SMO, which provides a solid conceptual base.
- gives initial insight, by means of an empirical study, into its potential for building more usable and maintainable software measurement models, as compared with textual representation..
- is integrated into the SMF for the automatic measurement of models.

As previously stated, there are other existing notations, but they use their own metamodel and consequently their own constructors. For this reason, the comparison of the SMML with respect to such notations was outside the scope of the present work.

## 7 Conclusions and future work

In this paper, the usability and maintainability (based on ISO/IEC 9126) of the SMML has been tested by means of an empirical study. The study has been carried out with the collaboration of a group of Software Engineering experts whose work consisted of defining software measurement models both with the SMML and with textual notation. Once the results of the experiment (graphical plots and the value of the significance levels for the measures of the dependent variables) had been studied, the main conclusions obtained were the following:

– The use of the SMML improves the modifiability of the software measurement models. Understandability is slightly better for the SMML, although statistical evidence was not obtained.
– The use of the SMML is considered easier (subjective valuation) than the use of textual notation in understandability or modifiability.
– The icons and their relationships used in the language are considered suitable.

These results demonstrate that it is of benefit to have a suitable notation to define software measurement models, elements which are a key factor in the SMF.

A case study applied in a real-world IT company (Mora et al. 2009) shows the potential benefits of using the SMF (the synergic combination of MDE and measurement). The company's measurement process was supported by the 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. In this case study,

Author's personal copy



the SMML has been used to define the source models (Software Measurement Models), which are important elements in the SMF.

The existence of the Software Measurement Modelling Language also improves the software measurement process carried out in the SMF.

Seeking to confirm the results obtained, one focus of future research will be the replication of this empirical study in new settings, particularly in those of industry, in which more realistic material and tasks will be considered. Some of the improvements considered are the following:

- An exercise that compares the difficulty in creating software measurement model by using SMML vs TEXTUAL from software measurement specifications.
- To modify the present qualitative questionnaire for the evaluation of the specific syntax, in which the icons are presented and the subjects must guess what they mean.
- Since the objective is to compare a set of software measurement models represented in different notation (visual vs textual), we have considered a not very complex model to speed up the execution—time of the exercises.

Within future work related to all we have presented here, one important effort will be the performing of an experiment to validate the usability and level of acceptance of the SMML editor integrated in the SMTool. Another important task to be undertaken will be to consider the size of measurement models as an experimental factor. This is an important point to address in prospective work, which will be done by enhancing the tool to support packaging and different views, as well as to conduct new empirical studies.

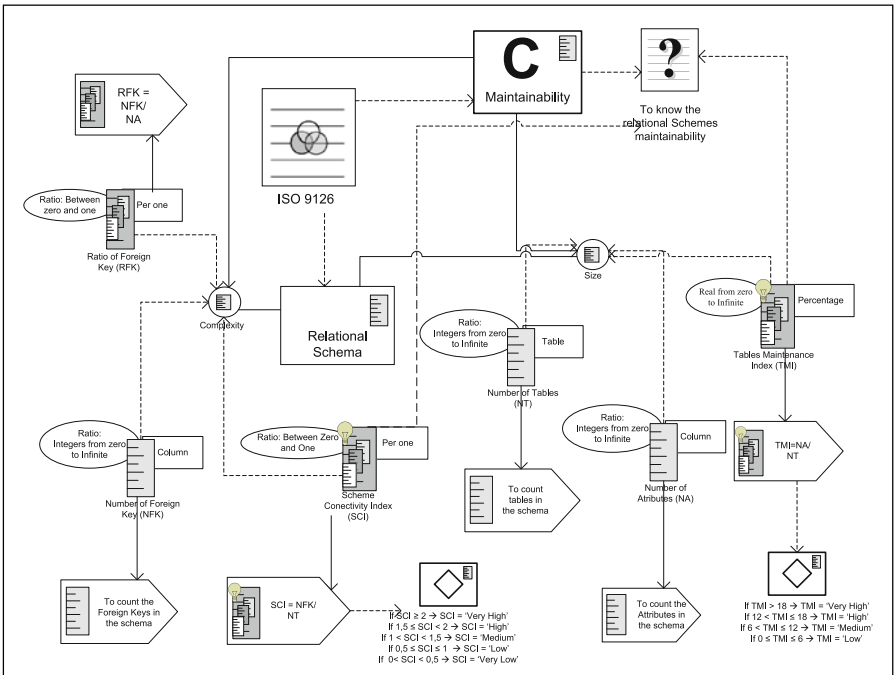
**Acknowledgments** This work has been partially supported by the projects: INGENIO (JCCM, PAC08-0154-9262), MEDUSAS (CDTI (MICINN), IDI-20090557), PEGASO/MAGO (MICINN and FEDER, TIN2009-13718-C02-01), and ALTAMIRA (JCCM, Fondo Social Europeo, PII2I09-0106-2463) projects.

## Appendix A: Example of definition of software measurement models

See Figs. 11, 12, 13.

Author's personal copy






**EXERCISE 1. Tasks to be carried out:**

Please note down starting time (indicating h:m:s): _____

1) **Answer the following statements with true or false according to the SMML diagram shown at the end of the exercise:**

   a) The Size, Complexity and Length *attributes* **are related** to the Maintainability *measurable concept*.

      T ☐ F ☐

   b) The NA *measure* **is defined for** the Size *attribute*.

      V ☐ F ☐

   c) The TMI *Indicator* **satisfies** the To know the maintainability of the relational schemas *Information need*.

      V ☐ F ☐

   d) The SCI *measure* **has** the *scale:* Ratio of between 0 and 1 and the *unit:* Foreign Key per Table.

      T ☐ F ☐

   e) The ISO9126 quality model **evaluates** the Usability *measurable concept*.

      T ☐ F ☐

   f) The RFK *derived measure* **is calculated** with the Count the keys not belonging to the schema *measure function*.

      T ☐ F ☐

Please note down finishing time (indicating h:m:s) _____

2) **Attempt to understand the SMML diagram and to evaluate the COMPLEXITY of the Software Measurement model according to your own criteria.**

   Very simple ☐ Quite simple ☐ Normal ☐ Quite complex ☐ Very complex ☐

**Fig. 11** Example of understandability exercise with SMML diagram

Author's personal copy



| Measurement metamodel elements (MOF classes) | E/R measurement model (instances M2) | Measurement metamodel relations (MOF associations) | | E/R measurement model (links M2) |
|---|---|---|---|---|
| Attribute | Size Complexity | Has | Entity class<br>Attribute | Relational Schema<br>Size, Complexity |
| Entity class | Relational Schema | Defined for | Quality model<br>Entity class | ISO 9126<br>Relational Schema |
| Measurable concept | Maintainability | Evaluates | Quality model<br>Measurable concept | ISO 9126<br><br>Maintainability |
| Quality model | ISO 9126 | Relates | Attribute<br>Measurable concept | Size Complexity<br><br>Maintainability |
| Information need | To know the Relational Schemas' maintainability | Is associated With | Measurable Concept<br><br>Information need | Maintainability<br><br>To know the Relational Schemas' maintainability |
| Measure | NFK, RFK, SCI | Defined for | Measure<br>Attribute | NFK, RFK, SCI<br>Complexity |
| Measure | NT, NA, TMI | Defined for | Measure<br>Attribute | NT, NA, TMI<br>Size |
| Indicator | TMI, SCI | Satisfies | Indicator<br>Information need | TMI, SCI<br><br>To know the Relational Schemas' maintainability |

| Base Measure | Description | Measurement method | Scale | Unit |
|---|---|---|---|---|
| NFK | Number of Foreign keys | Counting the foreign keys in the schema | Ratio: integers from zero to infinite | Column |
| NT | Number of Tables | Counting tables in the schema | Ratio: integers from zero to infinite | Table |
| NA | Number of Attributes | Counting the attributes in the schema | Ratio: integers from zero to infinite | Column |

| Derived Measure | Description | Measurement Function | Scale | Unit |
|---|---|---|---|---|
| RFK | Ratio of foreign key | RFK = NFK/NA | Ratio: between zero and one | Per one |

| Indicator | Description | Analysis Model | Decision Criteria | Scale / Unit |
|---|---|---|---|---|
| SCI | Schema Connectivity Index | SCI = NFK/NT | If SCI ≥ 2 → SCI = **'Very High'**<br>If 1,5 ≤ SCI < 2 → SCI = **'High'**<br>If 1 < SCI < 1,5 → SCI = **'Medium'**<br>If 0,5 ≤ SCI ≤ 1 → SCI = **'Low'**<br>If 0< SCI < 0,5 → SCI = **'Very Low'** | Ratio: between zero and one / Per one |
| TMI | Tables Maintenance Index | TMI = NA/NT | If TMI > 18 → TMI = **'Very High'**<br>If 12 < TMI ≤ 18 → TMI = **'High'**<br>If 6 < TMI ≤ 12 → TMI = **'Medium'**<br>If 0 ≤ TMI ≤ 6 → TMI = 'Low' | Ratio: Real from zero to infinite / Percentage |

**Fig. 12** Example of modifiability exercise with a TEXTUAL notation

Author's personal copy



**EXERCISE 1: Tasks to be carried out:**

Please note down starting time (indicating h:m:s): _____

1) **Make the necessary modifications to the tables shown at the end of the exercise to satisfy the following software measurement requirements (indicating the section number in each modification):**

   a) Delete the NT *base measure*. If it's necessary, delete the measures that use this base measure.

   b) Add the NPK (Number of primary keys) *base measure* **defined for** the complexity *attribute* which **uses** the count the number of primary keys *measurement method*. With the Ratio: whole numbers from 0 to infinity *scale* and primary key *unit*.

   c) Modify the ISO 9126 *quality model* so that it **evaluates** the Maintainability and Portability *measurable concepts*.

   d) Modify the to Know the portability of the relational schemas *Information need* so that it **relates** to the Portability *measurable concept*.

Please note down the finishing time (indicating h:m:s):_____

2) **Attempt to understand the tables and evaluate the COMPLEXITY of the Software Measurement model according to your own criteria.**

   Very simple ☐ Quite simple ☐ Normal ☐ Quite complex ☐ Very complex ☐

**Fig. 13** Example of modifiability exercise with a TEXTUAL notation (cont.)

## Appendix B: Boxplots

See Figs. 14, 15, 16, 17, 18.

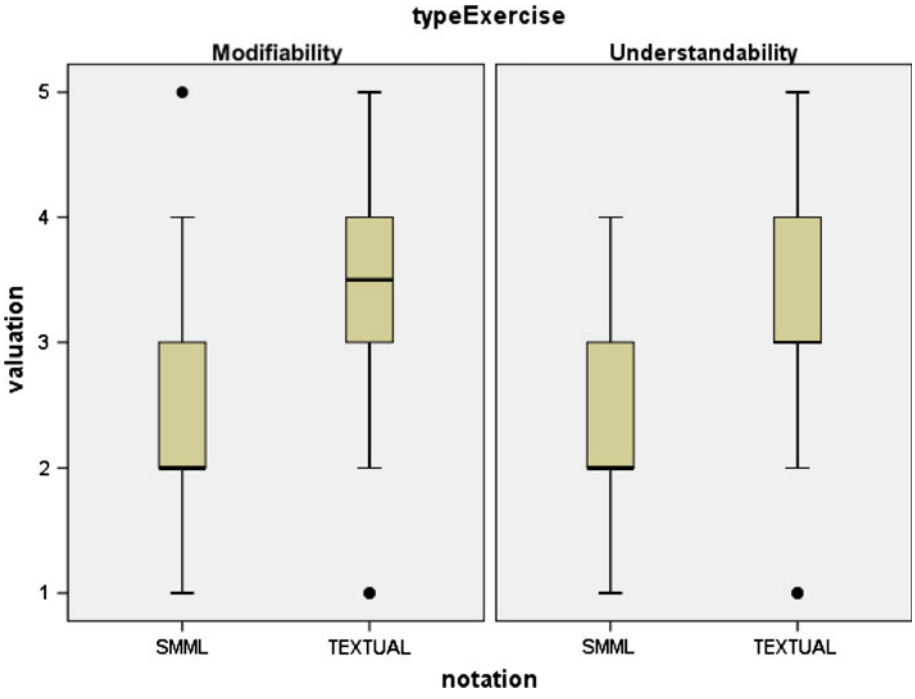



**Fig. 14** Boxplot diagram of the interaction of UoD × usage of SMML in the experiment for modifiability/understandability valuation

Author's personal copy



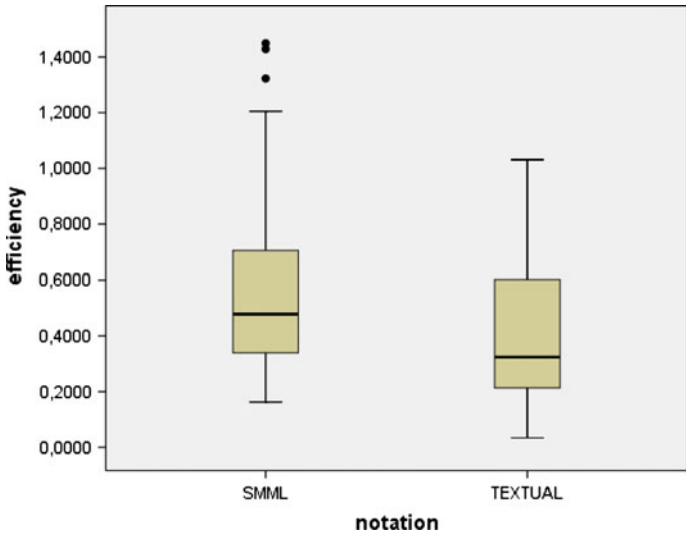



**Fig. 15** Boxplot diagram for the usage of SMML in the experiment for the efficiency

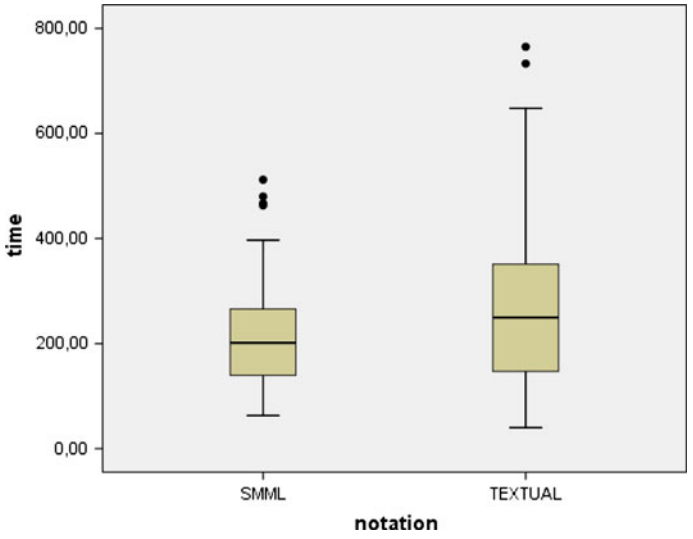



**Fig. 16** Boxplot diagram for the use of SMML in the experiment for the time

Author's personal copy



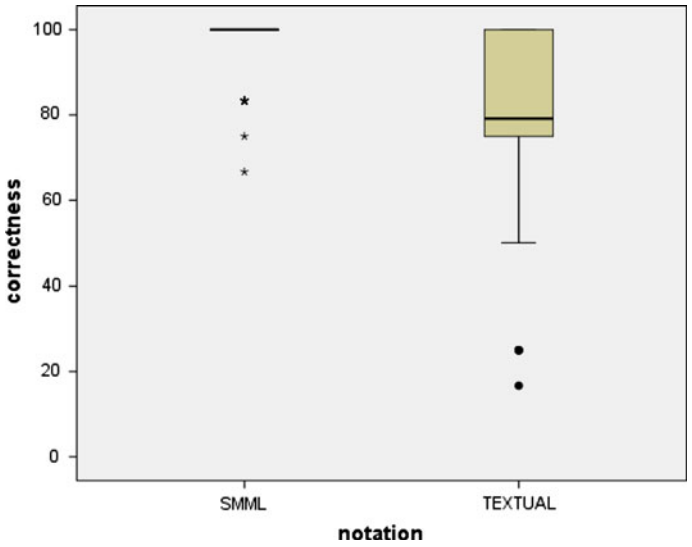



**Fig. 17** Boxplot diagram for the use of SMML in the experiment for the correctness

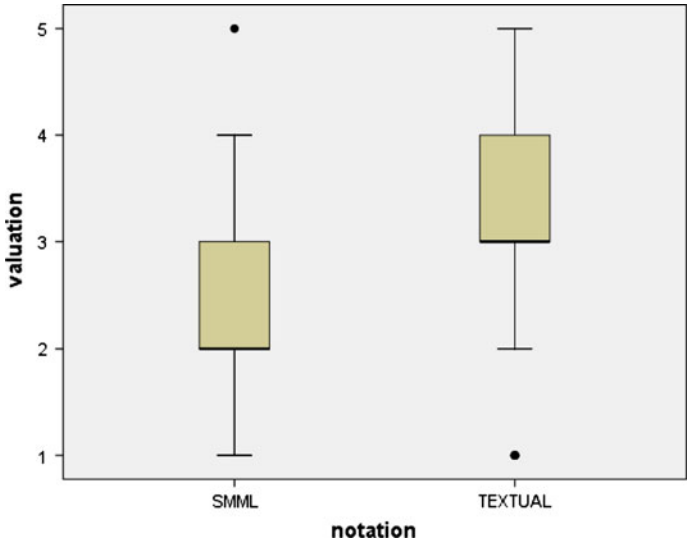



**Fig. 18** Boxplot diagram for the usage of SMML in the experiment for the valuation

## References

Ahmad, R. (1999). Visual languages: A new way of programming. *Malaysian Journal of Computer Science, 12*(1), 76–81.

Apple. (2010a). Automator, from www.macosxautomation.com/automator. Accessed May 2010.

Apple. (2010b). appleScript, from http://www.macosxautomation.com/applescript/). Accessed May 2010.

Arpaia, P., Buzio, M., Fiscarelli, L., Inglese, V., Commara, G. L., & Walckiers, L. (2009). Measurement-domain specific language for magnetic test specifications at CERN. 2009 IEEE instrumentation and measurement technology conference, CERN/TE 2009-002, 1716–1720.

Author's personal copy



Basili, V. R., Shull, F., & Lanubile, F. (1999). Building knowledge through families of experiments. *IEEE Transactions on Software Engineering, 25*(4), 456–473.

Bézivin, J. (2004). In search of a basic principle for model-driven engineering [Special Issue]. *Novatica Journal, 5*, 21–24.

Bézivin, J., Jouault, F., & Touzet, D. (2005). Principles, standards and tools for model engineering. 10th IEEE international conference on engineering of complex computer systems (ICECCS'2005), 28–29.

Cook, S. (2004). Domain-specific modeling and model driven architecture. In D. S. Frankel & J. Parodi (Eds.), *The MDA journal: Model driven architecture straight from the masters* (Chap. 5, pp. 1–10). Tampa, FL: Meghan-Kiffer Press.

Cunniff, N., & Taylor, R. P. (1987). Graphical vs. textual representation: An empirical study of novices' program comprehension. Empirical studies of programmers: Second workshop, pp. 114–131.

Deursen, A. v., Klint, P., & Visser, J. (2000). Domain-specific languages: An annotated bibliography. *SIGPLAN Notices, 35*(6), 26–36.

Eysenck, M. W., & Keane, M. T. (2005). *Cognitive psychology: A student's handbook*. Mahwah, NJ: Lawrence Erlbaum Associates.

Feilkas, M. (2006). How to represent models, languages and transformations? Proceedings of the 6th OOPSLA workshop on domain-specific modeling (DSM'06), pp. 204–213.

Fenton, N., & Pfleeger, S. L. (1997). *Software metrics: A rigorous & practical approach* (2nd ed.). Boston, MA: PWS Publishing Company.

García, F., Bertoa, M. F., Calero, C., Vallecillo, A., Ruíz, F., Piattini, M., et al. (2006). Towards a consistent terminology for software measurement. *Information & Software Technology, 48*(8), 631–644.

García, F., Ruiz, F., Cruz, J., & Piattini, M. (2003). Integrated measurement for the evaluation and improvement of software processes. Proceedings of the 9th European workshop on software process technology (EWSPT'9), Lecture Notes in Computer Science, 2786, pp. 94–111.

García, F., Serrano, M., Cruz-Lemus, J., Ruiz, F., & Piattini, M. (2007). Managing software process measurement: A metamodel-based approach. *Information Sciences, 177*, 2570–2586.

Genero, M., Moody, D., & Piattini, M. (2005). Assessing the capability of internal metrics as early indicators of maintenance effort through experimentation. *Journal of Software Maintenance, 17*(3), 225–246.

Green, T. R. G., Petre, M., & Bellamy, R. K. E. (1991). Comprehensibility of visual and textual programs: A test of superlativism against the match-mismatch conjecture. 4th workshop on empirical studies of programmers, pp. 121–146.

Guerra, E., Lara, J. d., & Díaz, P. (2008). Visual specification of measurements and redesigns for domain specific visual languages. *Journal of Visual Languages and Computing, 19*(3), 399–425.

Hull, D., Wolstencroft, K., Stevens, R., Goble, C., Pocock, M., Li, P., & Oin, T. (2006). Taverna: A tool for building and running workflows of services. *Nucleic Acids Research, 34*(Web Server issue) pp. 729–732.

ISO/IEC. (2001). *ISO/IEC 9126-1: Software engineering-software product quality—Part 1: Quality model.* Geneva, Switzerland, International Organization for Standardization.

Jedlitschka, A., & Ciolkowski, M. (2005). Reporting guidelines for controlled experiments in software engineering. ACM/IEEE international symposium on empirical software engineering, pp. 95–195.

Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., & Valduriez, P. (2006). ATL: A QVT-like transformation language. Companion to the 21th annual ACM SIGPLAN conference on object-oriented programming, systems, languages, and applications, OOPSLA 2006, pp. 719–720.

Jouault, F., & Bézivin, J. (2006). KM3: A DSL for metamodel specification. Formal methods for open object-based distributed systems, 8th IFIP WG 6.1 international conference, FMOODS 2006, 4037, pp. 171–185.

Juristo, N., & Moreno, A. M. (2001). *Basics of software engineering experimentation*. Boston, MA: Kluwer Academic Publishers.

Karsai, G., Krahn, H., Pinkernell, C., Rumpe, B., Schindler, M., & Völkel, S. (2009). Design guidelines for domain specific languages. The 9th OOPSLA workshop on domain-specific modeling, pp. 7–13.

Kelly, S., & Pohjonen, R. (2009). Worst practices for domain-specific modeling. *IEEE Software, 26*(4), 22–29.

Kitchenham, B. A., Pfleeger, S. L., Pickard, L. M., Jones, P. W., Hoaglin, D. C., Emam, K. E., et al. (2002). Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering, 28*(8), 721–734.

Kolovos, D. S., Paige, R. F., Kelly, T., & Polack, F. A. C. (2006). Requirements for domain-specific languages. First ECOOP workshop on domain-specific program development (ECOOP'06).

Kurtev, I., Bézivin, J., Jouault, F., & Valduriez, P. (2006). Model-based DSL frameworks. OOPSLA Companion 2006, pp. 602–616.

Mernik, M., Heering, J., & Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM Computing Surveys (CSUR), 37*(4), 316–344.

Author's personal copy



Moher, T. G., Mak, D. C., Blumenthal, B., & Leventahal, L. M. (1993). Comparing the comprehensibility of textual and graphical programs: The case of petri nets. Palo Alto, pp. 137–161.
Mora, B., García, F., Ruiz, F., & Piattini, M. (2008b). SMML: Software measurement modeling language. The 8th OOPSLA workshop on domain-specific modeling, pp. 52–59.
Mora, B., García, F., Ruiz, F., & Piattini, M. (2009). Model-driven software measurement framework: A case study. The 9th international conference on quality software, QSIC 2009, pp. 239–248.
Mora, B., García, F., Ruiz, F., Piattini, M., Boronat, A., Gómez, A., Carsí, J. Á., & Ramos, I. (2008a). Software measurement by using QVT transformation in an MDA context. 10th International conference on enterprise information systems—ICEIS 2008, 1, pp. 117–124.
Mora, B., Ruiz, F., Garcia, F., & Piattini, M. (2008c). *SMML: Software measurement modeling language*, Department of Computer Science. University of Castilla—La Mancha, from http://www.uclm.es/dep/tsi/pdf/SMML_Software_Measurement_Modeling_Language.pdf.
Oinn, T., Greenwood, M., Addis, M., Alpdemir, N., Ferris, J., Glover, K., et al. (2006). Taverna: Lessons in creating a workflow environment for the life sciences. *Concurrency and Computation: Practice and Experience, 18*(10), 1067–1100.
OMG. (2003). *OCL 2.0—OMG final adopted specification*, Object Management Group.
OMG. (2005a). Query/view/transformation (QVT) Standard Specification.
OMG. (2005b). *UML specification: Superstructure version 2.0*, Object Management Group, from http://www.omg.org/docs/formal/05-07-04.pdf.
Özgür, T. (2007). *Comparison of Microsoft DSL tools and eclipse modeling frameworks for domain-specific modeling in the context of the model driven development*. School of Engineering. Ronneby, Sweden, Blekinge Institute of Technology: p. 56.
Pandey, R. K., & Burnett, M. M. (1993). Is it easier to write matrix manipulation programs visually or textually? An empirical study. IEEE symposium on visual languages, pp. 344–351.
Patig, S. (2008). A practical guide to testing the understandability of notations. Fifth Asia-Pacific conference on conceptual modelling (APCCM 2008), pp. 49–55.
Pelechano, V., Albert, M., Javier, M., & Carlos, C. (2006). Building tools for model driven development comparing Microsoft DSL tools and eclipse modeling plug-ins. Desarrollo de Software Dirigido por Modelos—DSDM'06.
Shneiderman, B., Mayer, R., McKay, D., & Heller, P. (1977). Experimental investigations of the utility of detailed flowcharts in programming. *Communications of the ACM, 20*(6), 373–381.
Shull, F., Carver, J. C., Vegas, S., & Juzgado, N. J. (2008). The role of replications in empirical software engineering. *Empirical Software Engineering, 13*(2), 211–218.
Völter, M. (2009). MD* Best practices. *Journal of Object Technology, 8*(6), 79–102.
Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2000). *Experimentation in software engineering: An introduction*, International Series in Software Engineering.
Yoder, M., & Black, B. (2006). A study of graphical vs. textual programming for teaching DSP. American Society for Engineering Education Annual Conference & Exposition.
Zhao, Y., Hategan, M., Clifford, B., Foster, I., Laszewski, G. V., Raicu, I., et al. (2007). Swift: Fast, reliable, loosely coupled parallel computation. 2007 IEEE Congress on Services, pp. 199–206.

## Author Biographies

**Beatriz Mora** graduated in Computer Engineering at the University of Castilla la Mancha in 2005. She holds a Masters in Advanced Information Technologies from the School of Informatics of the University of Castilla-La Mancha, Ciudad Real, Spain, and is currently studying for her a doctorate. She is a member of the ALARCOS research group in the Department of Technology and Information Systems at the same university. Her research interests include generic software measurement, software quality, and the modelling and development of Domain Specific Languages. She is the author of several papers, one of which has been published in an international journal, and several of which have been presented at national and international conferences. She is currently working as an analyst and is the person responsible for quality assurance at Indra Software Labs, a software development company.

Author's personal copy



**Félix García** received his M.S. (2001) and Ph.D. (2004) degrees in computer science from the University of Castilla-La Mancha (UCLM). He is currently an associate professor in the Department of Information Technologies and Systems at the UCLM. He is member of the Alarcos Research Group, and his research interests include business process management, software processes, software measurement and agile methods.

**Francisco Ruiz** is PhD in Computer Science for the University of Castilla-La Mancha (UCLM), and MSc in Chemistry–Physics for the Complutense University of Madrid. He is an associate professor of the Department of Information Technologies and Systems at UCLM in Ciudad Real (Spain). He has been Dean of the Faculty of Computer Science between 1993 and 2000. Previously, he was Computer Services Director in this university (1985–1989) and he has also worked in private companies as analyst-programmer and project manager. His current research interests include business processes modelling, improvement and measurement; software process engineering; software maintenance; and methodologies for planning and managing software projects. He has published 26 papers in refereed international journals and more than one hundred papers in others journals, congresses, conferences or workshops, and several books and chapters. He has been member of more than 30 program committees and 8 organizing committees. He belongs to several scientific and professional associations (ACM, IEEE-CS, ISO JTC1/SC7, EASST, ATI-IFIP).

**Mario Piattini** is full professor at the UCLM. He holds the PhD degree in Computer Science from the Technical University of Madrid and leads the Alarcos Research Group. He is CISA, CISM and CGEIT by ISACA. His research interests include software quality, metrics and maintenance. He is the Director of the Joint UCLM-Indra Software Research and Development Center. He is member of ACM and IEEE Computer Society.