Design of a Functional Size Measurement Procedure for a Model-Driven Software Development Method[*]

Beatriz Marín[1], Nelly Condori-Fernández[1] and Oscar Pastor[1]

[1] Centro de Investigación en Métodos de Producción de Software,
Universidad Politécnica de Valencia,
Camino de Vera s/n,
46022 Valencia, Spain
{bmarin, nelly, opastor}@pros.upv.es

Abstract. The capability to accurately quantify the size of software developed with a Model-Driven Development (MDD) method is critical to software project managers for evaluating risks, developing project estimates, and having early project indicators. This paper presents a measurement procedure defined according to the last version of the ISO 19761 standard measurement method. The measurement procedure has been designed to measure the functional size of object-oriented applications generated from their conceptual models by means of model transformations. The measurement procedure is structured in three phases: the strategy phase, where the purpose of the measurement is defined; the mapping phase, where the elements of the conceptual model that contribute to the functional size are selected; and the measurement phase, where the functional size of the generated application is obtained.

Keywords: Conceptual model, Object orientation, Functional size measurement, COSMIC, MDD.

1 Introduction

Models are abstractions of the reality that help to understand complex problems and their potential solutions [22]. Model-Driven Development (MDD) methods have been developed to take advantage of the benefits of the use of models: a simplified view of the problem (using concepts that are much less bound to the underlying implementation technology and are much closer to the problem domain); and an easy way to specify, understand, and maintain the model. Since MDD methods are focused on models and model transformations, these allow the achievement of the automatic generation of the final product. To do this, the models (conceptual models) must have enough semantic formalization to specify all the functionality of the final application and also to avoid different interpretations for the same model.

[*] This work has been developed with the support of MEC under the project SESAMO TIN2007-62894 and co financed by FEDER.

The adoption of MDD methods has presented new challenges, such as the need to accurately quantify the functional size of the generated products from their conceptual models. Since the functional size of applications is essential to apply estimation models, defect models, and budget models [17], it is very important to obtain the functional size of the applications so that the project leader generates indicators to facilitate the project management and to assure the quality of the final product.

To measure the functional size of software applications, four measurement methods have been recognized as standards: IFPUG FPA [13], MK II FPA [14], NESMA FPA [15], and COSMIC FFP [12]. These methods have been illustrated in the measurement of the functional size of final applications. However, project leaders need indicators in the early stages of software development for a better management of MDD projects. For this reason, it is necessary to define how the measurement standards can be applied to the conceptual models that allow the generation of the final application. The specification of the way in which the measurement method must be applied to a phase of the development of a software application is named measurement procedure [9].

The COSMIC measurement method can be applied to any type of software and allows the measurement of multi-layer applications, in contrast to other functional size measurement methods (such as IFPUG FPA, NESMA FPA, and MK II FPA). For this reason, we have selected the COSMIC measurement method to specify a measurement procedure that can be applied to conceptual models.

The objective of this work is to design a measurement procedure that allows the application of the COSMIC measurement method to conceptual models, which are used by a MDD method to generate a final application by means of model transformations. Thus, the project leader will have the accurate functional size available to calculate productivity indicators, the price to be charged to clients, the defects in the models, etc.

The rest of the paper is organized as follows: Section 2 presents the phases and activities of the last version of the COSMIC measurement method and the measurement procedures based on COSMIC to measure conceptual models. Section 3 presents the design of a measurement procedure that applies COSMIC to measure the functional size of final applications from their object-oriented conceptual models. Finally, Section 4 presents some conclusions and further work.

2 Background and Related Works

The ISO/IEC 14143-1 [11] standard defines *functional size* as the size of the software derived by quantifying the functional user requirements. This standard also defines a Functional Size Measurement (FSM) as the process of measuring the functional size. In addition, this standard defines a FSM method as the implementation of a FSM that is defined by a set of rules, which is defined in accordance with the mandatory features defined in the ISO/IEC 14143-1.

The COSMIC measurement method was first recognized as a standard measurement method [12] because it fulfilled the characteristics defined in the ISO/IEC 14143-1 [10] and was verified with the ISO/IEC 14143-2 [11]. Later, the

COSMIC measurement method was improved maintaining the concepts and the characteristics that allowed it to be recognized as a standard method. The last version of the COSMIC measurement method is version 3.0 [1], which is different from the previous version mainly because it has a new phase to define the measurement strategy and it changes the concepts of end-user viewpoint and developer viewpoint for a generic concept named functional user, which allows the measurement of each piece of software that makes up an application. Next, we describe in more detail this version of the COSMIC measurement method.

2.1 The COSMIC Measurement Method

The application of the COSMIC measurement method [1] includes three phases: the measurement strategy, the mapping of concepts, and the measurements of the identified concepts (see Figure 1).

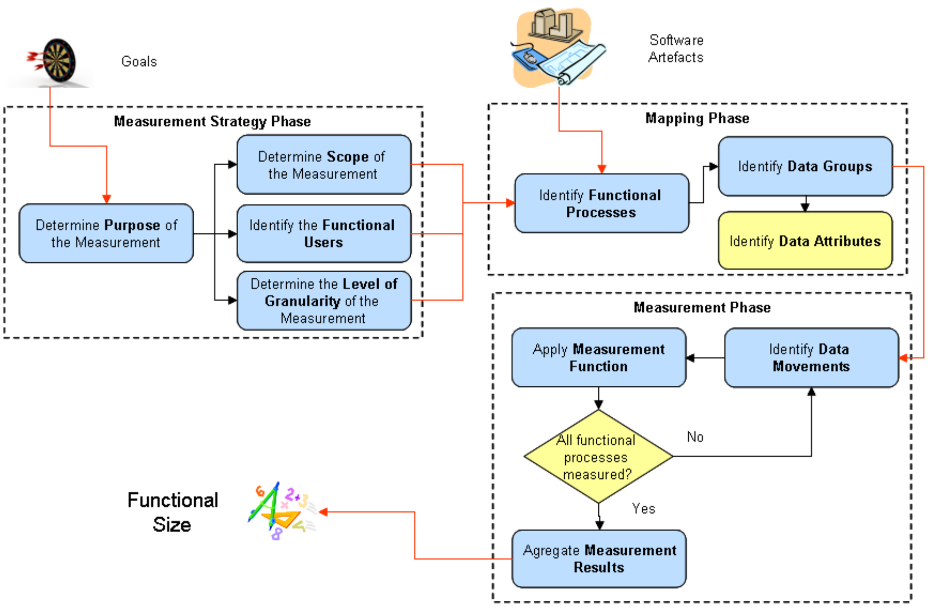
Goals
Software Artefacts
Measurement Strategy Phase
Determine Purpose of the Measurement
Determine Scope of the Measurement
Identify the Functional Users
Determine the Level of Granularity of the Measurement
Mapping Phase
Identify Functional Processes
Identify Data Groups
Identify Data Attributes
Measurement Phase
Apply Measurement Function
Identify Data Movements
All functional processes measured?
No
Yes
Agregate Measurement Results
Functional Size

Fig. 1. Phases and activities in the COSMIC measurement method.

In the *measurement strategy phase,* the purpose of the measurement exercise must be defined to explain why it is necessary to measure and what the measurement result will be used for. Next, the scope of the measurement must be defined in order to allow the set of user functional requirements that will be included in the measurement task to be selected. Then, the functional users of the application to be measured must be identified. The functional users are the types of users that send (or receive) data to (from) the functional processes of a piece of software. This phase also includes the identification of the boundary, which is a conceptual interface between the functional user and the piece of software that will be measured. Finally, the level of granularity of the description of a piece of software to be measured is identified.

In the *mapping phase,* the functional processes must be identified (i.e., the elementary components of a set of functional user requirements). Every functional process is triggered by a data movement from the functional user, and the functional process is completed when it has executed all the data movements required for the triggering event. It should be kept in mind that a triggering event is an event that causes a functional user of the piece of software to initiate one or more functional processes. Next, the data groups must be identified. This is a set of data attributes that are distinct, non empty, non ordered, non redundant, and that participates in a functional process. Finally, the identification of the data attributes, which comprise the smallest part of information of a data group, is optional.

In the *measurement phase,* the data movements (Entry, Exit, Read and Write) for every functional process must be identified. When all the data movements of the functional process are identified, the measurement function for the functional process must be applied. This is a mathematical function that assigns 1 CFP (Cosmic Function Point) to each data movement of the functional process. Then, after all the functional processes are measured, the measurement results are aggregated to obtain the functional size of the piece of software that has been measured.

2.2 Related Works

There are some approaches that apply COSMIC (in any of its versions) in order to estimate the functional size of future software applications from the conceptual model specifications [3] [5] [8] [16]. These proposals use scenarios, use case diagrams, sequence diagrams, and i* models to estimate the functional size. Therefore, these proposals estimate the functional size in conceptual models that not are used to generate the final application because these models do not have enough semantic expressiveness to specify all the functionality (for instance, in these models is not possible to specify the way in that the values of the attributes of a class change). Therefore, the functional size obtained by these proposals is not the accurate functional size of the final application. For this reason, the project leader can not use the functional size obtained to calculate indicators or use quality models (budget models, defect models, etc).

To avoid these problems, other proposals have been designed to measure the functional size of conceptual models that have more expressiveness to specify the functionality of the final applications and that allow the automatic generation of the final applications from these models. This is the case of Diab's proposal [7] and Poels' proposal [20]. Diab's proposal presents a measurement procedure to measure real time applications modelled with the ROOM language [23]. Diab's proposal uses a kind of statechart diagrams to measure the functional size. Poels' proposal presents a measurement procedure to object-oriented applications modelled with an event-based method named MERODE [6]. Poels' proposal allows the measurement of the functional size of Management Information Systems (MIS). The main disadvantage of both proposals is that the conceptual model does not allow the specification of all the functionality of the final application; for instance, that conceptual model does not allow the specification of the presentation of the application. Also, Poels' proposal is restricted to a specific technology because it uses the AndroMDA tool to specify the presentation of the application and to generate the final application. In addition, both proposals were defined using an old version of the COSMIC measurement method, and, therefore, these proposals do not take into account the improvements made to the COSMIC measurement method, for instance, the capability to measure the functional size of a piece of software of the application depending on the functionality that needs other piece of software.

None of the proposals for measurement procedures based on COSMIC allows the measurement of the accurately functional size of MIS applications in the conceptual model. Moreover, none of them take into account the improved version of COSMIC. The main limitation of the approaches presented above comes from the lack of expressiveness of the conceptual model that allows the generation of the final application. If the conceptual model has enough expressiveness to specify all the functionality of the final application, then a measurement procedure can accurately measure the functional size of the final application from its conceptual model.

The OO-Method approach [18] is an object-oriented method that allows the automatic generation of final applications by means of model transformations. It provides the semantic formalization needed to define complete and unambiguous conceptual models, allowing the specification of all the functionality of the final application in the conceptual model. This method has been implemented in a tool [4] that allows the automatic generation of fully working applications. The applications generated can be desktop or web MIS applications and can be generated in several technologies (for instance, java, C#, visual basic, etc.). The measurement procedure presented in this paper is based on this MDD method.

3 A FSM Procedure for Conceptual Models of an MDD Method

The design of a measurement procedure is a key stage in the development of a measurement procedure because the objective of the measurement, the artifact that will be measured, the measurement rules, and the measurement strategy are defined in this stage. It is very important to correctly perform the design of a procedure of measurement (correctly abstracting the elements that will be measured), since, otherwise, the procedure may not measure what should be measured according to the specifications in the base measurement method selected. It is also important to keep in mind the direct influence that the design of a measurement procedure has on the application of this procedure. For instance, if the design is incorrect, then the application of the procedure may be confused and erroneous measures may be obtained.

Since design is very important, in this section of the paper we present the design of a measurement procedure. In the design, the last version of the COSMIC measurement method has been selected. Therefore, the measurement procedure has three phases: strategy, mapping, and measurement. Moreover, the measurement procedure has been designed in the context of the OO-Method conceptual model because this model has the expressivity necessary to specify all the functionality of the final application. The following present the phases and the activities of the COSMIC method instantiated with concepts of the OO-Method conceptual model.

3.1 The Measurement Strategy Phase

Initially, the purpose of the measurement must be determined. The scope and the granularity level are determined depending on the purpose. Finally, the functional users are identified.

Purpose. The purpose of the measurement procedure has been defined in terms of a Goal-Question-Metric template [2]. Therefore, the purpose is:
To define a measurement procedure
with the purpose of applying the COSMIC measurement method to applications generated in an MDD environment
with respect to its functional size
from the point of view of the researcher
in the context of the conceptual model of an MDD development process named OO-Method.

Scope. The scope of the measurement procedure is the conceptual model of the OO-Method MDD technology. This conceptual model is comprised of four models: the object model, the dynamic model, the functional model, and the presentation model. The object model defines the structure and static relationships between the classes. The dynamic model defines the possible valid lives for the objects of a class and the interaction among objects. The functional model captures the semantics associated to object state changes, triggered by the occurrence of events. Finally, the presentation model allows the specification of the user interfaces in an abstract way. With all of these models, the conceptual model has all the details needed for the generation of the final application. The complete definition of the elements of the conceptual model of OO-Method is described in detail in [19].

Since the OO-Method software applications are generated according to a three-tier software architecture that is structured in a hierarchy, we distinguish three independent *layers* of the final application: the client layer, the server layer, and the database layer. Also, we distinguish three *pieces of software* that correspond to the parts of the application in each layer (see Figure 2).

Granularity Level. Since the conceptual models need the functional requirements to be detailed and validated to generate the final application, the granularity level is low.

Functional Users. The functional users in the final applications are: (1) the human users of the application, and (2) the pieces of software that interchange data between the layers of the application. The functional users are separated by a boundary from the pieces of software of the application.

The functional users can be specified in the conceptual model by the role that the user has been assigned in order to execute the services of the application. In the OO-

Method approach, the different roles of the users are specified in the object model as agents of the services of classes that can execute. These users are functional users of the client piece of software of the application because they send (or receive) data to (from) this piece of software (see Figure 2).

On the other hand, the functional users that correspond to the pieces of software of a three-tier application are the client piece of software and the server piece of software (see Figure 2). The client piece of software is a functional user of the server piece of software because it interchanges data with this piece of the application. The server piece of software is a functional user of the client piece and the database piece of software because it sends (or receives) data to (from) these pieces of software.

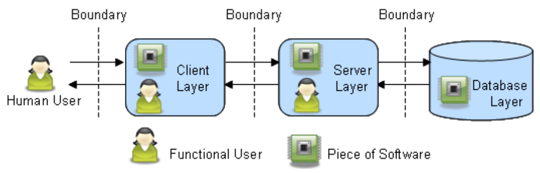

Fig. 2. Functional users and scope of an OO-Method application.

To avoid mistakes in the identification of the functional users and the boundaries of an OO-Method application, Table 1 shows three rules that have been defined to identify the functional users (Rule 1, 2, and 3) and one rule that has been defined to identify the boundaries (Rule 4).

3.2 The Mapping Phase

In this phase, the functional processes must be identified, and after that, the data groups and the data attributes must be identified.

Functional Process. A functional process is a set of functionalities of the application that allows the achievement of a functional requirement. Generally, in the final application, the functional requirements are presented in groups of functionality that can be directly accessed from the graphical user interface (GUI), for instance, in the menu options. Since the MDD conceptual models can specify the presentation of the final application, the groups of functionality (or interaction units) that can be directly accessed in the GUI of the final application are considered to be a functional process (see Rule 5 in Table 1).

It is important to note that all the functionalities that can be accessed or executed from the interaction units make up the functional process and not just the interaction unit that can be accessed directly from the menu of the application. Therefore, once all the elements that make up the interaction unit are identified, the functional process is correctly identified.

The interaction units can be used to (1) show information to the user or (2) to execute services by the user. The interaction units that show information can show data of an object or data of a set of objects. To do this, these interaction units basically use presentation patterns to display information of the objects, to filter the objects, and to access other interaction units. On the other hand, the service interaction unit uses presentation patterns to enter the arguments of the service and to access other interaction units (for instance, to search for an object that corresponds to an argument of the service). To completely identify the elements that make up a functional process, the following rules must iteratively apply:

Rule 5.a: Identify the **display pattern**, the **filter pattern** and the **interaction units** that can be accessed from the functional process as elements of the functional process.

Rule 5.b: Identify the **arguments** and the **interaction units** that can be accessed from the functional process as elements of the functional process.

With the rules described above, the functional processes can be identified several times if they are accessed from more than one access in the menu of the final application. Also, the interaction units that are elements of a functional process can be accessed by several components of the functional process. To avoid duplicity in the identification of the functional processes and the elements that compose it, the following rules have been defined:

Rule 5.c: Drop the interaction units contained in a functional process when these interaction units also correspond to a functional process.

Rule 5.b: Identify the interaction units contained in a functional process only once.

Data Group. The data groups are the conceptual objects of an application. In object oriented applications, the data groups correspond to the classes of the object model. However, if the class participates in an inheritance hierarchy, one data group must be identified for the father of the hierarchy, and one data group must be identified for each child that has attributes different from its father. Rules 6, 7, and 8 of Table 1 have been defined for the correct identification of the data groups.

Attributes. The attributes correspond to the attributes of the classes specified in the object model, which have been identified as data groups (see Rule 9 of Table 1).

Table 1. Mapping Rules.

COSMIC	OO-Method
Functional User	Rule 1: Identify 1 functional user for each **agent** in the OO-Method object model. Rule 2: Identify the client functional user for the **server piece of software** of an OO-Method application. Rule 3: Identify the server functional user for the **client piece of software** of an OO-Method application.
Boundary	Rule 4: Identify 1 boundary **between** a **functional user** and a **piece of software** of an OO-Method application.
Functional Process	Rule 5: Identify 1 functional process for each **interaction unit** that can be **directly** accessed in the menu of the OO-Method presentation model.
Data Group	Rule 6: Identify 1 data group for each **class** defined in the OO-Method object model, which does not participate in an inheritance hierarchy. Rule 7: Identify 1 data group for each **parent class** of an inheritance hierarchy defined in the OO-Method object model. Rule 8: Identify 1 data group for each **child class** of an inheritance hierarchy of the OO-Method object model, which has different attributes from its father.

Attributes	Rule 9: Identify the set attributes of the **classes** defined in the OO-Method object model.

3.3 The Measurement Phase

In this phase, the data movements of each functional process are identified. Then, a measurement function is applied, and the results are aggregated to obtain the functional size of each functional process. Finally, the functional sizes of the functional processes are aggregated to obtain the functional size of the piece of software that has been measured.

Data Movements. Each functional process has two or more data movements. Each data movement moves a single data group. A data movement can be an *Entry* (E), an *Exit* (X), a *Read* (R), or a *Write* (W) data movement.

An *Entry data movement* is a data movement that crosses the boundary from a functional user to a functional process. An *Exit data movement* is a data movement that crosses the boundary from a functional process to a functional user. A *Read data movement* is a data movement that crosses the boundary from the database to a functional process. Finally, a *Write data movement* is a data movement that crosses the boundary from a functional process to the database of the application. The data movements that can occur in an OO-Method application are shown in Figure 3.

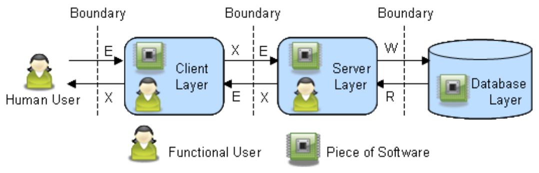

Fig. 3. Data movements that can occur in an OO-Method application.

To identify the data movements that occur in an OO-Method application, 29 rules were defined. These rules are grouped by the conceptual elements of the model. Each rule considers the type of the data movement, the piece of software, and the element of the OO-Method conceptual model.

In the identification of the functional processes, the smallest elements contained in the functional processes were identified as display patterns, filter patterns, and services. With the specification of these patterns in the conceptual model, it is possible to identify all the data movements that can occur in the final application.

The *display patterns* must be identified in the presentation model of the application. The display patterns define the attributes of classes of the object model that will be shown to the users of the application (human functional users). Rules 10, 11, 12, and 13 of Table 2 have been defined to identify the data movements of the display patterns of an application.

The *filter patterns* must also be defined in the presentation model of the application. The filter patterns specify the data that will be shown to the user, if a formula calculated with certain input variables has a particular value. These input variables must be entered by the users of the application (human functional users), and the application retrieves a set of data that satisfies the filter formula calculated with the values of the input variables. The filter patterns are defined in the context of a class of the object model and can retrieve information about this class and the classes related by association with this class. Rules 14, 15, 16, 17, and 18 of Table 2 have been defined to identify the data movements of the display patterns of an application.

The *services* are defined in the classes of the object model. The services have a set of inbound arguments that allows the execution of the service itself. This execution can be changes in the values of the attributes of the class that contains the service, creation of associations between classes, execution of a set of services of a class, execution of a set of services of the model, etc. Therefore, the services defined in the OO-Method object model can be classified as event, transaction/operation, and global services.

The events can be defined to change the value of the attributes of a class. To do this, the events use formulas called valuations (see Rules 19, 20, 21 of Table 2). Also, the events can be defined to create instances of a class or to destroy instances of a class by means of a property of the events. Finally, some events can be defined to create or destroy associations between classes (see Rule 21 of Table 2).

The transactions and operations are defined in a class to group a set of services that must be sequentially executed. These services can belong to the class that contains the transaction or operation. These services can also belong to the classes associated with the class that contains the definition of the transaction or operation. The global services are defined in the OO-Method object model in order to group a set of services of different classes, which may or may not be associated. These kinds of services have been considered in Rule 22 of Table 2.

The arguments of each service are defined in the object model, and a single default value for the arguments of the service can be specified. In addition, a formula can be specified to initialize the values of the arguments of a service. Occasionally, some arguments of a service can depend on the value of other arguments of the service. To represent this situation, dependency rules are used in the OO-Method conceptual model. To count the functionality related to the arguments of a service, Rules 23, 24, 25, 26, 27, 28, 29 and 30 of Table 2 have been defined.

Before the execution of a service, the preconditions of the service must be checked. If the preconditions are satisfied, it is possible to continue with the execution of the service. Otherwise, an error message is shown to the user of the application (human functional user). Rules 31, 32, and 33 of Table 2 have been defined to take into account the functionality of the preconditions of a service.

On the other hand, after the execution of a service, the integrity constrains of the class that contains the service are checked. If the constraints are satisfied, the service ends its execution. If not, the service performs a rollback of the execution and shows a message to the user of the application (human functional user). Rules 34, 35, and 36 of Table 2 have been defined to take into account the functionality of the integrity constraints of the class that contains the service.

Finally, the conditions (guards) that must be fulfilled to execute a service that changes the state of an object (see Rule 37 of Table 2) and the conditions that must be fulfilled to trigger a service (see Rule 38 of Table 2) can be specified in the dynamic model.

Table 2. Rules to identify the data movements of an OO-Method application.

Conceptual Element	Rules
Display Pattern	Rule 10: Identify **1X** data movement for the client piece of software for each **display pattern** in the interaction units that participate in a functional process. Rule 11: Identify **1E** data movement for the client piece of software, and **1X** and **1R** data movements for the server piece of software for each different **class** that contributes with attributes to the display pattern. Rule 12: Identify **1R** data movement for the server piece of software for each different **class** that is used in the **condition** of the **derivation formula** of derivate attributes that appear in the display pattern. Rule 13: Identify **1R** data movement for the server piece of software for each different **class** that is used in the **effect** of the **derivation formula** of derivate attributes that appear in the display pattern.
Filter Pattern	Rule 14: Identify **1E** data movement and **1X** data movement for the client piece of software, and **1E** data movement for the server piece of software for the **set of data-valued variables** of the filter patterns (represented by the class that contains the filter) of the interaction units contained in a functional process. Rule 15: Identify **1E** data movement and **1X** data movement for the client piece of software, and **1E** data movement for the server piece of software for each different **object-valued variable** of the filter patterns of the interaction units contained in a functional process. Rule 16: Identify **1R** data movement for the server piece of software for each different **class** that is used in the **filter formula** of the filter patterns of the interaction units that participate in a functional process. Rule 17: Identify **1E** data movement and **1X** data movement for the client piece of software, and **1X** data movement for the server piece of software for the **set of data-valued variables** with a **default value** of the filter patterns (represented by the class that contains the filter) of the interaction units contained in a functional process. Rule 18: Identify **1E** data movement and **1X** data movement for the client piece of software, and **1X** data movement for the server piece of software for each different **object-valued variable** with **default value** of the filter patterns of the interaction units contained in a functional process.
Service	Rule 19: Identify **1R** data movement for the server piece of software for each different **class** that is used in the **condition** of the **valuation formula** of events that participate in the interaction units contained in a functional process. Rule 20: Identify **1R** data movement for the server piece of software for each different **class** that is used in the **effect** of the **valuation formula** of events that participate in the interaction units contained in a functional process. Rule 21: Identify **1W** data movement for the server piece of software for each **create event**, **destroy event**, or **event that has valuations** (represented by the class that contains the service) that participate in the interaction units contained in a functional process. Rule 22: Identify **1R** data movement for the server piece of software for each different **class** that is used in the **service formula** of transactions, operations, or global services that participate in the interaction units contained in a functional process.

	Rule 23: Identify **1E** data movement and **1X** data movement for the client piece of software, and **1E** data movement for the server piece of software for the **set of data-valued arguments** of the services (represented by the class that contains the service) that participate in the interaction units contained in a functional process. Rule 24: Identify **1E** data movement and **1X** data movement for the client piece of software, and **1E** data movement for the server piece of software for each different **object-valued argument** of the services that participate in the interaction units contained in a functional process. Rule 25: Identify **1E** data movement and **1X** data movement for the client piece of software, and **1X** data movement for the server piece of software for the **set of data-valued arguments** with a **default value** of the services (represented by the class that contains the service) that participate in the interaction units contained in a functional process. Rule 26: Identify **1E** data movement and **1X** data movement for the client piece of software, and **1X** data movement for the server piece of software for each different **object-valued argument** with a **default value** of the services that participate in the interaction units contained in a functional process. Rule 27: Identify **1R** data movement for the server piece of software for each different **class** that is used in the **condition** of the **initialization formula** of the arguments of the services that participate in the interaction units contained in a functional process. Rule 28: Identify **1R** data movement for the server piece of software for each different **class** that is used in the **initialization formula** of the arguments of the services that participate in the interaction units contained in a functional process. Rule 29: Identify **1R** data movement for the server piece of software for each different **class** that is used in the **condition** of the **dependency formula** of the services that participate in the interaction units contained in a functional process. Rule 30: Identify **1R** data movement for the server piece of software for each different **class** that is used in the **dependency formula** of the services that participate in the interaction units contained in a functional process. Rule 31: Identify **1R** data movement for the server piece of software for each different **class** that is used in the **precondition formulas** of the services that participate in the interaction units contained in a functional process. Rule 32: Identify **1X** data movement for the client piece of software for all **error messages** of the **precondition formulas** of the services that participate in the interaction units contained in a functional process. Rule 33: Identify **1E** data movement for the client piece of software, and **1X** data movement and **1R** data movement for the server piece of software for each different **class** used in the **error messages** of the **precondition formulas** of the services that participate in the interaction units contained in a functional process. Rule 34: Identify **1R** data movement for the server piece of software for each different **class** that is used in the **integrity constraint formulas** of the class that contains each service that participates in the interaction units contained in a functional process. Rule 35: Identify **1X** data movement for the client piece of software for all **error messages** of the **integrity constraint formula** of the class that contains each service that participates in the interaction units contained in a functional process. Rule 36: Identify **1E** data movement for the client piece of software, and **1X** data movement and **1R** data movement for the server piece of software for each different **class** used in the **error messages** of the **integrity constraint formula** of the class that contains each service that participates in the interaction units contained in a functional process.

	<u>Rule 37:</u> Identify **1R** data movement for the server piece of software for each different **class** that is used in the **condition formula** of a **transition** that changes the state of an object by means of a service that participates in the interaction units contained in a functional process. <u>Rule 38:</u> Identify **1R** data movement for the server piece of software for each different **class** that is used in the **trigger formula** that triggers a service that participates in the interaction units contained in a functional process.

Measurement Function. The measurement function assigns 1 CFP (Cosmic Function Point) to each data movement that occurs in a functional process of the application.

Measurement Aggregation. Once the measurement function has been applied, the measures can be aggregated to obtain the functional size of each functional process of each piece of software of the application as well as the whole application. Since the functional size of each functional process corresponds to the addition of the data movements that occur in this functional process, the data movements are aggregated to obtain the functional size of each functional process. Using the same criteria it is possible to obtain the functional size of each piece of software. Therefore, the functional size of each piece of software corresponds to the addition of the data movements that occur in the functional processes that are contained in this piece of software. In the case of the functional size of the whole application, the same criteria have been used. Therefore, the functional size of the whole application corresponds to the addition of the data movements that occur in the functional processes that are contained in the pieces of software that are contained in the application. Table 3 presents the rules defined to obtain the functional size.

Table 3. Rules to obtain the functional size of the functional processes, the pieces of software of the application, and the whole application.

Conceptual Level	Rules
Functional Process	<u>Rule 39:</u> Aggregate the CFP related to the **data movements** identified in the **client piece** of software of each **functional process** to obtain the functional size of that process. <u>Rule 40:</u> Aggregate the CFP related to the **data movements** identified in the **server piece** of software of each **functional process** to obtain the functional size of that process.
Piece of Software	<u>Rule 41:</u> Aggregate the CFP related to the **data movements** identified in the **functional processes** identified in the **client piece** of software to obtain the functional size of that piece of software. <u>Rule 42:</u> Aggregate the CFP related to the **data movements** identified in the **functional processes** identified in the **server piece** of software to obtain the functional size of that piece of software.
Application	<u>Rule 43:</u> Aggregate the CFP related to the **data movements** identified in the **functional process** contained in the **pieces of software** identified in the **application** to obtain the functional size of the whole application.

Finally, with the rules of the measurement procedure, it is possible to accurately measure the functional size of the OO-Method software applications that are generated from their conceptual models. The conformity of the rules that are defined in this measurement procedure with the COSMIC measurement method has been validated by experts. In addition, this measurement procedure has been applied to OO-Method case studies, and the results have been compared with the measures obtained by experts. In terms of theoretical validation, since the validation of COSMIC has been carried out successfully from the perspective of measurement theory in [5] using the DISTANCE framework [21], and since the measurement procedure has been designed on the basis of COSMIC, we can infer that the measurement procedure has also been theoretically validated.

4 Conclusions and Further Work

In this paper, we have presented a measurement procedure, which is an FSM procedure for applications that are generated from object-oriented conceptual models of an MDD method. This procedure was designed in accordance with the COSMIC measurement method, which facilitates the functional size measurement of multi-layer applications (in contrast to traditional FSM methods).

The measurement procedure has been designed to obtain accurate measures of applications that have been generated from their conceptual models. It is important to note that it is possible to obtain the accurate functional size because all the functionality of the final application has been specified in the conceptual model, which is automatically transformed to the final application. In other cases (i.e., the conceptual model is not automatically transformed to the final application), it is only possible to obtain estimations of the functional size. Assuming that the conceptual model is of high quality (that is, the conceptual model is correct, complete, and without defects), the measurement procedure could be completely automated, providing measurement results in a few minutes using minimal resources. Obviously, if the conceptual model has incorrect information or missing information, the measures obtained will not be correct by any measurement procedure.

This paper defines a set of mapping rules that allow the selection of the relevant conceptual elements of a specific MDD method called OO-Method to measure the functional size according to the COSMIC concepts. Moreover, a set of measurement rules has been defined to obtain the functional size at three levels: the functional process level, the piece of software level, and the whole application level. The mapping and measurement rules were defined in the context of OO-Method, but many conceptual constructs of the OO-Method conceptual model can be found in other object-oriented methods. For this reason, the measurement procedure could be generalized to other object-oriented MDD methods.

The main limitation of the measurement procedure presented in this paper is the large amount of time required for the manual application of the procedure to models of real applications. We plan to develop a tool that automates the measurement procedure and to conduct empirical studies of the tool to ensure the accuracy of the measures.

References

1. Abran, A., Desharnais, J., Oligny, S., St-Pierre, D., Symons, C.: The COSMIC Functional Size Measurement Method, version 3.0. In: GELOG web site www.gelog.etsmtl.ca (2007)
2. Basili, V., Rombach, H.: The TAME Project: Towards Improvement Oriented Software Environments. IEEE Transactions on Software Engineering, 758--773 (1988)
3. Bévo, V., Lévesque, G., Abran, A.: Application de la méthode FFP à partir d'une spécification selon la notation UML: compte rendu des premiers essais d'application et questions. In: 9th IWSM, Lac Supérieur, Canada, pp. 230--242 (1999)
4. CARE Technologies Web Site, www.care-t.com
5. Condori-Fernández, N.: Un procedimiento de medición de tamaño funcional a partir de especificaciones de requisitos. Doctoral thesis, Univ. Politécnica de Valencia, Spain (2007)
6. Dedene, G., Snoeck, M.: M.E.R.O.DE.: A Model-driven Entity-Relationship Object-oriented Development Method. ACM SIGSOFT Software Engineering Notes 19(3), 51--61 (1994)
7. Diab, H., Frappier, M., St-Denis, R.: Formalizing COSMIC-FFP Using ROOM. In: ACS/IEEE Int. Conf. on Computer Systems and Applications (AICCSA), pp. 312 (2001)
8. Grau, G., Franch, X.: Using the PRiM method to Evaluate Requirements Model with COSMIC-FFP. In: International Conference on Software Process and Product Measurement (IWSM-MENSURA), Mallorca, Spain (2007)
9. ISO, International vocabulary of basic and general terms in metrology (VIM), Geneva, Switzerland (2004)
10. ISO, ISO/IEC 14143-1 – Information Technology – Software Measurement – Functional Size Measurement – Part 1: Definition of Concepts (1998)
11. ISO, ISO/IEC 14143-2 – Information Technology – Software Measurement – Functional Size Measurement – Part 2: Conformity Evaluation of Software Size Measurement Methods to ISO/IEC 14143-1:1998 (2002)
12. ISO/IEC 19761, Software Engineering – CFF – A Functional Size Measurement Method (2003)
13. ISO/IEC 20926, Software Engineering – IFPUG 4.1 Unadjusted Functional Size Measurement Method – Counting Practices Manual (2003).
14. ISO/IEC 20968, Software Engineering – Mk II Function Point Analysis – Counting Practices Manual (2002)
15. ISO/IEC 24570, Software Engineering – NESMA Functional Size Measurement Method version 2.1 – Definitions and Counting Guidelines for the application of Function Point Analysis (2005)
16. Jenner, M.S.: COSMIC-FFP and UML: Estimation of the Size of a System Specified in UML – Problems of Granularity. In: 4th European Conf. Soft. Measurement and ICT Control, pp. 173--184 (2001)
17. Meli, R., Abran, A., Ho Vinh, T., Oligny, S.: On the Applicability of COSMIC-FFP for Measuring Software Throughout its Life Cycle. In: 11th European Software Control and Metrics Conference, Munich (2000)
18. Pastor, O., Gómez, J., Insfrán E., Pelechano, V.: The OO-Method Approach for Information Systems Modelling: From Object-Oriented Conceptual Modeling to Automated Programming. Information Systems 26(7), 507--534 (2001)
19. Pastor, O., Molina, J. C.: Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling. Springer (2007)
20. Poels, G.: A Functional Size Measurement Method for Event-Based Object-oriented Enterprise Models. In: Int. Conf. on Enterprise Inf. Systems (ICEIS), pp. 667--675 (2002)
21. Poels, G., Dedene, G.: Distance-based software measurement: necessary and sufficient properties for software measures. Inf. and Software Technology 42(1), 35--46 (2000)
22. Selic, B.: The Pragmatics of Model-Driven Development. IEEE Sof. 20(5), 19--25 (2003)
23. Selic, B., Gullekson, G., Ward, P.T.: Real-time Object Oriented Modelling. Wiley (1994)