A Metamodeling Approach to Estimate Software Size from Requirements Specifications[1]

Silvia Abrahão and Emilio Insfran

[1]*Department of Information Systems and Computation*
Valencia University of Technology
Camino de Vera, s/n, 46022, Valencia, Spain
{sabrahao, einsfran}@dsic.upv.es

Abstract

Early software size estimation is essential for good project management. Although several proposals to estimate software size from requirements specifications exist, most of them have not been properly defined or automated. This paper presents the design and automation of a measurement procedure (ReqPoints) to estimate the size of software projects from a requirements specification. The procedure is based on a requirements engineering approach that provides a MDA framework for requirements specification and model transformations to obtain the architecture of UML models. Specifically, a set of measurement rules is defined as a mapping between the concepts of the Requirements Metamodel onto the concepts of the Function Point Analysis (FPA) Metamodel. A Requirements EStimation Tool (REST) was built to automate the measurement process. We demonstrate the feasibility of applying the estimation tool to a case study.

Keywords: *Model-driven Software Development, Requirements Engineering, Functional Size Measurement.*

1. Introduction

Estimates of cost and effort for software projects are based on a prediction of the size of the future system. Therefore, the capability to accurately quantify the size of software systems at early stages of the development lifecycle is a critical issue.

Functional Size Measurement (FSM) is supposed to be a suitable approach for early size measurement. It assesses the logical external view of the software from the users' perspective by measuring the amount of functionality to be delivered. However, current FSM methods (i.e., FPA) depend on the human interpretation, which leads to large variability in the measurement results. Some proposals for sizing object-oriented systems from requirements specifications have been defined in the last few years. The main limitation of these approaches is that they have not been properly defined in accordance with a standard FSM method (e.g., [13]). In addition, the requirements artifacts used as input for measurement do not have a well-defined traceability to other artifacts built in the upcoming phases of the software development lifecycle [5], [13]. This can probably affect the usefulness of the size measure obtained with these artifacts. Furthermore, most approaches are not automated [2], [4], [5], [6], [7], [8], [13], [16] which limits their use and adoption.

To address these limitations, this paper introduces a *measurement procedure* for object-oriented systems called Requirements Points (ReqPoints). A measurement procedure is a *"set of operations, described specifically, used in the performance of particular measurements according to a given method of measurement"* [11]. The aim of our procedure is to size a requirements specification that is developed using a Requirements Model [9] that follows an MDA-based approach. The construction of the requirements specification is supported by a Requirements Engineering TOol (RETO).

We present the design and application of ReqPoints using a process model for software measurement [12] (see Figure 1). This process model was successfully applied in a previous work [1]. According to this process, a measurement method is designed, that is, the concept to be measured is defined and the measurement rules are devised. Then, the measurement method is applied. The results of the method are then presented and verified. This verification includes determining whether the value that is produced is the result of a correct application and interpretation of the measurement rules. Finally, the results are used to build different types of models (e.g., productivity analysis models, effort estimation models).

[1] This work is supported by the META (Models, Environments, Transformations and Applications) project, with reference TIN2006-15175-C05-01.

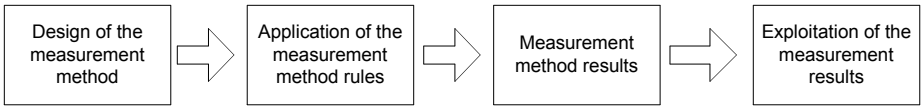

Figure. 1. Measurement process steps (Source: [12])

We detail how the first three steps of the process model are conducted to design and automate ReqPoints. The verification of the measurement results (e.g., evaluation of the reproducibility and accuracy of the results) is out of the scope of this work.

This paper is organized as follows. Section 2 presents an overview of related work. Section 3 presents the Requirements Model and the RETO tool. Section 4 presents the design of ReqPoints followed by its application in Section 5. Section 6 shows the automation of the measurement procedure in the Requirements EStimation Tool (REST). Finally, section 7 presents the conclusions and further work.

2. Related Work

The first proposal of size estimation for requirements specification (Use Case Points – UCP) was published in 1993 by Karner et al. [14]. Since then, another eight studies have proposed other functional size measurement procedures ([2], [4], [5], [6], [7], [8], [13], [16]). UCP was defined by practitioners, and the remaining proposals were defined by researchers. With the exception of UCP [14], all the other procedures have not been automated. Even though several tools to automate UCP have been developed, they provide only partial automation to extract the actors and use cases. Their complexity classification has to be done manually. In addition, the majority of the proposals have not been validated, either empirically or theoretically. A summary of each of these studies is presented in Table 1.

Table 1. Measurement procedures to estimate size from Requirements Specifications

Proposal	FSM method	Input Artifacts	Auto-mated	Vali-dated
Use Case Points [14]	IFPUG FPA-like	Use Case Model	Partial	Partial
Fetcke et al. [7]	IFPUG FPA v. 4.0	Use Cases, Class diagram	No	No
Bévo et al. [5]	COSMIC-FFP v. 2.0	Use Cases, Class diagrams	No	Partial
Jenner [13]	COSMIC-FFP v. 2.0	Use Cases, Sequence Diagrams	No	No
Tavares et al. [16]	IFPUG FPA v. 4.1	Use Cases, Class diagrams	No	No
Azzouz [2]	COSMIC-FFP v. 2.2	RUP	No	No
Bertolami et al. [4]	Mark II FPA	LEL	No	No
Condori-Fernandez et al. [6]	COSMIC-FFP v. 2.2	Use Cases, Sequence Diagrams	No	Yes
Habela et al. [8]	COSMIC-FFP v. 2.2	Use Cases, Sequence Diagrams	No	No

The main limitation of these approaches is that they are not properly defined in accordance with a standard FSM method. Although they are based on a FSM standard method, their measurement rules are not fully compliant to these standards. Only [6] conducted an evaluation conformance of the proposed measurement procedure with respect to a FSM standard.

In addition, the requirements artifacts used as input for measurement do not have a well-defined traceability to other artifacts in the upcoming phases of the software development lifecycle. For instance, the Jenner proposal [13] discusses the problem of granularity at the Use Cases level. However, it is not enough to simply indicate that Sequence Diagrams have the most appropriate level of granularity to measure the functional size. A method of how to construct these diagrams is also needed. The same happens with the Bévo et al. proposal [5].

Another drawback of these procedures is the lack of a clear definition of the different types of Base Functional Component (BFC) types. These are the important elements for measurement.

3. The Requirements Model

The Requirements Model [9] provides primitives for the specification of requirements following a Model-Driven Architecture (MDA) approach. The requirements of the software system are specified using the standard UML notation. The approach is completed with a Transformation Rules Catalog (TRC), which defines transformation rules to obtain the architecture of UML models, establishing clear traceability links. The Requirements Model is mainly composed by a mission statement, a function refinement tree, and a use case model.

The **Mission Statement** is a high-level description of the nature and purpose of the system (main goal). Considering the future software system as a black box, its visible (external) interactions are identified as functions. These functions are hierarchically structured and organized in a **Functions Refinement Tree (FRT),** where the root is the *mission statement*, the internal nodes are *functional groups*, and the leaves are the *elementary functions* of the system that correspond to the concept of *use case*. The **Use Case Model** includes all the identified functions from the FRT, which are the use cases, including their corresponding communication relationships with actors and the structural relationships among actors (inheritance).

Finally, a requirements specification is completed with a **Requirements Domain Model**, which defines the vocabulary of the problem domain including the relevant entities and their structural relationships, and an **Interaction Model**, which captures the main object

interactions to realize each identified use case using the UML Sequence Diagram notation.

This Requirements Model is defined by a MOF metamodel that is introduced in section 3.1 This approach is supported by a Requirements Engineering Tool (RETO), which can be downloaded at http://reto.dsic.upv.es.

3.1 The MOF Requirements Metamodel

Metamodeling is a key concept of the MDA paradigm and is used in Software Engineering (SE) to describe the basic abstractions that define the models and their relationships. The Meta Object Facility (MOF) [15] provides a multi-layer architecture for defining metamodels. Figure 2 shows an excerpt of the relevant parts of the MOF Requirements Metamodel.

The functions of the system are represented by the *Use Case* class. Each Use Case is specified by means of one or more *Sequence Diagrams* that are mainly composed of *Entities* and *Messages*. We distinguish three types of Entities when describing a Sequence Diagram: *Actor, Interface,* and *Class*. *Actor* represents the users of the Use Case, *Interface* represents the boundary among the actors and the internal classes of the system, and *Class* represents the different entity classes that participate in the realization of the Use Case.

Finally, to characterize the different nature of interaction between objects, we identify four types of messages:

- **Signal Messages:** they are labeled with the stereotype «signal» and represent interactions between an *Actor* and the system. The only property for this message type is the *direction*, which can have two types of values: input and output.
- **Service Messages:** they are labeled with the stereotype «service» and represent interactions where the objects of the receiver class changes their state. The changes can be of three types:
  - **New:** creation of a new object instance of the target class.
  - **Update**: change of state of an object instance of the target class.
  - **Destroy:** destruction of an existing object of the target class.
- **Query Messages:** they are labeled with the stereotype «query» and represent queries on related objects or on a class population.
- **Connect Messages:** they are labeled with the stereotype «connect» and are used to establish a structural relation between the participant objects in the interaction.

Moreover, these messages can also be labeled with a condition that, if satisfied, allows the interaction to occur. The syntax for this type of condition is: *[boolean-expression] message-name*.

An INCLUDE relationship allows a base Use Case to perform an explicit inclusion of an included Use Case. This relationship is represented in a base Sequence Diagram by a message call with the syntax: *[condition] INCLUDE IncludedUseCaseName*. The EXTEND relationship is only known by the Use Case that extends. For this reason, the base Sequence Diagram does not make any reference to the Use Case that extends.

4. The Design of the Procedure

The design of ReqPoints is done by following the activities suggested in the first step of the process model shown in Figure 1. Four activities are suggested for a complete design of a measurement method [12]: definition of the objectives, characterization of the concept to be measured, definition or selection of the metamodel, and definition of the numerical assignment rules.

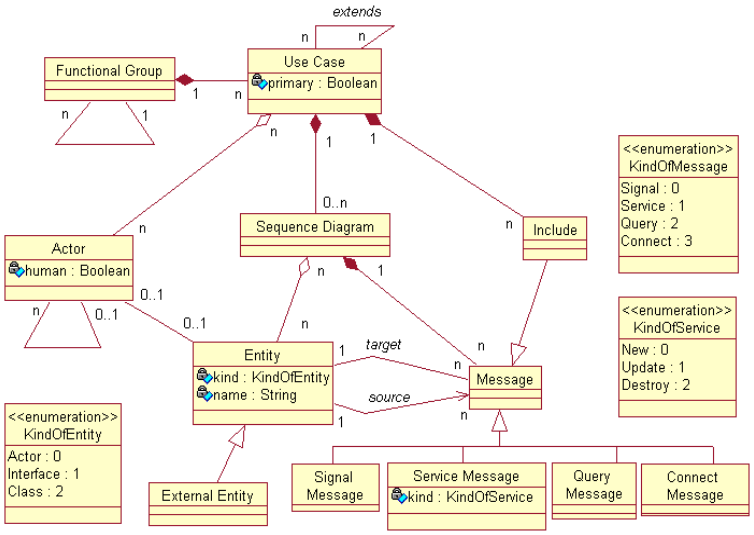

Figure 2. MOF Requirements Metamodel

4.1 Definition of the objectives

According to the GQM template [3], our goal is to **design** a FSM procedure **for the purpose of** sizing a requirements specification **with respect to its** functional size **from the point of view** of the researcher. The **context** is that this procedure should conform to the IFPUG FPA method version 4.1 [10], referred to here as FPA.

4.2 Characterization of the concept

The **entity** to be measured consists of a Requirements Model specification obtained with the RETO tool. It is mainly composed of a *Use Case Model* and a set of *Sequence Diagrams* that capture the system functionality and the object interactions necessary to realize each use case, respectively. The **attribute** to be measured is *functional size*, which is the size of the software derived by quantifying the functional user requirements.

4.3 Selection of the metamodel

The metamodel of a FSM method provides a precise basis to design the measurement rules that identify and measure the relevant concepts that contribute to the size of a system. As our measurement procedure is intended to conform to the IFPUG method [10], it assumes the same metamodel as FPA. Figure 3 shows the FPA metamodel for the IFPUG method. It illustrates the information that must be captured in order to size a software *project*. These concepts will be used in the definition of the mapping rules.

4.3.1 Definition of the mapping rules. The mapping rules help identify the elements in a Requirements Model that contribute to the functional size of the system. These rules are defined as a mapping between the concepts of the FPA Metamodel onto the concepts in the Requirements Metamodel [9].

First, Rules 1 to 4 are applied to establish the counting scope and the boundary of the system. Then, the data (ILF and EIF) and transactional (EI, EO, and EQ) functions are identified by applying Rules 5 to 11.

The *counting scope* defines what is going to be sized. In the Requirements Model, it *corresponds to the Use Case Model, which includes all the use cases* (**Rule 1**). However, other scopes might also be established considering any *functional group* of the Functional Refinement Tree.

The *boundary* indicates the border between the project or application being measured and the external applications or user domain. A Use Case Model is the visual representation of the actors who interact with the use cases that define the system's functionality. *An Actor can be a human actor such as a user of the application* (**Rule 2**) or a *non-human actor such as an external application* (**Rule 3**). *The boundary corresponds to an imaginary line traced in the Use Case Model. The actors are considered to be outside the boundary, whereas the Use Cases are considered to be inside the* boundary since they define the system's functionality **(Rule 4).**

Once the boundary has been established, the data and transaction functions can be identified. The data functions (ILF and EIF) are identified using the Interaction Model (Sequence Diagrams) as input.

An ILF is a user identifiable group of logically related data or control information maintained within the boundary of the system [10]. In the Requirements Model, *an ILF corresponds to an entity of type Class because it represents a collection of objects described structurally by a set of attributes* (**Rule 5**). An EIF is a user identifiable group of logically related data or control information referenced by the system, but maintained within the boundary of another system [10]. In the Requirements Model, *an EIF corresponds to an external entity of type Class because it represents objects or components that are outside of the system but that are referred to it* (**Rule 6**).

A transactional function (EI, EO, and EQ) is identified considering the different types of messages defined in the Sequence Diagrams (signal, query, service and connect).

An EI is an elementary process that manipulates data or control information that comes from outside the system boundary. Its goal is to maintain one or more ILFs and/or to alter the behavior of the system [10]. In the Requirements Model, *an EI corresponds to a «signal» message with an input value because it represents an interaction between the actor and the system* (**Rule 7**).

In addition, *the «service/new», «service/destroy», and «service/update» messages are also considered as EIs since they maintain the information of the target objects* (**Rule 8**). Finally, *a «connect» message is also an EI because it represents the establishment of a relationship between the source and target objects* (**Rule 9**).

An EQ is an elementary process that sends data or control information outside the system boundary. Its goal is to present information to a user through the retrieval of data [10]. In the Requirements Model, *an EQ corresponds to a «query» message with a «signal» message to show the result of the query to an actor* (**Rule 10**).

Finally, an EO has the same definition as an EQ, but its processing logic must contain at least one mathematical formula or calculation, or it has to create derived data [10]. In the Requirements Model, *an EO also corresponds to a «query» message together with a «signal» message* (**Rule 11**). *However, this rule is only applied if there are other messages indicating an internal process with the result of the query before showing the result to the actor.*

4.4 Definition of numerical assignment rules

The purpose of this phase is to produce a quantitative value that represents the functional size of the system. This is accomplished by applying two sets of rules that are introduced below.

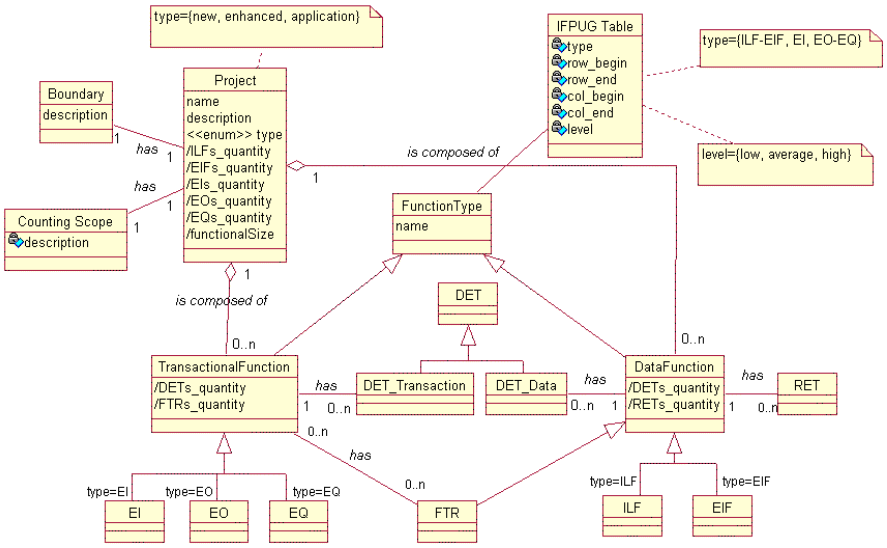

Figure 3. The FPA metamodel for the IFPUG method

4.4.1 Definition of the measurement rules. These rules are defined to count the number of DETs, RETs and FTRs of each data function and transactional function previously identified. Once these concepts have been counted, the FPA counting rules are applied to classify the function complexity as low, average, or high.

4.4.1.1 Establishing the complexity of an entity. According to the FPA metamodel, the complexity of an entity (ILF) or an external entity (EIF) is determined by counting the number of Data Element Types (DET) and Record Element Types (RET). A DET is a *unique user recognizable, non-repeated field*. A RET is a *user recognizable subgroup of data elements within an ILF or EIF*. We identify a RET for the entity itself as it represents a group of logically related data. Thus, the following rules are proposed:

- **Rule 12***: Count a RET for each entity of type class*
- **Rule 13***: Count a RET for each external entity*

In the Requirements Model, only the entities and possible relationships among them are identified. Therefore, at this level is not possible to know the number of DETs of each entity of type class. As a consequence, a *low complexity* is directly assigned to each entity of type class (ILF) and external entity (EIF). According to the complexity weights of the data functions provided by the IFPUG standard [10] (see Table 2), this corresponds to a function with one RET and up to fifty DETs.

Table 2. Complexity weights for ILF and EIF

RETs	1-19 DETs	20-50 DETs	51+ DETs
1	Low	Low	Average
2-5	Low	Average	High
6+	Average	High	High

4.4.1.2 Establishing the complexity of a message. The complexity of the different types of messages (EI, EQ, and EO) is determined by counting the number of Data Element Types (DET) and File Types Referenced (FTR). A FTR *is an entity of type class that is read or maintained by a transactional function or an external entity that is read by a transactional function*.

A DET could be identified for each single parameter of a message. However, because we are dealing with early object interactions, the number of DETs obtained by counting the number of message parameters is not meaningful. Usually, at this level, only the most relevant data is indicated. The precise amount of data interchanged among objects will be completed in the subsequent stages of the development lifecycle. Consequently, a fixed range of DETs are assigned to each transactional function taking into account the complexity weights described in Table 3 and Table 4.

The complexity of a signal, service and connect message (EIs) is established by directly assigning *5-15DETs* for each one of these functions. We use the middle column of Table 3 as a reference value. Two of these DETs are counted in order to be compliant to the IFPUG method: one DET for the capability of the application to send a message outside the boundary (error, confirmation, control), and another DET for the ability to specify an action to be taken. The remaining DETs are an estimation of the number of DETs that can be manipulated when a message is executed. In terms of FTRs, an FTR is counted for the entity class where the message is defined. Therefore, the following measurement rule is defined:

- **Rule 14***: Count a FTR for the entity class in which the message is defined (the target of the message).*

Additionally, other FTRs are counted for the *conditions* of a message, the *precondition* of the Use Case, and the *invariants* associated to an entity of type class.

The condition of a service, query or connect message may reference data from an entity class that is different from a target class. This implies recovering the value of the attributes involved in the condition in order to evaluate it. If this occurs, an additional FTR must be counted for each different entity class involved in the condition. Therefore, the following measurement rule is defined:

- **Rule 15***: Count a FTR for each single entity of type class referenced in a condition of a message.*

In addition, in the Use Case template specification, it is possible to define a precondition for the execution of the Use Case. A precondition can also increase the complexity of a signal, service and connect message as defined in the following rule:

- **Rule 16***: Count a FTR for each single entity of type class referenced in the precondition of the use case.*

The specification of an invariant is not specifically associated to a Sequence Diagram but rather to the entire system since it is defined as a property of an entity of type class. The following rule is considered as a complementary rule to the identification of FTRs since the invariant should be evaluated after any modification of the corresponding entity of type class. This rule is defined as follows:

- **Rule 17***: Count a FTR for each single entity of type class referenced in an invariant associated to an entity.*

Finally, the complexity of a signal, service and connect message is established by using the complexity weights for EIs (see Table 3) provided in the IFPUG standard [10]. Note that the complexity will depend on the amount of FTRs: a message with one FTR will have a *low complexity*, a message with two FTRs will have an *average complexity*, and a message with more than three FTRs will have a *high complexity*.

Table 3. Complexity weights for EI

FTRs	1-4 DETs	5-15 DETs	16+ DETs
0-1	Low	Low	Average
2	Low	Average	High
3+	Average	High	High

Similarly, the complexity of a query message (EQ or EO) is established by assigning *6-19DETs* for each one of these functions. Again, we use the middle column of Table 4 as a reference value. Rules 14 to 17 are used to count the number of FTRs for messages of this type.

Finally, the complexity of a query message is established by using the complexity weights for EQ and EO (see Table 4) provided in the IFPUG standard [10]. The complexity of a query message will also depend on the amount of FTRs of the function.

Table 4. Complexity weights for EQ and EO

FTRs	1-5 DETs	6-19 DETs	20+ DETs
0-1	Low	Low	Average
2-3	Low	Average	High
4+	Average	High	High

4.4.2 Calculating the functional size of the system. This step consists of assigning a value of Function Points (FP) to the classified functions and aggregating the assigned values into an overall functional size value for the software system. A FP value is assigned to each function depending on its type and complexity level. Table 5 shows the number of FPs per function type and complexity provided in the IFPUG counting manual [10]. For instance, an ILF with low complexity has 7 FPs.

Table 5. Number of FPs per function type and complexity

Function type	Low	Average	High
ILF	7	10	15
EIF	5	7	10
EI	3	4	6
EQ	3	4	6
EO	4	5	7

The sum of the FP values of the different types of messages (EI, EQ, and EO) is the size of a Sequence Diagram as indicated by the following equation:

$$Size(SequenceDiagram) = \sum_{i=1}^{n} Size(EI_i) + \sum_{j=1}^{n} Size(EQ_j) + \sum_{m=1}^{n} Size(EO_m) \quad (1)$$

The size of a Use Case is then determined by the size of its Sequence Diagram, as indicated as follows:

$$Size(UseCase_k) = Size(SequenceDiagram_k) \quad (2)$$

However, if a Use Case is related to other use cases by «include» or «extend» relationships, two additional rules are defined:

- **Rule 18***: The size of a use case extended by other use cases is equal to the sum of the size of the base use case plus the size of each use case that extends it.*
- **Rule 19***: The functional size of a use case that includes other use cases is equal to the sum of the size of the base use case plus the size of each included use case.*

These rules are expressed by the following equation:

$$Size(UseCase_r) = Size(SequenceDiagram_r + \sum_{i=1}^{n} Size(ExtensionUseCase_i) + \sum_{j=1}^{n} Size(IncludedUseCase_j) \quad (3)$$

The sum of the size of all Use Cases is the size of the transactional functions of the system, as indicated by the following equation:

$$Size(TransactionalFunction) = \sum_{i=1}^{n} Size(UseCase_i) \quad (4)$$

The sum of the size of all entities of type class and external entities is the size related to the data functions of the system, as indicated by the following equation:

$$Size(DataFunction) = \sum_{i=1}^{n} Size(ILF_i) + \sum_{j=1}^{n} Size(EIF_j) \quad (5)$$

Finally, the sizes of the transactional and data functions are summed to produce the functional size of the system in unadjusted function points, as indicated below:

$$Size(System) = Size(TransactionalFunction) + Size(DataFunction) \quad (6)$$

5. Application of the Procedure

This section illustrates the use of ReqPoints in a case study. This is done by following the activities suggested in the second step of the process model shown in Figure 1. These activities are described below.

5.1 Software documentation gathering

The documentation used to apply ReqPoints include: a Requirements Model of a Car Rental System obtained with the RETO tool and a set of guidelines explaining how to apply the measurement procedure.

The **mission** of the Car Rental system is *"to automate the management of cars and car rentals of the company. It also includes the car maintenance and repair, additional accessories to be rented (extras), and customer management"*. The **Functions Refinement Tree** is built based on the definition of the system's mission. Due to space limitations, we only consider the first-level functional groups: car management, customer management, user management, and contract management.

5.2 Construction of the software model

This step consists in identifying the elements in the Requirements Model specification that contribute to the functional size of the system. The result is a collection of data and transactional functions that can be quantified in the next step. The software model for the Car Rental system is built by applying the mapping rules described in Section 4.3.1.

5.2.1 Defining the counting scope and boundary. According to Rule 1, the *counting scope* includes all the Use Cases and their corresponding Sequence Diagrams that comprise the four first-level functional groups (car management, customer management, user management, and contract management).

The *boundary* is established by identifying the users and the external systems that interact with them. This is done by applying Rules 2 to 4. The boundary for the Car Rental System is shown in Figure 4. It shows the main Use Case diagram as packages and the two users of the system (User and Administrator).

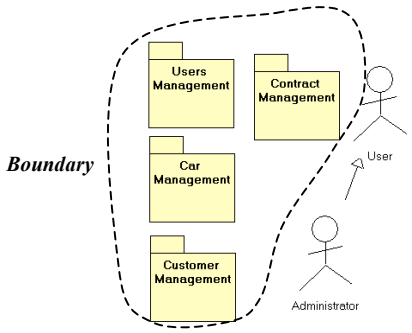

Figure 4. The boundary for the Car Rental System

5.2.2 Identifying the data functions. The entities of a Sequence Diagram can be *entities of type class* (ILF) or *external entities* (EIF). By applying Rules 5 and 6, we identify the following data functions:

- ILFs: *Access Manager, Administrator, Car, Contract, Customer, Direct Customer, Disabled Car, Extra Contract, Extra Type, Garage, Insurance, Insurance Company, User.*
- EIF: *Agency Customer*.

Note that *User* and *Administrator* are also considered as entities of type class because some information related to these actors is maintained inside of the system boundary.

5.2.3 Identifying the transactional functions. We explain the application of the appropriate mapping rules defined for the identification of transactional functions using an excerpt from the *Create Insurance* Sequence Diagram shown in Figure 5. In this scenario, after introducing the initial data, the existence of the car is verified. After that, the rest of the data is entered into the system. A new insurance policy is created and the corresponding insurance company and the car are related to the created insurance policy. Finally, the record of insurance policies of the insurance company is updated.

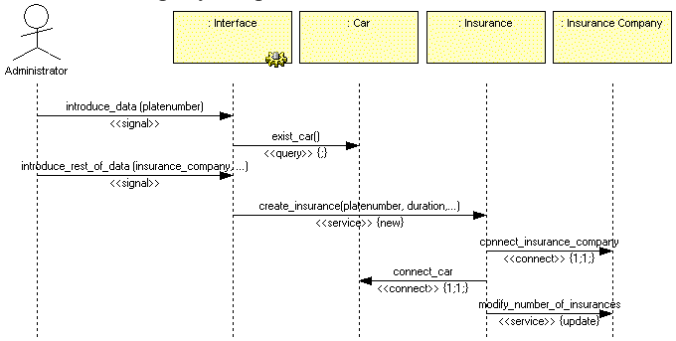

Figure 5. Create Insurance Sequence Diagram

By applying Rule 8, two External Inputs (EIs) are identified: one for the *create_insurance* message and another for the *modify_number_of_insurances* message. Similarly, by applying Rule 9, two additional EIs are identified: *connect_insurance_company* and *connect_car* messages.

5.3 Application of numerical assignment rules

Finally, to obtain the functional size value for the Car Rental system, we first apply the measurement rules described in Section 4.4.1 and then the aggregation rules described in Section 4.4.2.

5.3.1 Establishing the complexity of entities. By applying Rule 12, we identify a RET for the entity of type class itself since it represents a group of logically related data. As it is not possible to know the number of DETs of each entity at this development phase (Requirements), a low complexity is assigned to each one of the thirteen identified entities of type class (ILFs). Similarly, by applying Rule 13, a *low complexity* rate is assigned to the *Agency Customer* entity.

5.3.2 Establishing the complexity of messages. The complexity of a signal, service and connect message (EI) is established by assigning *5-15DETs* for each one of these functions. The number of FTRs is identified by applying Rules 14 to 17. They assist the project manager to identify the number of entities that participate in the message execution.

For the Car Rental system, we take all the Use Cases and their Sequence Diagrams. In total, we have identified 25 signal, service and connect messages that manipulate only one entity, 5 messages that manipulate two entities, and 5 messages that manipulate three or more entities. For instance, the messages *create_insurance*, *modify_number_of_insurances,* *connect_car,* and *connect_insurance_company* (see Figure 5) have low complexity since they manipulate only one entity class each. Consequently, there are 25 messages with *low complexity*, 5 messages with *average complexity*, and 5 messages with *high complexity*.

We have identified 14 query messages with signal messages and an output value (EQ). Of these, 8 manipulate only one entity and 6 messages manipulate two or three entities. Consequently, there are 8 messages with *low complexity* and 6 messages with *average complexity*. We also identified 3 query messages together with an internal process that show the result of the query (EO). One of them manipulates one entity and the other two manipulate three entities. Consequently, there is 1 message with *low complexity* and 2 messages with *average complexity*.

5.3.3 Calculating the functional size of the system. In this step, we assign a Function Point (FP) value for each one of the functions identified in the previous step and then aggregate the values to obtain the functional size of the Car Rental system.

By applying the complexity weights from Table 6 and equation 1, the size of the *Create Insurance* (see Figure 5) is as follows:

Size (Create Insurance) = 4 x 3 + 0 + 0 = 12 FP

Similarly, the other Sequence Diagrams of the Car Rental system are measured. As each Use Case is specified as a Sequence Diagram, the size of the Sequence Diagram corresponds to the size of the Use Case (see equation 2). The next step is to sum the values of each Use Case to obtain the size of the transactional functions of the system. This is done by applying equation 4:

Size (TransactionalFunction) = 187 FP

Then, equation 5 is applied to obtain the size of the data functions of the system:

Size (DataFunction) = 13 x 7 + 1 x 5 = 96 FP

Finally, equation 6 is applied to obtain the total size of the system in unadjusted function points.

Size (System) = 187 + 96 = 283 FP

6. Automating the Procedure

The automation of the measurement procedure is done by the REST tool. This tool is able to import a requirements specification that is built using the RETO tool. This step corresponds to the automation of the *software documentation gathering* activity. When the requirements specification is imported, the REST tool stores "on the fly" all the relevant information that is needed to get the functional size of the system. This step consists in identifying the elements in the requirements model that contribute to functional size. This step corresponds to the automation of the *construction of the software model* activity.

Finally, the functional size of the system is obtained through the application of the measurement rules (see section 4.4.1). The REST tool performs the queries to the stored requirements to get the values to calculate the functional size. This corresponds to the *application of the numerical assignment rules* activity. Although the weights suggested in the FPA standard are used, the tool has an advanced configuration module that allows the project manager to change the complexity weights. A running example of a requirements model being sized with ReqPoints is shown in Figure 6. Specifically, it shows the size estimation report for the Car Rental System. This tool provides a real-time detailed report of both the number of function points per function type and the number of function points for the overall system.

We compared the size estimate obtained by the REST tool with the estimate that was manually obtained by a Certified Function Point Specialist (CFPS). The CFPS sized the IEEE 830 specification of the Car Rental system (automatically obtained with the RETO tool). A small difference of only 6 FPs was observed due to the establishment of function complexity. The results were used to adjust the measurement rules and improve the REST tool.

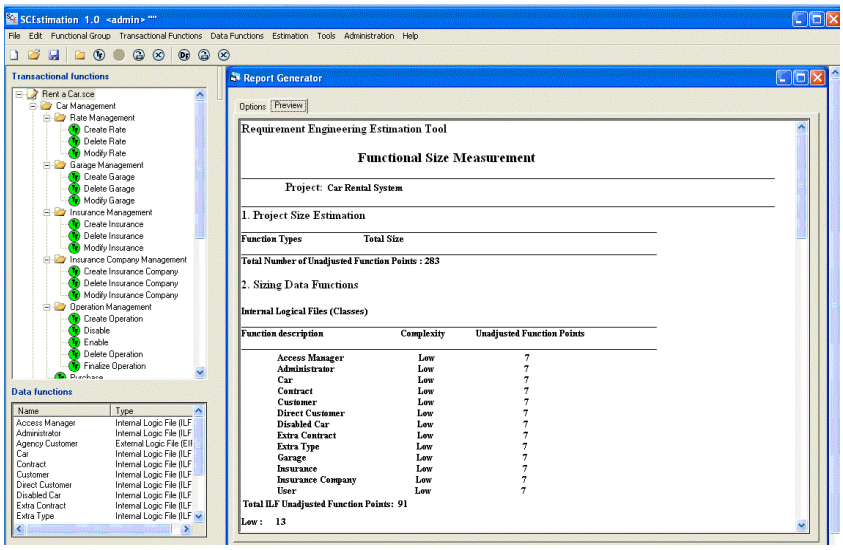

Figure 6. The REST tool

7. Conclusions

This paper has introduced ReqPoints as a measurement procedure for estimating the functional size of object-oriented systems from requirements specifications. The procedure is compliant to the IFPUG method, which is a widely used FSM method in industry. Since the procedure was designed as a mapping between the FPA metamodel and the requirements metamodel, a conformity evaluation was made during the design stage to assure that all the concepts in the standard were properly dealt with.

In addition, ReqPoints was automated in the REST tool. Thus, a size measure of a system can be easily estimated in an early stage of the development lifecycle when the requirements model is specified. This avoids the ambiguity of interpreting the FPA counting rules and the need for special training to count function points in an accurate and repeatable way. Since ReqPoints has been defined for a Requirements Model that follows a MDA approach, the measurement performed at this early stage of the software development process can be considered as representative of the size of the corresponding conceptual model, and consequently, of the final application. However, further studies have to be done in order to determine the accuracy of a size estimate obtained in the requirements level with respect to the size of the final application. Nevertheless, the more widely used MDA transformation processes become in the software development community, the greater the need for automated measurement procedures to be defined at higher levels of abstraction.

8. References

[1] Abrahão S., Poels P., Pastor O. A Functional Size Measurement Method for Object-Oriented Conceptual Schemas: Design and Evaluation Issues, Software & System Modeling 5(1): 48-71, 2006, Springer.

[2] Azzouz S., Abran A. A Proposed Measurement Role in the RUP and its Implementation with ISO 19761: COSMIC-FFP. Software Measurement European Forum, Italy, 2004.

[3] Basili, V., Rombach, H. The TAME project: towards improvement-oriented software environments. IEEE Transactions on Software Engineering 14(6): 728-738, 1988.

[4] Bertolami M., Oliveros A. Functionality Measurement Process on the Requirements Elicitation, Ibero-American Workshop on Requirements Engineering and Software Environments, 2004.

[5] Bévo V., Lévesque G., and Abran A. UML Notation for Functional Size Measurement Method. 9th International Workshop on Software Measurement, Canada, 1999, pp. 230-242.

[6] Condori N., Abrahão S., Pastor O. Towards a Functional Size Measure for OO Systems from Requirements Specifications, 4th Conference on Quality Software (QSIC 2004), 2004, pp. 94 - 101.

[7] Fetcke T., Abran A., and Nguyen T. Mapping the OO Jacobson approach to function point analysis. In Proc. of the TOOLS Conference, Santa Barbara, USA, 1997, pp. 1-12.

[8] Habela P., Glowacki E., Serafinski T. Adapting Use Case Model for COSMIC-FFP based Measurement. 15th International Workshop on Software Measurement, Canada, 2005.

[9] Insfran E., Pastor O., Wieringa R.: Requirements Engineering-Based Conceptual Modelling. Journal of Requirements Engineering 7(2): 61–72, 2002, Springer.

[10] ISO/IEC 20926: Software engineering- IFPUG 4.1 Unadjusted size measurement method - Counting manual, 2003.

[11] ISO, International Vocabulary of Basic and General Terms in Metrology, Second edition, 1993.

[12] Jacquet J. P., Abran, A. From Software Metrics to Software Measurement Methods: A Process Model, 3rd International Standard Symposium and Forum on Software Engineering Standards (ISESS'97), Walnut Creek, USA, 1997.

[13] Jenner M.S. COSMIC-FFP and UML: Estimation of the Size of a System Specified in UML-Problems of Granularity. European Conf. Soft. Measurement and ICT Control, 2001, pp. 173-184.

[14] Karner, G. Metrics for Objectory. Diploma thesis, University of Linköping, Sweden, December 1993.

[15] OMG, Meta Object Facility (MOF) 2.0 Core Spec., 2004.

[16] Tavares H., Carvalho A., Castro J. Function Points Measurement from Requirement Specification. 5th Workshop on Requirements Engineering, Spain, 2002, pp. 278-298.