Design of a Functional Size Measurement Procedure for Real-Time Embedded Software Requirements Expressed using the Simulink Model

Hassan Soubra
DEL'TA, GÉLOG, LISV
Renault SAS, ETS, UVSQ
Guyancourt, France
hassan.soubra.1@ens.etsmtl.ca

Alain Abran
GÉLOG-Laboratoire de Recherche en Génie Logiciel
ETS
Montréal, Canada
alain.abran@etsmtl.ca

Sophie Stern
Direction de l'Ingénierie Electrique et des Systèmes Electroniques
Renault SAS
Guyancourt, France
sophie.stern@renault.com

Amar Ramdan-Cherif
LISV – Laboratoire d'Ingénierie des Systèmes de Versailles
UVSQ
Vélizy, France
rca@prism.uvsq.fr

***Abstract*— To obtain the functional size of software and reduce measurement variance caused by the interpretations of individual measurers, a number of measurement procedures have been designed based on measurement methods approved as international standards. To date, most of these procedures have targeted Management Information Systems software, while only a few were designed for real-time and reactive embedded software. In this paper, we propose a functional size measurement (FSM) procedure for real-time embedded software, based on the COSMIC method, version 3.0.1 (ISO 19761), and with requirements documented using the Simulink model. The design of this FSM procedure is based on the mapping of key concepts in both Simulink and COSMIC, and the identification of the mapping rules for extracting the information stored in the Simulink files that is required for measurement. This procedure therefore provides the foundation for automating the measurement of software requirements documented using Simulink. The background study for this procedure was conducted at Renault SAS using ECU (Electronic Control Unit) functional requirements expressed with the Simulink tool.**

Functional size measurement; real-time systems; estimation; measurement procedure; automation tool; COSMIC ISO 19761

I. INTRODUCTION

The use of Electronic Control Unit (ECU) software in cars has grown considerably in recent years. In the car development process, the ECU's software development and validation tasks are clearly identified up front, and the corresponding milestones in global car development planning are strongly positioned. To manage more tightly the development costs of software suppliers, as well as its own validation activities, Renault SAS measures the functional size of their ECU software requirements using the international standard ISO 19761, and takes this information as the main input for estimating software development costs and schedules.

Software development is a complex process with a large number of drivers of both success and failure. While in general software projects frequently suffer cost overruns and are delivered late or not at all [1], in the highly competitive car industry very tight management of these projects is critical. To address this issue, Renault SAS has implemented innovative software measurement programs to develop and implement better models for estimating the effort and duration of software projects, so that they can take proactive action to manage the risks associated with carrying them out or avoid them altogether [2,3,4].

Software measurement is a powerful tool for managing software projects. It allows engineering principles to be applied to software development, which provides an objective and quantitative base for process and technology decisions. For instance, software size gives a measure of the software product itself, and can be used to obtain software development productivity ratios and to build objective estimation models for predicting project effort and duration. In practice, software size, measured in function points, for instance, is highly correlated with project work effort.

There are two approaches to measuring the size of software: physical size measures, such as SLOC [5], and functional size measures, such as the methods based on software functionality. Physical size measures include counts of source lines of code (SLOC), modules, object classes, and the like, and even lines of documentation. However, such measures suffer from several disadvantages. For example: their measurement results can vary depending on the line counting conventions adopted; they are highly dependent on programming languages; and their accuracy cannot be determined until the software is built. The functional size measures are independent of programming languages, and they allow early estimation of the size of the end-product.

Renault SAS has chosen the COSMIC–ISO 19761 standard for measuring the size of real-time embedded software and for estimating project costs [2] [3] [16]. They also use this standard to estimate the memory size of their embedded software [4]. This function point-based measurement method is applicable at both the beginning of the development process, in the requirements elicitation or specification phase, and at the end of the project, after implementation for benchmarking studies.

Currently, the measurement of functional size is mostly performed manually and it is time consuming. To speed up the measurement process and to reduce the possibility of human errors, several attempts have been made to automate functional size measurement (FSM) methods, in particular the first generation of these methods. Mendes *et al.* [6] propose a framework to classify the functions that could be automated. Stambollian *et al.* [7] present a reference framework made up of the set of functions that is of interest to practitioners who implement ISO FSM standards. In [7], a 2006 survey was presented of the COSMIC-related tools available, both on the market and in the research community, followed by a gap analysis in which the functions that still needed to be addressed by tool vendors are identified. In [8], the authors present a tool for the automated measurement of function points from the design specifications of an application expressed using UML (sequence diagrams and class diagrams). A tool for measuring the Function Point diagram and a Data Flow Diagram (DFD) is presented in [9]. First, the informal and general Function Point measurement rules are translated into rules expressing the properties of the entity relationship data flow diagram (ER–DFD). Next, the rules are translated into Prolog for verification purposes. There are some commercial tools available, like Saver [10] and Xupper & Xeadian [11], but only in Japanese and with no information provided about their measuring procedure. DIAB *et al.* work with COSMIC-FFP on ROOM and Rational Rose Real Time [12] [13], and propose some guidelines to deal with the mapping of the generic concepts of the embedded real-time type of system to COSMIC concepts.

To avoid measurement errors and to speed up their process of measurement of the many new ECU software requirements modeled and documented with the Simulink software tool, Renault SAS initiated a research project in 2009 to address the lack of tools for automatically sizing software requirements in the context of embedded real time reactive software systems documented using Simulink. This research project was carried out in collaboration with the software engineering research teams at École de Technologie Supérieure (University of Québec, Canada) and the UVSQ (University of Versailles St-Quentin en Yvelines France) [16].

[16] presents overviews of the steps designed to automate the COSMIC measurement method in the industrial context where Renault uses two different modeling tools (Statemate and Simulink) to prepare the functional user requirements (FURs) that must be allocated to software developed by third-party suppliers.

This paper presents the design of the FSM procedure based on the COSMIC method [14] to obtain the functional size of software requirements expressed with the Simulink tool. This includes the documentation of the mapping of the Simulink concepts to the COSMIC concepts, guidelines, rules, and heuristics related to the context of embedded real time reactive software systems. For instance, each element of the Simulink model had to be translated into a COSMIC concept using rigorous rules to avoid ambiguities and help reduce the discrepancies between different measurers of the same software application. The objective was to provide a foundation for an automated measurement system. In addition, a measurement tool prototype was developed at Renault SAS to implement this approach to generate the functional size of a piece of software expressed using a Simulink model.

This paper is organized as follows. Sections II and III provide overviews of the COSMIC method and of the Simulink modeling tool, and discuss how to apply the COSMIC method to Simulink, including a Simulink example. Section IV introduces the FSM procedure designed for software specified using Simulink and its rules for the development of new software modules. Section V presents an illustration of the proposed approach applied to the Simulink Equation System example. Sections VI and VII present respectively the steps followed in automating the FSM procedure, and the design of the automation tool. Section VIII presents our conclusions.

II. COSMIC OVERVIEW

The COSMIC method provides a standardized method for measuring the functional size of software from both the business application domain (also referred to as Management Information Systems or MIS) and the real-time domain. The COSMIC method is considered a second-generation FSM.

This method has been accepted as an International Standard: ISO/IEC 19761 Software Engineering – COSMIC – A functional size measurement method (hereafter referred to as ISO 19761). The latest version of the COSMIC manual available on the COSMIC website is version 3.0.1. While this release includes a number of refinements, the original principles of the COSMIC method have remained unchanged since they were first published in 1999.

The COSMIC method measures the Functional User Requirements (or FUR) of software. The result obtained is a numerical 'value of a quantity' (as defined by the ISO) representing the functional size of the software.

Functional sizes measured by the COSMIC method are designed to be independent of any implementation decisions embedded in the operational artifacts of the software to be measured. This means that the FUR can be extracted not only from actual software already developed but also from the model of the software before it is implemented.

The measurement process of the COSMIC method, version 3.0.1, consists of three phases:

A. The COSMIC Software Context Model is applied to the software to be measured. This is the Measurement Strategy Phase.
B. The COSMIC Generic Software Model is applied to the software to be measured. This is the Mapping Phase.
C. The Measurement Phase follows, in which the actual size measurement results are obtained.

The measurement result corresponds to the functional size of the FUR of the software measured, and is expressed in COSMIC Function Points (or CFP).

III. THE SIMULINK MODELING TOOL

Simulink is a Matlab [15] tool used to model, simulate, and analyze dynamic systems. With this tool, the behavior of a large number of dynamic systems can be studied, including electrical, mechanical, thermodynamic, and embedded real time systems.

Simulink creates a visual model of the system to be implemented, which can then be used to simulate its behavior.

Simulink takes the block diagram approach to modeling a system, since it is possible to model any type of dynamic system using the appropriate blocks. These blocks, which are the main elements of Simulink, are predefined in a model or user defined. They have inputs and outputs, and can also have configurable parameters. Every block acts as a basic system that implements a particular treatment, while parameters can influence the exact behavior of the block.

In some cases, blocks can have states. These are called StateFlow blocks. The state can change the output of blocks of this type, according to a value based on the previous state or previous input, or both.

Simulink also allows subsystems to be created, which in turn allows systems within systems to be modeled. Subsystems embody a group of blocks and signals in a single block, and so the inputs and outputs are specified in the construction of the block.

Blocks are connected via ‘lines’, which constitute the other type of element in Simulink. They allow the output(s) of one block to be linked to the input(s) of another block, for example.

To summarize, Simulink is an environment for multi-domain simulation and model-based design for dynamic and embedded systems, with which a detailed diagram of a system can be created, modeled, and maintained using a set of elements. It provides the means for hierarchical modeling, data management using subsystem customization, and model referencing.

A system’s behavior is described by the combination of blocks used and their types. In addition, the lines represent the data flows, and so the architecture of the model also reflects its behavior.

In other words, Simulink contains all the details necessary to identify and categorize the software features of a module and the data movements taking place during the processing of a feature.

A. *Simulink example: A Simple Equation System*

Fig. 1 shows an example of the high level design and structure of some functional requirements documented using the Simulink tool. The example, which we call the Simple Equation System, has two subsystems: Equation_1 and Equation_2.

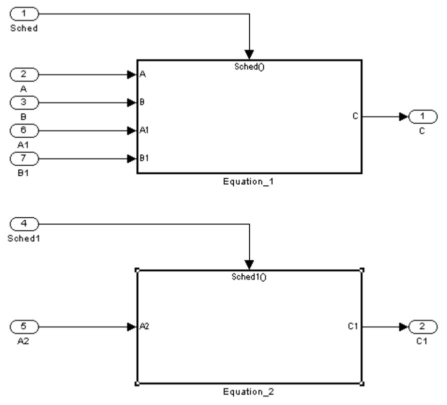


Figure 1. High level functional requirements of the Equation_1 and Equation_2 subsystems

Equation_1 executes two series of operations, as specified in its detailed functional requirements: when triggered, the first series of operations consists of adding 5 to the value of B, and then multiplying the result by the value of A. The final result is obtained as a value C. The mathematical equation for this series as modeled is: C = A x (B+5). The second series of operations consists of adding 5 to the value of B1, and then multiplying the result by the value of A1. But, instead of obtaining the final result immediately, as in the first series, the detailed functional requirement is that the result be written in memory through the ‘variable’ M. The mathematical equation of this series as modeled is: M = A1 x (B1+5).

The Simulink blocks used to model Equation_1, as presented in Fig. 2, are: 4 Inport blocks (A and B, A1 and B1), 1 Outport block (C), 2 Constant blocks (Constant1, Constant2), 2 Sum blocks (Add, Add1), 2 Product blocks (Product, Product1), 1 Data Store Write block (Data Store Write), and 1 Triggerport block (Sched). All the blocks are put into a subsystem block called Equation_1, as shown in Fig. 1.

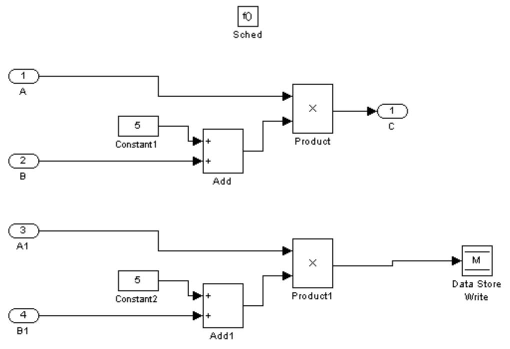


Figure 2. Detailed functional requirements of the Equation_1 subsystem

Equation_2, when triggered, consists of adding the value of M (in memory) to the value of A2. The final result of this sum is obtained as a value C1. The mathematical equation for this series, as modeled, is: C1 = A2 + M.

The Simulink blocks used to model this equation (Equation_2), as presented in Fig. 3, are: 1 Inport block (A2), 1 Outport block (C1), 1 Sum block (Add2), 1 Data Store Read block (Data Store Read), and 1 Triggerport block (Sched1).

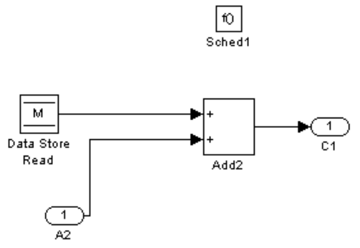


Figure 3. Detailed functional requirements of the Equation_2 subsystem

IV. AN FSM PROCEDURE FOR REAL TIME EMBEDDED SOFTWARE DESIGNED USING SIMULINK

To correctly measure the functional size of software requirements documented and modeled with the Simulink tool, the measurement objective, scope, and other elements must be identified and well defined. The FSM procedure must also have a set of measurement rules to be applied to the functional requirements modeled in order to obtain their functional sizes. For accurate results, to facilitate the measurement process, and to permit automation of the procedure, these rules must be clear and consistent. The FSM procedure proposed in this paper is based on version 3.0.1 of COSMIC (the latest version). In this section, the main concepts defined in COSMIC are mapped to the Simulink model notation.

In the Matlab Simulink R2006b block library, there are various block categories: Sink blocks, Source blocks, Logical Operators, etc., and in each category there are a number of blocks.

A. *The measurement strategy phase*

Purpose: The purpose of this FSM procedure is to apply the COSMIC method to the Simulink model, i.e. to measure the size of the FUR of any system based on its functional requirements described using the Simulink model. The size of the FUR will be used to estimate the effort required to develop the software (documented using Simulink).

Scope of the measurement: The scope of this FSM is at the detail level of the subsystems of the Simulink Model.

Level of granularity: The level of granularity is at the block level of the Simulink Model.

Functional User: The functional users are all subsystems interacting (sending or receiving data) with the specified software (Root Subsystem).

B. *The mapping phase*

Functional Process: A subsystem containing an elementary block is a functional process: when triggered, it receives, manipulates, and moves data groups. StateFlow blocks are also functional processes.

Boundaries: There is a boundary lying between any external software (functional user) and the software to be measured (application). There is also a boundary between any two subsystems (peer components in the same layer).

Data groups: Identifying the data groups is a key element in correctly identifying the movements of data groups in each functional process. A line conveys a single data group and is taken a priori as a data group.

The rules for identifying the functional users, the software boundary and the functional processes are presented in Table I. Tables II and III present the rules for identifying the data group movements. The rules for obtaining the functional size of each functional process and the whole software are presented in table IV.

V. APPLYING THE FSM TO THE SIMPLE EQUATION SYSTEM EXAMPLE

A. *Functional processes of the Simple Equation System*

According to rule 1, there are two subsystems that contain at least one elementary block in the example: Equation_1 and Equation_2 are the only two functional processes in this example. The next step consists of obtaining the functional size of each of the functional processes identified.

B. *The functional size of the 'Equation_1' functional process*

This functional process is described using 4 Inport blocks, 1 Outport block, 2 Constant blocks, 4 Elementary blocks (2 Sum blocks and 2 Product blocks), and 1 Data Store Write block. It is triggered by a Triggerport block. Table V shows all the data movements identified in this functional process. The total size of the Equation_1 functional process is 9 CFP.

TABLE I. SIMULINK/COSMIC MAPPING RULES

COSMIC element	**Rule number**	**Rule description**
Functional user (FU)	1	Identify 1 functional user for each external subsystem that interacts (sends and/or receives data to/from) any FP identified according to rules 4 & 5
Boundary	2	Identify 1 boundary between any external subsystem interacting with the subsystem to be measured
Boundary	3	Identify 1 boundary between two functional processes interacting with each other
Functional process (FP)	4	Identify 1 functional process for each subsystem containing at least one elementary block
Functional process	5	Identify 1 functional process for each StateFlow block

C. *The measurement phase*

The data group movements of each functional process are identified in this phase using the rules in Tables II and III. Once all the data movements in a functional process have been identified, the standard value of 1 CFP is assigned to each data movement. The final step consists of aggregating the results to obtain the functional size of each functional process (rule 18). The functional sizes of the functional processes are next aggregated to obtain the functional size of the software being measured (rule 19).

Data movement identification: The rules for correctly identifying the four types of COSMIC data movements are presented in Tables II and III. The COSMIC triggering events are identified using rule 6 and rule 13. The COSMIC Entry (E) and Exit (X) data movements are identified using the Source Port block type: rule 7 and the Sink Port block type: rule 9 respectively; or they can be related to lines coming from/going to Subsystems (rules 8 & 10). Rule 14 & 15 are used to identify the COSMIC Entry (E) and Exit (X) data movements from StateFlow blocks.

The Read (R) and Write (W) data movements are bound to the DataStoreRead blocks and the DataStoreWrite blocks respectively (rule 11 & 12). They are also identified when the functional process measured is a StateFlow block (rules 16 & 17), in Table III.

TABLE II. RULES TO IDENTIFY DATA MOVEMENTS OF A SIMULINK MODEL (FROM **NON** STATEFLOW BLOCKS)

Data group movements	6	Identify a 1E data movement – and COSMIC triggering event – for the TriggerPort, EnablePort, or Function-Call Generator
Data group movements	7	Identify a 1E data movement for each InPort (or any other Source Port) connected via a line to any elementary block inside this subsystem.
Data group movements	8	Identify a 1E data movement for each line from a subsystem (identified as a functional process) (the line's source) and connected to an elementary block (the line's destination)
Data group movements	9	Identify a 1X data movement for each OutPort (or any other Sink Port) connected via a line to any elementary block in this functional process
Data group movements	10	Identify a 1X data movement for each line to a subsystem (identified as a functional process) (the line's destination) and from an elementary block (the line's source) inside this FP
Data group movements	11	Identify a 1R data movement for each DataStoreRead identified in this functional process coming from an elementary block (the line's source) inside this FP
Data group movements	12	Identify a 1W data movement for each DataStoreWrite identified in this functional process and connected to an elementary block (the line's destination)

TABLE III. RULES TO IDENTIFY DATA MOVEMENTS OF A SIMULINK MODEL (FROM STATEFLOW BLOCKS)

Data group movements	13	Identify a 1E and COSMIC triggering event data movement for an event starting this functional process
Data group movements	14	Identify a 1E data movement for each data object that is *an input to the chart from a Simulink block* and used by a condition in this Stateflow chart
Data group movements	15	Identify a 1X data movement for each data object that is *an output from the chart to a Simulink block* and used by an action in this Stateflow chart
Data group movements	16	Identify a 1R data movement for each data object that is local to the Stateflow chart and used by a condition in this Stateflow chart
Data group movements	17	Identify a 1W data movement for each data object that is local to the Stateflow chart and used by an action in this Stateflow chart

TABLE IV. RULES FOR OBTAINING THE FUNCTIONAL SIZE OF THE FUNCTIONAL PROCESSES AND THE WHOLE SOFTWARE.

Functional Process (FP)	18	Aggregate the CFP related to the data movements identified in a specific FP to obtain the functional size of that process.
Whole Software	19	Aggregate the CFP related to the data movements of (identified in) the functional processes of (identified in) the whole system to obtain the functional size of that system.

TABLE V. FUNCTIONAL SIZE OF EQUATION_1

No. of the rule applied	Type(s) of block(s) identified	Name(s) of the block(s) identified	Data movement type	CFP value
6	Triggerport	Sched	E	1
7	Inport; Product	A; Product	E	1
7	Inport; Sum	B; Add	E	1
7	Constant; Sum	Constant1; Add	E	1
9	Outport; Product	C; Product	X	1
7	Inport; Product	A1; Product1	E	1
7	Inport; Sum	B1; Add1	E	1
7	Constant; Sum	Constant2; Add1	E	1
12	Data Store Write; Product	M	W	1
				Total: 9 CFP

D. *The functional size of the 'Equation_2' functional process*

This functional process is described using 1 Inport block, 1 Outport block, 1 Constant block, 1 Elementary block (1 Sum block), and 1 Data Store Read block. It is triggered by a Triggerport block. Table VI shows all the data movements identified in this functional process. The total size of the *Equation_2* functional process is 4 CFP.

TABLE VI. FUNCTIONAL SIZE OF EQUATION_1

No. of the rule applied	Type(s) of block(s) identified	Name(s) of the block(s) identified	Data movement type	CFP value
6	Triggerport	Sched1	E	1
7	Inport; Sum	A2; Add2	E	1
9	Outport; Sum	C1; Add2	X	1
11	Data Store Write; Product	M	R	1
				Total: 4 CFP

To obtain the functional size of the whole system, the size of all the functional processes of the module are aggregated (rule 27): Size (Equation System) = Size (*Equation_1*) + Size (*Equation_2*) = 13 CFP.

VI. AUTOMATING THE FSM PROCEDURE FOR SIMULINK

A. *Designing the common representation*

In order to automate the FSM procedure, a semi-formal representation has been proposed. This representation maps four key elements from the COSMIC method to different elements of the Simulink modeling tool. The main purpose of the semi-formal representation is to extract all the necessary information, and enough of it, to obtain the functional size of the software to be measured according to the COSMIC method. Thus, in this semi-formal representation the elements defined in the COSMIC method are identified to ensure a complete mapping between the COSMIC method rules and the Simulink real-time modeling tool model, that is:

1. Functional users: users that are senders and/or intended recipients of data in the FUR of a piece of software.
2. Functional processes: each functional process is an elementary component of a set of FUR comprising a unique, cohesive, and independently executable set of data movements. It is triggered by a data movement (an Entry) from a functional user that informs the piece of software that the functional user has identified as a triggering event. It is complete when it has executed all that is required to be done in response to the triggering event.
3. Data groups: distinct, non empty, non ordered, and non redundant set of data attributes where each data attribute included describes a complementary aspect of the same object of interest.
4. Data movements: A base functional component which moves a single data group type. There are four sub types of data movement types: Entry, Exit, Read, and Write.

Table VII shows the four different COSMIC elements mapped to different elements of the Simulink modeling tool. In addition to the figures, their descriptions, the corresponding rules used in building the semiformal presentation are also presented.

TABLE VII. COSMIC /SIMULINK MAPPING TABLE

Element name	Figure	Figure description	Simulink elements	No. of the corresponding rule(s)
Functional Users	⬚	Green dotted box	Subsystems	1
Functional Processes	▭	Blue box	Subsystems	4&5
Movements of Data Groups	→	Black arrow	{lines + blocks}	6 to 10 and 13 to 15
Persistent Storage	⛁	Purple cylinder	Data Store Write/ Data Store Read Blocks	11, 12,16, and 17

Fig. 4 shows the semi-formal representation of the example, the Simple Equation System. No specific functional users could be identified, because no specific External Subsystems were modelled in the requirements of the System, all Exit and Entry data group movements are therefore between the System and an entity called 'External'. Two functional processes are identified in this example:

Equation_1 and Equation_2. Nine data group movements are identified in Equation_1: 7E coming from outside the System, in addition to 1X going outside the System + 1W going to Persistent Storage. In Equation_2, four data group movements are identified: 2E coming from outside the System, in addition to 1X going outside the System + 1R coming from Persistent Storage.

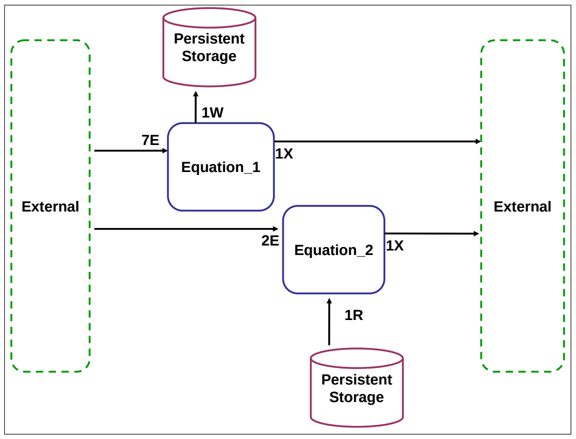


Figure 4. Semiformal representation of the Simple Equation System

VII. Design of the automation tool

A. Introduction

An automation tool should help in applying and using the selected FSM method in industry. At Renault SAS, a prototype was developed to automate the measurement of the functional size of the requirements made with the Simulink modeling tool.

B. Algorithm

The algorithm of the automation tool must follow the proposed FSM procedure step by step. In other words, the rules in Table VII for obtaining the semi-formal representation are mapped to this algorithm, which begins with identifying the functional processes. Next, the data movements in each functional process (FP) are identified and assigned a numerical value of 1 CFP each. Finally, the sizes of all the identified data movements of all the FP are aggregated into a single functional size value.

As a first step, the prototype shall identify all the possible functional processes (FP) from the input: i.e. search for all the subsystem blocks in the input.

1 Search for all the StateFlow blocks in the input. Each StateFlow block is a functional process.

2 Search for elementary blocks in every subsystem block. In doing so, each block should be tested if it is on the elementary block category list. If an elementary block is found in the subsystem, that subsystem becomes a functional process.

3 For each Non StateflowBlock FP (subsystem block containing at least one elementary block), check all its blocks to determine their types:

3.1 For each triggerport block, identify one Entry data movement.

3.2 For each source block, check the blocks to which it is connected. If it is connected to an elementary block, identify one Entry data movement.

3.3 For each sink block, check the blocks to which it is connected. If it is connected to an elementary block, identify one Exit data movement.

3.4 For each subsystem block identified as a functional process according to step 3:

3.4.1 Identify one Entry data movement for each of its outputs, if the next block it is connected to is an elementary block,

3.4.2 Identify one Exit data movement for each of its inputs, if the next block it is connected to is an elementary block,

4 For each DataStoreRead block, check the blocks to which it is connected. If it is connected to an elementary block, identify one Entry data movement.

5 For each DataStoreWrite block, check the blocks to which it is connected. If it is connected to an elementary block, identify one Exit data movement.

6 For each FP identified using the StateFlow blocks, according to step 2:

6.1 Identify 1 Entry data movement for every event starting this FP;

6.2 Test If each Data object used by a condition in this StateFlow chart is an input to the chart from a Simulink block outside the StateFlow Block:

6.2.1 THEN: identify 1 Entry data movement

6.2.2 ELSE: identify 1 Read data movement

6.3 Test IF each Data object used by an action in this StateFlow chart is an output to a Simulink block outside the StateFlow block:

6.3.1 THEN: identify 1 Exit data movement

6.3.2 ELSE: identify 1 Write data movement

7 Assign 1 CFP value for each Data Movement identified.

8 Aggregate the size of all the Data Movements (i.e. the Entry, Exit, Read, and Write identified in step 3) inside that FP.

9 Aggregate the size of all the functional processes identified in steps 2 and 3.

C. Measuring the Simple Equation System example using the prototype

The prototype tool for automating the measurement of the equation example was developed in Java at Renault SAS. Fig. 5 shows the measurement result of the Equation System obtained with the proposed prototype. It identified two functional processes: Equation_1 and Equation_2, for a total functional size of 13 CFP:

- In the functional process of Equation_1, it identified 9 data movements: 7 Entry data movements, 1 Exit data movement, and 1 Write Data movement (Sub-total = 9 CFP).

- In the functional process of Equation_2, it identified 4 data movements: 2 Entry data movements, 1 Exit data movement, and 1 Read Data movement (Sub-total = 4 CFP).

```
           ######       FP REPORT              ######

      # NAME OF FUNCTIONAL PROCESS 1 IS: Equation_1 #

      # NAME OF FUNCTIONAL PROCESS 2 IS: Equation_2 #

              ##            FP REPORT DONE!!     ##


  ##           DATA MOVEMENT DETAILS FOR FP: Equation_1 ##

NUMBER:1                 TYPE:E      DATA MOV: External(Constant2)
                              -->            Equation_1

    NUMBER:2               TYPE:E      DATA MOV: External(B1)
                              -->            Equation_1

     NUMBER:3              TYPE:W      DATA MOV: Equation_1
                  -->            Persistent_Strorage(DataStoreWrite)

 NUMBER:4                TYPE:E      DATA MOV: External(Constant1)
                              -->            Equation_1

     NUMBER:5              TYPE:E      DATA MOV: External(B)
                              -->            Equation_1

     NUMBER:6              TYPE:E      DATA MOV: Trigger
                              -->            Equation_1

    NUMBER:7               TYPE:E      DATA MOV: External(A1)
                              -->            Equation_1

     NUMBER:8              TYPE:E      DATA MOV: External(A)
                              -->            Equation_1

     NUMBER:9              TYPE:X      DATA MOV: Equation_1
                              -->            External(C)

  ##           DATA MOVEMENT DETAILS FOR FP: Equation_2 ##

          NUMBER:10              TYPE:R      DATAMOV:
     Persistent_Strorage(DataStoreRead)   -->            Equation_2

    NUMBER:11              TYPE:E      DATA MOV: External(A2)
                              -->            Equation_2

     NUMBER:12             TYPE:X      DATA MOV: Equation_2
                             -->             External(C1)

     NUMBER:13             TYPE:E      DATA MOV: Trigger
                              -->            Equation_2


 TOTAL FUNCTIONAL SIZE: 13 CFP              { E:9 X:2 R:1 W:1 }
```

Figure 5. Automated measurement results of the Simple Equation System

D. *Verification protocol*

A three-phase protocol was proposed in [16] for the verification of the tools implementing the COSMIC measurement method. This verification protocol uses samples at the input level of the automation tool to cover most types of input cases that could be encountered. The samples must also be measured manually, using the FSM procedure described in sections IV and V. This means that, in addition to the total functional size obtained in CFP, all the individual functional processes and data group movements are identified manually, for verification purposes, at the detailed level of the measurement process. Therefore, in the verification protocol, the causes of discrepancies across the parallel automated and manual measurement results can be identified and addressed. A detected error could be caused either by the manual measurement or by the automation tool, and the protocol makes it possible to determine what caused the error.

Moreover, if no difference is detected between the manual measurement result and the automated one, the protocol provides the option of stopping the verification process [16]. Experience has shown that, even though the parallel measurement results may be equal at the total CFP level, differences at the detailed level, i.e. at the level of the identification of individual functional processes or data movements, could have taken place. Therefore, as a refinement to the verification protocol proposed in [16], if the verification stops at phase 1, it is considered to be a high-level non detailed one. So, in order to fully verify an automation tool, the protocol should be applied in all three phases – see Fig. 3. This instantiation of the measurement protocol proposed in [16] revealed certain ambiguities in the order in which the subtasks should be executed in phases 2 and 3, and it is this updated protocol, which permits a clearer and more accurate description of the three phases of the protocol, that is provided here.

- Phase 1: Numerical results comparison. In this phase, the results (in CFP), both those produced automatically by the prototype and those obtained from the manual measurement of the same input, are compared. If there is a match between the results, then no difference is detected between the automated and manual measurement processes. However, this does not mean that the processes for identifying the individual COSMIC elements that are used to obtain the final results are similar. It is important to understand that if the verification stops at this phase, only the final results will have been verified.

- Phase 2: Detailed comparison. If the final numerical results at the end of phase 1 do not match, and to find the cause of the difference, phase 2 starts by comparing the results at the detailed level, that is, verifying the number of the functional processes obtained automatically and manually. It is also necessary to verify, using the detailed measurement results obtained by the prototype tool, whether or not there were any human errors in those results.

1 If there is no difference in the number of functional processes, each automatically obtained functional process is verified against its manually obtained “peer” to determine whether or not there is a difference in their names (or their identifiers)

2 If every functional process obtained automatically matches its “peer” obtained manually, then their functional sizes are compared. A difference indicates that one or more data movements in the functional process must be responsible. Then, at the end of this phase, any data movement responsible for an error is isolated.

- Phase 3: Prototype and input inspection. This phase is triggered when the possibility of a human error (in the manual measurement) is discarded at the end of Phase 2. A detected error could have two sources: a measurement error or an error in the requirements-input to the measurement process. Therefore, this phase consists of the following:

1 Inspecting the module of the automation tool that is responsible for the error. These modules can have sub modules; if so, the sub module causing the error is inspected.
2 Inspecting, in parallel, the requirements-input to detect a possible defect that may be causing the error.

Once an error is detected, the following steps are taken:

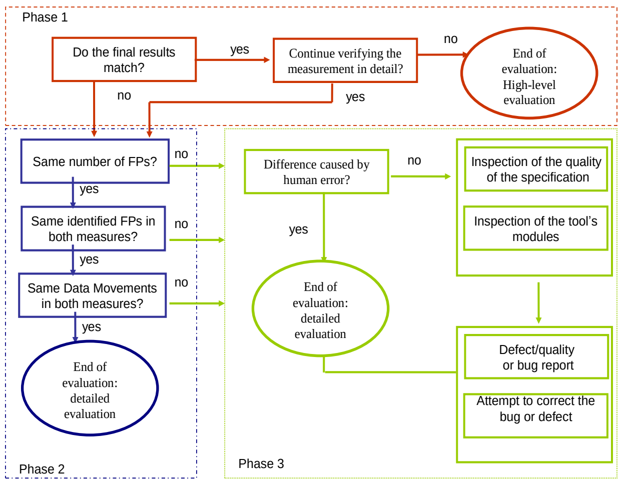
1 If the error was caused by the automation tool, a correction is made to the tool and the appropriate specification is re-measured with the new version of the tool, and then re-verified.
2 If the cause of the error was in the specification itself, the defect is recorded for possible future enhancements to the specification and/or a specific functionality, in order to bypass this defect in the tool.

Finally, the revised version of the input requirements is verified by the protocol to ensure that the results of the manual and automated measurements are the same.

Fig. 6 shows the phases of the verification protocol graphically.

E. *Verification of the Simple Equation System example*

Using the manual and automated measurement results from tables V and VI and Fig. 5 of the Simple Equation System example, the results of the application of the verification protocol are presented in Table VIII. We see that there is no difference in the final measurement results of the two measurement procedures (manual and automated): the number of functional processes and their identifications are exactly the same. Lastly, precisely the same data movements were identified in both measurement procedures.



Phase 1
Do the final results match?
yes
Continue verifying the measurement in detail?
no
End of evaluation: High-level evaluation
no
yes
Same number of FPs?
no
yes
Same identified FPs in both measures?
no
yes
Same Data Movements in both measures?
no
yes
End of evaluation: detailed evaluation
Phase 2
Difference caused by human error?
no
yes
End of evaluation: detailed evaluation
Inspection of the quality of the specification
Inspection of the tool's modules
Defect/quality or bug report
Attempt to correct the bug or defect
Phase 3

Figure 6. The 3 phases of the verification protocol

TABLE VIII. APPLICATION OF THE VERIFICATION PROTOCOL ON THE SIMPLE EQUATION SYSTEM

Phase no.	**Sub step**	**Manual measurement**	**Automated measurement**	**Outcome comment**
1	Check that the final results are the same	13 CFP	13 CFP	YES, the final results are equal
1	Continue verifying the measurement in detail?			YES: enter phase 2 (detailed evaluation)
2	Same number of FP?	2	2	YES
2	Same FP identified in both measurements?	Equation_1 Equation_2	Equation_1 Equation_2	YES
2	Same Data Movements in both measurements?	{ E:9 X:2 R:1 W:1 }	{ E:9 X:2 R:1 W:1 }	YES: End of the detailed evaluation

VIII. CONCLUSION

This paper has proposed a functional size measurement (FSM) procedure for real time embedded software requirements documented using the Simulink modeling tool. This procedure is based on the newest version of the COSMIC measurement method, which is designed, unlike the other traditional FSM methods, to measure both real time and management information systems.

The rules of this procedure cover the measurement of new software projects, and they can be considered as a set of rules that allow mapping the Simulink conceptual elements to the COSMIC concepts. However, the manual application of our procedure to models of a very large set of requirements takes time and requires specialized expertise when these requirements are also complex. Measurement with automated tools eliminates measurement variances caused by interpretations made by different measurers, which may lead to different measurement results for the same set of requirements. That is why a tool that automates the measurement procedure while ensuring the accuracy of the measurement results is useful, and can benefit organizations in terms of reducing the workload of measurement specialists, as well as eliminating measurement delays. This work also provides a basis for the development of other types of automated measurement tools.

We also presented overviews on the Simulink model and the COSMIC method, and explained how to map the COSMIC method to the Simulink model. To clarify the rules, an example of the measurement of a Simple Equation System modeled in Simulink was described and then used in this work.

Finally, a verification protocol for automation tools that implement an FSM procedure based on COSMIC was designed and applied with the prototype developed. An example of its application was presented in this paper using our Simple Equation System.

REFERENCES

[1] Sommerville, Ian., Software Engineering, Addison Wesley; 2006.

[2] Stern, S., "Practical experimentation with the COSMIC method in the automotive embedded software field", IWSM/Mensura, Amsterdam; 2009.

[3] Stern, S., and Gencel, C., "Embedded software memory size estimation using COSMIC: a case study," IWSM/MetriKon/Mensura, Stuttgart, Germany, 2010.

[4] Stern, S. and Guetta O., "Manage the automotive embedded software development cost by using a Functional Size Measurement Method (COSMIC) ", ERTS² 2010, 5th International Congress & Exhibition, Toulouse, France, 2010.

[5] Albrecht, A., Gaffney, J. E. Jr., "Software function, source lines of code, and development effort prediction: a software science validation", IEEE transactions on software engineering, vol. 9, no. 6, pp. 639-648 (9 ref.), 1983.

[6] Mendes, O., Abran, A., Bourque, P., "FP Tool Classification Framework and Market Survey", IFPUG Fall Conference, International Function Point Users Group, Dallas, 1996.

[7] Stambollian, A. and Abran, A., “Survey of Automation Tools Supporting COSMICFFP – ISO 19761”, International Workshop on Software Measurement – IWSM/MetriKon, Postdam, Germany, Shaker Verlag, pp. 435-454, 2006.

[8] Uemura, T., Kusumoto, S., and Inoue, K., "Function Point Analysis using design specifications based on the unified modelling language”, Journal of Software Maintenance and Evolution: Research and Practice, vol. 13, pp. 223-243, 2001.

[9] Evelina Lamma, Paola Mello, and Fabrizio Riguzzi, "A System for Measuring Function Points from an ER–DFD Specification", The Computer Journal 47(3):358-372, doi:10.1093/comjnl/47.3.358, 2004.

[10] Vest software Co. (3003) SAVER. http://www.vest.co.jp/saver/saver.php3

[11] Ken System Development C. (2003) Xupper & Xradian http://www.kensc.co.jp.accessed, July 7, 2003.

[12] Diab, H., Frappier, M., and St-Denis, R., “Formalizing COSMIC-FFP Using ROOM”, Int. Conf. on Computer Systems and Applications (AICCSA), pp. 312-318, Beirut , Lebanon, 2001.

[13] Diab, H., Koukane, F., Frappier, M., and St-Denis, R., “µcROSE: Automated Measurement of COS-MIC-FFP for Rational Rose Real Time”, Journal of Information and Software Technology, 47(3), pp. 151-166, 2005.

[14] Abran, A., Desharnais, J., Lesterhuis, A., Londeix, B., Meli, R., Morris, P., Oligny, S., O’Neil, M., Rollo, T., Rule, G., Santillo, L., Symons, C., and Toivonen, H., The COSMIC Measurement Manual, version 3.0.1, http://www.cosmicon.com/

[15] http://www.mathworks.com/

[16] Soubra, H., Stern, S. Abran, A., Ramdan-Cherif, A., “Automation of Functional Size Measurement of the Model-based Requirements of Real-time Embedded Software”, IWSM/MetriKon/Mensura, Stuttgart, Germany, 2010