$\mu_c ROSE$: Automated Measurement of COSMIC-FFP for Rational Rose RealTime

H. Diab, F. Koukane, M. Frappier [*], and R. St-Denis

Département d'informatique, Université de Sherbrooke, Sherbrooke (Québec), Canada, J1K 2R1

Abstract

During the last ten years, many organizations have invested resources and energy in order to be rated at the highest level as possible according to some maturity models for software development. Since measures play an important role in these models, it is essential that CASE tools offer facilities to automatically measure the sizes of various documents produced using them. This paper introduces a tool, called $\mu_c ROSE$, that automatically measures the functional software size, as defined by the *COSMIC-FFP* method, for *Rational Rose RealTime* models. $\mu_c ROSE$ streamlines the measurement process, ensuring repeatability and consistency in measurement while reducing measurement cost. It is the first tool to address automatic measurement of *COSMIC-FFP* and it can be integrated into the *Rational Rose RealTime* toolset.

Key words: Measure, measurement tool, COSMIC-FFP, function points, real-time system, Rational Rose RealTime

1 Introduction

This paper describes $\mu_c ROSE$[1], a CASE tool that automatically measures COSMIC-FFP for Rational Rose RealTime (RRRT) models. COSMIC-FFP

[*] Corresponding author.
Email addresses: `Hassan.Diab@USherbrooke.ca` (H. Diab), `Marc.Frappier@USherbrooke.ca` (M. Frappier), `Richard.St-Denis@USherbrooke.ca` (R. St-Denis).

[1] $\mu_c ROSE$ is pronounced *McRose*, and derived from the concatenation of μ, c, and Rose, which stand for measure, COSMIC-FFP, and Rational Rose RealTime.

is a functional size (FS) measure initially designed for real-time systems [4]. It was inspired from Albrecht's function points (FP), widely used in industry for information systems [2]. COSMIC-FFP addresses some shortcomings of FP for real-time systems and simplifies the measurement task. It should be noted that COSMIC-FFP is not restricted to real-time systems; it can also be used for information systems. It is the first method that conforms to ISO/IEC 14143-1, which specifies a set of generic mandatory requirements for a method to be called a *functional size measurement method*; COSMIC-FFP has also been recently adopted as an ISO/IEC standard (19761).

RRRT is a CASE tool for designing embedded, real-time, and distributed systems. It originates from the acquisition and evolution of the *ObjecTime* toolset by *Rational*. *ObjecTime* was created to support the *Real-Time Object-Oriented Modeling* (ROOM) language [17]. RRRT is mainly used in telecommunications, avionics, and process control. The main reasons for selecting RRRT as the first target for COSMIC-FFP measurement automation are its market penetration and close correspondence with COSMIC-FFP.

FS measures like COSMIC-FFP and FP are applied in several key areas of the software process. The first motivation behind the development of FS measures was to estimate the software development cost (effort), because size is the main factor influencing cost. FS is more relevant than *lines of code* (LOC) to estimate cost early on in the software development process, the latter being measured too late. FP and COSMIC-FFP have been successfully used to build cost estimation models. They can also be used to measure productivity (by computing FS unit per effort unit) or quality (by computing defect per FS unit). An organization may also benchmark its productivity across several organizations using FS measures. For instance, the *International Software Benchmarking Standards Group* [12] collects and provides data to its members, while preserving anonymity. Members can benchmark their productivity based on FP or COSMIC-FFP.

COSMIC-FFP can be measured from various documents and at various times during the software process. At the beginning of a project, it is measured from the functional requirements and can be applied in a cost estimation model to estimate cost. Costs derived in the requirements definition phase can be adjusted during design or implementation by re-measuring COSMIC-FFP from design documents or the source code, since the functionalities usually evolve during a project. When a project is coming to an end, the final COSMIC-FFP size and actual effort can be stored in a database of completed projects in order to build or improve a cost estimation model.

Currently, the COSMIC-FFP measurement process is manual, performed by experts who must gather the appropriate information from project documentation. COSMIC-FFP is defined in plain natural language and therefore not specific to a particular functional specification notation. It is formulated in terms of simple concepts like functional process, data group, and data movement. A measurer must interpret these concepts for the specification notation at hand. This interpretation can be subjective (i.e., the measurement may vary depending on the measurer) and may lead to repeatability problems (measurement variance). A recent report on field trials of COSMIC-FFP notes that nearly perfect repeatability is obtained by experienced engineers, but junior engineers show poor repeatability [1]. The measurement rules are subject to interpretation, as noted in a preliminary study based on an experiment with a substantial system [8]. FP is also subject to repeatability problems and interpretations. Studies revealed differences varying from 11% to 30% between several measurements of the same specification by different measurers [9,13–15]. These figures hold when the count boundary is well-defined, but larger variations are observed when boundaries are not clearly delineated. There are various causes for measurement variance: different (acceptable) interpretation of a measure, lack of proper training on measurement, and ambiguous or incomplete functional requirements document.

$\mu_c ROSE$ improves the COSMIC-FFP measurement process in two ways. First, it almost eliminates measurement costs and advanced measurement training, because measurement is automatic. The measurer only has to identify the subset of the system, expressed as a list of capsules, that has to be measured; $\mu_c ROSE$ handles all the analysis and calculations of COSMIC-FFP by processing an RRRT model. Second, under the assumption that measurers select capsules that belong to the same layer, $\mu_c ROSE$ removes measurement variance and ensures perfect repeatability, because the measurement algorithm is completely automated. This algorithm is based on the author's interpretation of the COSMIC-FFP definition and has been validated by experts who participated in defining the COSMIC-FFP measure. This interpretation has been formalized in a mathematical definition that is publicly available in a companion paper [5]. $\mu_c ROSE$ implements this mathematical definition.

The current version of $\mu_c ROSE$ requires an all-inclusive RRRT model, so that it can be used from the end of the programming phase to measure the actual value of COSMIC-FFP for the corresponding system. This means that $\mu_c ROSE$ can assist in building a database of completed development projects (containing actual COSMIC-FFP size, actual cost, and other measures) in order to build cost estimation and defect prediction models, and manage software maintenance and outsourcing, all of which are based on functional software size.

Automation of FS measurement has attracted the interest of both case tools vendors and researchers. According to Mendes et al. [16], most commercial tools provide clerical support for storing FP components and calculating FP from a list of FP components. Eight tools claim to measure FP directly from functional requirements models (e.g., data flow diagrams, entity-relationship diagrams, or object models), but their accuracy has not been independently validated. To the best of our knowledge, no measurement algorithm implemented in any of these tools has been published by vendors or researchers. *HierarchyMaster FFP* is a clerical tool that allows the recording of functional processes and data movements, and the computation of COSMIC-FFP [11]. In contrast, $\mu_c ROSE$ automatically extract the list of functional processes, data groups, and data movements from an RRRT model.

The rest of this paper is organized as follows. Sections 2 and 3 summarize the main concepts of COSMIC-FFP and RRRT, respectively. They also introduce the terminology used in this paper. Section 4 presents a tour of $\mu_c ROSE$, particularly its main functionalities, architecture, and interface. Section 5 contains a discussion about problems inherent to the formalization of the COSMIC-FFP measurement rules, the solutions selected with their consequences, and alternatives. Section 6 gives two class models that capture concepts of COSMIC-FFP and RRRT related to the measurement process and establishes connections between them. Section 7 concerns the validation and verification of $\mu_c ROSE$. Finally, Section 8 concludes with a summary of the key points presented in this paper, $\mu_c ROSE$ limitations, and future directions.

2 A Brief Overview of COSMIC-FFP

In COSMIC-FFP, a system can be decomposed into layers. Each *layer* represents a level of abstraction that performs a set of functional processes. In this context, software in one layer exchanges data with software in another layer through their respective functional processes. A *functional process* is defined as a unique set of data movements. Each *data movement* is considered as a particular *subprocess* that is either an *entry*, an *exit*, a *read*, or a *write*. The four types of subprocess represent an input received from the environment (entry), an output sent to the environment (exit), data read from the storage side (read), and data updated within the storage side (write). Elementary data used by a functional process is called *data attribute*. A *data group* is defined as a set of data attributes that are logically related based on a functional perspective.

As stated in the COSMIC-FFP measurement manual [4], the *measurement function* and *measurement standard* are as follows:

> The COSMIC-FFP measurement function is a mathematical function which assigns a value to its argument based on the COSMIC-FFP measurement standard. The argument of the COSMIC-FFP measurement function is the data movement type.

The COSMIC-FFP measurement standard, 1 Cfsu (Cosmic Functional Size Unit), is defined as one elementary data movement.

The following expression describes how the number of COSMIC-FFP is calculated:

$$\begin{aligned} Size_{Cfsu}(\mathtt{layer_i}) \triangleq\ & \mathbf{Card}(\mathtt{entries}) + \mathbf{Card}(\mathtt{exits}) + \\ & \mathbf{Card}(\mathtt{reads}) + \mathbf{Card}(\mathtt{writes}) \end{aligned}$$

The term $Size_{Cfsu}(\mathtt{layer_i})$ denotes the FS of a software at $\mathtt{layer_i}$. Variables `entries`, `exits`, `reads`, and `writes` denote the sets of all the entries, exits, reads, and writes that are identified in $\mathtt{layer_i}$, respectively. To compute the number of elements into each of these sets, the cardinality operator is used. For instance, the term $\mathbf{Card}(\mathtt{entries})$ represents the number of elements into `entries`.

The aforementioned expression is used in the measurement phase of the COSMIC-FFP measurement method. Prior to this phase is the mapping of the software to be measured to the COSMIC-FFP generic software model based on rules and procedures prescribed by the COSMIC-FFP measurement method. $\mu_c ROSE$ implements theses rules and procedures, and applies them to RRRT models.

The COSMIC-FFP measure ignores the algorithmic complexity and timing constraints, two distinguishing features of real-time software systems that require as much attention as data movements. In the context of COSMIC-FFP they are entangled in data movements, thus implicitly and partially captured by the size of functional processes. For instance, each timer is a functional process that accepts an entry for recording the tick. The execution of a functional process is also triggered by an event that occurs in the environment, which includes other software layers, users, and physical devices. This may results in a wider range of interactions compared with those that are considered in information systems.

3 An Overview RRRT

The paradigm behind RRRT allows developers to model architectures and designs of software systems that are highly event-driven, concurrent, and distributed. In addition, these systems must generally satisfy timing constraints. RRRT models result from the use of a graphical and textual notation based mainly on UML [3]. Embedded segments of executable code written in an object-oriented programming language like C++ form a large part of RRRT models. Hence, RRRT supports a software life cycle from use case analysis through design, code generation, and model execution. A toolset, which comprises model editors, navigators, an incremental model compiler, static and dynamic checkers, and code generators, makes it possible to create, modify, simulate, and implement models.

In an RRRT model, the design might be observed via two different viewpoints: structure and behavior. The structure describes the system's architecture in terms of components and links between them. The behavior represents system dynamics. It shows how the system may evolve over time and depends on the time and occurrence of some events.

The structure of an RRRT model is built from three kinds of entity: capsules, protocols, and data classes. A *capsule* is an active entity that exhibits both static and dynamic semantics. The RRRT paradigm integrates the encapsulation principle. A capsule is encapsulated and has restricted visibility with respect to other capsules. A capsule offers its services that might be reached through its *ports* and *connectors*. Each service request corresponds to a specific *message*, also called *signal*. A port must refer to a *protocol* that represents a set of possible messages exchanged between capsules. Finally, a *data class* is the basic unit for data representation. An object of a data class may be sent or received in conjunction with a message. A data class is also used to define capsule attributes.

The behavior of an RRRT model specifies the dynamic aspects of each capsule by using an *extended finite state machine* inspired from Harel's *Statecharts* [10]. A state may be decomposed into substates. States that are not decomposed are called leaf states. A state machine is defined by a state context that may have *variables*, an *entry action*, an *exit action*, *substates*, and *transitions*. Executable state machines are a feature of RRRT. An action associated with a transition or state may incorporate code written in an object-oriented programming language.

To illustrate, Figure 1 contains a part of an RRRT model for an elementary factory system that includes two producer machines and one consumer machine with a table separating them. In a cycle, a producer machine takes components from its own basket, produces a part, and puts the part on the table. Similarly, a consumer machine removes a part from the table, performs some operations, and feeds its own basket. A controller must supervise the factory in accordance with two constraints: i) the number of parts on the table must not exceed its capacity); ii) a consumer machine must not try to take a part from an empty table. The structure diagram (left part in Figure 1) identifies the components of the factory system represented by the capsules `Producer` (a replicated capsule with an instance factor fixed to 2), `Consumer`, and `Controller`. The state machine (right part in Figure 1) gives the behavior

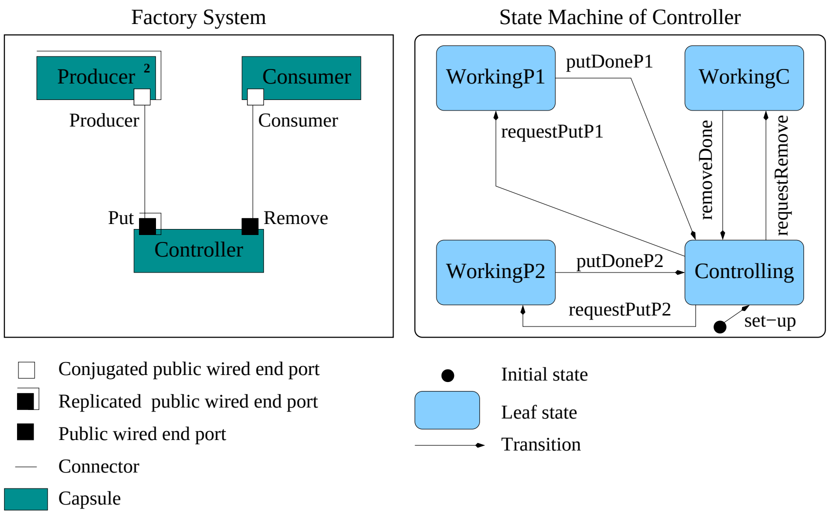


Fig. 1. The structure diagram of the factory system and state machine diagram of capsule `Controller`

of the capsule `Controller`. For instance, the transition labeled `requestPutP1` from the state `Controlling` to the state `WorkingP1` indicates that the first producer has the permission to put a part on the table. The transition labeled `putDoneP1` from `WorkingP1` to `Controlling` represents the fact that the first producer has effectively put a part on the table. The other transitions act in a similar way. Segments of executable code written in C++ are added to actions associated with the different transitions.

4 A Tour of $\mu_c ROSE$

$\mu_c ROSE$ offers basic facilities to aid measurers and project managers in the measurement process. It accepts software requirements of any size written in the RRRT notation and calculates its FS based on the COSMIC-FFP method. A user can select any collection of capsules included in an RRRT model and obtain measurement data with a small amount of interactions. The main functionalities supported by $\mu_c ROSE$ are: i) visual support of the COSMIC-FFP measurement process; ii) generation of an RRRT model in XML format; iii) extraction of RRRT entities required in the measurement process; iv) analysis of C++ code included in the RRRT model; v) identification of functional processes, data groups, and data movements; vi) calculation of COSMIC-FFP, representing the FS; and vii) aggregation and reporting of measurement results.

These functionalities represent the initial core to be considered as a minimum to make experiments on COSMIC-FFP with large real-time software systems over a short period and validate more deeply the theoretical framework implemented in $\mu_c ROSE$, which maps RRRT concepts to COSMIC-FFP concepts. Finally, $\mu_c ROSE$ has been developed in Java using *Sun microsystems* JDK 1.3 on a *Windows NT* platform.

4.1 The Architecture

Figure 2 shows the functional architecture of $\mu_c ROSE$. It comprises four main modules organized around the `Manager` module, which coordinates their activations and manages data exchange.

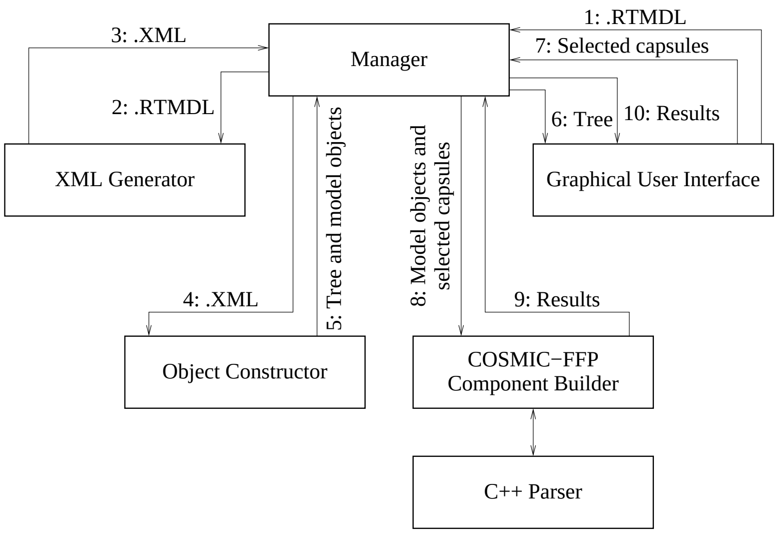


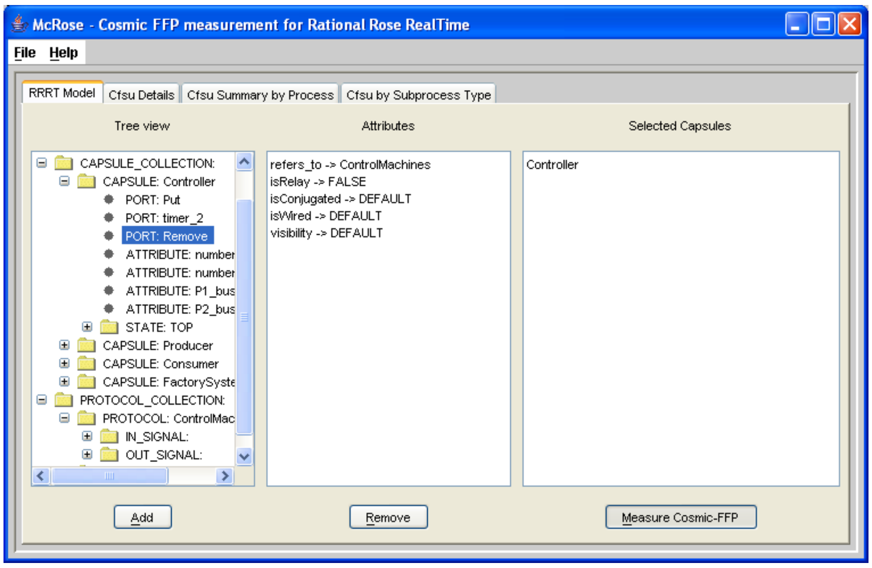
Fig. 2. The functional architecture of $\mu_c ROSE$

The `XML Generator` reads an RTMDL file, which contains the internal representation of an RRRT model, analyzes it, and produces a corresponding lightened file in XML format. The analysis includes the extraction of RRRT entities that are pertinent for the application of the COSMIC-FFP method and omission of irrelevant data. `XML Generator` output is used by the `Object Constructor` as input in order to build and store in the main memory instances of Java classes that correspond to the extracted RRRT entities. The `Object Constructor` also supports the visualization of these entities, since it constructs a hierarchy of RRRT entities that is presented in a tree view by the `Graphical User Interface` module.

From the selected capsules, the `COSMIC-FFP Component Builder` determines the functional processes, data groups, and data movements inside the boundary. It classifies data movements according to their type, assigns a numerical value to each functional process, and aggregates the measurement results. This module is strongly connected with the `C++ Parser` module that accepts a piece of C++ code and a list of attributes. The parser analyzes the syntax of the code to characterize the attributes as modified attributes or referred attributes in order to classify data movements. Finally, the `Graphical User Interface` module provides a user-friendly interaction between the user and $\mu_c ROSE$.

4.2 The Interface

The interface includes a main window with four tab boxes. In the following figures, the tab boxes contain data specific to the factory system introduced in Section 3. As shown in Figure 3 the `RRRT Model` tab box is composed of three panels: a tree view of RRRT entities, a list of attributes, and a list of selected capsules.



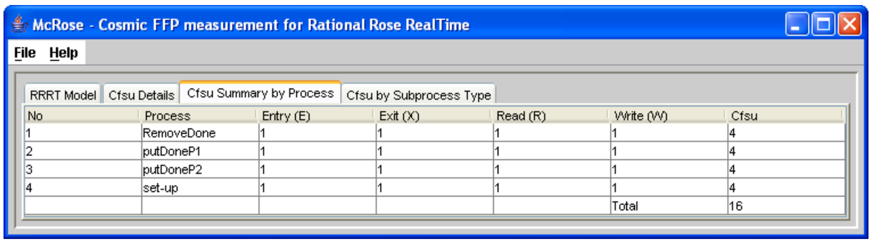
```
CAPSULE_COLLECTION:
  CAPSULE: Controller
    PORT: Put
    PORT: timer_2
    PORT: Remove
    ATTRIBUTE: number
    ATTRIBUTE: number
    ATTRIBUTE: P1_bus
    ATTRIBUTE: P2_bus
    STATE: TOP
  CAPSULE: Producer
  CAPSULE: Consumer
  CAPSULE: FactorySyste
PROTOCOL_COLLECTION:
  PROTOCOL: ControlMac
    IN_SIGNAL:
    OUT_SIGNAL:
```

```
refers_to -> ControlMachines
isRelay -> FALSE
isConjugated -> DEFAULT
isWired -> DEFAULT
visibility -> DEFAULT
```

```
Controller
```

Fig. 3. The RRRT model tab box

The measurement process starts by opening an RTMDL file, which contains an RRRT model, from the `File` menu. The entities extracted from the RRRT model by the `XML Generator` are brought in the tree view panel. The user then selects capsules one-by-one from the tree view panel and adds them to the selected capsules panel. The selected capsules must belong to the same level of abstraction to be considered as a layer in the COSMIC-FFP context. Note that $\mu_c ROSE$ cannot verify if the selected capsules belong to the same layer. The concept of layer is subjective, hence it is difficult to formalize. The `Add` (`Remove`) button is used to add (remove) capsules to (from) the selected capsules panel. Finally, the functional size is calculated by clicking the `Measure Cosmic-FFP` button. A summary of the measurement results is then displayed in the `Cfsu Summary by Process` tab box (see Figure 4). It contains the functional processes, number of Cfsu assigned to each functional process, and total number of Cfsu representing the FS. More details about the measurement results can be displayed by selecting the `Cfsu Details` (Figure 5) and `Cfsu` `by Subprocess Type` (Figure 6) tab boxes.

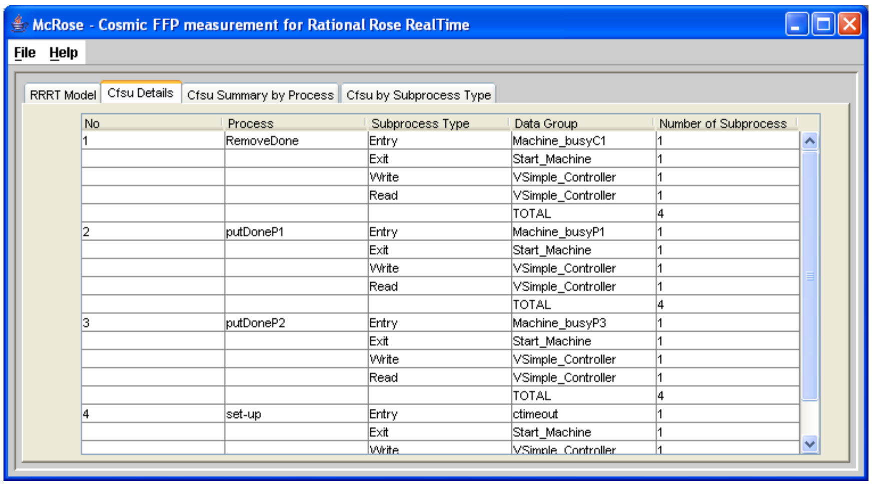

McRose - Cosmic FFP measurement for Rational Rose RealTime

File Help

RRRT Model | Cfsu Details | Cfsu Summary by Process | Cfsu by Subprocess Type

No	Process	Entry (E)	Exit (X)	Read (R)	Write (W)	Cfsu
1	RemoveDone	1	1	1	1	4
2	putDoneP1	1	1	1	1	4
3	putDoneP2	1	1	1	1	4
4	set-up	1	1	1	1	4
					Total	16

Fig. 4. Summary by process of COSMIC-FFP measurement

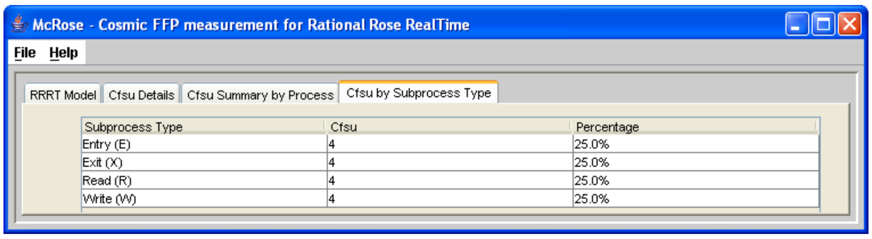

McRose - Cosmic FFP measurement for Rational Rose RealTime

File Help

RRRT Model | Cfsu Details | Cfsu Summary by Process | Cfsu by Subprocess Type

No	Process	Subprocess Type	Data Group	Number of Subprocess
1	RemoveDone	Entry	Machine_busyC1	1
		Exit	Start_Machine	1
		Write	VSimple_Controller	1
		Read	VSimple_Controller	1
			TOTAL	4
2	putDoneP1	Entry	Machine_busyP1	1
		Exit	Start_Machine	1
		Write	VSimple_Controller	1
		Read	VSimple_Controller	1
			TOTAL	4
3	putDoneP2	Entry	Machine_busyP3	1
		Exit	Start_Machine	1
		Write	VSimple_Controller	1
		Read	VSimple_Controller	1
			TOTAL	4
4	set-up	Entry	ctimeout	1
		Exit	Start_Machine	1
		Write	VSimple_Controller	1

Fig. 5. Details of COSMIC-FFP measurement


McRose - Cosmic FFP measurement for Rational Rose RealTime

File Help

RRRT Model | Cfsu Details | Cfsu Summary by Process | Cfsu by Subprocess Type

Subprocess Type	Cfsu	Percentage
Entry (E)	4	25.0%
Exit (X)	4	25.0%
Read (R)	4	25.0%
Write (W)	4	25.0%

Fig. 6. Summary by type of COSMIC-FFP measurement

5 Formalization of the COSMIC-FFP Measurement Rules

The formalization of the COSMIC-FFP measurement rules represents a challenge in regard to the variety of problems to be solved because nothing is recommended in the literature or COSMIC-FFP manual concerning this aspect. Four main related issues are examined. They concern the diversity of measurement rules, the essence of the COSMIC-FFP measurement manual, the representation of the measurement rules as well as the underlying vocabulary, and the selection of a functional specification notation as a language to describe software to be measured.

5.1 Diversity of Measurement Rules

The measurement rules can be classified according to their role in the COSMIC-FFP method and the effort that must be invested to implement them in a measurement tool. Four kinds of rules are considered: counting rules, identification rules, mapping rules, and expert rules.

Counting rules are very simple because they are described by arithmetic expressions as the one given in Section 2. They can be easily implemented in a measurement tool, but at the expense of greater effort from measurers if the tool does not support other kinds of rules, especially those that are necessary to identify software elements used in the computation of arithmetic expressions. Such simple tools are limited to the acquisition and recording of COSMIC-FFP elements, and the calculation of COSMIC-FFP.

Identification rules concern the identification of software layers, software boundaries, functional processes, data groups, data attributes, and data movements. These rules, prescribed by the COSMIC-FFP method, are not by themselves sufficient unless the software to be measured is described with respect to the COSMIC-FFP generic software model. This is rarely the case in practice. Generally, other rules are required for instantiating the COSMIC-FFP generic software model from a software given in a specific form. Mapping rules are used for this purpose, but they are not are included in the COSMIC-FFP method. They must be determined and implemented in a measurement tool for each specification language. The richer the set of mapping rules, the higher the sophistication of a measurement tool. Identification and mapping rules are concretized by a fine-grained parser and an extractor based on the syntax and semantics of the specification language.

Additional knowledge is relevant to better understand, apply, or implement the COSMIC-FFP method. This knowledge can be acquired from experienced software measurers and translated into executable forms. Expert rules mimic procedures and heuristics used by experts when they apply identification and mapping rules in a specific context. They also represent alternatives to consider in the interpretation of rules because software artifacts can be given in different forms. Finally, they may convey a history of decisions in order to provide a coherent measurement process. A measurement tool can implement and maintain expert rules by adopting a knowledge-based approach.

5.2 Essence of the COSMIC-FFP Measurement Manual

In the FS software metrics area, the greater the degree of independence from specification languages, the lower the degree of formality used in their definition. This viewpoint adopted by the international group of software measurement experts that participated in the definition of COSMIC-FFP has significant consequences for measurers who want to apply or formalize the COSMIC-FFP method. Most of the measurement rules are defined in plain natural language and are not specific to a particular specification language. Furthermore, the COSMIC-FFP measurement manual is subject to successive revisions by experts in order to clarify the measurement rules as they are used in various contexts. Therefore, ambiguities and misinterpretations necessarily arise in the interpretation of the COSMIC-FFP measurement rules.

Ambiguities arise when measurers must interpret definitions or some aspects of vocabulary, for example, the definition of functional process given on page 36 of the COSMIC-FFP measurement manual [4].

> “A functional process is an elementary component of a set of Functional User Requirements comprising a unique cohesive and independently executable set of data movements. It is triggered by one or more triggering events either directly, or indirectly via an actor. It is complete when it has executed all that it is required to be done in response to the triggering event (-type).”

The term “cohesive” appearing in this definition is highly subjective. Furthermore, it is very difficult to define precisely. In our formalization for RRRT, we have interpreted this definition as all the transitions that are triggered by the reception of a message from outside the boundary. Another example is the following sentence (page 36): “A triggering event is an event(-type) that occurs outside the boundary of the measured software and initiates one or more functional processes.” It is not clear what kind of triggering events may initiate several functional processes and how to be sure to not count them as one functional process. In our formalization for RRRT, we identify a single process for each triggering event type. Finally, the notion of data group is also subjectively defined as follows (page 37): “A data group is a distinct, non empty, non ordered and non redundant set of data attributes where each included data attribute describes a complementary aspect of the same object of interest.” We have chosen to count each class that is used for typing an attribute of a capsule as a single data group; moreover, capsule attributes of elementary types are grouped to form a single data group.

The major source of misinterpretation arise in the mapping phase of the measurement process because software to be measured is represented in a given notation. Some software artifacts particular to the notation do not have a direct representation in the COSMIC-FFP method; other artifacts are broader than COSMIC-FFP concepts. The lack of a one-one onto mapping results in a situation in which no universal agreement on a single interpretation of the COSMIC-FFP rules emerges.

These drawbacks may lead two persons to come up with different size measures when applying the FS measurement method to the same software document. For a given functional specification notation, a mapping may be developed by an organization in collaboration with experts in the domain. Organizations document these rules in a local guide to increase measurement consistency within the organization.

5.3 Representation of the Measurement Rules and the Underlying Vocabulary

Despite the interpretation issue, additional ingredients are necessary in the formalization of the measurement rules: a mathematical representation of the measurement rules and an ontology of the underlying FS measurement domain. By merging these two ingredients we obtain a rigorously defined framework that provides a shared and common understanding of the measurement scheme in order to implement a measurement tool.

An ontology of the underlying FS measurement domain entails building at least two formal models: one from the informal COSMIC-FFP generic software model and another from an accurate description of a functional specification notation. Data models or class models may be used for that purpose in order to represent elements, properties, and relationships of the essential subjects in the domain. These two models make easier the instantiation of the COSMIC-FFP generic software model from a piece of software.

The formalization of the measurement rules is strongly motivated by the fact that it is important to proceed in a thoughtful methodical way especially since software measurement is an inherently mathematical activity. Measurers are confronted with several rules of different degrees of complexity used in various settings. A measurement tool should be based on a formal model of rules with mathematical fidelity that everyone will embraced instead of a measurement process only guided by experience, intuition, and experimental case studies. A mathematical approach helps removing ambiguities and inconsistencies. It eliminates subjective interpretations and clarifies the measurement rules. First-order logic and set theory appear to be the relevant mathematics to represent measurement rules because these formalisms facilitate reasoning from instances of ontological models.

5.4 Selection of a Specification Language

The COSMIC-FFP method can be applied to any specification language. An evaluation of the appropriateness of several specification paradigms may, however, provide guidelines in the selection of a software notation as a starting point in the formalization of the measurement rules.

Trace-based specifications are not adequate for the formalization of COSMIC-FFP measurement rules. Because they abstract from states, it is not possible to identify data groups. The same conclusion can be drawn for algebraic specifications. Sorts may represent data groups or data attributes, without any systematic way of distinguishing between them. Furthermore, it seems laborious to establish a procedure for characterizing an operation as a functional process or as an auxiliary operation. Finally, the process algebra paradigm raises similar difficulties in the identification of data groups and functional processes. In this paradigm, event parameters may be input parameters, output parameters or state information, but without any syntactic feature.

There is a natural correspondence between COSMIC-FFP and model-based specifications, but not all the notations are appropriate. For instance, in the Z notation, it is not always obvious to distinguish between a modified variable and a referred variable that are used in a predicate. In semi-formal object-oriented notations (e.g., UML), behavior diagrams are not always precise enough to properly identify data that are modified or referred by a service; these details being generally described in a natural language. In formal object-oriented notations, RRRT constitutes a natural choice because of its close correspondence with COSMIC-FFP. This point is detailed in the next section. Founded on first-order logic, set theory, and theory of refinement, the B language is also adequate for the formalization of COSMIC-FFP. Indeed we have already formalized IFPUG's Function Points measure with respect to this model-based language [6]. B models represent software by typed data, characterized by their invariant properties, and by operations which handle these data. A model defines a set of state machines, where each machine includes data and operations which represent services of a software. A simple analogy with information systems is to consider that machines are specifications of modules interacting with users. An operation corresponds to a functional process and its signature may be used to identify the software inputs and outputs. Furthermore, the use of elementary substitutions in operations facilitates the classification of data.

5.5 Epilogue

Among all the solutions presented in the previous subsections, some are embodied in $\mu_c ROSE$. First, it has been design to perform measurements on RRRT models. Second, the knowledge-based approach has been discarded, but counting rules, identification rules, and mapping rules have been formalized by using first-order logic and set theory. Third, some rules have been clarified with the help of experts of the COSMIC-FFP group. Finally, two class models, one for COSMIC-FFP components and another for RRRT entities, have been developed to support the execution of identification and mapping rules. They can be considered as a first step to a genuine ontology of the FS measurement domain.

The design of $\mu_c ROSE$ was not only based on these decisions. It results from a methodical approach which can be replicated for the development of other COSMIC-FFP measurement tools, but for different functional notations, because it is not specific to RRRT. The class model for COSMIC-FFP components can also be reused in that case. The main steps are the following: make a feasibility study concerning the appropriateness of a functional notation, develop a formal model of the selected functional notation, determine the scope of the new measurement tool, formalize the measurement rules with respect to ontological models, validate the formal rules with experts, verify the measurement tool from a sample of projects, and eventually obtain a certification for the new measurement tool.

Some design choices in $\mu_c ROSE$ may not be obvious to apply for another notation, particularly those choices made during the development of mapping rules. An analogy with the LOC measure helps to precise this point. If this specific measure is applied to programs with the same functionalities, but written in different programming languages, various sizes will be obtained. Normalization factors can then be determined from a sample of projects. Normalization is certainly required to obtain equivalent COSMIC-FFP sizes for similar documents, but written in different notations.

6 The Class Models of $\mu_c ROSE$ and Formalization of Mapping Rules

In addition to the functional architecture introduced in Section 4.1, several models have been derived during the design of $\mu_c ROSE$, in particular RRRT and COSMIC-FFP class models. These models capture RRRT entities (e.g., capsules, protocols, ports, messages, attributes) and COSMIC-FFP components (e.g., functional processes, data groups, data movements), respectively. Thus, each class is considered an abstraction of a group of objects (entities or components) that share the same attributes, operations, relationships, and semantics.

The processing performed by the `Object Constructor` and `COSMIC-FFP Component Builder` largely depends on these models. On one hand, the `Object Constructor` creates instances of classes in $\mu_c ROSE$'s RRRT model to represent entities from which different measures can be derived. On the other hand, the `COSMIC-FFP Component Builder` uses these instances in a specific context. It instantiates objects in $\mu_c ROSE$'s COSMIC-FFP model in order to calculate COSMIC-FFP. This processing expresses a mapping from a set of entities in an RRRT model to a representation in a COSMIC-FFP model.

6.1 The RRRT Class Model

A capsule is the fundamental modeling element of an RRRT model. A capsule may have the following properties: attributes, ports, operations, behavior, supercapsule, and subcapsules. Figure 7 shows the class `Capsule` and its associations with the classes `Attribute`, `Port`, `Action` (representing capsule operations), and `State` (representing capsule behavior). These associations describe the relationships between a capsule and its properties. The class `Capsule` also has two reflexive associations. The association `inherits` describes the inheritance relationship between capsules while the association `is_composed_of` describes the relationship between a composite capsule and its components. A capsule is said to be composite if it contains one or several capsules.

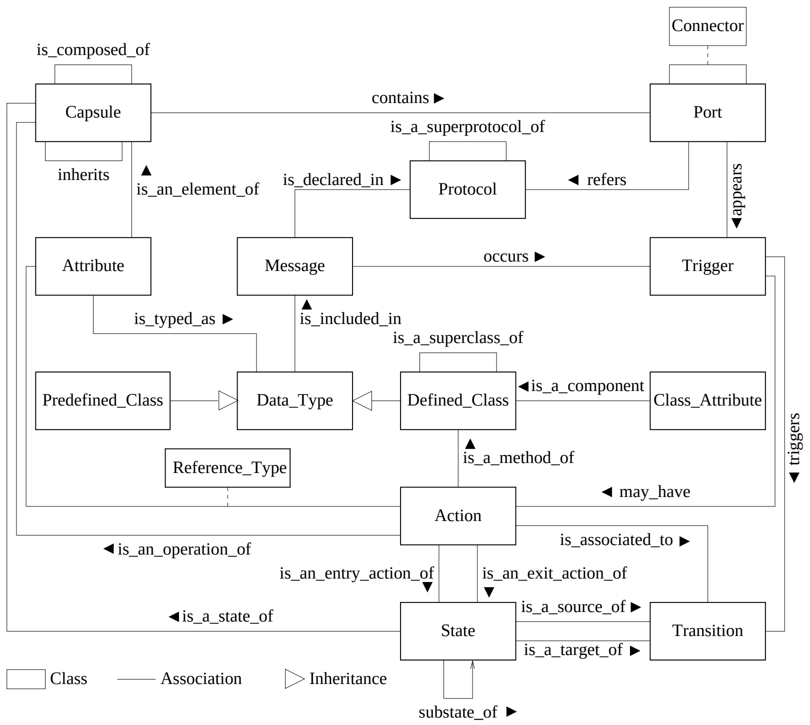


Fig. 7. Class model for RRRT models

The associations between the classes `Port`, `Protocol`, and `Message` reflect the exchanges of messages between capsules. A capsule is awakened by the arrival of a message through one of its ports. A port refers to a protocol that specifies a set of incoming messages that can be received and a set of outgoing messages that can be sent through the port. In an RRRT model, the inheritance concept is also applicable to protocols.

A hierarchical state machine is used to describe a capsule's behavior. In Figure 7, a hierarchical state machine is bound to the class `State`. It has a reflexive association necessary to express the concept of composite state that contains a lower-level state machine. The class `Action` plays a central role in the RRRT class model. An action is materialized by statements written in a programming language. It may be associated with a state as an entry or exit action. An entry action is executed whenever the corresponding state is entered. An exit action is executed whenever an outgoing transition from the corresponding state is taken. An action may also be associated with a transition and is executed whenever the transition is triggered, but after the exit action of its source state and before the entry action of its target state. The class `Transition` has associations with others classes (e.g., `State`, `Trigger`). Besides an action, a transition has a source state, a target state, and triggers that specify the names of signals that trigger the transition and ports from which these signals are received. Furthermore, a trigger may have an optional guard condition defined as a Boolean expression. This is captured by the association `may_have` between `Trigger` and `Action` classes.

A data class in an RRRT model is similar to a class in C++ or Java. It includes one or several attributes (data fields) and operations (methods). A data class may be used to define the type of an attribute peculiar to a capsule or another data class. An operation is a service that affects the internal state of an object. Data classes provided by the RRRT toolset as well as C++ primitive data types like `int` and `char` are considered as instances of `Predefined_Class`. Classes defined in an RRRT model (i.e., introduced by designers) are considered as instances of `Defined_Class`. During the computation of COSMIC-FFP, all attributes of type `Predefined_Class` in a capsule are grouped together to form a single data group. Each capsule attribute of type `Defined_Class` is identified as a single data group. As shown in Figure 7, classes `Defined_Class` and `Predefined_Class` are two subclasses of the superclass `Data_Type`. The class `Data_Type` has an association with the class `Attribute`, since each capsule attribute must be typed.

6.2 The COSMIC-FFP Class Model

A class model for COSMIC-FFP components is shown in Figure 8. It shows only those components required for measurement automation. For instance, the class `Functional_Process` denotes the functional processes of the COSMIC-FFP generic software model. It has associations with different classes to take into consideration definitions and limitations specified by the COSMIC-FFP method.

First, the association `initiates` reflects the fact that the execution of a func-

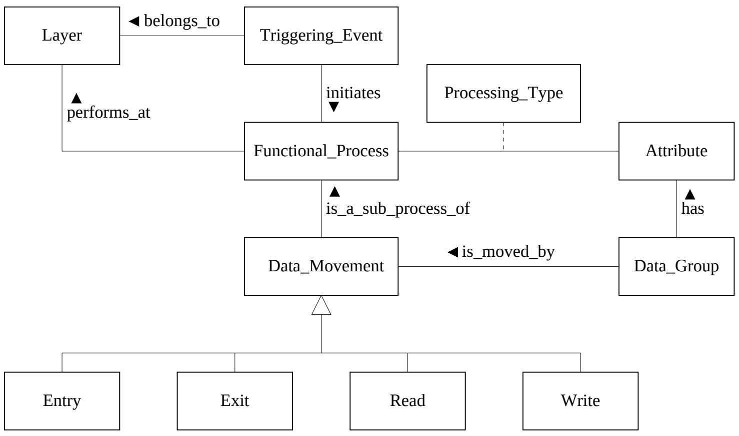


Fig. 8. Class model for COSMIC-FFP software models

tional process must be triggered by the occurrence of an event from outside the boundary. A triggering event may initiate one or several functional processes.

Second, the association `performs_at` indicates that a functional process implements a service that is provided at a specific layer. According to the COSMIC-FFP method, the measurement process is separately applied to each layer of a software model. A boundary represents the conceptual demarcation of a layer.

Third, a functional process is defined as a unique set of data movements. Based on the way attributes are used by a functional process, data movements can be identified as well as their type. A functional process may: i) receive one or several data groups from outside the boundary as input; ii) produce one or several data groups to outside the boundary as output; iii) refer to one or several data groups from the storage side without modification of their values; and iv) refer to one or several data groups from the storage side with modification of their values. These four cases are captured by the association `is_moved_by` between the classes `Data_Movement` and `Data_Group` and the specialization of the class `Data_Movement`.

Finally, it should be noted that a functional process must have at least one entry and one write or exit.

6.3 The Mapping between COSMIC-FFP and RRRT Class Models

To overcome the misinterpretation problem, $\mu_c ROSE$ is based on a theoretical framework [5] that maps RRRT concepts to COSMIC-FFP concepts. The mapping is based on mathematical definitions of COSMIC-FFP concepts with respect to RRRT concepts. A summary of the mapping, implemented through the `COSMIC-FFP Component Builder`, is given in Table 1 and explained in the following subsections with a representative subset of formal rules and references to the RRRT class model. The complete set of formal rules is given in [7].

Table 1
Correspondence between COSMIC-FFP and RRRT concepts

COSMIC-FFP Concepts	RRRT Concepts
layer	set of capsules
boundary	conceptual demarcation of a set of capsules
data group	message or a data type
functional process	one or several transitions
triggering event	arrival of an external message
attribute	attribute
entry	incoming external message
exit	outgoing external message
read	referred attributes in an action
write	updated attributes in an action

6.3.1 Identification of Layers

In an RRRT model, software is represented as a collection of capsules that might perform at one or several levels of abstraction. Given capsules, it is difficult to automatically determine these levels of abstraction. Therefore, human judgment is required to identify the capsules forming a layer. It is assumed that the capsules selected by a user correspond to a COSMIC-FFP layer. Once a selection is made, an instance of the class `Layer` is created with appropriate references to the selected capsules.

From an instance of the RRRT class model given in Figure 7, it is possible to determine the set of data and protocol classes referenced in a layer. These classes are essential in the identification of data groups. For example, associations `contains` and `refers` allow to identify protocols that belong to a capsule. The conceptual demarcation of a set of capsules, which represents the boundary, is denoted by B.

6.3.2 Identification of Data Groups

A data group in the COSMIC-FFP class model can be derived from:

- a collection of simple attributes used in a selected capsule (an attribute is simple if its type is an instance of the class `Predefined_Class`);

- an instance of the class `Defined_Class` used to specify the type of an attribute that belongs to a selected capsule; or
- an instance of the class `Message` that participates in a communication between capsules within the boundary and the environment.

Therefore, the set of data groups, denoted by DG and identified from the capsules within the boundary, is defined as the union of three sets: $DataAttributes$, $DataClasses$, and $ExternalMessages$.

$$DG \triangleq DataAttributes \cup DataClasses \cup ExternalMessages$$

The following formal rule defines the set $DataClasses$. It contains the data classes used in the declarations of attributes contained in the capsules.

$$DataClasses \triangleq \{dc \mid dc \in \texttt{Defined\_Class} \land \exists\, c, a : c \in B \ \land \ a \in attributes(c) \ \land \ type(a, dc)\}$$

where the set $attributes(c)$ (concretized by the association `is_an_element_of`) includes all the attributes declared in capsule c and the predicate $type(a, dc)$ (concretized by the association `is_type_as`) holds if the type of a is dc.

The other two sets can be defined formally in a similar manner.

6.3.3 Identification of Functional Processes

A functional process in COSMIC-FFP corresponds to a chain of transitions in an RRRT model. A chain must contain:

- transitions triggered by the same message sent from outside the boundary (called external transitions); and
- the transitions that are recursively triggered by messages generated and exchanged between capsules within the boundary during the execution of an action associated with an external transition.

Thus, an instance of the class `Functional_Process` in the COSMIC-FFP class model is created from one or several instances of the class `Transition` in the RRRT class model. An instance of the class `Triggering_Event` corresponds to the arrival of a message from outside the boundary in an RRRT model.

Formally, a transition is defined by a 5-tuple:

$$t \triangleq \langle s, input_t, A_t, output_t, d\rangle$$

where s is the source state and d is the destination state; $input_t$ is the set of all incoming messages appearing in the trigger of t; A_t is the set of all the actions associated with the transition label, the exit action of s, and the entry action of d; $output_t$ is the set of all outgoing messages generated by the actions of A_t.

The source state s and destination state d are determined by using the associations `is_a_source_of` and `is_a_target_of`, respectively. The set of actions A_t is obtained from the associations `is_associated_to`, `is_an_exit_action_of`, and `is_an_entry_action_of`. The set of incoming messages $input_t$ is derived from the associations `occurs` and `triggers`. The set of outgoing messages $output_t$ is determined by searching specific patterns that appear in actions of A_t. The form of these patterns is $p.m.\texttt{send}()$, where p is a port, m is a message, and `send()` is the method used to transmit a message.

6.3.4 Identification of Data Movements

An instance of the class `Entry` (`Exit`) in the COSMIC-FFP class model is created for each instance of the class `Message` in the RRRT class model that:

- refers to the timer; or
- is sent from the environment (by a capsule within the boundary) and received by a capsule within the boundary (to the environment).

The following rule provides a formal definition of the entries associated to an external transition t, which are data groups of incoming messages of t.

$$Entry(t) \triangleq \{dg \mid dg \in DG \ \wedge \ external(t) \ \wedge \ dg \in input_t\}$$

Finally, an instance of the class `Read` (`Write`) is created from an instance of the class `Data_Group` when one of its attributes is used (updated) in an action associated with a transition (including its source and target states as well as its guards) without (with) modification of its value. Of course, the transition should be a part of a functional process.

The following formal rule defines the set of data groups maintained by the actions associated with a transition t.

$$\begin{aligned} Write(t) \triangleq \{dg \mid dg \in DG \wedge \\ \exists a : a \in Attributes(dg) \ \wedge \ maintains(A_t, a)\} \end{aligned}$$

where the predicate $maintains(A_t, a)$ is satisfied if the attribute a is on the left-hand side of an assignment, which appears in an action of A_t, and the set *Attributes*(dg) contains all the attributes of a data group (a collection of simple attributes or a defined class)

7 The Methodology for Validating and Verifying $\mu_c ROSE$

The validation process of $\mu_c ROSE$ started early by examining and reviewing the theoretical framework with a COSMIC-FFP expert. The main goal of this process was to uncover errors in our formal model due to improper formalization or a misinterpretation of COSMIC-FFP measurement rules. Errors were detected during the review process and modifications have been made to ensure correctness and completeness.

7.1 Testing

During the development phase, unit testing and integration testing were conducted following conventional practices. For system testing, a set of scenario-based tests for verifying the functionalities and ensuring quality have been systematically derived from the formal specification of the measurement rules. Scenarios have been classified under two categories.

(1) Scenarios for testing classes that allow identification of data groups – This includes trivial and nontrivial cases for verifying the identification of the data groups from capsule attributes and messages exchanged between capsules within a boundary and its environment. Trivial cases correspond to the “lower” bound of input domains (e.g., a class without attributes, a class with one attribute).
(2) Scenarios for testing classes that allow identification of functional processes and their data movements – This includes the analysis of code embedded in actions associated with states and transitions in order to determine the type of each attribute (referred or modified) involved in the actions. This also includes the identification of chains of transitions associated with a functional process.

$\mu_c ROSE$ has been executed from various RRRT models according to different boundaries. Table 2 presents some results of experiments conducted with $\mu_c ROSE$, in order to illustrate the size of the case studies. The first model (`Factory System`) is the model introduced in Section 3. The second model (`Manufacturing System`) is a variation of the first one, in order to test several complementary aspects. The last model (`Integrating Data`) comes from case studies provided with the RRRT documentation.

The second model (`Manufacturing System`) is similar to the first model. The main difference concerns how a transition is triggered. In the first model, each transition of the state machine, that gives the behavior of the capsule `Controller`, is triggered by a particular message while there are transitions that are triggered by the same message in the second model. Thus, there are more data groups in the first model. This allows to show what are the effects of some small modifications in a model with respect to COSMIC-FFP. The critical points concern the identification of functional processes represented by a sequence of transitions instead of only one transition and the partitioning of internal and external messages with respect to a boundary. The last model (`Integrating Data`) contains two capsules: i) the `Receiver` that receives messages containing data of various types and uses the log service to display the data values and ii) the `Sender` that sends messages containing data of various types. When the two capsules are selected, the messages sent by the `Sender` to the `Receiver` are now considered as internal messages. This reduces the number of entries and the number exits with respect to the case in which only the `Sender` is selected.

Table 2
Some RRRT models used during validation and verification with their FS

RRRT Model	Capsules in the Boundary	Cfsu
Factory System	Controller	16
	Controller and Consumer	21
Manufacturing System	Controller	15
	Controller and Consumer	18
	Controller and Producer	18
	Consumer and Producer	16
	Controller, Consumer, and Producer	21
Integrating Data	Sender	10
	Receiver and Sender	5

7.2 A Validation of $\mu_c ROSE$ with an Expert

The `Factory System` case study has been measured by an independent COSMIC-FFP expert, who has more than 20 years of industrial experience in FS measurement; he also participated in the definition of COSMIC-FFP. He had not seen the results provided by $\mu_c ROSE$ before conducting the measurement. Table 3 provides the results of both the expert and $\mu_c ROSE$. The similarities and differences between the two measurements are as follow.

Data Group. The expert identifies *one data group per simple attribute* of a capsule; $\mu_c ROSE$ identifies *one data group for all simple attributes* of a capsule. This difference has a substantial impact on the Cfsu size, since the number of reads and the number of writes depends on the number of data groups accessed in a functional process. We discussed this issue with the expert and another one, who also has several years of industrial experience in FS measurement; they both agree that identifying one data group for all simple attributes is an acceptable interpretation of the COSMIC-FFP definition. The first expert has a preference for the one-variable-one-data-group interpretation, while the second one endorses the all-variables-one-data group interpretation.

Functional Process. Both the expert and $\mu_c ROSE$ identify the same functional processes when each capsule is measured separately. When two communicating capsules are included in the same boundary, the expert identifies the same functional processes as when the capsules are measured separately, whereas $\mu_c ROSE$ identifies a different set of functional processes. $\mu_c ROSE$ considers messages exchanged between capsules within the boundary as *internal messages*, since these messages do not cross the boundary; hence they cannot be considered as triggers for functional process identification. Internal messages allow a functional process to be executed in parallel by two communicating capsules. Consequently, $\mu_c ROSE$ identifies a smaller set of functional processes. The number of data movements typically decreases when two capsules are grouped within the same boundary. This is illustrated by Figure 9 and Table 4. Dotted lines denote messages that are received and sent by a transition. In this example, we assume that capsule C_1 communicates with capsule C_2 through ports P_2 and P_3, respectively. Moreover, each capsule has only one data group. When C_1 and C_2 are measured separately, transitions t_1, t_2, and t_3 constitute three functional processes. When C_1 and C_2 are included in the same boundary, messages m_2 and m_3 become internal messages, which entails that transitions t_1, t_2, and t_3 form a single functional process which is carried out by two capsules in parallel. The reads of t_1 and t_3 are merged into a single read, because the same data group is accessed in each transition; the writes of t_1 and t_3 are also merged for the same reason. The messages sent and received in t_2 are not counted, because they are internal messages; the message sent by t_1 and the message received by t_3 are not counted for the same reason. The sum of t_1, t_2 and t_3, when measured separately, is 9; when measured as a single functional process, it falls to 3. The sum of C_1 and C_2 is 12, whereas the size of C_1 and C_2, grouped in the same boundary, is 6.

Entry. Both the expert and $\mu_c ROSE$ identify the same entries.

Exit. The expert counts one exit per send action; $\mu_c ROSE$ counts one exit per message type sent. Hence, $\mu_c ROSE$ identifies only one exit when messages of the same type are sent more than once within a functional process. There are other potential deviations. For instance, an expert could count only one exit

Table 3
A comparison of expert and $\mu_c ROSE$ measurements of the `Factory System`

Capsules in the Boundary	Cfsu Expert	Cfsu $\mu_c ROSE$
Controller	38	16
Producer	8	10
Consumer	6	8
Controller and Producer	46	18
Controller and Consumer	44	21

when two messages are similar. Consider the message types `MachineBusyP1` and `MachineBusyP2`; they could be seen by an expert as the same message type, `MachineBusy`(m), where m indicates which machine is busy. $\mu_c ROSE$ cannot analyze message type identifiers and infer these facts.

Read and Write. The expert and $\mu_c ROSE$ differ, since they do not identify the same data groups. However, they do identify data movements in the same manner.

Coverage. The expert missed references to a data group in a few send actions. This raises the issue of the tediousness of manual measurement. It is time consuming and uneasy for an expert to browse the source code and the diagrams of an RRRT model to identify COSMIC-FFP components. Our case study is very simple in terms of structure and LOC size. Yet, it took the expert two hours to conduct the measurement. The identification of functional processes comprising internal messages is particularly difficult to conduct, since the information is scattered over several parts.

We draw two conclusions from this validation. First, $\mu_c ROSE$ provides an acceptable interpretation of the COSMIC-FFP measure. Its measurement differs from the one provided by the expert, but the differences are due to either a distinct but acceptable interpretation, or a more systematic application of the COSMIC-FFP definition, when dealing with fine details like messages sent, internal messages, and access to variables. Second, automatic measurement is almost mandatory for RRRT models, because the number of details to manage for a human being is overwhelming; small elements, which significantly affect Cfsu size, can be easily overlook.

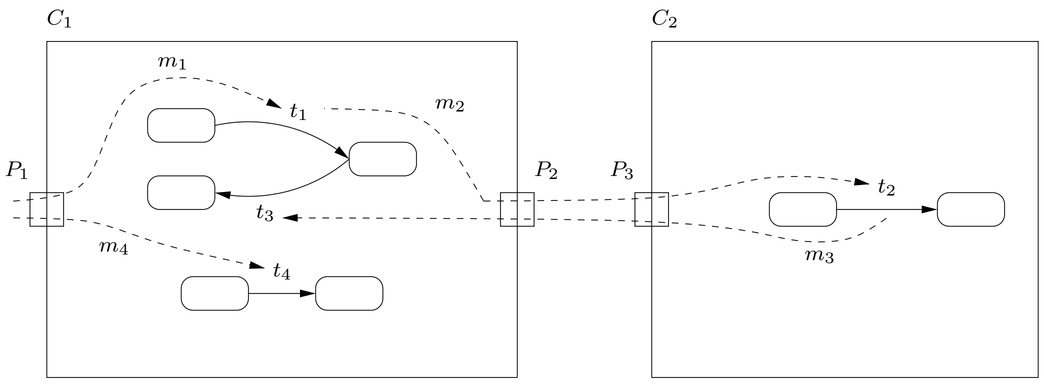


Fig. 9. A small example to illustrate the effect of grouping capsules

Table 4
The effect of grouping capsules on measurement

Functional Process	Entry	Read	Write	Exit	Cfsu
	Capsule C_1 alone				
t_1	1	1	1	1	4
t_3	1	1	1	0	3
t_4	1	1	1	0	3
Total	3	3	3	1	10
	Capsule C_2 alone				
t_2	1	0	0	1	2
	Capsules C_1 and C_2 in the same boundary				
$t_1t_2t_3$	1	1	1	0	3
t_4	1	1	1	0	3
Total	2	2	2	0	6

8 Conclusion

The ever-growing need for effective support to software project management has been exemplified with the advent of maturity models that advocate monitoring and controlling development and maintenance activities related to the software process. In order to satisfy this need, measurement components suitable for evaluating both the process and documents must be integrated into existing CASE tools. $\mu_c ROSE$ constitutes a valid contribution to this effort because it is the first measurement software component to provide a systematic, stepwise, and complete way to measure the FS of RRRT models according to the COSMIC-FFP method. It yields an objective and unobtrusive measurement process with perfect repeatability involving minimal human assistance.

$\mu_c ROSE$ represents the tangible part of a research project oriented towards the formalization of rules and procedures prescribed by methods related to FS measures not only for RRRT models, but also for B models. The fundamental part of this research project was a theoretical framework that formally expresses FS measure concepts and knowledge accumulated by experts in this area as well as their mapping with formal specification languages (ROOM and B) in a mathematical form. This kind of framework provides the means for better understanding COSMIC-FFP and clarifying some of its obscure points. Although some details of the framework will need to be changed and refined according to the evolution of COSMIC-FFP and intense use of $\mu_c ROSE$, it turns out that the development of a mathematical framework and class models has provided a sound foundation for addressing automatic measurement of COSMIC-FFP. Each modification or refinement will be repeatedly transposed into future versions of $\mu_c ROSE$.

Presently, the main limitation of $\mu_c ROSE$ concerns the `C++ Parser` module that performs the code analysis by only examining some basic operators (e.g., assignment, arithmetic and logical operators) of the C++ programming language, because arrays, pointers, and functions may cause serious problems. For instance, the parser does not analyze the code of functions called in actions associated with transitions and states. So, it is assumed that there is no side effect in the measurement when such complex operators appear in actions. These problems are not specific to $\mu_c ROSE$ but are encountered in program slicing and static data-flow analysis. According to the present state of art, no general solutions exist, but the use of heuristics allows to obtain non-optimum results when applied in some specific contexts. Heuristics appropriate to static data-flow analysis in the context of COSMIC-FFP measurement should be proposed to improve the `C++ Parser` module. We hope to address this weakness in a subsequent version of $\mu_c ROSE$.

The measurement algorithm is essentially based on rules that map RRRT concepts to COSMIC-FFP concepts. It would be inappropriate to ignore these rules and compute the functional size only from a C++ program, even if it is automatically produced by the RRRT code generator from entities of an RRRT model and segments of code attached to its actions. This would require to map the syntax and semantics of any C++ program to the COSMIC-FFP generic software model. Generally, this kind of mapping constitutes an intractable problem. Furthermore, an RRRT model contains more software artifacts that can be easily identified at a higher level of abstraction than a C++ program. Nevertheless, the segments of C++ code remain significant because they are used to classify data movements.

In addition to the aforementioned issues, we plan to fully integrate $\mu_c ROSE$

into the RRRT toolset in such a way that it loses its own identity. We also plan to connect $\mu_c ROSE$ to a project database in order to automatically populate it with FS of finished projects. As suggested in the literature, such data are required to build and improve estimation models. Finally, we will investigate how $\mu_c ROSE$ could be adapted to measure the FS of incomplete RRRT models. An RRRT model may be incomplete for two reasons. First, entities, such as capsules, data classes, protocols, or state machines, are missing while others are partially described. Second, the segments of C++ code attached to actions have not yet been provided by programmers. The former is related to design aspects. Theoretical and empirical studies should be conducted to precise what means the measurement of FS throughout the construction of RRRT models. The latter concerns programming practices. A realistic approach to tackle this issue consists in encouraging programmers to include, as soon as possible in the design phase, small fragments of code, like `x = ...`, or `... = ... y ...`, or useful comments, which will allow the parser to characterize an attribute as a modified attribute or referred attribute. The C++ parser should be modified to recognize and process such partial segments of code or codification. All these improvements should significantly reduce the measurement cost for RRRT models and yield a better control and monitoring during their development and maintenance.

Acknowledgements We would like to express our thanks to Jean-Marc Desharnais for his useful comments during the validation of the theoretical framework implemented in $\mu_c ROSE$. Of course, any remaining errors in this framework are ours. We would also like to express our thanks to Stéphane Beaudoin, Ning-Sun Cao, Ly Thanh-Le, Huu-Baoan Trinh, and Hai-Pho Truong for the intense work carried out in programming $\mu_c ROSE$.

The research described in this paper was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Fonds québécois de la recherche sur la nature et les technologies (FQRNT).

References

[1] Abran, A., Symons, C., and Oligny, S. 2001. An overview of COSMIC-FFP field trial results. 12th European Software Control and Metric s Conf. London, England.

[2] Albrecht, A.J. and Gaffney, J.E. 1983. Software function, source lines of code, and development effort prediction: A software science validation. *IEEE Trans. Soft. Eng.* SE-9 (6): 639–648.

[3] Booch, G., Rumbaugh, J., and Jacobson, I. 1999. The Unified Modeling Language User Guide. Reading, MA: Addison-Wesley.

[4] Common Software Measurement International Consortium. 2003. COSMIC FFP measurement manual, version 2.2. Software Engineering Management Research Laboratory, UQÀM, Montréal, Canada.
`http://www.lrgl.uqam.ca/cosmic-ffp/`

[5] Diab, H., Frappier, M., and St-Denis, R. 2001. A formal definition of COSMIC-FFP for automated measurement of ROOM specifications. Proc. 4th European Conf. on Software Measurement and ICT Control. Heidelberg, Germany, 185–196.

[6] Diab, H., Frappier, M., and St-Denis, R. 2002. A formal definition of function points for automated measurement of B specifications. in *Formal Methods and Software Engineering*, Lecture Notes in Computer Science, vol. 2495 , 483–494, C. George and H. Miao, Eds. Berlin, Germany:Springer.

[7] Diab, H. 2003. Formalisation et automatisation de la mesure des points de fonction. Ph.D. Thesis. Département d'informatique, Faculté des sciences, Université de Sherbrooke, Sherbrooke (Québec) Canada.

[8] Dion, F. 2000. Gendarme: Mesure de la taille fonctionnelle, ESI Software Inc., Montréal, Canada.

[9] Furey, S. and Kitchenham, B. 1997. Point (why we should use function points) and counterpoint (the problem with function points). *IEEE Soft.* 14 (2): 28–31.

[10] Harel, D. 1987. Statecharts: A visual formalism for complex systems. *Science of Computer Programming* 8 (3): 231–274.

[11] HierarchyMaster Pty Ltd. 1999. Overview – HierarchyMaster FFP.
`http://www.hmaster.com/FFP/overview.htm`.

[12] International Software Benchmarking Standards Group. 2002. Benchmark release 6. `http://www.isbsg.org.au`.

[13] Jeffery, D.R. and Low, G.C. 1993. A comparison of function point counting techniques. *IEEE Trans. Soft. Eng.* 19 (5): 529–532.

[14] Kemerer, C.F. 1993. Reliability of function points measurement: A field experiment. *CACM* 36 (2): 85–97.

[15] Kemerer, C.F. and Porter, B.S. 1992. Improving the reliability of function point measurement: An empirical study. *IEEE Trans. Soft. Eng.* 18 (11): 1011–1024.

[16] Mendes, O., Abran, A., and Bourque, P. 1996. Function point tool market survey. Software Engineering Management Research Laboratory, UQÀM, Montréal, Canada.

[17] Selic, B., Gullekson, G., and Ward, P.T. 1994. Real-Time Object-Oriented Modeling. New York: John Wiley & Sons.