A Case Study in COSMIC Functional Size Measurement: the Rice Cooker Revisited

Luigi Lavazza[1,2] and Vieri Del Bianco[3]

[1] University of Insubria,
Computer Science and Communication Department,
Via Mazzini, 5 – 21100 Varese (Italy)
[2] CEFRIEL,
Via Fucini, 2 – 20133 Milano (Italy)
[3] University College Dublin,
Systems Research Group, CASL
Dublin, Ireland

luigi.lavazza@uninsubria.it, vieri.delbianco@ucd.ie

Abstract. UML models have been successfully used to support COSMIC-based functional size measurement. UML-based measurement is of great interest for industry, because of the popularity of the language. However, industry needs well defined, easy to learn and apply methods. It is therefore necessary to provide measurement procedures that are well defined, that require relatively little effort and that are coherent with the COSMIC measurement rules, in order to ease their adoption in enterprise environments. This paper contributes to such goal: we show how to build UML models that are easy to measure according to the COSMIC rules; we provide a case study based on the well-known example of the rice cooker; we show how the usage of UML can actually improve the practice of COSMIC measurement, by making the COSMIC measurement rules applicable in a UML context.

Keywords: Functional Size Measurement, COSMIC method case study, measurement-oriented modelling, requirements modelling, UML.

1 Introduction

Functional Size Measurement (FSM) [2] is of great importance in industrial software development, since it provides the necessary input to effort estimation models and tools.

The COSMIC FSM method [4,5] is an emerging FSM technique that aims at overcoming several limitations of traditional FSM methods, like Function Point Analysis [3]. COSMIC FSM method is well defined form a theoretical point of view, but this is not enough to be widely and successfully used in industry: it needs methodological support. In particular, it is important that 1) the measurement method

fits nicely in the industrial development practices, and 2) it is provided with guidelines, including case studies.

Indeed, in order to improve the applicability of the COSMIC method in industry, we have considered one of the case studies published by the Common Software Measurement International Consortium (COSMIC) as a complement to the measurement manual [5], namely the Rice Cooker controller case study [6,7].

As a first step, we have modelled the Rice Cooker controller by means of UML [1]. This choice was motivated by the following observations:

- UML was already successfully used to model the software to be measured according to the COSMIC method (see for instance [9,10,11,12]).
- Since UML is widely used in industry, a COSMIC compliant measurement procedure that uses UML as supporting graphical language fits nicely in several industrial development processes. However, we have to show the minimum amount of information that UML diagrams have to provide in order to make a model measurable, and how this information is most suitably represented with UML diagrams.
- The COSMIC measurement process involves concept (like the triggering events, data groups, functional processes, etc.) that are not always easy to apply in practice. Establishing a mapping with the familiar UML concepts can make measuring easier for analysts.

The rice cooker controller has been chosen because:

- It is an embedded, real-time application. Therefore, the COSMIC method is the most suitable for measuring the functional size of the application [13,14].
- It is sufficiently small to be dealt with in a paper.
- It belongs to the 'official' set of case studies proposed by COSMIC.

The paper has two main purposes. On the one hand we want to assess the usage of UML for supporting COSMIC FSM, possibly establishing a mapping of *all* the COSMIC concepts onto UML elements, and defining a general and systematic measurement procedure based on UML models. On the other hand, we need to assess the measurement procedures employed in the measurement of the rice cooker controller [7], identify possible weaknesses and limits, and propose improvements (only in the measurement procedures, not in the definition of the metrics). To this end, an extension of the case study is proposed.

With respect to other proposals concerning the measurement of UML models according to the COSMIC method, our work is different under two main aspects:

- It aims at mapping *all* the COSMIC concepts onto UML elements, while other approaches do not address the mapping of some aspects (like triggering events, for instance) to corresponding UML elements.
- It proposes to use UML in a measurement-oriented way; i.e., we do not propose to apply the COSMIC method to general-purpose models; instead, we propose to build models that are specifically conceived to support COSMIC measurement as completely and seamlessly as possible. Hence, the construction of such models has to be done according to COSMIC model and rules, and using UML as the modelling notation.
- It strives to make the COSMIC method applicable in industrial practice as easily as possible.

The paper is organized as follows: Section 2 reports the requirements for the rice cooker controller (as given in [7]); Section 3 illustrates the UML models of the requirements. Section 4 illustrates the UML-based measurement. In Section 5 the proposed model and measurement procedure is compared with the official one, presented in [7] and [9]. Section 6 proposes an enhanced version of the rice cooker controller that improves the requirements and extends the responsibility of the software part of the system. Section 7 accounts for related work; finally Section 8 draws some conclusions.

2 The Rice Cooker Controller: Functional User Requirements

In [7] the specification of a first simple version of the Rice Cooker is given. The high-level specifications for the Rice Cooker controller, as reported in [7] are the following:

1. The Rice Cooker must be able to cook rice in three Modes: fast, normal and gruel. The user selects the mode via the selection buttons (see Fig. 1).
2. When the user pushes the START button, the Rice Cooker starts cooking and the cooking lamp (see Fig. 1) is lit. If the START button is pressed without the user selecting a Mode, the Rice Cooker automatically starts cooking in normal Mode.
3. The heater must be controlled according to the cooking specification, a function that determines the cooking temperature on the base of the mode and the elapsed time (see Fig. 2).
4. The time reference for the whole system will be provided by a specific function that will issue three signals, namely the elapsed time at one second intervals since the START button was pressed, and cycle signals at 5 and 30 second intervals.
5. Every 30 seconds, a new target temperature will be set.
6. Every 5 seconds, the actual temperature will be obtained from an external temperature sensor and compared with the target temperature. The heater will be switched ON if the actual temperature is lower than the target one, OFF otherwise.

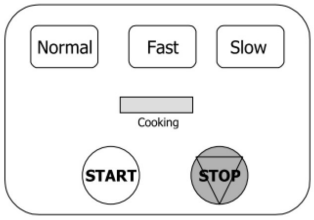

Fig. 1. The user interface of the rice cooker.

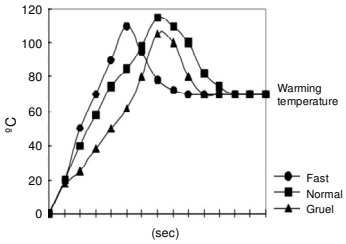

Fig. 2. The specification of cooking temperature and times depending on cooking mode.

In the first version of Rice Cooker, some functions are implemented through hardware devices and the remaining ones through software. In particular, the Functional User Requirements (FUR)

allocated to the Software are requirements 5 and 6 concerning the heater control and the control of the cooking lamp according to requirement 2, in particular, as soon as the Elapsed time signal is activated, the software sends a 'Turn on' command to the Cooking Lamp.

- The software controller receives data signals from the timer.
- The software controller receives data signals from the temperature sensor.
- The software controller reads the content of the Cooking Mode status switch.
- The software controller sends On-Off commands to the Cooking lamp and to the Heater.

The hardware controller's functions are as follows.

- The hardware controller manages the 'Start button', including starting a hardware Timer device that provides the sole time reference for the whole software system.
- The hardware Timer issues three signals to the software controller: the elapsed time at one second intervals since the START button was pressed, a 5 second cycle signal and a 30 second cycle signal.
- The cooking lamp is turned on by On commands from the software controller; actually there is no requirements specifying that the software controller should issue any Off command to the cooking lamp. The Off command is introduced in Section 6 in the specifications of the enhanced rice cooker.
- The Rice Cooker hardware has a ROM (Read Only Memory) which contains the cooking specifications (Fig. 2).
- The heater hardware is controlled by the software controller.
- The Cooking Mode switch status is stored in RAM, where it is available to the software controller. After the Start button is pushed, the hardware prevents the Cooking Mode status from changing.
- When the Stop button is pushed, the timer is stopped and reset, the Cooking Mode is reset to Normal. All other devices are switched off.
- Safety interactions between the Start and Stop buttons, the power supply and the Rice Cooker door are also controlled by hardware.

The requirements concerning the mode selection can be interpreted in different ways: either the state of the selector is accessed by means of memory-mapped I/O, i.e., the selector is just an external element that provides data upon request, or there is a full-fledged independent (hardware) process that actively stores the state of the selector in a given RAM location, so that the software controller just has to read the value from memory. The latter interpretation is adopted in the following models.

3 The UML Model of the Rice Cooker Controller

The first step in modelling the application to be measured consists in drawing the use case diagram, which is useful for several purposes:

- It shows the boundary of the application;
- It shows the functional user(s);
- It indicates the external elements with which the application exchanges data (i.e., the targets of exits and the sources of entries).

The use case diagram of the rice cooker controller is shown in Fig. 3.

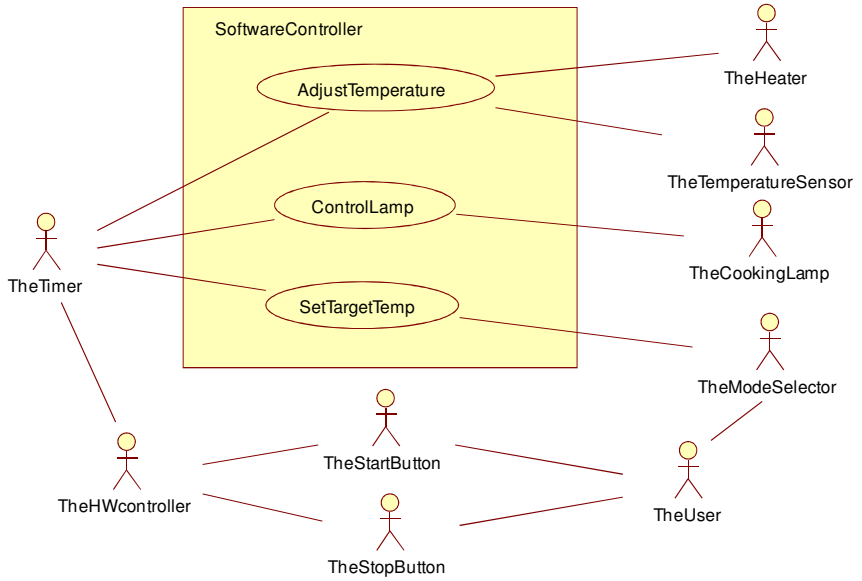

Fig. 3. The use case diagram of the rice cooker controller.

It is possible to see that the definition of the use case diagram in Fig. 3 matches the informal requirements reported in Section 2.

- Actor[1] "TheHeater" corresponds to the heater mentioned in requirements 3 and 6.
- Actor "TheTemperatureSensor" corresponds to the sensor mentioned in requirement 6.
- Actor "TheCookingLamp" corresponds to the lamp mentioned in requirement 2.
- Actor "TheTimer" corresponds to requirement 4. Actually, the requirement does not impose the existence of a timer, but this is the way time signals are usually modelled in COSMIC: the Measurement Manual [5] suggests that "clock-triggered events are considered external".
- Actor "TheModeSelector", corresponds to the selector mentioned in requirements 1 and 2. Actually, the selector does not participate directly in the use case "SetTargetTemp"; rather, it provides data that will be used by the target temperature setting function.
- Use case "AdjustTemperature" corresponds to the functionality described in requirements 3 and 6.
- Use case "SetTargetTemp" corresponds to the functionality described in requirement 5.
- Use case "ControlLamp" corresponds to the functionality described in requirement 2.

[1] Remember that in UML an actor, even though represented as a human, can be any device that interacts with the system.

The actors “TheUser”, “TheHWcontroller”, “TheStartButton” and “TheStopButton” are not strictly necessary for measuring the application, nonetheless they have been included, in order to take into account the presence of all the elements –including those managed by the HW– and make the use case more complete and representative of the requirements allocations.

As a second step in modelling the application to be measured it is useful to define a component diagram. It ‘refines’ the use case diagram by introducing the groups of data managed by the application, and making the interfaces between the application and the external elements explicit.

The component diagram of the rice cooker controller is shown in Fig. 4. Since several functions are allocated to the hardware, the component diagram in Fig. 4 – coherently with the use case in Fig. 3– shows also the hardware controller and the devices that are connected uniquely with it. Note that also the human user is modelled as a component, named “User”.

In accordance with the definition of UML, interfaces are provides by a component (connected with a broad-headed dotted arrow) and used by another component (connected with a narrow-headed dotted arrow).

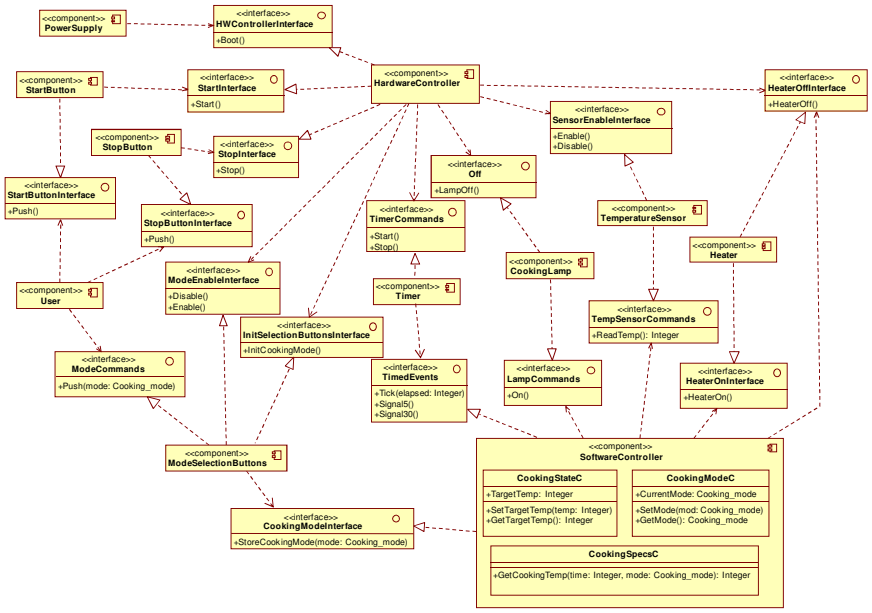

Fig. 4. The component diagram of the rice cooker.

Fig. 5 shows the portion of the component diagram centred on the software controller, i.e., the portion that is of greatest interest for the COSMIC measurer.

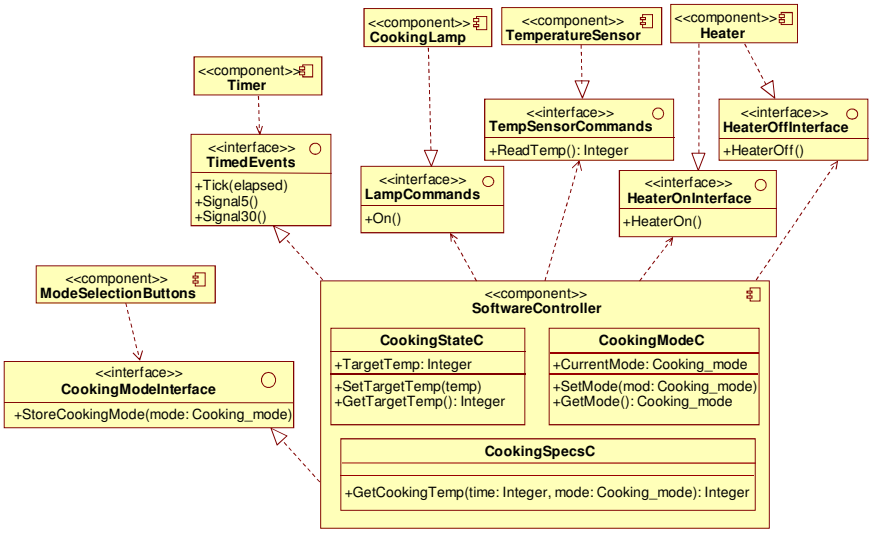

Fig. 5. The component diagram of the software controller of the rice cooker.

The sequence diagrams in Fig. 6 to Fig. 9 represent the processes that are allocated to the hardware controller.

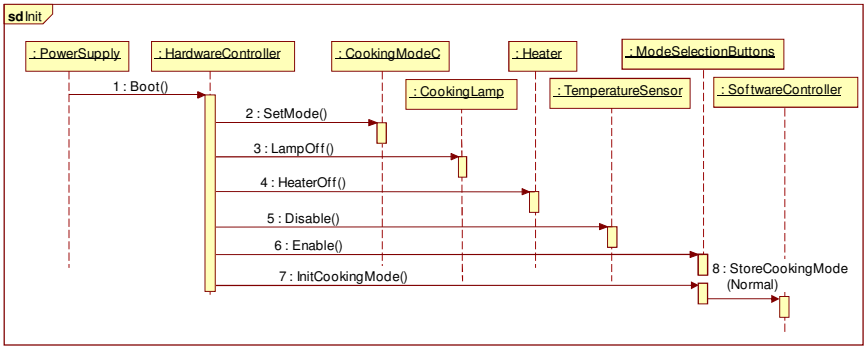

Fig. 6. The sequence diagram that represents the initialization process.

Note that it is the hardware that initializes the cooking mode to 'Normal'.

Fig. 7 illustrates the mode selection process: in particular, it shows that before the user pushes the Start button every mode selection action is recorded, while after the Start button has been pushed –being the mode selection disabled (Fig. 8)– pushing the mode selection buttons has no effect.

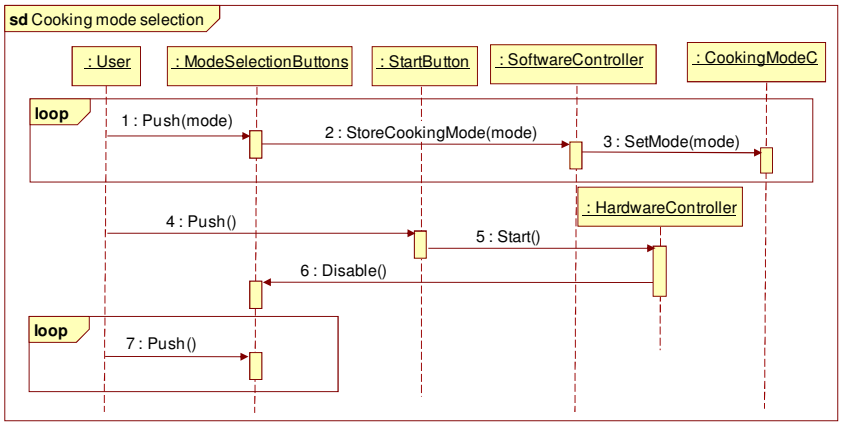

Fig. 7. The sequence diagram that represents the cooking mode selection process.

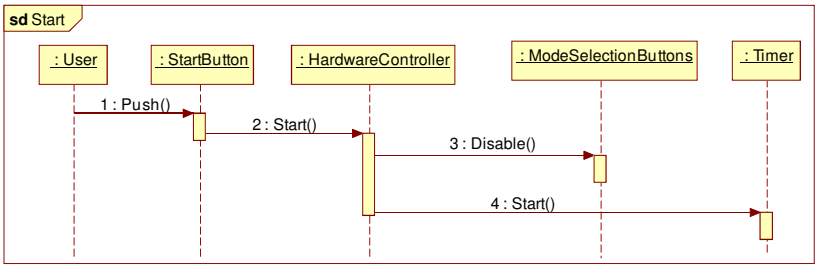

Fig. 8. The sequence diagram that represents the cooking start process.

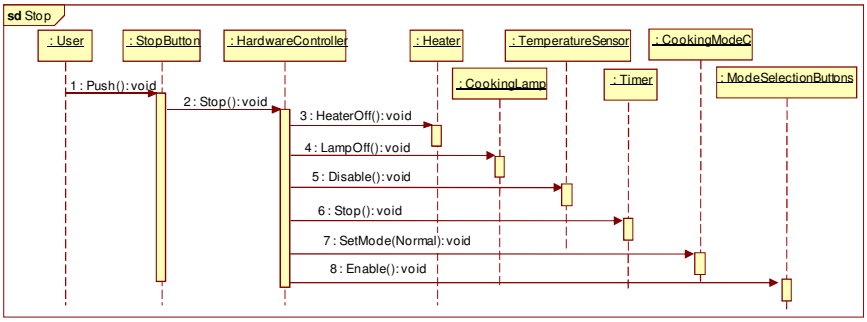

Fig. 9. The sequence diagram of the Stop process.

The sequence diagrams in Fig. 10 to Fig. 12 represent the processes that are allocated to the software controller. They are triggered from the signals issued by the timer: Tick (every second), Signal5 (every 5 seconds) and Signal30 (every 30 seconds). The timing of these signals has not been modelled, since it is not essential with respect to sizing. However, for the sake of clarity it could be modelled by means of state diagram, or via a sequence diagram decorated with duration constraints.

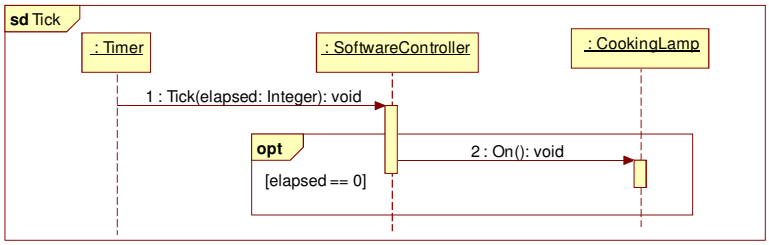

Fig. 10. The sequence diagram that represents the process that controls the cooking lamp.

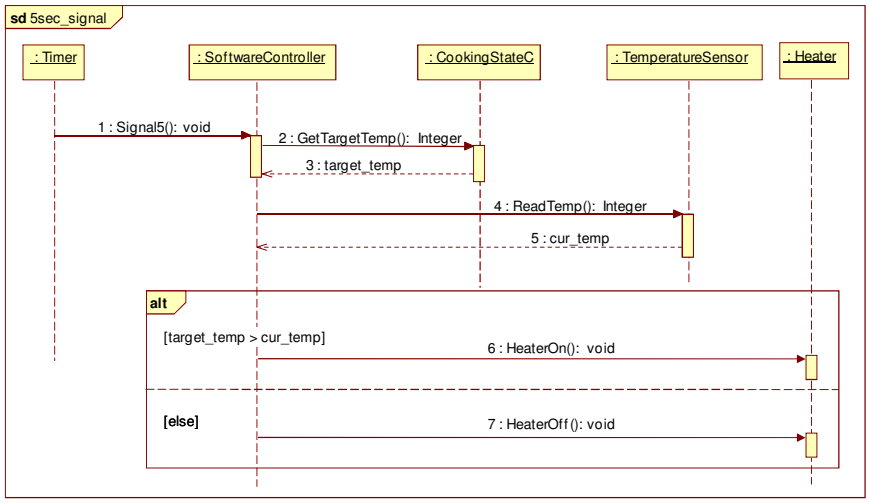

Fig. 11. The sequence diagram of the process invoked every 5 sec in order to control the heater.

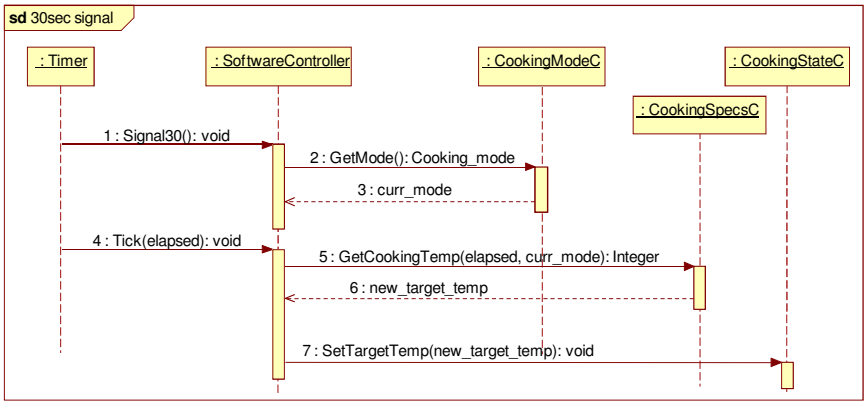

Fig. 12. The sequence diagram of the process invoked every 30 sec in order to set the target temperature.

4 Measurement

We start the measurement of the one-layer software controller with the identification of the boundary of the application according to the FUR.

Both the use case diagram and the component diagram show clearly that the person who operates the cooker (component User) is not a Functional User; similarly, the Start and Stop buttons are not Functional Users. In fact, none of these elements is connected directly with the software controller. The human user interacts with the software controller only indirectly, via the mode selector and the hardware controller (when he/she pushes the Start button).

The component diagram in Fig. 5 shows quite clearly that we have the following Functional Users:

- The Timer, which proactively sends data to the software;
- The Temperature sensor, which makes data available to the software;
- The Cooking lamp and the Heater, which receive control data (namely, On-Off commands) from the software.

The explicit representation of interfaces helps making the role of each functional user clear. In particular, it makes easy to represent which components are active (the timer), which are passive (the remaining ones), which send data (the timer and the sensor), and which receive data (the lamp and the heater).

Since there is a full-fledged independent (hardware) process that actively stores the state of the selector in a given RAM location, so that the software controller just has to read the value from memory, the selector is not considered a functional user. For further details, see the requirements assumptions described in Section 2.

The application boundary is explicitly represented in the component diagram (Fig. 5): it shows the elements that –according to the FUR– are outside the application: the lamp, the heater, the temperature sensor, and the timer. The timer is not mentioned explicitly by the FUR, but, according to the COSMIC Measurement Manual [5], "clock-triggered events are considered external", therefore we need an external element that produces the signals. Both the use case diagram and the component diagram describe the boundary in a very effective manner.

The other elements that play a relevant role in the requirements (namely, the Cooking Mode state in RAM and the cooking specifications in ROM) are also represented explicitly in the component diagram, as classes within the software controller component.

Candidate Triggering Events can also be identified on the basis of the information provided by the component diagram (Fig. 5); in fact, the interfaces provided by the system indicate the triggers that are available to external elements. The triggers are the operations that are available to active external components (in our case, just the timer). The following triggering events are identified:

- 5 sec. Clock Signal event (operation Signal5 of interface TimedEvents).
- 30 sec. Clock signal event (operation Signal30 of interface TimedEvents).
- Tick event every second (operation Tick of interface TimedEvents). The Tick carries the value of the elapsed time, according to requirement 4.

The following candidate functional processes are identified:

- Lit cooking lamp (triggered by the Tick event that occurs when the elapsed time is nil);
- Control heater (management of 5 sec. clock signals);
- Set target temperature (management of 30 sec. clock signals).

The candidate processes are identified by considering the triggering events and the use cases reported in the diagram (Fig. 3).

The examination of the sequence diagrams associated with the candidate processes (Fig. 10 to Fig. 12) shows that all the candidate processes satisfy the conditions to be classified as functional processes: each one operates on a unique and ordered set of data movements performing a set of FURs, and is triggered by an event that occurs outside the boundary of the software.

It is interesting to note that the UML model suggests the existence of other processes –namely Initialization, Cooking mode selection, Start, Stop cooking– that are needed for the correct operation of the rice cooker. However, these processes are implemented in hardware, therefore they are not counted among the functional processes.

With the help of the component diagram, the following data groups are identified:

- Class CookingModeC: it stores the current cooking mode.
- Class CookingStateC: it stores the current target temperature.
- Class CookingSpecsC: it contains the description of the cooking temperature as a function of time and cooking mode (Fig. 2).

The component diagram is useful also to identify transient data groups, which do not correspond to classes, but to data that cross the boundaries of the system. Such data correspond to operations of the interfaces, or to parameters of these operations:

- The actual temperature read from the external TemperatureSensor via the ReadTemp operation.
- The commands for the heater: operations HeaterOn and HeaterOff.
- The commands for the lamps: operation On.

In addition, every triggering event (Elapsed Time, 5 sec. and 30 sec. Timer Signals) is also a transient data group.

Finally, we have to count the data movements of each process. This task is made quite simple by the availability of the sequence diagrams: we just have to observe the messages that involve the system (i.e., that are received or sent by the system). In particular:

- messages sent by external components to the system are entries;
- messages sent by the system to external components are exits;
- messages sent by the system to internal components and that obtain persistent data are reads, while messages that send data to internal components in order to make such data persistent are writes.

Table 1 illustrates for each process the messages sent or received by the application, and the corresponding data movements.

Table 1. Detail of functional processes and the involved data movements.

Process	Message sending		Data movement		CFP
	Message	Component or object involved	Data group	Type	
Tick (control lamp)	Tick (elapsed)	from Timer	TimedEvents	Entry	2
	On	to CookingLamp	CookingLamp	Exit	
5 sec. signal management (control heater)	Signal5	from Timer	TimedEvents	Entry	4
	GetTargetTemp	to CookingState	CookingState	Read	
	ReadTemp	to TemperatureSensor	TemperatureSensor	Entry	
	HeaterOn or HeaterOff	to Heater	Heater command	Exit	
30 sec. signal management (set target temperature)	Signal30	from Timer	TimedEvents	Entry	5
	GetMode	to CookingModeC	CookingMode	Read	
	Tick(elapsed)	from Timer	TimedEvents	Entry	
	GetCookingTemp	to CookingSpecs	CookingSpecs	Read	
	SetTargetTemp	to CookingState	CookingState	Write	
Total					**11**

It is possible to see that in most cases the involved data group is determined immediately by the element involved in the message passing; similarly the fact that the element is internal (a class of the SoftwareController component) or external (a component that interacts with the SoftwareController component) and the direction of the message (to/from the SoftwareController component) are usually enough to determine the type of data movement.

There is an important exception: it is often the case that an event is sent to actually retrieve some information, in this case we have a 'reversed' direction. In the model we can see this particular behaviour when reading Temperature from the TemperatureSensor: a message is sent to TemperatureSensor (ReadTemp) (an outgoing message), the TemperatureSensor sends back the temperature reading (an Entry).

5 Discussion

The system presented in Section 2 (requirements) and Section 3 (actual model) was derived from the requirements given in [7]; it is therefore interesting to compare the measurement procedure applied in Section 4 with the procedure followed in [7].

- A first observation is that in general UML guides to writing more precise requirements. For instance, in [7] FUR3 states that *as soon as the Elapsed time signal is activated, the software sends a 'Turn on' command to the Cooking Lamp*. When specifying the requirements in UML, it appears clearly that the requirement is impossible to achieve; it must be rephrased as follows: *the first time the Elapsed time signal is received, the software sends a 'Turn on' command to the Cooking Lamp*. In fact, the software cannot react to the activation event, which is performed in hardware and is hidden to the software controller.

- UML modelling highlights that a critical point in the requirements is the management of the time signal that are issued when the elapsed time (in seconds) is a multiple of 30. In these cases, all the three signals (Elapsed Time, 5 sec. and 30 sec. Timer Signals) are issued. The requirements do not say explicitly how this situation should be handled.
- UML helps noticing that when the controller enters the warming mode, no specific event is generated. Therefore, the user is not notified that the cooking is finished. This is a rather strange behaviour of the user interface. Actually, a warming lamp (as in [6]) should be included.
- According to [7] the temperature sensor is a functional user, while the cooking mode selector is not. This classification appears a bit unbalanced, since both devices make data available to the software controller. The fact that one makes the data available in RAM, while the other requires a proper I/O operation does not seem sufficient to justify the difference. Also the fact that no triggering event is associated with the temperature sensor casts some doubts on the classification of the sensor as a functional user.
- As far as the target temperature is concerned, the need to explicitly represent data in the component diagram (Fig. 5) helps realizing that the system involves two notions of target temperature: one is given by the function in Fig. 2 on the basis of elapsed time and cooking mode (method CookingSpecsC:: GetCookingTemp(...) in Fig. 5), the second one is the variable which is updated every 30 seconds and used every 5 seconds (attribute CookingStateC::TargetTemp in Fig. 5). In [7] these concepts tend to be confused; in fact, there is no data group corresponding to the target temperature variable which is updated every 30 seconds. This kind of issue does not cause problems in a small application like the rice cooker, but could cause waste of time or even measurement errors when dealing with a more complex system.
- UML helps identifying critical situations in processes. Consider for instance the "Select Target Temperature" as specified in [7] (see Table 1 on page 13) and modelled by the sequence diagram in Fig. 12. The process manages a signal (30 sec. Timer Signal) but during the management stops to wait another signal (the Elapsed Time). This is hardly a good programming practice: the real process would probably be organized differently, and –accordingly– could have a different size.

The usage of UML for modelling the software to be measured by means of the COSMIC method was also advocated in [9]. Unfortunately [9] makes reference to the older specifications of the rice cooker controller [6], which were rather ambiguous (for instance, they did not specify which requirements where allocation to hardware and which to software). Therefore, a punctual comparison with the modelling and measurement described here is not possible. However, it is still possible to perform a comparison of the two approaches at a rather high level.

Actually, [9] proposes the same usage of use case diagram and sequence diagrams that we advocate. However, the intermediate part of the measurement procedure (which involves identifying functional users, triggering events, data groups, etc.) is not supported by UML in [9]. For big complex applications such lack of support could create problems and void the benefits of using UML for the initial and final step of the modelling and measurement process. In fact, writing sequence diagrams is

easier if you have already identified classes that represent data objects and methods that correspond to data movements. In our procedure the component diagram provides such information, while in [9] the sequence diagrams have to be derived from the requirements expressed textually.

Table 2 summarizes the mapping between COSMIC concepts and UML elements that we propose.

Table 2. COSMIC – UML mapping.

COSMIC concept	UML diagram	UML element
Application border	Use case	Boundary of the subject
	Component	Boundary of the system component
Functional User	Use case	Agent directly connected with a use case
	Component	External component directly connected with the system
Triggering event	Component	Operation in interface realized by the system and invoked spontaneously by an active external component
Persistent Data group	Component, class	Class
Transient Data group	Component	Data that cross the boundaries of the system: operations of the interfaces, or to parameters of these operations
Process	Use case	Use case
	Sequence	Sequence (seq) interaction
Entry Data Movement	Sequence	Message from external component to the system
Exit Data Movement	Sequence	Message from the system to external component
Read/Write Data Movement	Sequence	Message involving persistent data from system to instance of class within the system

Note that in general UML elements indicate *potential* COSMIC elements. The usual COSMIC counting rules have to be applied; for instance, UML indicates the cooking mode selector as a potential functional user: considering it as an actual functional user or not depends on the application of COSMIC rules.

Finally, it is to be noted that one of the properties of a COSMIC functional process is uniqueness: that is, the process should "operate on an ordered and unique set of data movements performing a set of FURs". It is hard to satisfy this requirement before identifying the data movements involved in the process. This can lead to serious problems if a straight waterfall approach is used, but with the iterative, incremental approach –which is typical of object-oriented modelling– all the dependencies between functional process identification and data movement identification can be resolved smoothly.

6 An Enhanced Rice Cooker Controller

The observations reported in the previous sections suggest that we can redefine the requirements of the rice cooker controller by allocating all the control functionality to the software, improving the usability of the cooker by adding a warming lamp, and rationalizing the management of the signals from the timer.

The warming lamp has to be lit when the cooker moves from the cooking phase to the warming phase; correspondingly the cooking lamp is switched off.

Instead of three distinct types of signal, the timer issues just a Tick signal every second: this is easy to handle, and is sufficient to let the software compute the elapsed time and decide when it is time to update the target temperature or to control the heater.

Moreover, the access to the state of the Cooking Mode Button is defined as a memory-mapped I/O operation.

The use case diagram of the new rice cooker controller is shown in Fig. 13: it is possible to see that it accounts for the functions that have been moved from the hardware to the software controller. It should be noted that we have only four functional processes, while there are several use cases in Fig. 13. This is due to the fact that multiple use cases can be triggered by a single event: in fact the Timer's Tick triggers a process that involves setting the new target temperature, adjusting the actual temperature (by turning the heater on or off) and controlling the lamps. Therefore, the identification of functional processes based on use cases should take into account that several functionalities can be included in a single use case (which represents the interaction with the user).

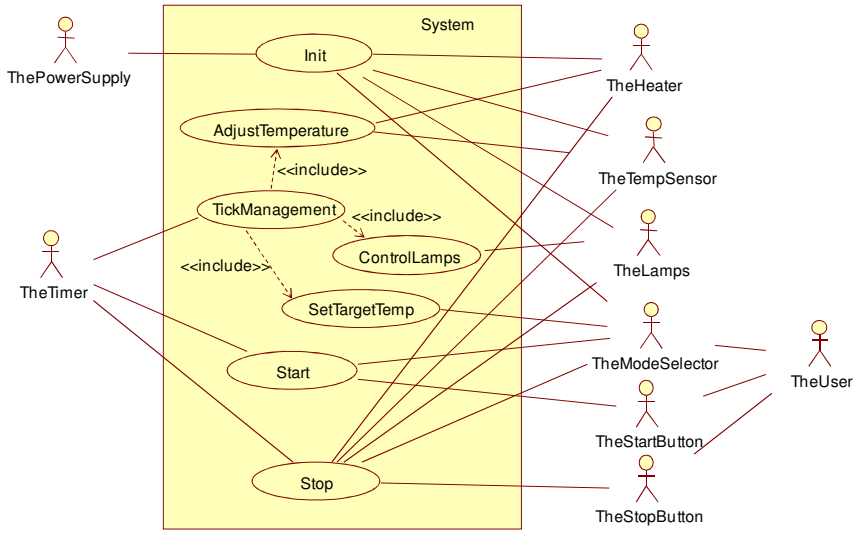

Fig. 13. The use case diagram of the enhanced rice cooker.

The component diagram of the enhanced rice cooker is reported in Fig. 14, while the sequence diagram corresponding to the functional processes are given in Fig. 15, Fig. 16 and Fig. 17.

Note that these is no 'Mode selection process': according to the new specifications the mode selection is confined in the model selection hardware, which is seen by the software controller as a plain peripheral that is able to provide some data.

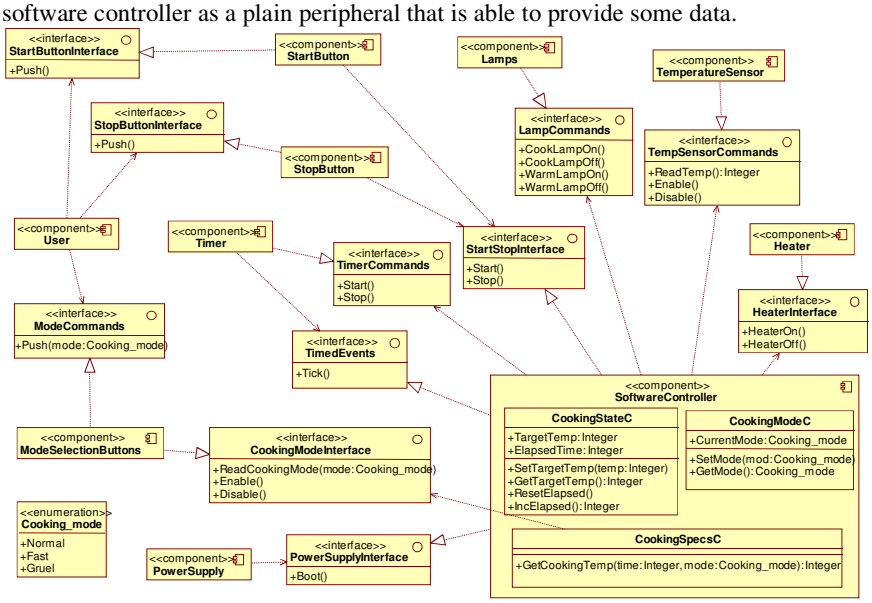

Fig. 14. The component diagram of the enhanced rice cooker.

The sequence diagram describing the initialization process is not reported: in fact, it is identical to the one illustrated in Fig. 6, except that the software controller is involved instead of the hardware controller.

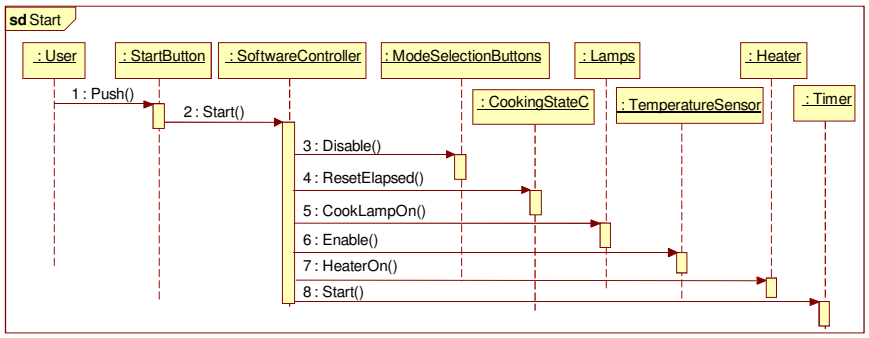

Fig. 15. The sequence diagram representing the start cooking process.

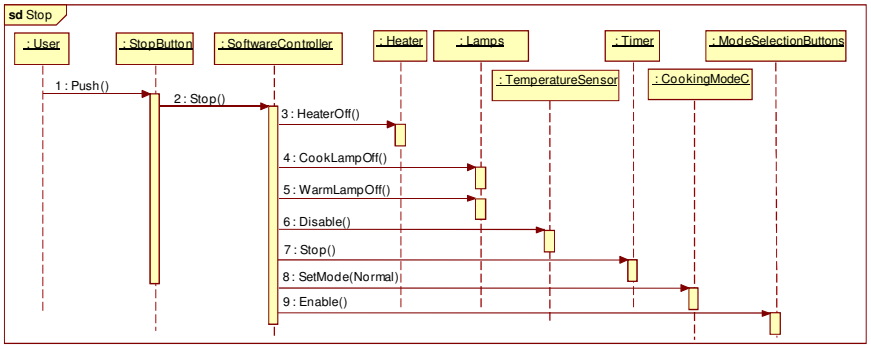

Fig. 16. The sequence diagram representing the stop cooking process.

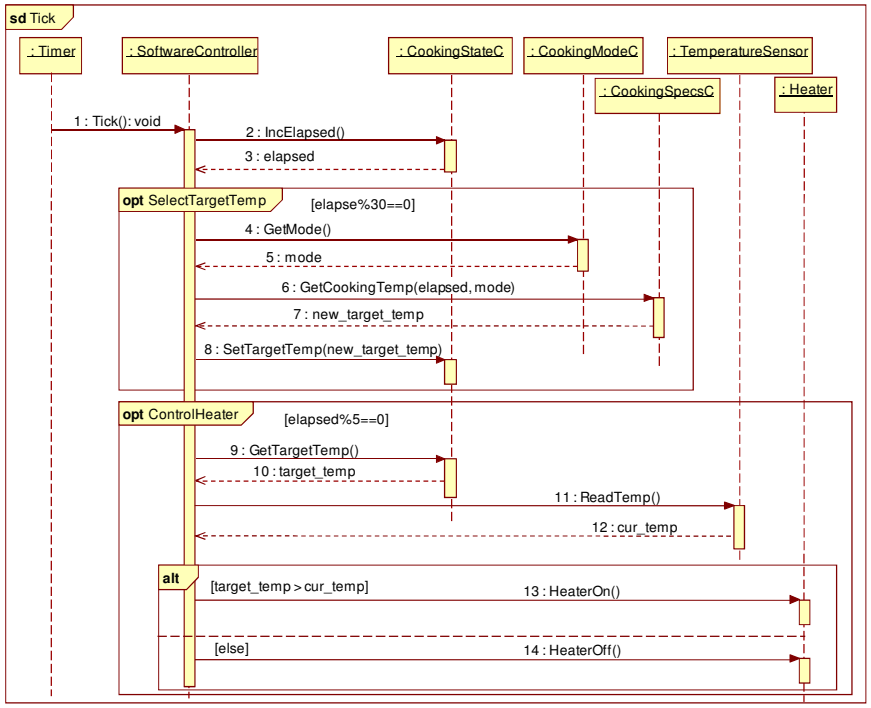

Fig. 17. The sequence diagram representing the management of ticks, including the selection of target temperature and the control of the heater.

The measurement of the model of the enhanced rice cooker is straightforward. By applying the same considerations as in Section 4 we get the results summarized in Table 3.

Table 3. Identification of COSMIC elements in the model of the enhanced rice cooker.

COSMIC element	Model element
Functional User	StartButton, StopButton, Timer, TemperatureSensor, Lamps, Heater, ModeSelectionButtons, PowerSupply
Triggering Events	Boot, Start, Stop, Tick
Functional Processes	Init, Start, Stop, Tick management
Data Groups	Classes CookingModeC, CookingStateC, CookingSpecsC
Transient Data Groups	TemperatureSensor, Heater commands, Lamps commands, Timer, ModeSelectionButtons, PowerSupply, StartButton, StopButton

Table 4 illustrates for each process the messages sent or received by the application, and the corresponding data movements.

Table 4. Detail of functional processes and the involved data movements.

Process	Message sending		Data movement		CFP
	Message	Component or object involved	Data group	Type	
Init	Boot	From Power supply	Boot	Entry	6
	SetMode	To CookingModeC	CookingModeC	Write	
	CookLampOff, WarmLampOff	To Lamps	Lamps	Exit	
	HeaterOff	To Heater	Heater	Exit	
	Disable	To TemperatureSensor	TemperatureSensor	Exit	
	Enable	To ModeSelection Buttons	ModeSelectionButtons	Exit	
Start	Start	From StartButton	StartButton	Entry	6
	Disable	To ModeSelection Button	ModeSelectionButtons	Exit	
	ResetElapsed	To CookingStateC	CookingStateC	Write	
	CookLampOn	To Lamps	Lamps	Exit	
	HeaterOn	To Heater	Heater	Exit	
	Start	To Timer	Timer	Exit	
Tick	Tick	From Timer	Timer	Entry	7
	IncElapsed, SetTargetTemp	To CookingStateC	CookingStateC	Write	
	GetMode	To CookingModeC	CookingModeC	Read	
	GetCookingTemp	To CookingSpecsC	CookingSpecsC	Read	
	GetTargetTemp	To CookingStateC	CookingStateC	Read	
	ReadTemp	To TemperatureSensor	TemperatureSensor	Entry	
	HeaterOn, HeaterOff	To Heater	Heater	Exit	
Stop	Stop	From StopButton	StopButton	Entry	7
	HeatOff	To Heater	Heater	Exit	
	CookLampOff, WarmLampOff	To Lamps	Lamps	Exit	
	Disable	To TemperatureSensor	TemperatureSensor	Exit	
	Stop	To Timer	Timer	Exit	
	SetMode	To CookingModeC	CookingModeC	Write	
	Enable	To ModeSelection Buttons	To ModeSelectionButtons	Exit	
Total					**26**

Actually, IncElapsed is both write and read operation; however, since the CookingStateC data group is already read by GetTargetTemp this makes no difference in this case. In general, however, it would be better to specify operations that either read or write a given data group: this makes easier to identify data movements.

7 Related work

Several approaches were proposed to measure FUR documented via UML diagrams. A survey of such approaches was published by Marín et al. [10]. One of the

first among such techniques is due to Bévo et al. [11]. They map COSMIC concepts on a few UML diagrams: use cases, sequence diagrams, and classes. Each use case corresponds to a functional process. The data movements are represented in the scenarios, which are sequences of interactions that occur within a use case. Each class corresponds to a data group. The triggering events are not represented with UML concepts. A tool named Metric Xpert supports the automatic application of the measurement procedure. The experimental application of the tool showed that it is able to produce measures that differ between 11% and 33% from measures obtained by experts.

van den Berg et al. [8] study the demands that FSM methods (both FPA and COSMIC) pose to FUR expressed in UML. They conclude that class diagrams are needed to represent data structure and used attributes, use case diagrams are needed to represent the actors and system boundary, finally a behavioural diagram is considered useful to represent the flow of events and to locate functional transactions and functional processes in use cases. For this purpose they use activity diagrams, although other choices are feasible.

The correspondence of a set of COSMIC concepts (boundary, user, functional process, data movement, data group and data attribute) with UML counterparts has been established in [15].

UML State diagrams and COSMIC are used together in [16]: the State diagrams are synthesized from UML sequence diagrams and dependency diagrams (the latter focus on dependency relations between scenarios; they are not UML standard diagrams). A mapping of concepts found in UML State diagrams and COSMIC method has been built, based on the synthesis algorithm. The State diagrams are necessary for assessing the reliability of the system using Markov processes; they are not used for modelling or sizing the system.

8 Conclusions

We have shown that UML can be conveniently used to build models that are relatively easy to measure according to the COSMIC rules. Establishing a mapping between *all* COSMIC concepts and UML constructs allows the modeller to build measurement-oriented models, and the measurer to apply the COSMIC counting rules in a fairly straightforward way.

By applying the proposed modelling and mapping technique to a simple but realistic example we hope to contribute convincing industry that COSMIC measurement can fit quite well in the software development process.

However, in order to favour the adoption of the COSMIC method by industry and enterprise environments, we believe that case studies, examples, etc. should be equipped with explicit models (not necessarily in UML) to ease both the understanding by readers and the verification of the completeness and correctness of the specifications and corresponding measurement.

Future work includes the application of the proposed measurement-oriented modelling techniques to larger (hopefully real) applications, and the experimental evaluation of the sensitivity of the obtained measures with respect to the

analyst/modeller. In fact, it is possible that different analysts produce different models and possibly measures for the same set of original requirements. The authors have already carried out some experimental validation of measurement-oriented modelling for function point analysis [17], which showed that the measures are fairly independent on the analysts (in any case, less sensitive than with the "traditional" measurement procedures). Such experimentation will be repeated for COSMIC functional measurement.

Another topic for further research concerns the sizing of hardware functionality. In fact, we showed that the functionality allocated to hardware can be modelled just like the functionality allocated to software. Therefore, in principle the same COSMIC concepts that apply to software sizing should be applicable to hardware sizing as well. We shall thus explore to what extent it is easy to size hardware functionality in a way that is compatible with the COSMIC method and provides measures that are compatible with the software size expressed in CFP.

Acknowledgments

The research presented in this paper has been partially funded by the IST project QualiPSo – Quality Platform for Open Source Software (www.qualipso.org), sponsored by the EU in the 6th Framework Program (IST-034763), and by the project "Elementi metodologici per la descrizione e lo sviluppo di sistemi software basati su modelli", funded by the Università degli Studi dell'Insubria.

9 References

1. Object Management Group, Unified Modeling Language: Superstructure version 2.1.1 formal/2007-02-05, February 2007.
2. ISO/IEC 14143-1:1998. Information technology -- Software measurement -- Functional size measurement -- Part 1: Definition of concepts, International Organization for Standardization, Geneva.
3. A.J. Albrecht, "Measuring Application Development Productivity", Proc. Joint SHARE/GUIDE/IBM Application Development Symp., pp. 83-92, 1979.
4. ISO/IEC19761:2003, "Software Engineering -- COSMIC-FFP – A Functional Size Measurement Method", ISO, 2003.
5. COSMIC – Common Software Measurement International Consortium, The COSMIC Functional Size Measurement Method - version 3.0 Measurement Manual (The COSMIC Implementation Guide for ISO/IEC 19761: 2003), September 2007.
6. COSMIC – Common Software Measurement International Consortium, Case study – Rice Cooker, version 2.1, January 14, 2000.
7. COSMIC Group, Case Study: Rice Cooker, Version May 22, 2008
8. K. van den Berg, T. Dekkers, and R. Oudshoorn, "Functional size measurement applied to UML-based user requirements", 2nd Software Measurement European Forum (SMEF2005), 16-18 March 2005, Rome.
9. G. Levesque, V. Bevo, D. T. Cao, "Estimating Software Size with UML Models". C3S2E Conference, Montreal, 2008.

10. B. Marín, G. Giachetti and O. Pastor, “Measurement of Functional Size in Conceptual Models: A Survey of Measurement Procedures based on COSMIC”, IWSM/Metrikon/Mensura 2008, Munich, November 2008.
11. V. Bévo, G. Lévesque, and A. Abran, “Application de la méthode FFP à partir d'une spécification selon la notation UML: compte rendu des premiers essais d'application et questions”. 9th International Workshop Software Measurement, Lac Supérieur, Canada, 1999.
12. V. del Bianco and L. Lavazza, “Applying the COSMIC Functional Size Measurement to Problem Frames”, 14th IEEE International Conference on Engineering of Complex Computer Systems ICECCS ’09, Potsdam (Germany), June 2-4 2009.
13. J.M. Desharnais, A. Abran, and D. St-Pierre, “Functional Size of Real-Time Software,” 11th International Conference - Software Engineering and its Applications, Paris, France, 1998.
14. D. St-Pierre, A. Abran, M. Araki, and J.-M. Desharnais, Adapting Function Points to Real-Time Software, IFPUG 1997 Fall Conference, Scottsdale, AZ, September 15-19, 1997.
15. M. Jenner, “Automation of Counting of Functional Size Using COSMIC-FFP in UML”, *12th International Workshop on Software Measurement (IWSM 2002)*, Magdeburg (Germany), October 2002.
16. M. Abu-Talib, A. Abran, and O. Ormandjieva, “Markov Model and Functional Size with COSMIC-FFP”, *IEEE International Symposium on Industrial Electronics (IEEE-ISIE3006)*, vol.4, Montreal (Canada), July 2006.
17. V. del Bianco, C. Gentile, and L. Lavazza, “An Evaluation of Function Point Counting Based on Measurement-Oriented Models”, *Evaluation and Assessment in Software Engineering – EASE 2008*, Bari (Italy), June 2008.