AS-TRM AND FUNCTIONAL SIZE WITH COSMIC-FFP

Manar Abu Talib
Zayed University
manar.talib@zu.ac.ae

Olga Ormandjieva
Concordia University
ormandj@cse.concordia.ca

Alain Abran
École de Technologie Supérieure
alain.abran@etsmtl.ca

Abstract: **COSMIC-FFP – ISO 19761 – is a functional size measurement method developed by the Common Software Measurement International Consortium (COSMIC). The COSMIC-FFP measurement model and related definitions are generic, and this paper investigates the feasibility of their application in specification languages. More specifically, it proposes a formalization of the COSMIC-FFP definition for the Autonomic Systems Timed Reactive Object Model (AS-TRM). This would allow the integration of functional complexity and functional size monitoring during autonomic system specification construction and/or evolution. The Steam Boiler case study is introduced to demonstrate the applicability of functional size measurement in terms of AS-TRM modeling.**

Keywords: **Autonomic Reactive Systems, Functional Size, COSMIC-FFP, ISO 19761**

I. INTRODUCTION

Software measurement is one of the key technologies for controlling and managing the software development process. Measurement is also the foundation of both the sciences and engineering, and much more research on software is needed to ensure that software engineering be recognized as a true engineering discipline.

Fenton and Pfleeger [1] have defined software measurement as the process of quantifying the attributes of software in order to characterize them according to clearly defined rules. For instance, it can be used to identify an anomaly within the development phase in which it originated, as well as to measure development progress. Thus, each software development phase should contain measures to enable high project visibility and quality control to be achieved [2].

The term "size" in software measurement can mean either project size or software size. The first refers to total effort, estimated or actual, and is expressed in units like work-hours or staff-months. The latter refers to the size of the requirements (functions) or the deliverables, and is expressed in units such as modules or lines of code.

Fenton and Pfleeger [1] have identified three attributes of software size, which are length, complexity and functionality.

Length refers to the physical size of the product. It is meaningful to the technical staff, and useful for measuring specifications, designs and codes. The lengths of the specifications may help in predicting the length of the design, and the length of the design may help in predicting the length of the code. For example, software size can be measured by counting the number of lines of code (LOC), a useful measurement in deciding how big a file needs to be to store code. However, it does not tell us anything about the software's functionality or about the quality of the coding.

Complexity is an essential characteristic of the software process/product, and is a multifaceted notion which is context-dependent [1], [3], [4], [5], [6]. As with the Whitmire [3] classification, the complexity of a software system is viewed in different dimensions, in this case four: computational, representational, structural and functional. Computational complexity is often associated with the study of algorithmic efficiency. Representational complexity considers the tradeoffs between graphical and textual notations for unambiguous representations of the system model, system interactions and system behavior. Structural complexity is viewed in terms of coupling and cohesion, without considering the complexity of the individual components. Functional complexity characterizes the dynamic performance of the system seen as a sequence of events required to fulfill a certain functionality of the system.

Functionality, as discussed in [1], embodies the functions supplied by a product. Many software engineers argue that code length is a misleading indicator, and there is little consensus on how to measure complexity. By contrast, functionality is a meaningful way to measure software product size as far as users and management are concerned, It must be independent of the effort, the method and the technology. There have been several approaches to measuring the functionality of software products, among them:

- Albrecht's function points [7].
- DeMarco's specification weight [8].
- COSMIC-FFP [9].

All three approaches measure the functional size of software specification documents, and they can all be applied later in the life cycle. Functional size measurement (FSM) has been used mostly for productivity, benchmarking and estimation purposes, and it can be used very early in the software development life cycle, such as when measuring the functional user requirements which are known prior to the design, architecture, code and test phases. There are some hints in the literature and in practice that, in addition to measuring productivity, and in benchmarking and effort estimation, FSM could also be used for quality purposes. In this paper, we are interested in the innovative use of the FSM method developed by the Common Software Measurement International Consortium (COSMIC), referred to as COSMIC-FFP, to contribute to early assessment of software complexity and quality. The COSMIC-FFP size measurement model and

related definitions are generic, and we provide here an initial investigation of the feasibility of their applicability in specification methods. More precisely, we look into the initial step in the formal definition of the COSMIC-FFP method in the specific context of the Autonomic Systems Timed Reactive Model (AS-TRM) specifications.

There are currently no adequate frameworks for measuring size in autonomic systems with self-monitoring functionality. Defining such a framework could allow integration of the FSM derived during the specification construction phase into the formal AS-TRM, and consequently its automatic self-monitoring during the evolution of the autonomic system. Our aim is to assess the size of an evolving AS-TRM system before a change is authorized, in order to monitor the system's complexity and quality.

Integrating the COSMIC-FFP size measurement model into the AS-TRM would ensure automatic and consistent feedback on the functional size of the autonomic system. The Steam Boiler case study is introduced next to illustrate the applicability of the FSM in terms of AS-TRM modeling in the specification phase.

The paper is organized as follows: Section II presents the key elements of the AS-TRM, and section III the key elements of COSMIC-FFP. Section IV discusses the mapping of concepts across the two fields. In section V, the Steam Boiler case study is introduced to illustrate the approach. Finally, section VI identifies research directions for the formalization of functional complexity and functional size during specification construction.

II. AS-TRM

Here, we introduce the AS-TRM architecture for modeling reactive autonomic distributed systems [17].

Reactive systems are computer systems that continuously react to their physical environment, continually sensing and responding to it, and at a speed determined by it. The behavior of these systems is infinite, and they must always be able to respond to a stimulus from that environment. Moreover, the time lapse between a stimulus and its response must be acceptable to the relative dynamics of the environment so that the environment is still receptive to the response.

The AS-TRM architecture builds on the TROM formalism [10] for modeling reactive systems by adding more tiers and including the following specifications (see Figure 1):

- Data modeling
- Timed Reactive Object Model (TROM)
- Timed reactive autonomic component (AC);
- Group of synchronously interacting ACs (ACG);
- Autonomic system (AS), consisting of asynchronously communicating ACGs.

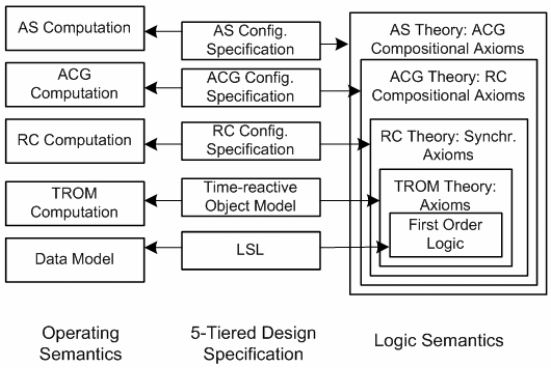

Figure 1: AS-TRM Formal Model [17]

The lowest tier is the Larch Shared Language (LSL) tier, which specifies the Abstract Data Types (ADTs) used in the reactive classes [10]. The TROM tier specifies the reactive classes, called Generic Reactive Classes (GRCs) [10]. A GRC is a hierarchical finite state machine augmented with ports, attributes, logical assertions on the attributes and time constraints.

The AC tier encapsulates TROM objects into an AS-TRM AC. The synchronous interaction between the ACs allows a reactive task to be realized. The communication between an AC and its upper tier ACG is realized through an interface, and is asynchronous.

The ACG is a set of synchronously communicating ACs cooperating in fulfillment of a group task. Each ACG can independently accomplish a complete real-time reactive task. The self-monitoring behavior in the ACG tier and the asynchronous interaction between an ACG and the ACs is realized by an ACG Manager (AGM). The responsibilities of an AGM include the continuous monitoring of the ACG quality level required by the evolving nature of the peer group.

The AS tier abstracts a set of asynchronously communicating ACGs. The self-monitoring behavior and the asynchronous interaction between AS and the ACGs is realized by the Global Manager (GM). The GM consists of a set of intelligent agents responsible for self-configuration, self-healing and self-optimization, and self-protection of the ACGs. The GM's responsibilities include the continuous monitoring of the AS quality level required to endure the quality requirements of the autonomic system.

Every ACG communicates its status and measurements to the GM. According to the input received from the ACGs, the GM makes decisions based on policies, facts and rules (stored in the AS repository) and communicates instructions to the corresponding AGMs.

Figure 2 depicts the hierarchical structure of the communicating entities of the AS-TRM.

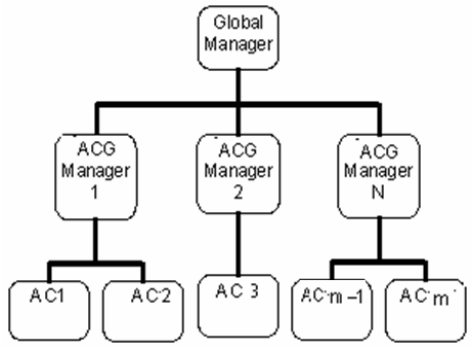

Figure 2: Hierarchical view of the AS-TRM [10]

The AS-TRM *evolves*: the composition of the ACs in the peer group can be changed in terms of run time and functionality, and synchronization axioms among ACs in the peer group can be changed during run time. The FSM and quality self-assessments at the AS level would allow the reconfigured system to be deployed by the GM if and only if the maximum functional size level is not reached and/or the quality requirements are met.

The process of functional size self-assessment is based on the concept of the Intelligent Control Loop [11][12][13] and consists of the following steps:

(i) **Monitor:** continuously track changes within the AS-TRM, either from the environment or from the GM, such as changes in configuration, functional requirements or quality requirements.

(ii) **Analyze:** assess functional size and quality requirements according to the changes collected at the monitoring stage, and make decisions as to whether or not those changes are acceptable.

(iii) **Plan:** decide on how the changes will be adapted, such as setting the appropriate timing and checkpoints to apply the changes, and how the ACG status will be restored if the execution fails.

(iv) **Execute:** apply the changes that were accepted at the Analyze stage, and follow the plans made at the Plan stage. The successful changes will be stored in the group repository, and the assessment, along with execution result, will be reported to the GM and stored in the AS repository.

The FSM method COSMIC-FFP has been chosen to monitor the size of an evolving AS-TRM system because of its objectiveness and solid theoretical foundation. In the following section, the COSMIC-FFP method is introduced.

III. COSMIC-FFP MEASUREMENT METHOD

A. COSMIC-FFP OVERVIEW

COSMIC-FFP, the FSM method developed by COSMIC [9], has now been adopted as an international standard (ISO 19761 [14]). It was designed to address some of the weaknesses of earlier methods like FPA [15], the design of which dates back almost 30 years to a time when software was much smaller and much less varied.

In the measurement of software functional size using the COSMIC-FFP method, the software functional processes and their triggering events must be identified. In COSMIC-FFP, the unit of measurement is the data movement, which is a base functional component that moves one or more data attributes belonging to a single data group. Data movements can be of four types: Entry, Exit, Read or Write. The functional process is an elementary component of a set of user requirements triggered by one or more triggering events, either directly or indirectly, via an actor. The triggering event is an event occurring outside the boundary of the measured software and initiates one or more functional processes. The sub processes of each functional process are sequences of events, and comprise at least two data movement types: an Entry plus at least either an Exit or a Write. An Entry moves a data group, which is a set of data attributes, from a user across the boundary into the functional process, while an Exit moves a data group from a functional process across the boundary to the user requiring it. A Write moves a data group lying inside the functional process to persistent storage, and a Read moves a data group from persistent storage to the functional process. See Figure 3 for an illustration of the generic flow of data through software from a functional perspective.

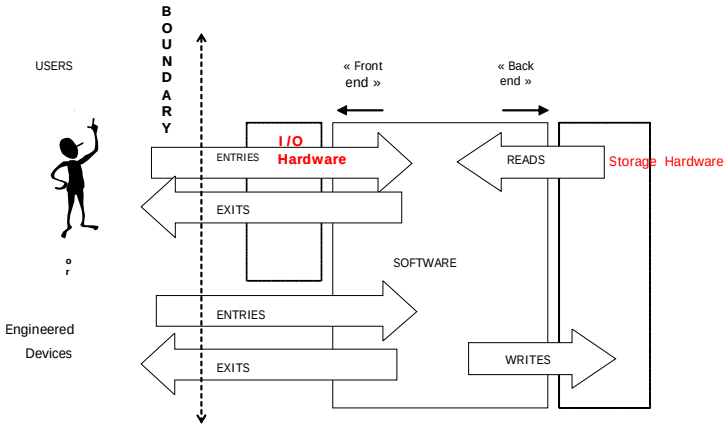

Figure 3 Generic Flow of Data through Software from a Functional Perspective [9]

A generic procedure for measuring software functional size with COSMIC-FFP is illustrated in Figure 4. The measuring process is performed through five steps.

First, the boundary of the software to be measured is identified by the measurer based on the requirements and the specifications of the interaction between the hardware and software. Second, the measurer identifies all possible functional processes, triggering events and data groups from the requirements. These are considered as candidate items at this stage. Third, the candidate items (i.e. functional processes, triggering events and data groups) are mapped into the COSMIC-FFP software context model (Figure 4) based on COSMIC-FFP rules. In this mapping, each functional process must be associated with a triggering event and with the data group(s) manipulated by it. This mapping also allows identification of the layers. Fourth, COSMIC-FFP sub processes (i.e. data movements of the following types: Entry, Exit, Read and Write) will be identified within each functional process. The COSMIC-FFP measurement function will be

applied to the sub processes identified to determine their respective COSMIC-FFP Cfsu[1] size measure. Finally, the measurer will compute an aggregate of the measurement results to obtain the total functional size of the software being measured.

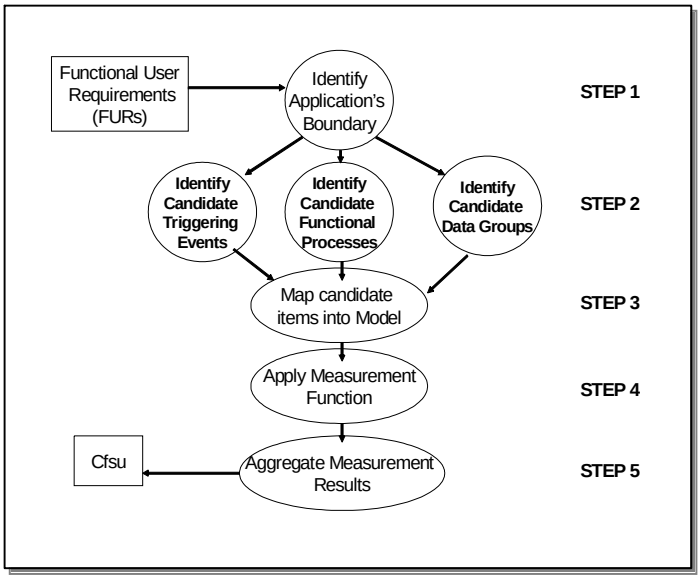

Figure 4: General Procedure for Measuring Software Size using COSMIC-FFP – ISO 19761 [14]

IV. Analysis of Linkages Across Models

Since the COSMIC-FFP measurement model and related definitions and rules are generic, it is of interest to investigate the feasibility of their applicability in specification languages to figure out whether or not the COSMIC-FFP method could complement the complexity management in the AS-TRM, which would allow early complexity assessment from the formal specifications. To achieve this, it is necessary to investigate a formalization of COSMIC-FFP within the AS-TRM formal model.

This formalization of COSMIC-FFP for AS-TRM requires a mapping of the COSMIC-FFP concepts (boundary, layer, functional process, triggering event, data group, movement and attributes, etc.) to the AS-TRM notation. Clear rules of COSMIC-FFP measurement shall be defined for AS-TRM specifications according to the COSMIC-FFP informal textual definitions given in [9]. The rules for measurement of autonomic elements and systems' size have to be specified formally and mapped to system behavior so that the functional size can be monitored automatically.

The initial mapping analysis between COSMIC-FFP and AS-TRM is presented in Table 1.

Table 1 COSMIC-FFP & AS-TRM

COSMIC-FFP concepts	*AS-TRM notations*
Boundary	Reactive Component interface
Layer	Tier in the formal model
Functional process	Reactive task or self-management task
Triggering event	Shared input event
Data group	LSL trait
Data movement	Internal & External event (input & output)
Data attribute	Operation in the LSL trait

The boundary concept in COSMIC-FFP corresponds physically to the "Reactive Component interface" notation in the formal AS-TRM. Since only the specification level is considered here, the ports that model the interfaces of the generic reactive classes (GRCs) [10] during the design phase are therefore not considered, but rather the Reactive Component interface. In specification, the user (either a human or an engineered device) wants to communicate with the software in order to receive/enter information. For example, the user communicates with the elevator through buttons in order to go either up or down. The communication design details are not of interest at the specification level. As is also shown in Table 1, the layer concept in COSMIC-FFP corresponds to the tier in the formal AS-TRM, where each upper tier communicates only with the tier immediately below it. The tier structure describes the system configuration, the ACGs, the ACs, the GRCs and their relative ADTs used to model the attributes in the AS-TRM (see section II). The functional process and the sub functional process concepts in COSMIC-FFP are mapped to "Reactive task or self-management task" notation in the formal AS-TRM. A reactive task is carried out during the synchronous communication between the ACs belonging to one ACG, which cooperates in the fulfillment of a group task. Each ACG can independently accomplish a complete real-time reactive task. A transition specification describes the computational step associated with the occurrence of an event. A transition is triggered by an event and causes a reaction in the form of the occurrence of either an internal event or an output event. There may be a timing constraint on the occurrence of the reaction. Triggering event and data movement (i.e. Entry) concepts in COSMIC-FFP correspond to the external (input) event that occurs at a port, and represent a message being transmitted in the formal AS-TRM. Also, a data movement corresponds to an internal (Read from internal storage, Write) or output (Exit) event in the AS-

[1] Cfsu = COSMIC-FFP functional size unit

TRM. Finally, a data group corresponds to an LSL trait, the lowest tier and the basic specification unit, and represents the ADT used in modeling the system and generic reactive class attributes. The operation in this LSL trait corresponds to a Read or Write data movement in COSMIC-FFP, and an LSL trait included in a given data movement corresponds to a data attribute in COSMIC-FFP. The above mapping between the COSMIC-FFP and AS-TRM concepts conforms to the AS-TRM event-driven modeling paradigm, where the components communicate through events and these events carry information.

The applicability of the COSMIC-FFP FSM in terms of the formal AS-TRM is illustrated in the next section.

V. CASE STUDY: STEAM BOILER

The Steam Boiler Control specification problem of J. R. Abrial and E. Brger [16] was derived from an original text by J. C. Bauer for the Institute for Risk Research at the University of Waterloo, Ontario, Canada. The original text has been submitted as a competition problem to be solved by the participants at the International Software Safety Symposium organized by the Institute for Risk Research. It provides the specification design that will ensure safe operation of a steam boiler by maintaining the ratio of the water level in the boiler and the amount of steam emanating from it with the help of the corresponding measurement devices. The Steam Boiler System consists of the following physical units:

- Steam Boiler: the container holding the water;
- Pump: the device for pouring water into the steam boiler;
- Valve: the mechanism for evacuating water from the steam boiler;
- Water Level Measurement device: a sensor to measure the quantity of water q (in liters) and inform the system whenever there is a risk of exceeding the minimum or maximum amounts allowed.

Figure 5 shows the Steam Boiler and the relationships between its components. The Steam Boiler is assumed to start up with a safe amount of water. The Controller runs a control cycle every 5 minutes to check on the amount of water currently in the system, and then triggers the Water Level Measurement device and sends the result to the Controller. The Controller receives the current level and checks whether it is normal, above normal or below normal: if the water level is normal, it will do nothing; if there is a risk that the minimum safe level will be reached, the Pump will be triggered to pour more water into the Steam Boiler; and if there is a risk that a level higher than normal will be reached, the Valve will be triggered to evacuate water from the Steam Boiler.

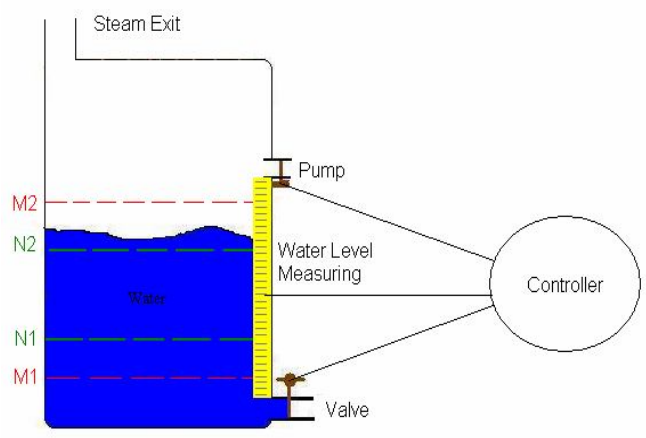

Figure 5: Steam Boiler Controller

It is of interest to measure the Steam Boiler Controller component (Figure 6), which is located within the AC tier. The Steam Boiler Controller is bounded by its interface, which separates it from the other components.

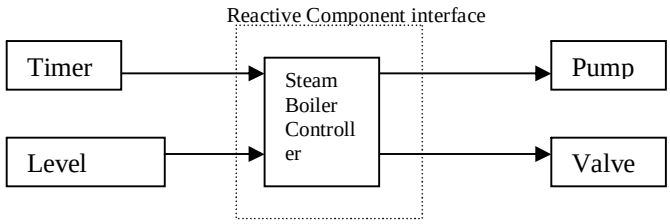

Figure 6: Steam Boiler Controller and its Interface

There are two reactive (or self-management) tasks that the Steam Boiler Controller has to accomplish. They are shown in Figures 7 and 8 as a sequence of events (e.g. corresponding to data movements in COSMIC-FFP) to be considered in FSM. The total number of these events in one sequence diagram corresponds to the total functional size for one reactive task. In other words, one event corresponds to one data movement, the basic elementary unit used in the COSMIC-FFP measurement method.

Figure 7 shows the first reactive task accomplished by the interactions of the Controller with other components, that is, when the water level is below the minimum. The Controller sends an “open” message to the Pump, which reacts accordingly. It also sends a “close” message to the Valve. That reactive task is triggered by a shared input event, which is the “cycle” that is received on the request of Controller to check the measurement level every 5 minutes.

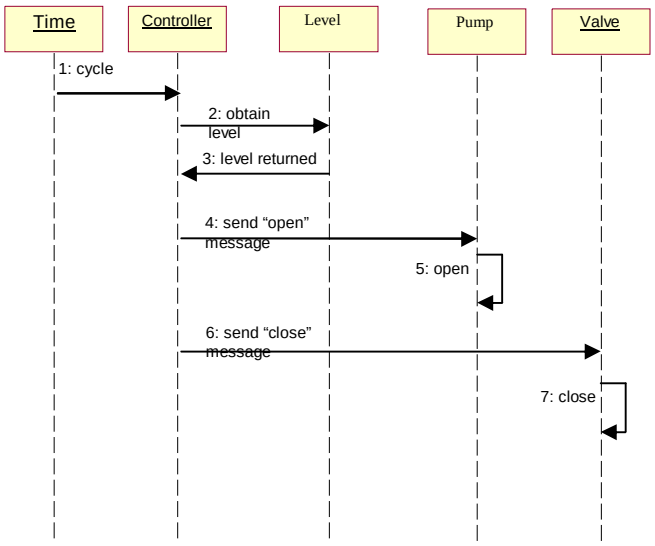

Figure 7: Controller Reactive Task (1)

The second reactive task shown in Figure 8 is also triggered by the "cycle" event. It checks the water level to note whether or not that level is above the maximum. It sends a "close" message to the Pump component and an "open" message to the Valve component. As a result, they react accordingly.

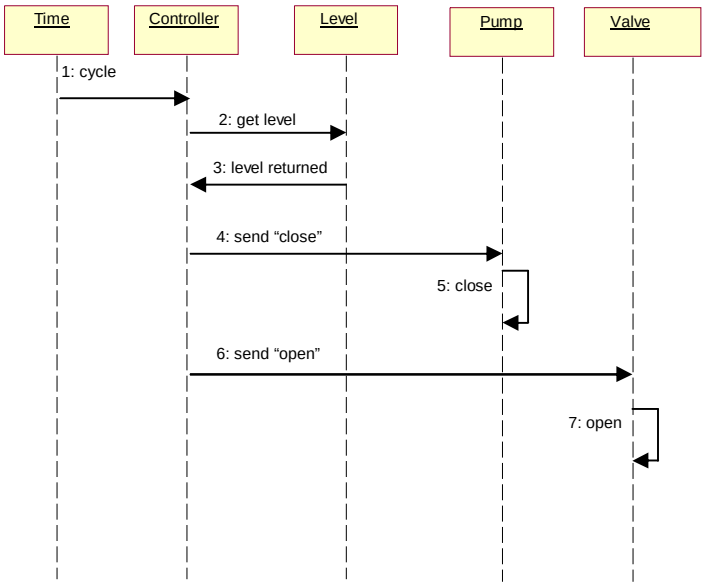

Figure 8: Controller Reactive Task (2)

The total functional size for both tasks is shown in Table 2.

Table 2: Total Function Size for Steam Boiler using AS-TRM terms

Tier	Reactive task	Sequence of events	Type of event	Corresponding functional size
AC	Acts when the water level is below, at or above the required level (see Figures 7 & 8)	1. Receive data from 5-minute time cycle check	Shared input event	1
		2. Obtain the water level measurement (value = below normal, normal or above normal)	Internal input event	1
		3. (Logic) Check if any action is needed; if not, terminate the cycle		
		4. Send message to Pump (value = open or close)	External output event	1
		5. Send message to Valve (value = open or close)	External output event	1
		6. The Boiler Controller mode is set (value = below normal, normal or above normal)	Internal output event	1
Total Functional size of Steam Boiler Controller software				5 Cfsu

VI. Discussion and Next Steps

In this paper, the candidate linkages between the AS-TRM and the COSMIC-FFP FSM method are investigated. We have set out the initial step for formalizing COSMIC-FFP in the AS-TRM context by mapping the COSMIC-FFP concepts (boundary, layer, functional process, triggering event, data group, movement and attributes, etc.) to the AS-TRM notation. Clear rules of COSMIC-FFP measurement have been defined for AS-TRM specifications according to the generic COSMIC-FFP definitions.

Research in progress is looking into an automatic COSMIC-FFP FSM in the AS-TRM context. This will allow a uniform application of the rules of the COSMIC-FFP FSM method to AS-TRM specifications across different case studies, making it possible to analyze several case studies for validation purposes. Moreover, based on this mapping, we may determine whether or not there is a possibility of allowing for an automatic early complexity self-assessment of evolving autonomic systems from their formal specifications.

REFERENCES

1. N. E. Fenton and S. L. Pfleeger, Software Metrics: A Rigorous and Practical Approach, 2nd ed: PWS Publishing Company, 1998.
2. O. Ormandjieva, "Deriving New Measurement for Real Time Reactive Systems," in Computer Science & Software Engineering Department. Montreal: Concordia University, 2002.
3. S. A. Whitmire, Object Oriented Design Measurement: John Wiley & Sons, 1997.
4. B. Henderson-Sellers, Object Oriented Metrics: Measures of Complexity. New Jersey: Prentice-Hall, 1996.
5. H. Zuse, Software Complexity Measures and Methods. Berlin, New York: Walter de Gruyter, 1991.
6. J. S. Davis and R. J. Leblanc, "A Study of the Applicability of Complexity Measures," presented at IEEE Transactions on Software Engineering, 1988.
7. A. Abran and P. N. Robillard, "Function Points: A study of Their Measurement Processes and Scale Transformations," Journal of Systems and Software, vol. 25(2), pp. 171-184, 1994.
8. T. DeMarco, Controlling Software Projects. New York: Yourdon, 1982.
9. Abran, A., Desharnais, J.-M., Oligny, S., St-Pierre, D., Symons, C., (2003), *COSMIC-FFP Measurement Manual: The COSMIC Implementation Guide for ISO/IEC 19761*: *2003*, Version 2.2, January 2003, The Common Software Measurement International Consortium, École de technologie supérieure – Université du Québec, Montréal, Canada http://www.gelog.etsmtl.ca/cosmic-ffp/manual.html
10. R. Achuthan, "A Formal Model for Object-Oriented Development of Real-Time Reactive Systems," Ph.D. thesis, Department of Computer Science, Concordia University, Montreal, Canada, October 1995.
11. IBM, An Architectural Blueprint for Autonomic Computing. IBM and Autonomic Computing, 2003.
12. IBM, An Architectural Blueprint for Autonomic Computing. IBM and Autonomic Computing, 2004.
13. IBM, An Architectural Blueprint for Autonomic Computing. IBM and Autonomic Computing, 2005.
14. ISO/IEC 19761. Software Engineering − COSMIC-FFP − A functional size measurement method. International Organization for Standardization − ISO, Geneva, 2003.
15. Albrecht, A. J. and Gaffney, J. E., Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. IEEE Trans. Software Eng. vol. SE-9, no. 6, pp. 639-648, Nov. 1983.
16. J. R. Abrial, "Steam Boiler Control Specification Problem," 1991.
17. E. Vassev, H. Kuang, O. Ormandjieva, E., Paquet. Reactive, Distributed and Autonomic Computing Aspects of AS-TRM. In the proceedings of the 1st International Conference on Software and Data Technologies-ICSOFT2006, September 11-14, 2006, Setubal, Portugal