A SEMI-FORMAL METHOD TO VERIFY CORRECTNESS OF FUNCTIONAL REQUIREMENTS SPECIFICATIONS OF COMPLEX EMBEDDED SYSTEM

Nihal Kececi
Department of Computer Science
Université du Québec à Montréal
Software Engineering Management Research Laboratory
nkececi@lrgl.uqam.ca

Wolgang A. Halang
Computer Engineering and Real Time Systems
Faculty of Electrical Engineering FernUniversitat
Wolfgang.Halanf@FernUni-Hagen.de

Alain Abran
École de Technologie Supérieure - ETS
1100 Notre-Dame Ouest, Montréal, Canada H3C 1K3
abran.alain@uqam.ca

Abstract: Many standards mandating verification of requirements correctness do not comprehensively state what information should be captured and used for verification and quality assurance activities. Therefore, a wide range of methods, from simplistic checklists to comprehensive formal methods, is used to verify correctness of system and software requirements. In this paper, a semi-formal method to verify functional requirements using a graphical logic-based structured architecture referred to as Graphical Requirement Analysis is proposed and illustrated with a case study. Its architecture allows to trace functional system requirements and to show correctness (non-ambiguity, consistency, completeness) of specifications. The support of graphical system engineering descriptions greatly facilitates to simulate requirement specifications and designs. Such capability is believed by many to be an essential aspect of developing and assuring the quality of highly complex systems requiring high integrity.

Key words: Requirements engineering; functional correctness, verification; quality; modular programming; functional decomposition; modeling of complex embedded systems specifications**.**

1. INTRODUCTION

To deal with inconsistent and incomplete requirements, many approaches to software requirement analysis have been developed over the last few years. According to a survey and assessment of conventional software V&V methods [1], requirements and design techniques consist of four major classes and various subclasses. These major classes of techniques and the total number of individual techniques are as follows:

Formal methods are based on a translation of requirements into mathematical form. Eight different techniques were discovered.

Semi-formal methods are based on the expression of requirement specifications in a special requirement language. Eleven different individual techniques were discovered.

Reviews and Analysis (informal method) are based on reviews by special personnel of the adequacy of the requirement specification according to a pre-established set of criteria and detailed checklists and procedures. Seven different techniques were identified.

Tracing and Analysis Techniques of the requirements are based on matching of each unique requirement element to design elements and then to the elements of the implementation. Two different techniques were identified.

However, there are some common problems not fully tackled by most of these techniques:
Formalizing the requirements (in total or in part) presents a new viewpoint. But formalization itself cannot guarantee to detect error, nor can it prove that the requirement specification is correct.
Mathematical verification of requirements does not seem to greatly simplify development.

Testing a specification will not find all the possible errors. For instance, combinatory effects are unlikely to be encountered.

Also, the ultimate interrelationship of the various aspects of the specification may not be known until the implementation is complete.

Graphical Requirement Analysis (GRA) is a framework that describes a function into graphical logic based with a structured form. To build a module (functional block) it requires identifying inputs, outputs and logic into multilevel hierarchy. The architecture of GRA methodology provides for functional system requirements traceability as well as correctness (ambiguity, consistency, completeness) of specification. GRA support of system-engineering descriptions in a graphical mode greatly facilitates simulating requirements specification and designs [2-3]. Such capability is believed by many to be an essential aspect of developing and assuring the quality of highly complex systems requiring high integrity. In this paper benefits of such a graphical functional module, and lessons learned from implementation is presented.

Summary: In this section, we highlight some limitations of the V&V Requirements and Design Methods. In section 2, we present the key aspects of the proposed GRA approach, in section 3, a case study of its implementation, and observations in section 4.

2. GRAPHICAL REQUIREMENT ANALYSIS (GRA)

The GRA framework [3-4] was proposed as a verification and integration tool for software and system specification as well as design specification. It aims to visualize functional specifications (input-process-output-interfaces) using a logic-based graphical framework. Figure 1 illustrates the framework's modelling philosophy for complex embedded systems. A software module performing a specific action is invoked by the appearance of its name in an expression; it may receive an input value and return a single value. When a module (function) is decomposed, sub-modules (functions) can be identified. The GRA framework's core concept is derived from the functional modelling methodology and a function's basic characteristics as defined in the standard IEEE 610.12 [5]. However, it also captures some system and software engineering approaches such as hierarchy theory, success failure mechanism, function block diagram paradigm, modular programming, and the COSMIC-FFP [6] functional size measurement procedure.

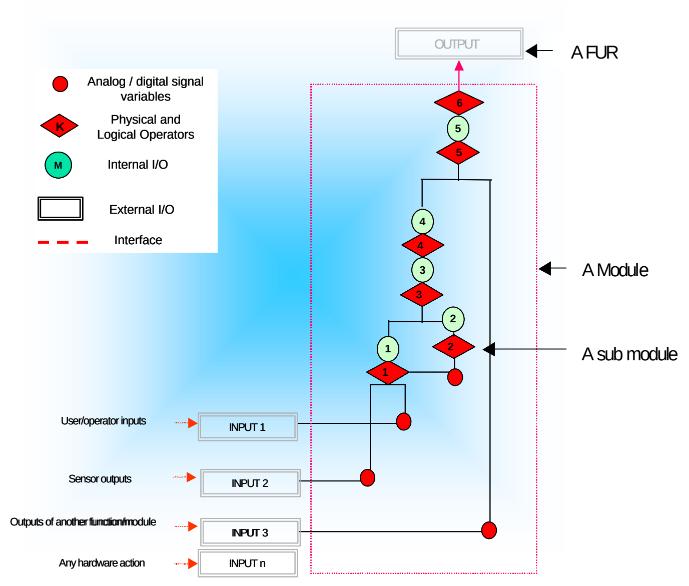

***Figure** 1* Modelling philosophy of the GRA framework for complex embedded systems**.**

1. Identifying a functional user requirement FUR (effort at requirement specification level)

* Hierarchy theory and the success-failure paradigm are used to break a FUR into sub-modules (components) to facilitate design and development (design level effort).

* Inputs/outputs and algorithms of modules are identified (architectural level effort).

2. Defining the relationships in the hierarchies, which show connections between different nodes of a hierarchy or between nodes of two different hierarchies. The relations can be characterized as:
* Logical (Boolean) connectivity relationships are used to show the redundancy and

connectivity between various components; in a logical relationship the states of the input and output nodes are binary.

* Physical connectivity relationships refer to node relations described by some physical laws, and are mostly represented by a continuum of values; accordingly, physical relationships are analogue in nature.

* Uncertain (fuzzy) connectivity relationships are appropriate when relationships are not fully known, or physical descriptions are either not available or uncertain.

3. Defining the operators based on the relationships between sub-modules

* Physical operators are presented in the GRA framework with a macro function such as mathematical, data/time, string, aggregate, data type conversion, array, system, graph, hierarchy and database functions.

* Logical operators in an expression describe the type of action the expression should perform, or how the expression should compare or relate two values, e.g., arithmetical, text, comparison, and conditional or loop control operators.

Summary: Graphical Requirement Analysis (GRA) is a methodology *for translating textual functional requirements to logic based graphical format* that may or may not be understood exactly, where the user is responsible for telling the developer what the functional requirements should do or vice versa (where the developer is responsible for verifying correctness of functional requirements).

3. CASE STUDY

The detailed FUR descriptions for a pressure controller were transformed from textual to graphical form. This process captured functional traceability as well as data and component traceability analysis. To build the pressure control function within the GRA architecture, the following documents had to be reviewed to ensure that all related functions were indeed included (and traceable backward and forward): the "Software Requirements Specification" (SRS) to understand what the function was supposed to do, the "Software Design specification" (SDR) to identify the control blocks to be used, the "Theory Manual" to understand the pressure control system procedure, and the "User Guide" to identify the system variables and the I/O of the pressure controller. The main purpose of this reviewing process is to identify typical characteristics of functional user requirements: outputs, inputs (user inputs, system variables, other function outputs), arithmetical operators or algorithms describing how these input variables provide desired output values, and the control procedure defining the relationships between inputs, outputs and logical operators. Figure 2 illustrates the results of this review process for a FUR of a pressure control system: user inputs are identified from the SRS, system variables and the function's output are identified from the User Guide, algorithms are identified from the Software Design Specification, and the control procedure is identified from the Theory Manual.

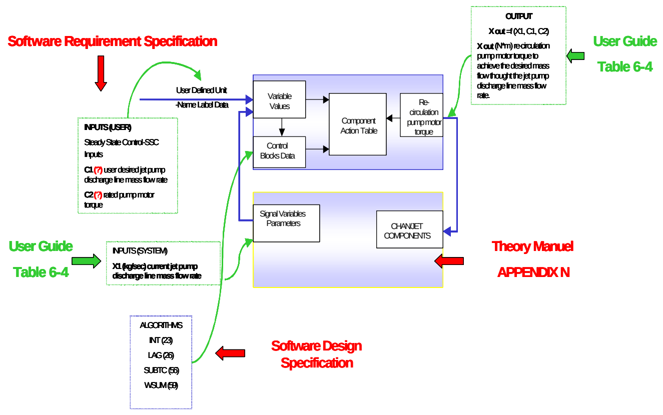
Software Requirement Specification
OUTPUT
Xout =f (X1, C1, C2)
Xout (N*m) re-circulation pump motor torque to achieve the desired mass flow thought the jet pump discharge line mass flow rate.
User Guide
Table 6-4
User Defined Unit
-Name Label Data
Variable Values
Component Action Table
Re-circulation pump motor torque
Control Blocks Data
INPUTS (USER)
Steady State Control-SSC Inputs
C1 (?) user desired jet pump discharge line mass flow rate
C2 (?) rated pump motor torque
Signal Variables Parameters
CHAN/JET COMPONENTS
User Guide
Table 6-4
INPUTS (SYSTEM)
X1 (kg/sec) current jet pump discharge line mass flow rate
Theory Manuel
APPENDIX N
ALGORITHMS
INT (23)
LAG (26)
SUBTC (56)
WSUM (59)
Software Design Specification

Figure 2. Pressure Control Function Procedure

The pressure control function detects the main steam line pressure X1, and adjusts the main steam line valve to achieve the pressure set point. Its implementation is composed of so-called control blocks, which are operators producing output parameter signals out of input parameter signals. Output signals can then be used as input parameters to other control blocks. Thus, a control procedure for a component action can be constructed by coupling control block operators in series and in parallel. Here, five control blocks are used to compose the pressure controller. Their types and identification numbers are Integration-INT23, Subtraction-SUBTC56, First-order lag-LAG26, weighted summation-WSUM59 and Addition-ADD3 yielding the Pressure controller-204. Figure 3 presents the functionality of the pressure control system within the GRA framework. It clearly shows all attributes of a Functional User Requirement.

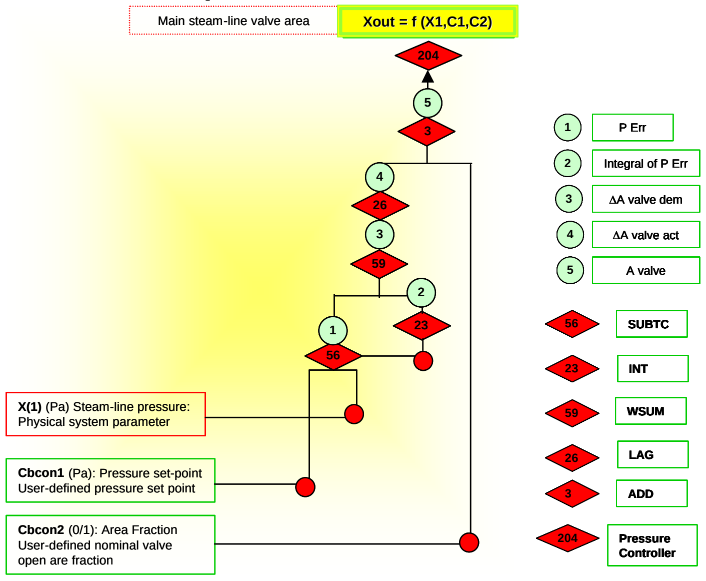
Main steam-line valve area
Xout = f (X1,C1,C2)
204
5
3
4
26
3
59
2
1
23
56
X(1) (Pa) Steam-line pressure: Physical system parameter
Cbcon1 (Pa): Pressure set-point User-defined pressure set point
Cbcon2 (0/1): Area Fraction User-defined nominal valve open are fraction
1 P Err
2 Integral of P Err
3 ΔA valve dem
4 ΔA valve act
5 A valve
56 SUBTC
23 INT
59 WSUM
26 LAG
3 ADD
204 Pressure Controller

Figure 3. Verification of pressure control FUR within the GRA framework

4. LESSON LEARNED

From this case study one can draw several conclusions beneficial in addressing the problems of functional requirements correctness. Being based on checking listed items and criteria, formal review and inspection techniques are able to identify functional information at certain levels, but not functionalities in full detail. Table 1 states the levels of information detectable by GRA and by review and inspection. Review and Inspection criteria are selected from [7-8]. Software quality engineers, configuration managers, and project managers evaluating products and projects can all use the information provided within the GRA framework. Even though we emphasize here verification of software requirements specifications, software never runs alone, but always as part of a larger system consisting of other software products with which it has interfaces, of hardware, humans and workflow. The GRA framework captures both system and software specifications and design attributes. Therefore, it lends itself as an alternative method to establish the requirements' correctness of embedded systems.

Table 1. Formal review and inspection versus GRA

REVIEW & INSPECTION “Criteria”	GRA Analysis Functional Description	GRA Analysis Graphical Description
Inputs	External Inputs	▭
Output	External Output	▭
Algorithms -	Logic, Control Block	◆
-	Internal Input	🔴
-	Internal Output	🟢
Interfaces	Interfaces	- - - - - - -
Time	Data Flow Sequence	Ⓣ

REFERENCES

1. Groundwater E.H., Miller L.A., Mirsky S.M. 1995. *Guidelines for the Verification and Validation of Expert System Software and Conventional Software. Survey and Document of Expert System Verification and Validation Methodologies* NUREG/CR-6316, SAIC-95/1028. Vol.1-7.

2. Kececi N., Abran A., *"An Integrated Measure for Functional Requirements Correctness"*. IWSM2001, 11th International Workshop on Software Measurement, August 28-29, 2001 Montréal (QC) Canada

3. Kececi, N., M. Modarres, and C. Smidts. "System Software Interface for Safety-Related Digital I&C Systems", European Safety and Reliability Conference, TUM Munich- Garching, September 13-17, 1999. http://www.lrgl.uqam.ca/team/membres.html

4. Kececi N., Abran A. "Analyzing, Measuring and Assessing Software Quality in a Logic Based Graphical Model" 4th International Conference on Quality and Dependability-QUALITA 2001, 22-23 March 2001 Annecy France.

5. Kececi N., M. Li, C. Smidts, C. "Function Point Analysis: An Application to a Nuclear Reactor Protection System," International Topical Meeting on Probabilistic Safety Assessment –PSA'99, Washington, DC, August 22-25, 1999.

6. IEEE Std. 610.12-1990. *IEEE Standards Glossary of Software Engineering Standards.*

7. Abran, A., Desharnais, J.M., Oligny, S., St-Pierre, D. and Symons, C. "COSMIC-FFP Measurement Manual, version 2.1", Software Engineering Management Research Laboratory, Université du Québec à Montréal, Montreal, Canada, 2001. Downloadable at http://www.lrgl.uqam.ca/ffp.html

8. IEEE 830-1998, "IEEE Guide to Software Requirements Specification".

9. IEEE 1012-1998, "IEEE Standards for software Verification and Validation Plan"