A research on Generic Functional System Requirement Engineering Concept and Management Tool

Kim, Seungbeom
Control System Research Team
Hyundai Carnes Co. Ltd.
Hyundai Motor Group Institute, Uiwang-si, Korea
E-mail: seungbeom.kim@carnes.co.kr

Championnet, Guy
Control System Research Team
Hyundai Carnes Co. Ltd.
Hyundai Motor Group Institute, Uiwang-si, Korea
E-mail: championnet.guy@carnes.co.kr

Lee, Louis
Control System Research Team
Hyundai Carnes Co. Ltd.
Hyundai Motor Group Institute, Uiwang-si, Korea
E-mail: louis@carnes.co.kr

***Abstract*—This paper presents a new requirement management and creation approach in the area of system engineering. The new way of vehicle requirement management is used the generic function elements for creation of project specific requirements. To save manual working time, a requirement management tool has been developed. This tool provides the user with a simple and easy way to define for specific projects a set of requirements aggregating and configuring a set of generic requirement elements, improving reusability between projects.**

Keywords- Requirement Engineering, Generic Functional System Requirement, GFS+, GFS+Aggregator, GFS+ Element

I. INTRODUCTION

With the increasing complexity and improvement of technology in vehicle system, the importance of system engineering, which should be designed and managed over the life cycle of the project, is crucial. The first step of system engineering is requirement engineering, and it is the field of engineering for analyzing, determining, and managing requirements. [1]

The conventional way of requirement management requires lots of effort for analyzing, updating, and managing the elements of the requirements. Most of the times, repetitive and simple manual works can't be avoided for parallel extension of generic requirement to vehicle specific project requirement. In addition, depending on the knowledge and skill of the engineer in charge, maturity of the vehicle project requirement issued is variable and will affect the requirement reliability.

In this paper, an efficient way for developing and managing requirements is proposed in regard to requirement engineering. (Called GFS+). Also, the tool to manage and create requirements is introduced.

II. REQUIREMENT DEVELOPMENT

General development way of system consists in eliciting requirements, modeling and analyzing requirements, development of requirements, and development of component. [2] In this paper, we will focus on the analyzing and developing phase of requirements.

The conventional development of vehicle system requirement is done by applying for each project a change request.

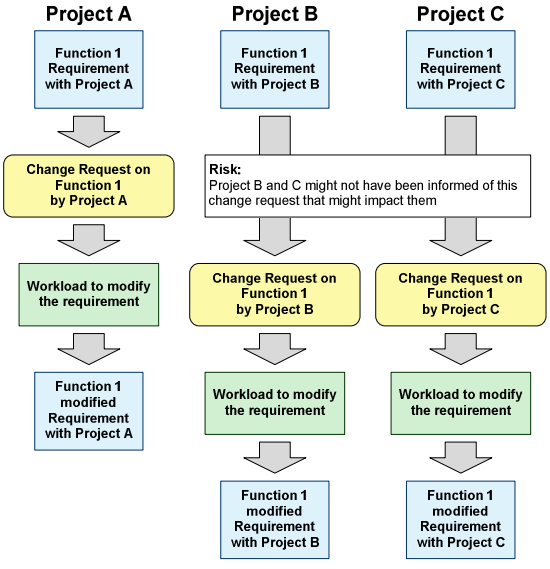


Figure 1. Conventional way of requirement development

Assuming that different vehicle projects share in common a function 1. (Figure 1), in this case, even though the change request to be applied for each project consists of a same

978-1-4673-0990-5/12/$31.00 ©2012 IEEE

functional change, each project requirement will have to be modified separately. In addition, it is worth mentioning that the standardization of function change across projects is made difficult as each project's development is in general distinct, and the occurrence of a change request applied on one project are not necessary known to other projects.

Because each requirement is managed in general by each vehicle project dedicated workforce, the same change will be applied on each project by different person/method with limited possible re-use leading to an inefficient use of resource and as well making difficult a global tracking of change requests in the perspective of development process.

In this context requirements management, an appropriate definition of generic functional system requirement and possible reuse of those generic requirements is very useful.

Generic Function Specification+ (GFS+) methodology for requirement development is maximizing the reuse of Generic Functional System Requirement elements. After creation of generic requirements(GFS+ Element), each vehicle project requirement is managed and generated by a tool (GFS+ Aggregator) using sets of GFS+ Elements and with their project specific configuration.

In figure 2, because each vehicle project requirements are reused from generic requirements, working time can be saved and there is no behavior gap between each project requirement.

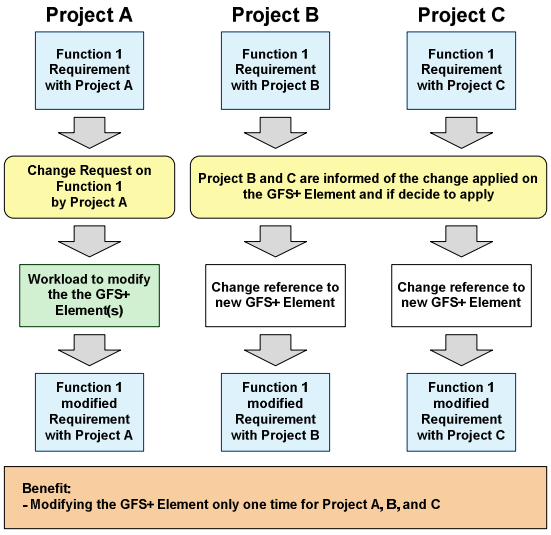


Figure 2. The way of GFS+ requirement development

III. GENERIC FUNCTION SPECIFICATION+ (GFS+)

A. *General concept of the GFS+*

GFS+ requirements development aims at providing a simple and easy way to user to define for project specific a set of requirements, aggregating a set of requirement elements, improving reusability between projects. For that, all functional behaviors are divided in atomic level elements, named GFS+ Element.

GFS+ Elements are divided into two categories, generic level elements and specific level elements. Generic level elements are characterized by common functionalities that are used by more than one project. For example, if element A is used in two different ECUs(Electronic Control Unit), element A is a generic level element. Specific level elements are the specific functionalities that are different for each project specific requirements. For example, if element B is only used in one ECU, element B is specific level element. GFS+ Aggregator does the aggregation of these GFS+ Elements and generates the project specific requirements.

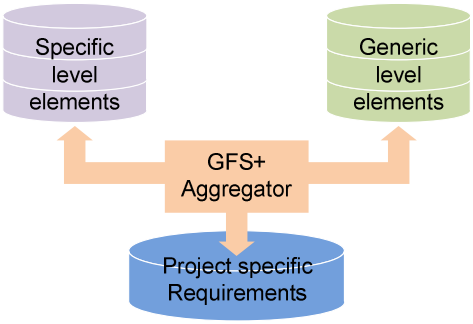


Figure 3. General concept of the GFS+

B. *GFS+ Element*

To avoid creation of different GFS+ Elements motivated by different available options or different input/output signals names, the functional logic is dissociated from signals the interfacing. The derived specific vehicle project requirements contains the common functional logic plus with the specific project signal names and when required, specific functional interface functions. (Figure. 4)

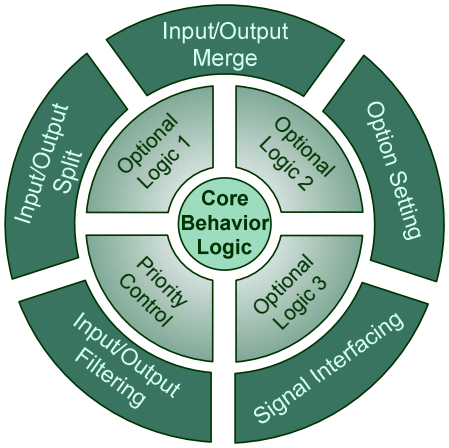


Figure 4. Basic structure of the GFS+ Requirement

Depending on each project, signal and functional interface can be changed by mean of configuration. Generally, the creation of core behavior logic function is more difficult than the creation of signal and functional interface function. The maximum reuse of logic elements can decrease the workload for elements creation.

Inputs/outputs specifics to each project are assigned to the generic interfacing signals and/or functional interfaces without changing the core logic of the function. (Figure 5)

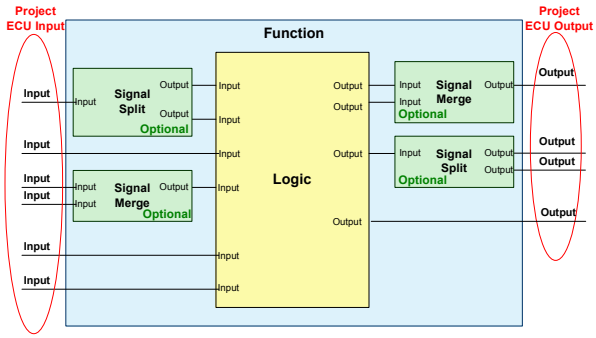


Figure 5. Structure of the GFS+ Requirement

A logic function consists of core behavior logic and optional logic function(s). Optional logic function can be selected or removed. (Figure 6)

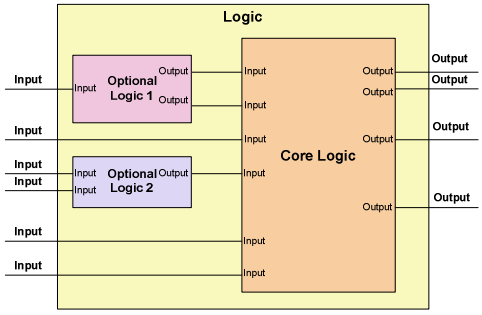
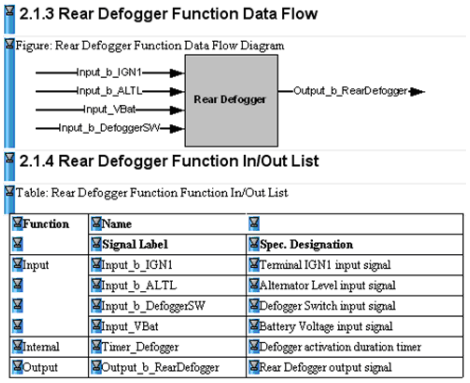


Figure 6. Logic Level Structure

C. *Input/Output Signal Correspondence Management*

The GFS+ Elements are focusing only on the logic of the functions and the interactions between functions, using for inputs and outputs generic variable names according to the following principle:

- All the inputs variables begin with: Input_...
- All the outputs expected to have a correspondence outside of the ECU begin with: Output_...
- All the outputs expected not to have a correspondence outside of the ECU begin with: Output_OPT_...
- All other variables not starting with Input_... or Output_... are internal variables used within the GFS+ Element.



2.1.3 Rear Defogger Function Data Flow

Figure: Rear Defogger Function Data Flow Diagram

2.1.4 Rear Defogger Function In/Out List

Table: Rear Defogger Function Function In/Out List

Function	Name	
	Signal Label	Spec. Designation
Input	Input_b_IGN1	Terminal IGN1 input signal
	Input_b_ALTL	Alternator Level input signal
	Input_b_DefoggerSW	Defogger Switch input signal
	Input_VBat	Battery Voltage input signal
Internal	Timer_Defogger	Defogger activation duration timer
Output	Output_b_RearDefogger	Rear Defogger output signal

Figure 7. Signal correspondence– At the GFS+ element level

Depending on the assignment of the GFS+ Elements within the requirements and the ECU, some inputs are having their ECU external signal correspondence required as this is illustrated in the following figure 8.

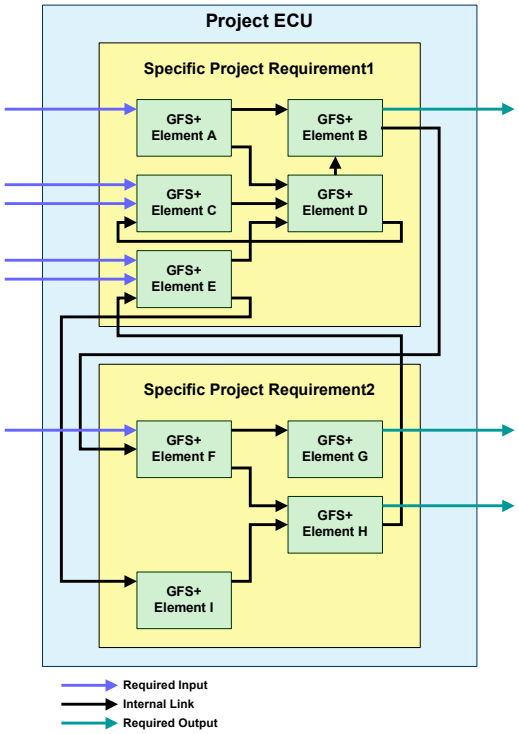


Figure 8. Signal correspondence– Example1

If for instance “Specific Project requirement 1” is no longer assigned to this Project ECU the Signal correspondence scheme will be change as figure 9.

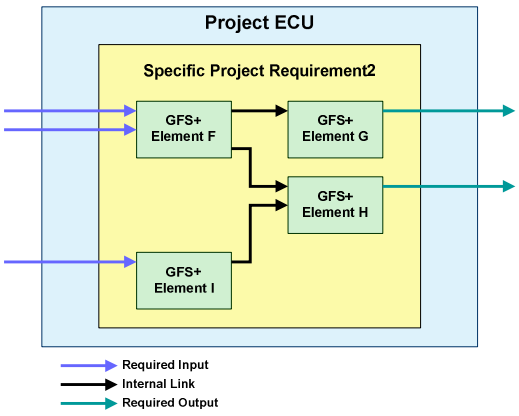
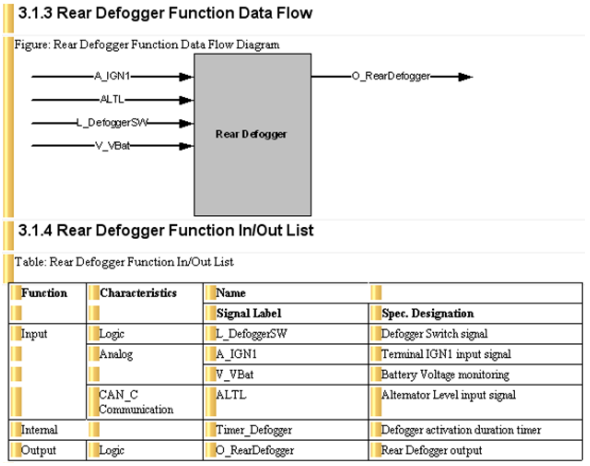


Figure 9. Signal correspondence– Example2

At the specific project level some of the Inputs will be communication signals input (e.g. CAN, LIN, KWP,…) some of the Inputs will be hardware inputs (e.g. PWM signals, Analog signals, Logical signals…) and the same goes for outputs; thus the need to define for the specific project the real Inputs/Outputs correspondence to make them match with the generic variables used in the GFS+ Elements.

This necessity is achieved by the definition of signal correspondence and will result in the replacement in the generated requirements of Input/Output variable names by their project specific Input/Output names.



Function	Characteristics	Name	
		Signal Label	Spec. Designation
Input	Logic	L_DefoggerSW	Defogger Switch signal
	Analog	A_IGN1	Terminal IGN1 input signal
		V_VBat	Battery Voltage monitoring
	CAN_C Communication	ALTL	Alternator Level input signal
Internal		Timer_Defogger	Defogger activation duration timer
Output	Logic	O_RearDefogger	Rear Defogger output

Figure 10. Signal correspondence– At the Generated Requirement level

In some cases, it might be required when an input is not having any correspondence at the ECU level but is required by the internal logic of the function to "stub" this input.

The action of stubbing an input consists in fixing its value to a fix value in order to pass successfully the consistency check of Input/Output signals.

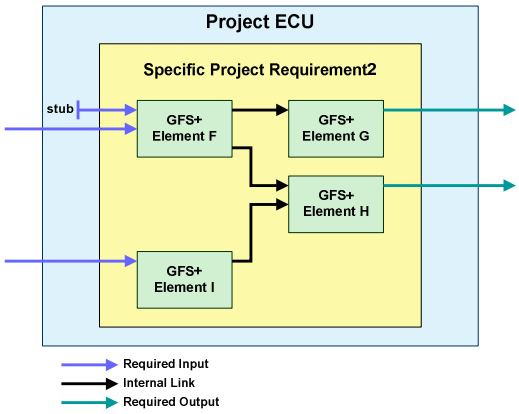


Figure 11. Signal correspondence– Stub

D. *Structure of the GFS+ Element Database*

GFS+ Element database is configured in DOORS [3]. Each GFS+ Element has below content.

- Document Change History
- Overall description
- Variants/Option/Calibration/Parameter
- Data Flow and In/Out list
- Function Description
- State Chart

(1) Document Change History: To check at a glance what have been the evolutions of the GFS+ Elements and to know who has modified what, when and according to which Change Request reference.

(2) Overall description: To provide reader with a short description

(3) Variants/Option/Calibration/Parameter: Presenting the availability of the function, the options, the calibrations, the parameter settings corresponding to various country variants

(4) Data Flow and Inputs/Outputs list: Presenting the Inputs Outputs interface of the function that can be connected to other GFS+ Elements functions

(5) Function Description: Presenting the description of the functionality for this GFS+ Element

(6) State Chart: Providing the reader with the logic of the function using state chart(s) aspiring at describing a clear behavior of a function in order to facilitate its software implementation.

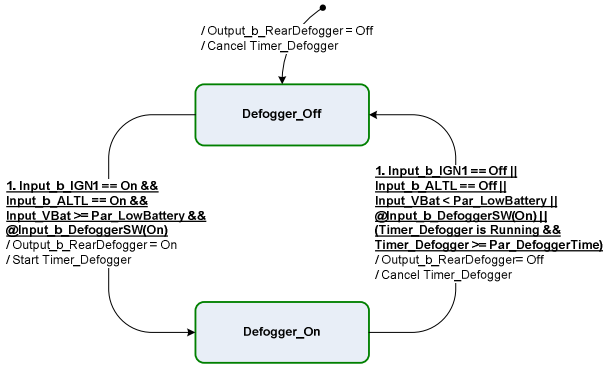


Figure 12. Example of the GFS+ State Chart

State charts require that the system described is composed of a finite number of states and transitions event-driven or state-driven, each transition composed with condition and action to perform when the condition is met.

E. *Tool - GFS+ Aggregator*

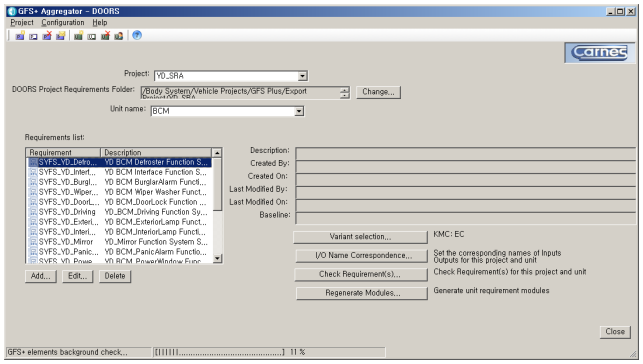


Figure 13. GFS+ Aggregator

Even if GFS+ Elements are well defined, if user would have to configure all these elements for specific projects manually, it would be a huge workload and would require a significant amount of time to complete the task. To limit the

manual workload, a tool, GFS+ Aggregator, has been developed using DXL language. [4]

There are similar tools [5] but GFS+ Aggregator is optimized and tailored to our needs and creates vehicle project requirements, allowing handling a large number of variants of requirements with minimized efforts and time.

GFS+ Aggregator does the aggregation of the GFS+ Elements, checks the inputs/outputs consistency within the GFS+ element, at the requirement level and at the ECU level, creates the functions interfacing diagrams, applies the country variant selection and generates at the end the specific project requirements under DOORS as DOORS Formal Modules.

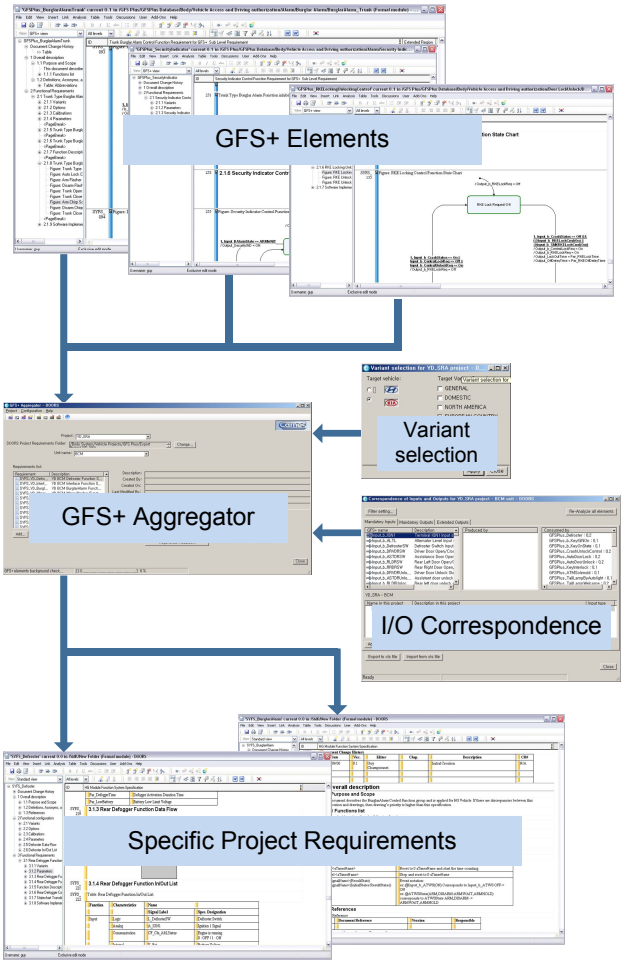


Figure 14. General concept of the GFS+ Aggregator

This tool automatically analyzes inputs and outputs of each elements and matches internal signals links. User just needs to assign ECU level required Input/Output names. (I/O Correspondence).

When specific project requirements are generated, assigned GFS+ Elements signal names are automatically replaced by project specific signal names.

IV. CONCLUSIONS

GFS+ provides the user with a simple and easy way to create, manage and generate project specific requirements.

The GFS+ requirement development methodology and tool has been conceived and applied to body control system requirement and is used to create and manage requirements in Hyundai Motor Company, now.

Figure 15 shows an actual comparison of the requirements development time between two projects with similar complexity and functionalities HG project(Hyundai Grandeur) not using GFS+, and VG project(Kia K7) using GFS+.

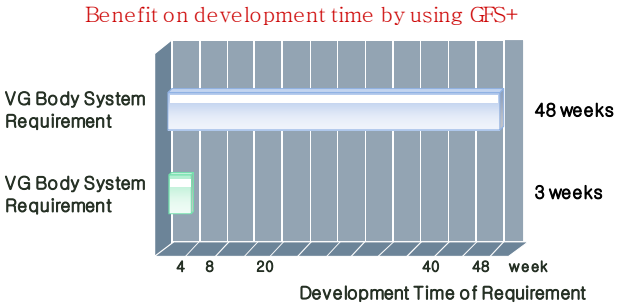


Figure 15. The benefit on development time

VG Project body system has 15 functional requirements with a similar complexity as HG Project body system having 16 functional requirements. VG Project body system requirements have been based on other projects already existing and available requirements, but different functional Input/Output architecture. HG Project body system requirements has been generated using GFS+.

Using GFS+ methodology and tool promotes generic developments, combined with automatic verifications and well defined atomic generic function elements, allowing reducing development time, the more the projects the more the common usage of elements. The four main benefits are:

(1) Dramatic decrease of development time

(2) Prevention of duplicated requirement development

(3) Parallel extension of new/updated requirement

(4) Limitation of manual work error

REFERENCES

[1] Karl E. Wiegers “More about Software Requirements” 2006 Microsoft Press

[2] B. Nuseibeh and S. Easterbrook. “Requirements engineering:a roadmap” In A. Finkelstein, editor, ‘The Future of Software Engineering’, Special Volume published in conjunction with ICSE 2000, 2000.

[3] “IBM Rational DOORS”, http://publib.boulder.ibm.com/infocenter/rsdp/v1r0m0/index.jsp?topic=/com.ibm.help.download.doors.doc/topics/doors_version9_2.html

[4] “DXL Reference Manual” http://publib.boulder.ibm.com/infocenter/rsdp/v1r0m0/index.jsp?topic=/com.ibm.help.download.doors.doc/topics/doors_version9_2.html

[5] “Doors dxl tool product” http://www.berner-mattner.com/en/berner-mattner-home/products/meran/index.html