International Journal of Mechanical, Industrial and Aerospace Sciences
ISSN: 2517-9950
Vol:11, No:3, 2017

VISMA: A Method for System Analysis in Early Lifecycle Phases

Walter Sebron, Hans Tschürtz, Peter Krebs

***Abstract*—**The choice of applicable analysis methods in safety or systems engineering depends on the depth of knowledge about a system, and on the respective lifecycle phase. However, the analysis method chain still shows gaps as it should support system analysis during the lifecycle of a system from a rough concept in pre-project phase until end-of-life. This paper's goal is to discuss an analysis method, the VISSE Shell Model Analysis (VISMA) method, which aims at closing the gap in the early system lifecycle phases, like the conceptual or pre-project phase, or the project start phase. It was originally developed to aid in the definition of the system boundary of electronic system parts, like e.g. a control unit for a pump motor. Furthermore, it can be also applied to non-electronic system parts. The VISMA method is a graphical sketch-like method that stratifies a system and its parts in inner and outer shells, like the layers of an onion. It analyses a system in a two-step approach, from the innermost to the outermost components followed by the reverse direction. To ensure a complete view of a system and its environment, the VISMA should be performed by (multifunctional) development teams. To introduce the method, a set of rules and guidelines has been defined in order to enable a proper shell build-up. In the first step, the innermost system, named system under consideration (SUC), is selected, which is the focus of the subsequent analysis. Then, its directly adjacent components, responsible for providing input to and receiving output from the SUC, are identified. These components are the content of the first shell around the SUC. Next, the input and output components to the components in the first shell are identified and form the second shell around the first one. Continuing this way, shell by shell is added with its respective parts until the border of the complete system (external border) is reached. Last, two external shells are added to complete the system view, the environment and the use case shell. This system view is also stored for future use. In the second step, the shells are examined in the reverse direction (outside to inside) in order to remove superfluous components or subsystems. Input chains to the SUC, as well as output chains from the SUC are described graphically via arrows, to highlight functional chains through the system. As a result, this method offers a clear and graphical description and overview of a system, its main parts and environment; however, the focus still remains on a specific SUC. It helps to identify the interfaces and interfacing components of the SUC, as well as important external interfaces of the overall system. It supports the identification of the first internal and external hazard causes and causal chains. Additionally, the method promotes a holistic picture and cross-functional understanding of a system, its contributing parts, internal relationships and possible dangers within a multidisciplinary development team.

W. Sebron is with the Vienna Institute for Safety and Systems Engineering (VISSE) at the FH Campus Wien University of Applied Sciences, Vienna, Austria (e-mail: walter.sebron@fh-campuswien.ac.at).

H. Tschürtz is head of the Vienna Institute for Safety and Systems Engineering (VISSE) at the FH Campus Wien University of Applied Sciences, Vienna, Austria.

P. Krebs is with the Vienna Institute for Safety and Systems Engineering (VISSE) at the FH Campus Wien University of Applied Sciences, Vienna, Austria.

***Keywords*—**Analysis methods, functional safety, hazard identification, system and safety engineering, system boundary definition, system safety.

I. Introduction

SYSTEMS engineering in general, and especially safety engineering, uses a variety of different analysis methods in order to ensure the correct functioning and safety of a system. Some of these methods are even used iteratively, integrating the growing knowledge about a system as development proceeds. However, these analysis methods are bound to a specific phase in the engineering lifecycle w.r.t. to their first usage, which is depending on what minimal depth of knowledge about the system is necessary to apply that respective method.

Fig. 1 shows the "hazard filter" [1, ch. 4.2], exemplarily filled with single methods according to the lifecycle phases of a system. The more a system development advances, the more sophisticated the necessary analysis methods become. These methods need deeper details about the system in order to produce meaningful results. Especially the classical (safety) analysis methods (e.g. FMEA, FTA, HAZOP) need detailed knowledge about a system [2] (p. 27). Only the early method types during conceptual design can be applied with a low level of system detail knowledge [1, ch. 4.3.1].

In order to apply an analysis method to a system correctly, it is necessary to first understand its basic requirements and design [1, ch. 5.2.2], and to have clearly defined system boundaries [1, p. 20]. Without those boundaries, the system cannot be distinguished exactly from its environment. It also becomes difficult to define the scope of analyses.

System engineering and system safety should look at *all* aspects of a system as an integrated whole, rather than looking at individual components *in isolation* from the system [1, p. 4]. This is often termed *holistic* approach and is also advocated in [2, p. 19]. Though, most, if not all, of the analysis methods applicable at an early lifecycle stage (some of which are also listed in Fig. 1) exhibit at least one of the following shortcomings:

1) Not following a structured and systematic approach (e.g. brainstorming)
2) Not defining the system boundaries (e.g. PHL, PHA)
3) Not defining a system structure (e.g. HAZOP)
4) Not supporting a holistic view (e.g. FMEA, checklists)

But when analyzing a (sub-)system-to-be-built at a very early stage (e.g. during concept phase), it is almost inevitable to come across the need for a structured method that aides in defining the overall system boundaries and its environment,

but also allows to focus upon the specific (sub-)system as it is embedded in the overall system.

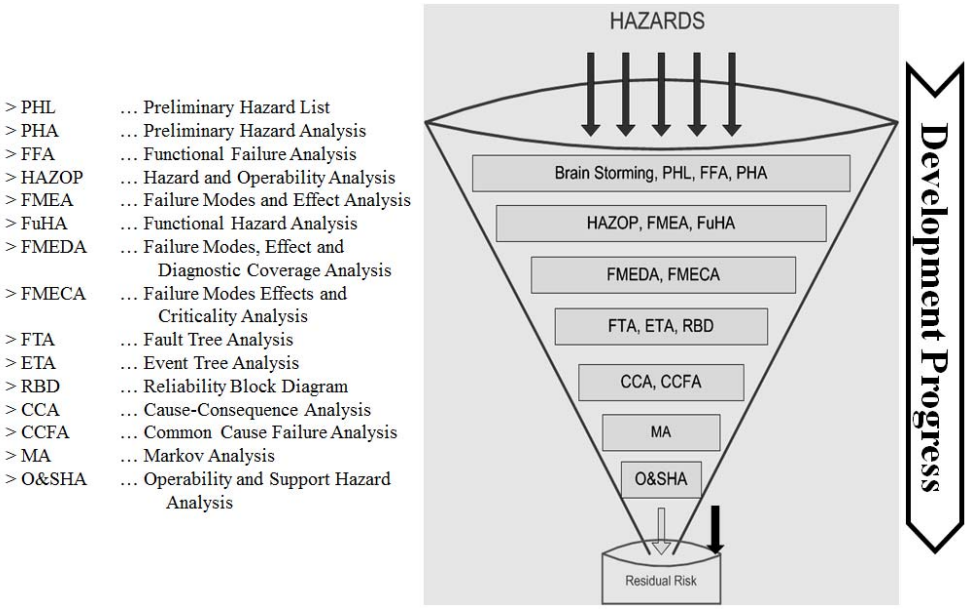

Fig. 1 The "Hazard Filter"

II. The Basic Idea

A. Motivation

The Vienna Institute for Safety and Systems Engineering (VISSE) conducted several support activities for safety analyses of new systems under development in different domains and industries over the past seven years. For this purpose, the systems in question had to be understood in great detail, mainly by conducting interviews with domain experts, developers and members of the project teams.

At first, classical building block diagrams were used to depict the static build-up of the systems-to-be-analyzed. For dynamic behavior, description state diagram methods were used. But the more people were interviewed about a specific system, the more different and often divergent viewpoints were encountered, especially about the boundaries of the system. There were even ambiguities about the newly intended functioning of the system and its components. This resulted in quite some confusion and showed the need for a method that supports and emphasizes the understanding of a system as a whole, and the definition of its boundaries and environment. Such a method should be applicable already at a very early project stage with only cursory knowledge on the system itself and the overall system it is embedded into. It should facilitate a description of the components and their relationships within the overall system. It should foster awareness of possible environmental influences which may vary due to different usage scenarios. This method should serve as an entry point for further system and/or safety analyses; the focus on the However, looking into different descriptions [4]-[6] and collections [1], [3], [7] of analysis methods, no method was found that would satisfy the above mentioned needs. This suggested that there exists a method gap.

B. Methodical Background

As stated in [2, ch. 2.3], when developing a (safety critical) system, understanding of the system, its environment, its workings, its failure modes, etc., is crucial. The main obstacle in this context is the ever-increasing complexity of newly developed systems. Reference [2] claims that "[simplicity] is the only real way to ensure understanding, or understandability, of the systems being produced." This simplicity can be achieved by use of appropriate *structure*. This is why VISMA uses the concept of *simplification* by focusing on one element of interest, called the SUC and its interactions with other system elements. This way, "unnecessary" details are omitted, thus limiting the efforts for modelling while still allowing purposeful input to subsequent analyses.

Additionally, *structuring* is achieved by placing the different *elements* of the overall system into different layers, the so-called "shells." These layers are arranged in a concentric manner in such a way that an onionskin-like model is generated. Furthermore, the structure gets enhanced via depiction of the relationships of the different components via arrows, thus defining also their basic interfaces.

A preliminary version of VISMA was defined and described in [8], and used as basis for the method described in this paper.

C. Goals of the Method

Since the basic idea for the method was to support the build-up and structure of and within a system, its boundaries and the influencing factors, the main goals of the method are:

1) Definition of a system's external boundary
2) Depicting the intended inner structure of a system and the relationship between its components
3) Consideration of environmental influences
4) Definition and consideration of use cases, usage scenarios and situations
5) Applicability at a very early lifecycle phase, e.g. pre-concept or early concept phase

Additional goals were identified during development of the VISMA method, in order to support, and provide input to, subsequent analysis methods:

1) Definition and consideration of the top level functions
2) Listing and considering the modes of the components and their influences on functions
3) Listing of communication transfers in between the components and defining their interfaces
4) Supporting the discovery and definition of preliminary hazards
5) Providing a holistic picture of the system

III. The VISMA Method

A. Prerequisites

As the VISMA should be applicable at a very early lifecycle stage, little input is required:

1) Basic idea or rough concept of the overall system
2) Selection of the SUC itself
3) Basic functional requirements of the SUC.

B. Basic Concepts of VISMA

The VISMA represents the structural and functional relationship between the SUC and its surrounding systems graphically as an interconnected structure, organized into a series of concentric stratums, or "shells". The elements of the structure depict the individual elements of a larger system which communicate with one other to execute functions. In addition, relevant influences from the environment and usage scenarios complement the view. A complete structure represents a system in its environment and is termed a *shell model* (SM) for the system.

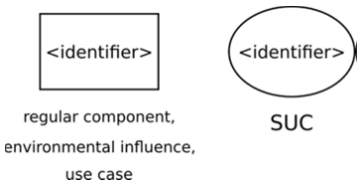

Fig. 2 Elements of the VISMA

Fig. 2 depicts the elements of VISMA. A *component* is a part of a larger system and can be clearly delineated (e. g. a dedicated HW element, such as a sensor or actuator). Each component is identified by an arbitrary *component identifier* (such as a name or numerical ID) that must be unique over the shell model.

The SUC is as a special kind of component and exists exactly once in a shell model - hence a unique symbol is reserved for it. Definition of and assignment of an identifier to the SUC forms the initial step in creating a shell model.

In addition to the SUC, a shell model can contain an arbitrary number of *regular* (i.e. non-SUC) components, or even none at all. The regular components form the "inner" or system environment of the SUC which directly influence or depend on its behavior.

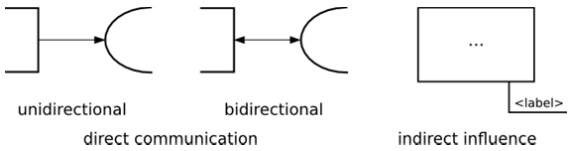

Fig. 3 Arrows for direct communication and notation for indirect influence of component

Direct communication between components is depicted by arrows between the component symbols for the exchange of *messages*. Messages include transfer of energy (e.g. electricity) or information (e. g. analogue or digital signals) - either unidirectional or bidirectional. Additionally, regular components can assert an *indirect influence* on the SUC (e. g. via heat or EMI) which is depicted on the component symbol by a "fringe" with a suitable label. Fig. 3 shows the arrows for direct communication and an indirect influence.

Components are grouped into the eponymous *shells* to convey further information on the logical or physical structure of a system and to facilitate the definition of the analysis scope. Shells are (with one exception) depicted as closed curves of dashed lines which are aligned in a concentric fashion, similar to the layers of an onion. Each shell must possess a meaningful identifier placed inside the shell's outline which must be unique over the shell model. Therefore, the VISMA shells are non-empty subsets of components (i.e. each shell must contain at least one component) that form a hierarchy, starting from the innermost to the outermost shell. The meaningful definition of shells is an important aspect in the construction of a shell model and directly influences its readability and usefulness.

C. Defining Shells

After selection of the SUC, all components that directly communicate with the SUC have to be identified. This includes:

- Components that provide input to the SUC (e. g. sensors, user input elements like buttons)
- Components that receive output of the SUC (e. g. actuators, user output elements like displays)
- Components that communicate in a bidirectional way with the SUC (e. g. clients, servers)

Each of the identified components directly impacts or is impacted by the SUC and is termed an *Adjacent Component* (AC) of the SUC. The SUC and its ACs are grouped into a shell which is used as the innermost shell of the shell hierarchy as the shell model is further developed. As the ACs are situated at a "hop distance" of 0 from the view point of the SUC, this initial shell is termed the *zero shell* (Z-shell). A

well-formed shell model must contain exactly one Z-shell which must consist of only the SUC and its ACs linked by respective communication arrows.

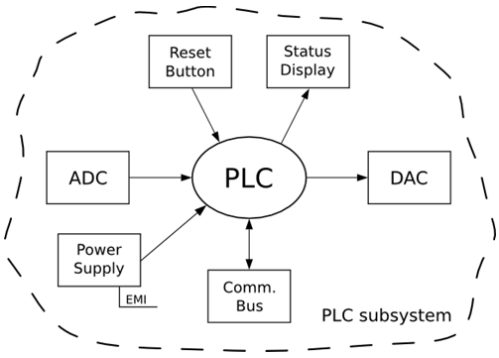

Fig. 4 Example of a zero shell depicting a Programmable Logic Controller (PLC) and its ACs

Fig. 4 shows a typical example of a Z-shell with a generic PLC as the SUC together with its surrounding components as the ACs. Note the bidirectional communication to the component "Communication Bus", which is defined as an AC even though it could physically extend from the SUC by quite a distance.

After definition of the Z-shell, further analysis might identify components that provide input or receive output from the ACs of the SUC, i.e. communicate only indirectly with the SUC. These "next order components" are placed outside the Z-Shell and are grouped together into a new shell that fully encloses the Z-shell. This *outer shell* (O-shell) of order one (O_1-shell) therefore contains only components with a distance of one hop from the SUC.

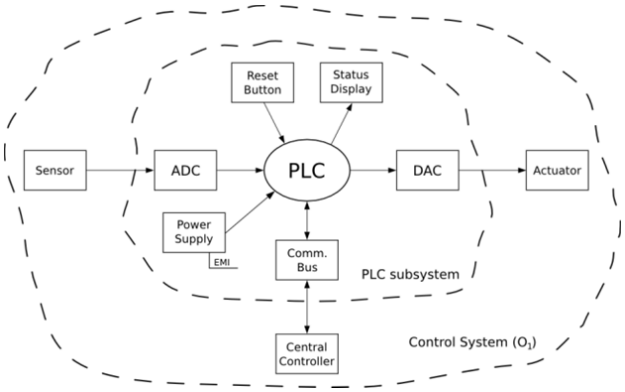

Fig. 5 Adding the first O-shell with indirectly communicating components

Fig. 5 shows how the running example of the PLC is extended by an O_1-shell containing components that provide/receive indirect input (Sensor) or output (Actuator) to/from the SUC. Note that some ACs might not have a corresponding component in the O_1-shell.

The principle of identifying indirect senders/receivers and adding them by means of outer shells can be repeated an arbitrary number of times. This results in a hierarchy of shells with increasing order, where a higher-order shell encloses all shells of smaller order. The amount of O-shells to add is a matter of judgement by the constructor of the shell model and, in general, depends mainly on the complexity of the analyzed system and the intended scope of the analysis. After adding the last O-shell, this outermost O-shell is designated as the border of the system and is termed the *system border shell* (S-shell). To facilitate visual recognition, the border of the S-shell is drawn with a solid line.

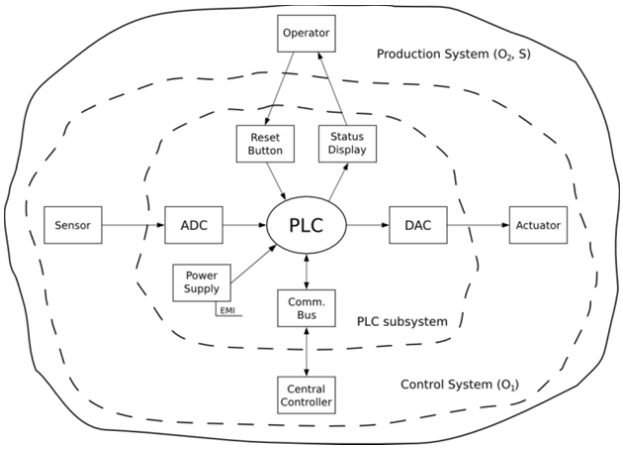

Fig. 6 Adding a second O-shell (O2-shell) as the S-shell of the example system

Fig. 6 shows the addition of a second O-shell (O_2-shell) with a component "Operator" representing the user interacting over the user interface elements with the SUC. As the construction should end at this point, the O_2-shell is marked as S-shell by its solid line. Of particular note is that the "Operator" communicates directly with two ACs, effectively "skipping" the O_1-shell. The order of an O-shell should therefore not be conflated with the "hop order" of its components. Note however, that only O-shells might be skipped in this way - direct communication between the SUC and a component of an O-shell is not allowed.

D.Further Shells

The shells contained within the system border describe the structure of the analyzed system and can be considered as an "internal" environment for the SUC. Information on the environment external to the system is added via two specialized shells.

The *environment shell* (E-shell) surrounds the S-shell and contains elements that represent *environmental influences* that might plausibly affect the SUC or regular components. A typical example is the influence of high or low temperature on the performance of electronic components. Environmental influences differ from the indirect influences exerted by regular components in that they originate from outside the system and are not limited to E/E/PE-related phenomena. Each environmental influence is identified by a suitable identifier (e. g. "heat", "moisture", “vibration”).

The *use case shell* (U-shell) surrounds the E-shell and contains *use case elements* that represent usage scenarios or situations of the system and the SUC. The intention is to define a set of disjoint "profiles" for further analyses, which can be used to analyze the SUC under different circumstances (e.g. low vs. high demand).

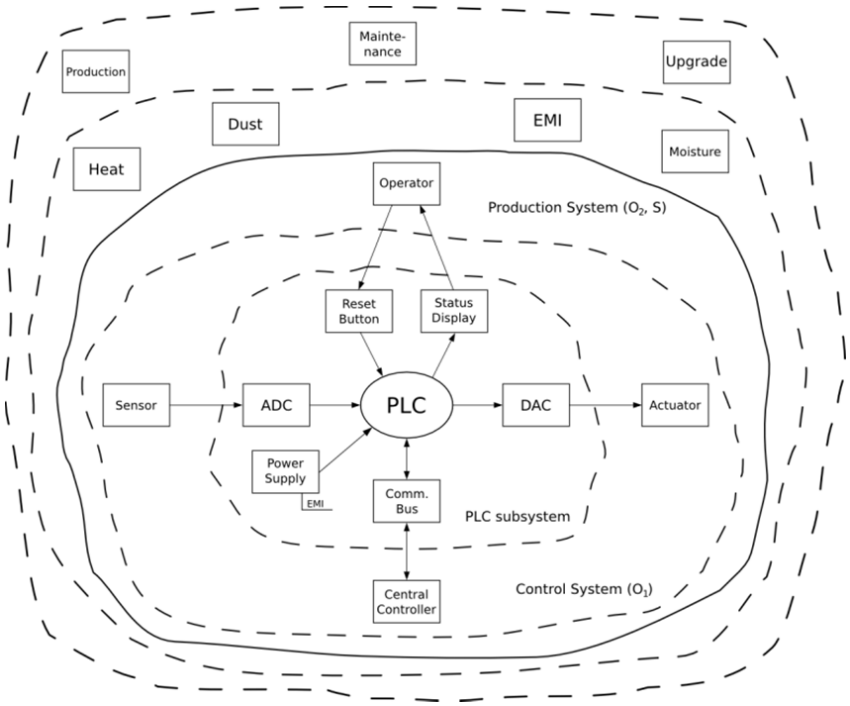

Fig. 7 Complete PLC example with added E-shell and U-shell

A complete shell model for the running example is depicted in Fig. 7. Note that no communication arrows originate or end at elements of the E- and U-shell.

E. Complementary Information

The information conveyed by the elements of a shell model is, in general, not enough to fully understand the workings of a system. This is deliberate in order to avoid overloading the graphical depiction. However, for a thorough understanding of the system, additional information is necessary. Specifically, the following complementary data should be specified for a shell model:

- For each component, including the SUC, a description of the component's purpose in the system and its interfaces for direct communication
- For each regular component with indirect influences, a description of each indirect influence asserted by the component and detailed constraints (e. g. exact thresholds for temperature, EM field strength, etc.)
- For each communication arrow, the set of messages that can be exchanged between the communicating components
- For the Z-shell and each O-shell, a description of the shell's meaning (i.e. the criterion for choosing this shell)
- For each environmental influence, detailed constraints similar to the indirect influences
- For each use case, a description of the use case focusing on important influences on the system and SUC (e. g. expected demand rate)

The formal representation of the complementary information is not prescribed by VISMA. One possibility is to use a "data dictionary" - basically a set of simple tables with references to the elements of a shell model. An excerpt of a data dictionary for the running example can be found in the appendix. It is suggested to use a configuration management system for maintaining a shell model and its associated data dictionary.

F. Derivation of System Functions

A shell model is a static depiction of a system's structure. However, it is possible to describe the dynamic behavior from the relationships between communicating components. This allows further analyses of a system's intended and, more importantly, unintended behavior.

A *system function* (or simply, *function*) is defined as an alternating sequence of components and unidirectional communications between them. The most basic function consists of one component, the sender, communicating some energy or information to another component, the receiver. Obviously, only components linked by a communication arrow in the shell model can be part of a function and the communication must match the direction and definition of the arrow. The communication between sender and receiver can consist of a non-empty subset of messages defined for the

communication arrow. By prefixing a proper message subset with a '!', the subset consisting of the set difference between the given subset and the complete set of messages for this arrow can be stated for convenience. In addition, a '*' symbol represents the complete set of messages.

In order to specify changes in the status of a component in response to communications or as part of a function execution, each component is augmented by a *mode*. To symbolize an internal mode change, a dashed arrow is used that links two instances of the same component with differing modes.

Fig. 8 depicts a basic function (a) and an internal mode change (b) in a graphical and sequence-based notation, where the latter represents a function/mode change as a sequence of (component, mode) tuples and {message} subsets. By chaining basic functions and mode changes, complete functions over the components of a shell model can be defined. The component at the start of the chain is termed the *trigger*, the component at the end of the chain is termed the *terminator* of a function.

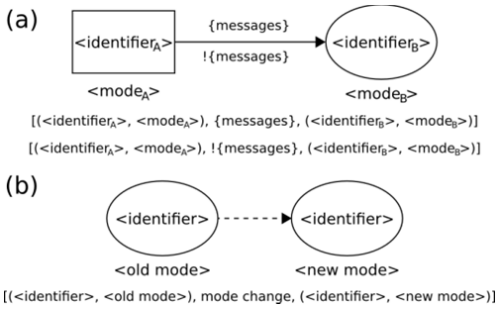

Fig. 8 (a) Basic function and (b) internal mode change in graphical and sequence notation

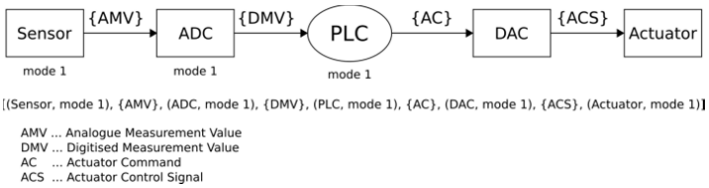

Fig. 9 A simple example function for the PLC shell model

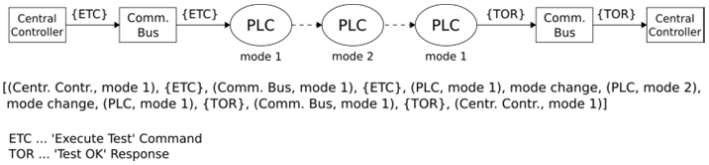

Fig. 10 Example PLC function with an internal mode change of the SUC

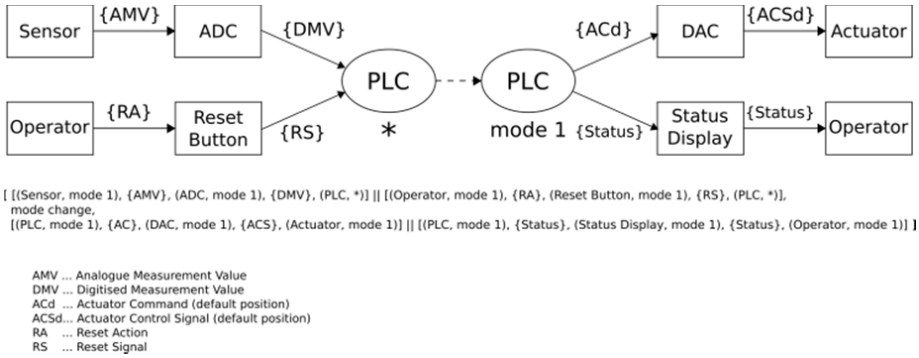

Fig. 11 Example PLC function showing path merging/splitting and wildcard mode

Fig. 9 shows a simple example function for the "PLC" shell model with the components "Sensor" and "Actuator" as the trigger and terminator, respectively. Note that the mode is omitted for some components - in this case, the component is understood to assume its *default mode* ("Mode 1", in this case) which must be defined for each component.

A more elaborate example function is depicted in Fig. 10. Here, the "PLC" undergoes a mode change in response to a communication from the "Central Controller", reverting to the original mode afterwards (supposedly after executing the command). Note that only the order of function steps can be modelled - the real time behavior of functions (such as exact timing for communications and mode changes) is out of scope of the notation.

Fig. 11 shows an advanced example of a function with parallel paths symbolizing input from and output to more than one component. Merging of two or more paths at a component implies that all input communications to the component must

be present at the same time. Splitting of two or more paths at a component does not imply any order, i.e. an indeterministic choice is made as to which path is executed first. Parallel paths are symbolized in the sequence notation as two or more sub-sequences concatenated by '||' with identical first or last tuple.

Note the use of the '*' symbol for the mode of "PLC" prior to the mode change - as expected, this *wildcard mode* states that the component can be in an arbitrary mode.

IV. Modelling Guidelines

Following the syntactical rules of the VISMA leads to well-formed shell models but does not guarantee maximum readability and suitability for further analyses. In the following, a sample of important aspects and advice concerning the construction of practically usable shell models is given.

A. Placement of ACs

In many cases, ACs assume clearly defined roles in their communication with the SUC. By consistently placing ACs according to their role at certain positions relative to the SUC, the amount and ratio of interfaces of the SUC can be easily discerned. The following placement is suggested:

- ACs which only provide input, to the left of the SUC (e. g. sensors)
- ACs which only receive output, to the right of the SUC (e. g. actuators)
- ACs which represent user interface elements, to the top of the SUC (e. g. buttons, displays)
- ACs which communicate bidirectionally with the SUC, to the bottom (e. g. busses, networks)

All examples in this paper follow these rules.

B. Functional Relevance of Communication

When adding communication arrows, the choice should be guided by the relevance of the communication for subsequent analyses based on the shell model. A principle rule is that only communication paths that include the SUC should be added. This excludes arrows between regular components that are not a part of a function where the SUC is involved. An example of such an "irrelevant" communication for the "PLC" shell model would be an exchange of information purely between the "Operator" and the "Central Controller".

For functions where the SUC is included, the scope of communication must be limited to a reasonable amount in order to balance completeness of the system view against complexity of further analyses. For example, a communication between the "Power Supply" and the "Sensor" (which is assumed to be an active one) could be added to the "PLC" example and the derived function in Fig. 9. This would allow an analysis to consider failures of the power supply and its consequences on the function.

C. Logical versus Structural Definition of Shells

Shells can be defined in a variety of ways, with the "ideal" choice depending on the system and the choice of subsequent analyses. Often, the most suitable choice is based on logical distance of regular components to the SUC. In this case, an outer shell's order is synonymous with the amount of components that must be traversed over communication arrows in order to reach the SUC. The O_1-shell in the "PLC" example was defined in this way.

An alternative to logical order is to define shells according to a system's physical layout. In this case, shells of higher order contain components which are situated at a further physical distance from the SUC. By aligning shells with physical demarcations (e. g. containments, fire barriers), analyses regarding the impact of environmental influences on the system components can be facilitated.

Both ways of defining shells may also be combined in the same shell model. For example, the "Central Controller" could be moved to an outer shell of order 3 to symbolize its placement in a facility separate from the location of the "PLC". Note however, that a suitable AC (usually some kind of network) must always be present to link the distant component to the SUC.

V. Application of VISMA

In this chapter, the application of VISMA is shown via an explanatory example. For the sake of brevity, the example is only shown and formulated partially, but should suffice to support understanding.

The basic idea of the example system is to develop a new, motor operated sidestand for motor bikes. Little is known yet about the system except for two requirements:

1. Automatic expansion and retraction can be activated via a toggle switch (“rocker switch”) that is mounted on the handlebar.
2. The current status of the motor bike as well as of the sidestand shall be delivered via CAN bus messages.

A. Constructing the Shell Model

When building up the VISMA system view, first the innermost system in focus, the SUC, has to be selected. From the above requirements it is easy to deduct that the system will contain a small logic unit to control the sidestand, the Sidestand Control Unit (SCU). This will serve as SUC for our further analysis. Furthermore, external converters and drivers are used to keep the SUC small and simple. Two sensors shall detect if the sidestand has reached its upper or lower end position to make sure that the sidestand is either fully retracted or expanded, providing the SUC with the correct information. Further considerations for the building up of the system resulted in the shell model depicted in Fig. 12. For this shell model, the logical distance order is preferred. Note, however, that not each component in the S-shell is at a "hop distance" of 2 from the SUC. This is to emphasize the physical distance between the “Vehicle Battery” or “Bike Motor Control Unit” and the SUC. The reasoning for this design choice is rather simple: those components provide necessary functionality to the sidestand sub-system (contained in O_1-shell in Fig. 12), but are not part of it.

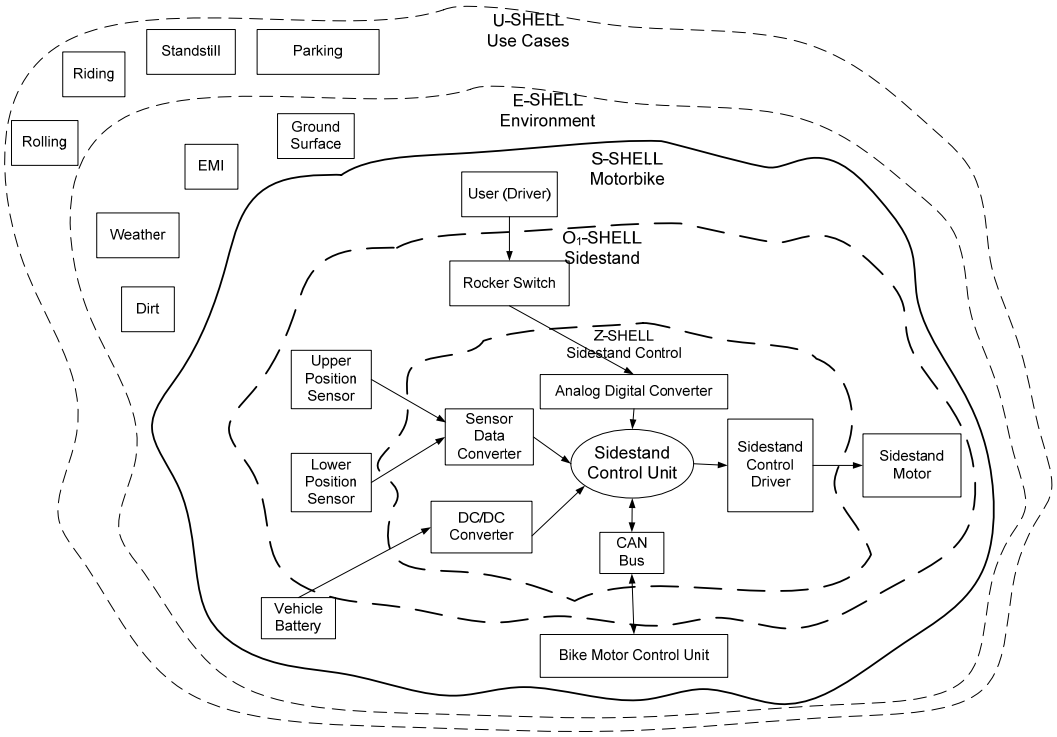

Fig. 12 The shell model for the sidestand example

B. Sidestand Functions

In order to derive the functions, all potential function triggers must be identified first. Components at the begin of a communication line to the SUC are prime candidates for triggers. Looking at Fig. 12, one obvious trigger is the component "User (Driver)."

Fig. 13 shows the corresponding function derivation for the "sidestand expansion" function, following the shell model in Fig. 12 upon the "Driver Action Down" trigger event (i.e. message).

This function consists of three input paths. On the first path, the "User (Driver)" triggers the "Driver Action Down" by pressing the "Rocker Switch" to the downward position, forcing the "Rocker Switch" to change its mode to "Down pressed". This sends the "ASD" then "DSD" to the SUC. On the second path, the "Lower Position Sensor" does not detect the position of the sidestand, identified by the "DSN" signal (while the "Upper Position Sensor" is irrelevant for this function), thus the "Sensor Data Converter" sends "!SPB" signal to the SUC. On the third path, the "V0" message is transferred from the "Bike Motor Control Unit" via the "CAN Bus" to the SUC. The SUC reacts with a mode transition to "Expanding" mode which results in the "Sidestand Motor" changing its mode to "Left turning."

Note that for enabling the mode transition of the SUC all three messages must be present at the same time: "ASD", "!SPB" and "V0".

Fig. 14 shows a similar function as Fig. 13, but this time the trigger is the detection of the upper position by the "Upper Position Sensor" ("DSD" signal), which results in the SUC switching to "Idle" mode, and the "Sidestand Motor" switching to "Stop" mode respectively. Note that the "User (Driver)" who initiated the "DAU" earlier, does not release the "Rocker Switch" meanwhile, as the "DSU" is still active at the SUC input. Note also that the "CAN Bus" status is irrelevant in this case.

C. Use of Derived (Sidestand) Functions

The functional derivations of the VISMA are supporting several outputs which are valuable, or even necessary, input to further analysis methods. This way, top level functions can be identified and described. E.g. for the sidestand example, the two functions for "Sidestand Expansion" triggered by the "User (Driver)," and for stopping the sidestand triggered by reaching the upper position, can be formulated, including their logical conditions with respect to the inputs.

Consideration of possible functional deviations can be facilitated this way, which can lead to undesired modes, events or situations and may even result in a first list of (functional) hazards. E.g. for the expansion function in the sidestand example in Fig. 13, a possible functional deviation would be the "CAN Bus" always sending "V0" for some reason. This results in the hazard that the expansion function could be activated by the "User (Driver)" although the vehicle is moving.

The functional descriptions can also form the input to a HAZOP analysis. E.g. using the function diagrams of the sidestand example, each single component can be subjected to a wrong mode change, and each communication could be subjected to a wrong message or message set transferred,

resulting in the typical deviations in a HAZOP analysis.

The different components linked throughout the shells can support a first structural or functional FMEA, even though at a high level, and ease the analysis of the consequences therein due to the holistic system view. E.g. in an FMEA of the sidestand example, each single component of the Z-shell (including the SUC) as in Fig. 12 could be analyzed for the causes for and direct consequences of each possible failure mode. The effects of the failures can be followed through the shell model till a *terminator* has been reached.

The undesired events or found hazards can be used for first high-level FTAs, if reasonable at this point in the project. E.g. the already mentioned hazard for the sidestand example (Activation of expansion function by the "User (Driver)" although the vehicle is moving) could be starting point for one fault tree, in order to look for the possible causes. However, the scope of such an FTA is limited to stop at single component level.

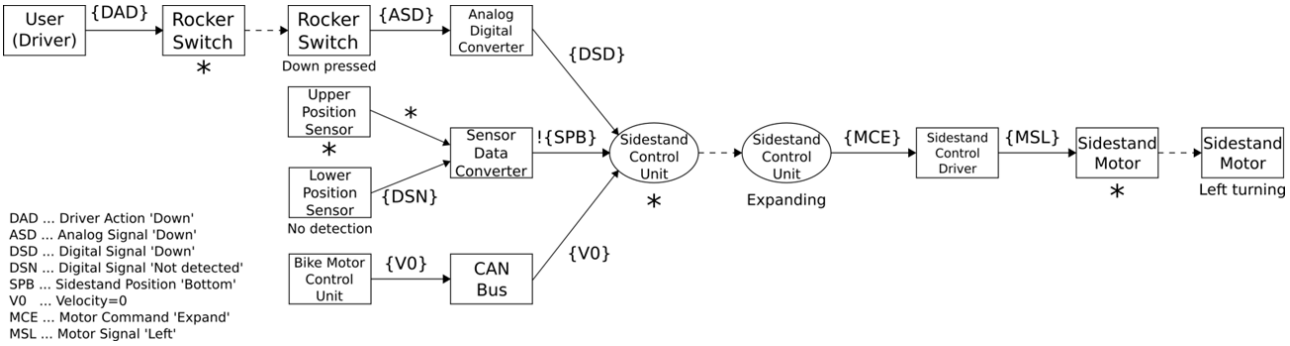

Fig. 13 "Sidestand Expansion" Function

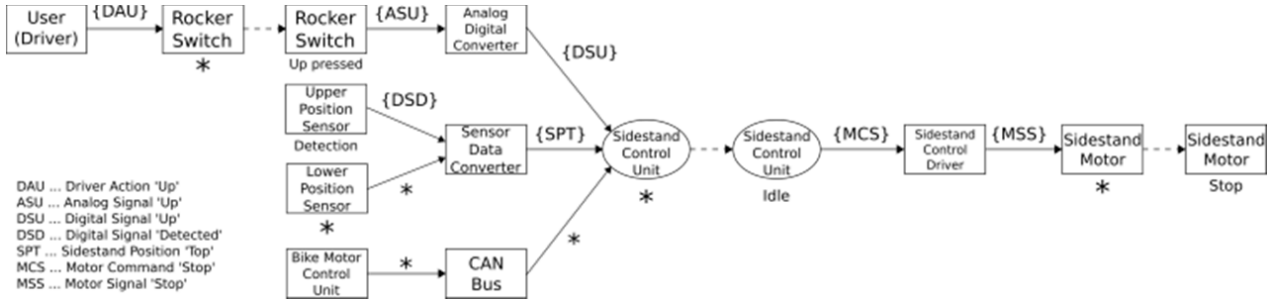

Fig. 14 "Sidestand Stop" Function upon reaching upper position

VI. Further Considerations

A. Benefits of the VISMA Method

As VISMA is representing a method at a very early lifecycle stage, its findings and results can be used as inputs for further analysis methods. This way, VISMA closes the method gap at this stage, providing input to:

1. HAZOP Analysis
2. FMEA
3. Qualitative FTA, limited to single component level
4. Further requirements analysis
5. Use-case and scenario analysis
6. Further functional analysis

B. Limitations of the Method

VISMA only supports qualitative analysis, as quantification is infeasible at such an early lifecycle stage. Also, VISMA is not suitable for describing or handling parallel structures in the sense of redundant solutions. As of now, VISMA is restricted to electrical, electronical, and programmable electronic (E/E/PE) systems.

C. Future Prospects

Application of VISMA to technologies other than E/E/PE systems (such as mechanics, hydraulics, and pneumatics) and necessary extensions to the method are currently under investigation.

As limitations of "classical" drawing tools became apparent during development of the method, a specialized tool that implements the method would significantly facilitate the application of VISMA.

Appendix

Figs. 15-26 are excerpts of the data dictionary for the example in Fig. 7.

Identifier	Description	Interfaces
PLC (SUC)	PLC for machine control	Input: Digital In x2, Power In (24Vdc) Output: Digital Out x2 Bidirectional: RS-485 interface
ADC	External ADC	Input: Analog In Output: Digital Out
DAC	External DAC	Input: Digital In Output: Analog Out
Power Supply	External switching power supply	Input: 230Vac Output: 24Vdc
Reset Button	External, debounced reset button	Input: User action Output: Digital Out

Fig. 15 Component list (excerpt) for example in Fig. 7

Shell	Description
PLC subsystem (Z)	PLC inside PLC cabinet
Control system (O_1)	PLC cabinet plus control elements
Production system (O_2, S)	Control system plus operator

Fig. 16 Shell list for example in Fig. 7

Unidirectional			
Sender	**Receiver**	**Type**	**Message Set**
ADC	PLC	Digitised measurement value	{Digitised measurement value}
Power Supply	PLC	Supply voltage	{No voltage, Arbitrary voltage, Nominal voltage}
Reset Button	PLC	Reset signal	{No Reset, Reset}

Fig. 17 Unidirectional communication (excerpt) for example Fig. 7

Bidirectional			
Peer	**Peer**	**Type**	**Message Set**
PLC	Comm. Bus	Control commands/responses	{'Execute Test' Command, 'Test OK' Response, ...}
Central Controller	Comm. Bus	Control commands/responses	{'Execute Test' Command, 'Test OK' Response, ...}

Fig. 18 Bidirectional communication for example in Fig. 7

Component	Indirect Influence	Description
Power Supply	EMI	Radiation EMI of switching power supply, 70µV/m@1GHz

Fig. 19 Internal indirect influences for example in Fig. 7

Identifier	Description
Heat	Heat buildup in PLC cabinet due to thermal radiation of nearby machines, maximum expected temperature +60°C
Dust	Accumulation of metal particles in PLC cabinet from metal-cutting work, grain sizes of 50-100µm
EMI	Radiation EMI from nearby machines and wireless networks, limits acc. to ISO 62061, Annex E
Moisture	Moisture buildup in PLC cabinet due to water-based coolant of nearby machines, expected absolute air humidity in cabinet 10g/m³

Fig. 20 Environmental influences for example in Fig. 7

Identifier	Description
Production	Default use case, continuous demand on PLC for control purposes
Maintenance	Maintenance of machines, no demand on PLC, MTTR = 1d
Upgrade	Upgrade and/or replacement of PLC

Fig. 21 Use cases for example in Fig. 7

Sender	Receiver	Type	Message Set
Sidestand Control Unit	Sidestand Control Driver	Motor command	{Motor Command 'Expand', Motor Command 'Retract', Motor Command 'Stop'}
Upper Position Sensor	Sensor Data Converter	Sidestand position detection information	{Digital Signal 'Detected', Digital Signal 'Not Detected'}
Lower Position Sensor	Sensor Data Converter	Sidestand position detection information	{Digital Signal 'Detected', Digital Signal 'Not Detected'}
Sidestand Control Driver	Sidestand Motor	Analog motor signal	{Motor Signal 'Left', Motor Signal 'Right', Motor Signal 'Stop'}
User (Driver)	Rocker Switch	Driver action	{Driver Action 'Down', Driver Action 'Up', Driver Action 'Neutral'}

Fig. 22 Unidirectional communication (excerpt) for example Fig. 12

Peer	Peer	Type	Message Set
Sidestand Control Unit	CAN Bus	Vehicle data messages	{Velocity=0, ...}
Bike Motor Control Unit	CAN Bus	Vehicle data messages	{Velocity=0, ...}

Fig. 23 Bidirectional communication for example in Fig. 12

Identifier	Description	Interfaces	Modes
Sidestand Control Unit (SUC)	Control unit for sidestand control	Input: Digital In x2, Power In (5Vdc) Output: Digital Out Bidirectional: CAN interface	Expanding Retracting Idle
Rocker Switch	3-position switch (down, up, neutral)	Input: User action Output: Analog Out	Up pressed Down pressed Neutral
Upper Position Sensor	Limit switch for detecting retraction of sidestand	Input: Position Output: Digital Out	Detection No detection
Lower Position Sensor	Limit switch for detecting expansion of sidestand	Input: Position Output: Digital Out	Detection No detection
Sidestand Motor	Electrical motor driving sidestand expansion or retraction	Input: Analog In Output: Torque	Left turning Right turning Stop

Fig. 24 Component list (excerpt) for example in Fig. 12

Shell	Description
Sidestand Control (Z)	Control unit plus auxiliary components
Sidestand (O_1)	Control HW plus external input elements and motor
Motorbike (O_2, S)	Complete vehicle

Fig. 25 Shell list for example in Fig. 12

Identifier	Description
Rolling	Vehicle moves, engine running
Riding	Vehicle moves, engine not running
Standstill	Vehicle doesn't move
Parking	Vehicle rests on sidestand

Fig. 26 Use cases for example in Fig. 12

REFERENCES

[1] C. E. Ericson, II, *Hazard Analysis Techniques for System Safety*, 2nd ed., New Jersey: Wiley, 2016, ch. 5.2.2
[2] J. McDermid, "Issues in development of safety-critical systems," *Safety-critical Systems*, first ed.London: Chapman & Hall, 1993, pp. 16–42.
[3] IEC, *IEC 61508-6 Functional safety of electrical/electronic/ programmable electronic safety-related systems*, ed. 2.0, part 6, Geneva: IEC, 2010
[4] IEC, *IEC 60812- Analysis techniques for system reliability – Procedure for failure mode and effects analysis (FMEA),* ed. 2.0, Geneva: IEC, 2006
[5] U.S. Nuclear Regulatory Commission, *NUREG-0492 - Fault Tree Handbook*, Washington: U.S. Government Printing Office, 1981
[6] C. E. Ericson, II, *Fault Tree Analysis Primer*, 2nd ed., Charleston: CreateSpace Independent Publishing Platform, 2011
[7] R. Preiss, *Methoden der Risikoanalyse in der Technik*, Vienna: Edition TÜV Austria, 2009
[8] H. Tschürtz, *Safety-Vorgehensmodell zur Konzeption und Entwicklung von sicherheitskritischen Systemen*, Dtechn Thesis, Institute for Engineering Design and Logistics Engineering, Vienna University of Technology, Austria, 2016