DEFINITION OF STATES AND MODES AS GENERAL CONCEPTS FOR SYSTEM DESIGN AND VALIDATION

R. BADUEL [1,2] **J.-M. BRUEL**[1] **I. OBER**[1] **E. DOBA**[2]

[1]IRIT / University of Toulouse
31400 Toulouse - France
{ronan.baduel,bruel,iulian.ober}@irit.fr

[2]Bombardier Transport
59154 Crespin - France
eddy.doba@rail.bombardier.com

ABSTRACT: *This paper advocates the need of a precise definition and consistent use of the notion of states and modes in system engineering and why this is important for the early validation of system behavior. This work is part of a solution developed for Bombardier Transport in order to provide continued validation of a train behavior across all development steps. The aim is to define concepts that support and generalize the application and use of states and modes in a way that is compatible with current practice, languages and tools. The finality of our work is to provide models that specify and integrate the whole system behavior at a high level of abstraction (in both granularity and details of the information characterized), enabling its validation and then supporting traceability of the behavior and its validation across the development process. This work is conducted under the principles of Model Based Systems Engineering (MBSE) and is linked to the use of SysML models. The examples presented are taken from the train industry.*

KEYWORDS: *Early Validation, MBSE, SysML, State, Mode*

1 INTRODUCTION

Specifying a system behavior is generally done early in the development process through behavior models, such as state diagrams. If we consider the state diagram in SysML (OMG, 2017), which comes from UML (OMG, 2015), we see that what a state represents regarding the system or its behavior is not specified. It is an issue raised by Naumenko in his thesis (Naumenko, 2002) regarding UML: "*existing UML semantics are very ambiguous in presenting relations between models constructed using the language on the one hand and the subject that is being modeled on the other hand*". While the representation and semantics of states regarding the model are specified in the standard, it is not clear what a state actually represents regarding the system to be modeled. Aside from UML, there are "state" elements or concepts used in different languages, tools and methods with different semantics, such as UPPAAL (Behrmann et al., 2006) or in Abstract State Machines (ASM) (Börger & Stärk, 2003).

There is no consensus on what a state is (Wasson, 2010 & Olver, 2014), and it is often confused with the notion of mode, which definition is also debated. When referring to the standard ISO/IEC/IEEE 24765, which references other standards, we see that there are a variety of definitions of what a state is. Since there is a confusion between states and modes but both are widely used in systems engineering, it is necessary to define both concepts in the scope of our work. Bombardier Transport (BT) uses states and modes in its models but lacks a definition of the corresponding concepts. This leads to a difficulty in capturing and integrating the right information (such as preconditions of functions) in executable models, which are themselves state machines.

Beyond the definitions of states and modes, we are interested in their utilization. While in SysML the states are used to model a behavior, we see in the definitions proposed by Wasson that we can define different kinds of states and modes, capturing different information. Such information can describe or impact the behavior of the system without being able to fully describe, condition or model it. When modeling the system behavior in a SysML state machine, as a synthesis of the information on those conditions, the information used for the conditions has its own evolution (transitions) with specific conditions. This leads to the creation of other state diagrams or constraints on variables characterizing information about the system and not its behavior. It means that we have to identify and define each type of condition-related information in order to model the behavior.

This paper is organized as follows: in section 2 we present some issues encountered in the train industry that we address through a proper definition of states and modes. In section 3 we analyze different definitions of states and modes and propose our owns, for

which we present a concrete application in section 4. Section 5 is a conclusion on what is presented in previous sections and introduce the future applications of this work for early validation.

2 CONTEXT AND NEEDS

BT has expressed the following needs concerning the train design:

- There is a need for means to validate the behavior early and then across the whole development process. This is currently done in a test-driven approach, which does not cover unwanted and unexpected behaviors nor does it validate specifications before implementation.
- There is a need for the reuse of the functional specifications of a train. While train components and subsystems are well-mastered, what changes from one train to another is the functional specification, which is difficult to reuse, change or adapt.
- There is a need to model the information used in preconditions of the system functions and scenarios. They are currently specified only in requirements, matrix and spreadsheet files which renders them difficult to analyze and refer to in behavior specifications.

In order to validate the train behavior at each step of the development process, we need an abstract representation of the expected behavior that we can compare to the implementation. This is why we need an executable model integrating the information of the specifications. This model would be built around state machines. We aim for an approach similar to the one using the OMAG tool (Chapurlat & Daclin, 2013). Such a representation would allow to define a general and reusable functional structure, which is part of BT needs. BT is developing a solution using state machines to represent lower levels of granularity, such as functions in subsystems. Those executable models would include preconditions and information linked to them, which we have to represent. Such a representation exists for example for the train operability (the train readiness and activation status regarding the functions that can be performed), using an informal state machine. We want to formalize this process and extend it to other kinds of information.

As each language and tool have their own definition and utilization of the notion of state and/or mode, the concepts defined here will refer to common characteristics taken from several sources. They are developed so as to be useful in the scope and needs defined by BT but are not specific to the train domain and could be used for general system development.

3 STATES AND MODES

3.1 State definition

3.1.1 State of the art

We consider several definitions:

- According to the standard ISO/IEC/IEEE 24765, a state can be a characterization of the system at a given time, the value of the variables defining the system, a condition to a behavior or a function, something that determines the set of functions that can be performed, and other meanings.
- According to a system engineering handbook supported by INCOSE (Walden et al., 2015): "*A system is in a state when the values assigned to its attributes remain constant or steady for a meaningful period of time*". It is otherwise specified that the handbook refers to the norm ISO/IEC/IEEE 24765 for vocabulary.
- According to (Naumenko, 2002), a state is one of two concepts at the basis of the semantic he defines, and its definition is: "*an information about a thing (object) at a given time (point in continuum) inside a context (time continuum)*".
- According to (Wasson, 2010), a state is "*An attribute used to characterize the current logistical employment, status, or performance-based condition of a system*".

Not all definitions available or considered are given here, only a representative group issued from formal or common sources. Some of the sources cited in this document, such as (Olver & Ryan, 2014), have already done similar work and reference different definitions, and can be consulted as additional references.

Besides the definition of states, we consider different kinds of elements used to represent them. In many finite state-machines such as UPPAAL, we have one state active at a time inside a state space without hierarchy or concurrency, at least in one diagram. In UML and SysML state machines or in Harel statecharts (Harel, 1987), we have complex state structures with concurrent states in one diagram, meaning we can have several states active at the same time. In Abstract State Machines, or ASM (Börger & Stärk, 2003), the state of the system is represented by a set of variables, called states variables, and a state is the valuation of these variable at a point in time. This can be linked to one of the definitions found in ISO/IEC/IEEE 24765. As in state diagrams, ASM allows to define transitions, conditions and constraints.

Another point is that state can be represented by discrete or continuous components, something that is highlighted in hybrid automata (Henzinger, 2000).

The state definition in UML can not be used as a definition of the concept. As per the analysis made by Naumenko, these elements are not directly associated to the system of study. In addition, UML defines states to model a behavior in what is called "behavioral state machines", but we can define states that affect the behavior without modeling it. Taking the example of a train, we can define a state that indicate the current energy supply, be it internal or external sources. It will affect its operability and hence its behavior but does not condition capabilities directly. States can capture information used to condition capabilities. This information cannot be used to directly model the behavior, as each piece of information could be used as part of the preconditions of any capability. To represent the behavior, we have to define elements characterizing the execution of capabilities and not the information used to allow it, which seems to be the goal of the state machines in UML. This is supported by the definition found in the standard UML 2.5 (OMG, 2015): speaking of the State model element,"*a state models a situation during which some (usually implicit) invariant condition holds*".

3.1.2 Analysis

Cross-referencing the definitions given above, we can establish a few characteristics of states:

- They characterize a thing (e.g., a system).
- They relate to a specific kind of information, knowledge domain or way of being regarding this thing (e.g., operations, readiness, energy, ...).
- They are evaluated or considered *at a given time*.

The most appropriate definition would be the one proposed by Naumenko, as it is high level, relates to the object of study that we want to model, and summarizes the main characteristics of states that are common in the definitions considered. In addition, it was thought as a core element of a semantic used to give a common meaning and understanding in the definition of models regarding the system they represent. It is one of our objectives regarding modes and states: to make it clear how to define them and what they represent regardless of which language or tool we use. It also presents the advantage of considering a temporal context in which states can be defined, which should be the system life-cycle. We will keep this notion of context as it can be applied to the other definitions.

This seems to contradict the definition given in the INCOSE engineering handbook (Walden et al., 2015), but we argue that it is not the case. A state will always be considered at a point in time, where it will be evaluated. If we get the same values or few variations of them in regard of the information characterized during meaningful periods of time, then they can be abstracted as fixed values over time. Searching for persistent values or intervals of values can be a criterion to create state diagrams or other such models, but we consider that is it not a part of the state concept definition. We consider that a state has one fixed value at a point in time, and it can be different or not from the value at the previous or next point in time. That is why states can be modeled by variables as well as diagrams.

In SysML state machines, we can have several "states" active at the same time. By "state" we mean the modeling element rather than the concept. An "active state" is either one of the higher level "states", or one of the "sub-states" contained in a higher level "state" or a concurrent partition. It means that there can be only one "active state" in a group of "states" or "sub-states" at a same hierarchical level and per concurrent process. We hence consider that each of these "active states" exist in their own context. It enables us to keep the definition (concept) of state as an evaluation in a time context, though defining several types of states characterizing different targets in a same diagram may not be pertinent.

The point of view from which the state is defined is also important. If we consider an unmoving train and want to evaluate its operational use, then depending on the person, it could be considered "parked", "waiting", "in a mission", "reserved", "ready for departure", etc. When evaluating an information through the measure of physical attributes, the point of view indicates who access the information: the train control system, the pilot or an external monitoring system. Depending on how the information is accessed and on the scope of study, there can be differences in the information received.

Engineers try to define a hierarchy of states. For example, Wasson (Wasson, 2010) considers that "system states" (characterizing the use being made of the system) contain "operational states" (characterizing the mission readiness of the system). But it seems logical that a system could have a same readiness status in two different missions. Acknowledging that Wasson's scope is oriented to the definition of scenarios and mission specifications phase by phase, we consider that in general two types of state of a same object characterize different information that may have dependencies and constraints between them but exist at a same level.

3.1.3 Definition

Using the definition proposed by Naumenko, we consider the concept of *state* as "*an information about a thing at a given time inside a context*". In order to be independent of the representation of a state, we want to use informal attributes to characterize it. In order to adapt our definition to the development of complex systems, we specify two attributes impacting the information characterized: the point of view and the level of granularity. We saw that the definition of a state was different depending on who is considering the information regarding the object characterized. The level of granularity, here divided into system, subsystem and component level, enables us to define a same information at different level of details along the development process. For example, a same state of the train energy supply can be evaluated as "internal supply depleted", "battery depleted" or a voltage measure depending on the level of granularity. This is also how we could ensure traceability of information and state along the development process, by refining states. According to this analysis, the attributes necessary to define a *type of state* are given in figure 1.

ATTRIBUTE	DEFINITION
Target	The entity characterized
Information	The type of information considered
Context	The full scope under which the state exist and can be defined
Granularity level	The level of granularity of the elements linked to the information considered
View	The point of view by which the state is expressed

Figure 1 – Attributes used to define a state type

We link our definition of *state* to concepts enabling to model it using different languages and modeling objects:

- A *type of state* is a type of information linked to the object it characterizes and the time context where both the information and the object are defined. A type of state is defined using a set of states variables.
- A *state variable* is used to express the information characterized by a type of state. A state variable has values defined over the whole time context where the type of state is defined.
- A *state space* is the set of all possible configurations of values of the state variables of a type of state.
- A *state* is the evaluation of the state variables of the type of state it belongs to. Each Element of a state space correspond to a possible state.

We do not consider modeling concepts such as transitions, initial states or guards in our definitions. Our goal is only to know what a state represents regarding the object to model and what are the kind of elements needed to model it. The representation and evolution depend on the modeling language.

3.1.4 Example

We illustrate our definition on a simple example: checking whether a common type of door is open or closed. Our target is a door, the information is the opening status and the time context is the time where the door remains installed and in good shape. The point of view is the one of the door, the level of granularity is the door as a system. We use as a *state variable* the angle made by the plane of the door regarding the one of its frame. The *state space* is any value between 0 and 90 degrees, assuming that the door can not be opened further than at a right angle. The *state* of the door is evaluated by measuring the angle at a given time. If the angle is less than 5 degree, the door is considered closed. We can abstract the *state variable* as one having for state space only two elements, "open" and "closed", as it is the only information we are interested in. We can then represent the *state variable* in a state diagram, with two state values. We see that a same *type of state* can be represented in several ways, and that is does not necessarily keep the same information value for extended periods of time, as we could place the door at an angle over 5 degree just for an instant before closing it again.

3.2 Mode definition

3.2.1 State of the art

Again, we consider several definitions:

- According to the standard ISO/IEC/IEEE 24765, a mode is a set of related features or functional capabilities of a product.
- According to the SMC Systems Engineering Handbook (SMC, 2005), "*The condition of a system or subsystem in a certain state when specific capabilities (or functions) are valid*".
- According to Wasson (Wasson, 2010), a mode is "*An abstract label applied to a user (UML Actor) selectable option that enables a set of use case-based system capabilities*".
- According to the Arcadia method (Voirin, 2017), a mode is "*a behavior expected of the system [...] in some chosen conditions*".

Modes are linked to actions, functions or capabilities. As there are debates about what is a state and a mode, we also analyze states definitions that relate to capabilities:

- According to the standard ISO/IEC/IEEE 24765, a state can be "*something that determine the set of functions that are possible or can be performed*".
- According to the SMC Systems Engineering Handbook (SMC, 2005), a state is "*the condition of a system or subsystem when specific modes or capabilities (or functions) are valid*".
- According to the Arcadia method (Voirin, 2017), tooled by Capella, a state is "*a behavior undergone by the system [...] in some conditions imposed by the environment*".
- According to Jenney (Jenney, 2011), "*States define an exact operating condition of a system, where modes define the set of capabilities or functions which are valid for the current operating condition*".

3.2.2 Analysis

Considering those definitions and other sources, we have identified several attributes and characteristics about modes:

- They characterize the behavior of a thing (e.g., the behavior of the system under this mode).
- They express a behavior regarding a set of capabilities, functions or actions (e.g., moving forward or backward, performing flight maneuver, etc.).
- They are defined for a set of conditions (e.g., specific states of the system).

We use the following definition found in ISO/IEC/IEEE 24765 for the concept of behavior: "*the peculiar reaction of a thing under given circumstances*". There is a distinction between the behavior characterized and the behavior expressed. The former is the whole behavior of the thing characterized, hence its reaction under any circumstances. The behavior expressed by a mode represents part or whole of the behavior of the thing, that is specific reactions under the conditions for which a mode is defined. For example, there are capabilities on a smart-phone that are valid when the phone is connected to a wifi, and that can be characterized by a "wifi mode", but this mode does not characterize capabilities such as calls, SMS, etc. which are part of the global behavior of the phone. "Wifi mode" could have sub-modes, such as "automatic updates", meaning we can characterize behaviors that are not always those of the phone, but those of its main capabilities.

Behavior under specific conditions is something that we find in UML "states". "State" elements in UML enable to call operations and are used to model a behavior. Regarding the "sub-states", we saw they could not always be considered as state variables, making it hard to link them to the state concept. However, according to the criteria listed above, UML "states" and "sub-states" elements could correspond to modes regarding the concept. A type of mode is not always defined for the whole time context of the object it characterizes.

Individual capabilities can be characterized by more than one mode. They can have their own conditions or events needed to execute them, which can be modeled as preconditions (guards) or with others modes. Most capabilities in a train are conditioned by the train operability modes, as what can be done depends on the energy supply and the level of activation. But there are then functional modes conditioning their execution even when they are technically possible, like opening doors or going forward in a train fully activated. Modes can share the same conditions, meaning they can exist at a same time, but still be independent from each other, meaning we can not build a hierarchy around them. That may be why in the UML definition of state, which we associate to the notion of mode, the conditions are said to be often implicit.

Being in a mode or not can be an information characterized by a state. We can have a set of modes linked by transitions and defined over the whole system lifecycle. But a mode is a notion different from the one of state. A state is evaluated at a point in time, and its value(s) can possibly hold only for an instant. A mode is defined independently from time, as its conditions can hold only for an instant but characterize behaviors that last in time. As a mode is linked to conditions on state values, it may be why states are sometime defined as values holding for a period of time. We can conclude that while modes and states can be linked, they correspond to two different notions. A state characterizes a way of being of the system at a given time, a mode characterizes the way the system behaves for a particular state value. A type of state represents some knowledge about the system, a mode is part of the specification of its behavior.

Note that a mode can be abstract. A mode could for example characterize capabilities according to the way the system is used, meaning characterizing a performed behavior, which may not cover all potential behaviors. There are capabilities one wishes to in-

hibit, authorize or constrain but not as part of the system specifications. For a train, there are restrictions when you drive in a station or in an urban area, representing an information that could be included in a model of the behavior.

3.2.3 Definition

We define a mode as a characterization of a set of capabilities of a thing under a set of invariant conditions. It specifies part of its target's behavior. To characterize a mode, we need to know the source of the behavior, the kind of characterization made, the capabilities and the conditions. In order to adapt this definition to system modeling, we have to consider another attribute: the level of granularity, which characterize the level of detail of the capabilities and whether they are attributed to the system, a subsystem or a component. The point of view is always internal, as we characterize the system capabilities. The conditions and characterization could originate from an external source, for example if we define modes of utilization, but the behavior characterized would be either the system's or the user's. According to our previous analysis, the attributes necessary to define a mode are given in figure 2.

ATTRIBUTE	DEFINITION
Source	The source of the behavior
Capabilities	The capabilities characterized by the mode
Conditions	The conditions for which the mode is defined
Characterization	The ways capabilities are characterized by the mode
Granularity level	The level of granularity linked to the capabilities characterized

Figure 2 – Attributes used to define a mode

We then link this definition to concepts enabling to model it using different languages and modeling objects:

- A *type of mode* specifies the behavior of a thing (e.g., system, function, group of functions) regarding a specific scope of study (operability). This scope is defined by sets of capabilities linked to a kind of characterization and sets of conditions.
- A *capability* is something realized by the thing characterized (e.g., actions, operations or functions)
- A *characterization* is a constraint put on a capability (e.g., enabling, conditioning, inhibiting or calling)
- A *set of conditions* is defined by conditions on a set of states variables
- A *mode space* is the set of all possible configurations of characterized capabilities and sets of conditions for a type of mode
- A *mode* is a behavior of a thing. Each element of a mode space correspond to a mode.

3.2.4 Example

We illustrate our definition using the door example from earlier: we qualify the behavior of a door. We characterize the *capability* of a door to be opened. The *characterization* of this *capability* is enabling it. The *condition* for which the *mode* is defined is that the door is unlocked. The *level of granularity* is the one of the SOI. At the *condition* that the door is unlocked, we are in a *mode* where the *capability* "open the door" is enabled. Whether the door is locked or not could be evaluated with a *state*, which would not characterize the behavior.

4 APPLICATION

4.1 State and mode definition

An example of a state and mode definition according to our approach is given here. We start from a real train state diagram given in figure 3, which is used to specify scenarios and represent high-level conditions of the main functions of a *consist* (a consist is a vehicle forming a train. A train can have several consists). This diagram is also part of a document describing an offer to a client, meaning BT is contractually bound to respect its specifications. The guards on the transitions have been removed for more clarity. We use simple graphical representations that are not linked to a specific tool or language, as we could use several of them to represent or use the information considered.

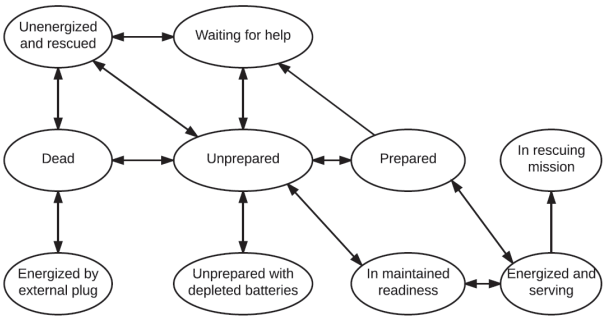

Figure 3 – Original state diagram characterizing the *consist* operability

We now evaluate the attributes that we used to characterize states in figure 4.

ATTRIBUTE	DEFINITION
Target	Consist
Information	Operability, energy supply, mission
Context	Train life-cycle
Granularity level	System or Component
View	Consist

Figure 4 – Operability state attributes

The states in this diagram provide three different kinds of information: the operability, clearly defined, the energy supply, only partially defined, and in one instance the mission (being in a rescue mission). All information are not defined over the whole context, meaning that the state space is incomplete. Mentioning batteries relate to the components granularity level, while the rest relate to system level. To settle those issues, we define two new types of states in figure 5 and figure 6, one for the operability and one for the energy supply. Dismissing the information regarding the *consist* mission, which was not in the original scope, both states now relate to the same level of granularity as the SOI. The other attributes do not change.

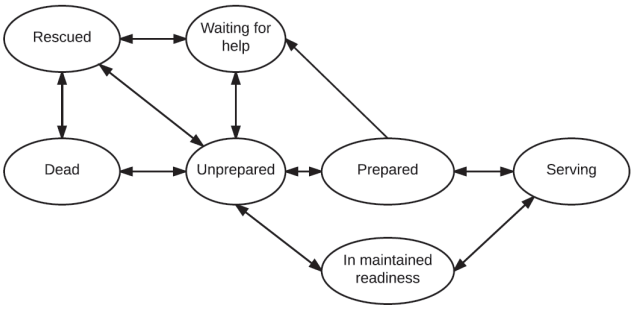

Figure 5 – Operability state of the *consist*

The dot lines in figure 6 characterize possible transitions that were not considered in the original diagram and became evident after a separate analysis. Note that the operability is only partially determined by the energy supply. The next step would be to define the constraints between the operability and the energy supply.

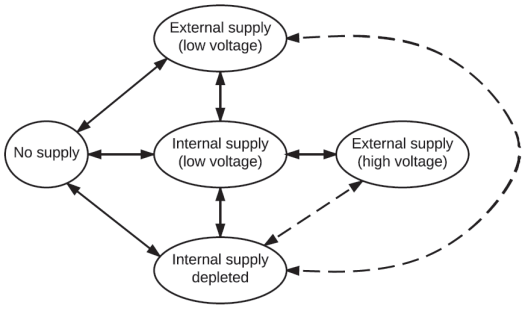

Figure 6 – Energy supply state of the *consist*

The operability state can be linked to operability modes, as we can characterize the train behavior depending on the *consist* readiness state. We define a type of mode for the operability in figure 7. For each mode of this type, we would have to specify the value(s) of the operability state as conditions as well as the detail of every functions and their characterization under this mode. For example, we could define a mode that has for condition the state "dead" and that enables the capability "power the train".

ATTRIBUTE	DEFINITION
Source	Consist
Capabilities	All capabilities of the consist needing energy and system activation status to be performed
Conditions	Operability
Characterization	Enabling capabilities
Granularity level	System

Figure 7 – Attributes defining the operability modes

We consider another type of mode, illustrated in figure 8. This is a SysML state machine illustrating the behavior of a *TCMS* function, "Manage the TCMS HMI", the *TCMS* being a subsystem of the *consist* and our SOI here. Another mode of the function is represented in the diagram, represented by a black circle connected to "Manage TCMS HMI". In this mode, none of the capabilities are enabled. As said earlier, we consider the SysML "state" elements as modes. We define the attributes of the mode "Manage the TCMS HMI" in figure 9.

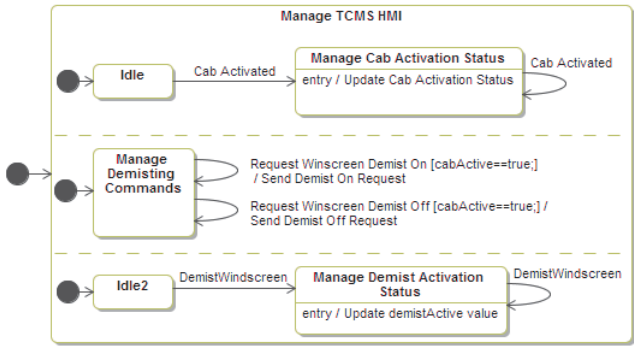

Figure 8 – Modes of a *TCMS* function

The conditions are not visible in this diagram, they are found in the documentation. This missing information could be represented by the system's and sub-system's states and modes, using variables. The mode characterizes the function's behavior, not the SOI's. Functions are abstract notions, and we prefer to not define states for them aside from modes modeling their behavior, as they are too numerous and can exist in specific time contexts. We also see that each capability is characterized by a different be-

ATTRIBUTE	DEFINITION
Source	A function managing the system interface
Capabilities	Managing the cab activation status Managing the demisting status Managing the demisting requests
Conditions	Unknown/not specified
Characterization	Enabling capabilities
Granularity level	Subsystem

Figure 9 – Attributes of "Manage the TCMS HMI"

havior with their own modes. There are information about components that could be abstracted. For a same level of granularity regarding the SOI, we can have a hierarchy of capabilities with their own behaviors, modes and time contexts. It should also be possible to model such a behavior for the *consist* itself, without referring to lower levels of granularity regarding the system. We see that beyond describing the expected behavior of the system, modes can be used to describe how it will be implemented.

4.2 Benefits

An improper definition of states and modes can have serious consequences when designing a system. There was a case where BT, as per contract, had to provide a train with troubleshooting functions executable when alimented by batteries at low train voltage (under 63V). However, the supplied battery displayed a low level status under 50 volts, for which no functions can be executed. Due to those similar terms, BT let suppose it provided troubleshooting functions available for any voltage under 63V, even under 50V where it was no longer possible. This would have been avoided by specifying the proper target, information and granularity level for each type of state.

5 CONCLUSION

We presented in this paper a definition of states and modes as concepts that can be used in different languages and semantics, and hinted at their usefulness in specification and design of the system behavior. This is a first step in a work aiming to perform early validation in a continuous way. Indeed, states as they are defined here represent information about the system that follow its development through abstraction levels, and modes correspond to specifications of its behavior, based on conditions on state values. This should allow us to analyze and validate both the system and its behavior before even defining its functions, and then trace it to any model or product developed after. A method is currently developed with this in mind, which will be supported by a tool chain.

ACKNOWLEDGMENTS

This work is supported by Bombardier Transport SAS and the ANRT CIFRE grant #2016/0262.

REFERENCES

Behrmann, G., David, A. & Larsen, K. G., 2006. A Tutorial on Uppaal 4.0 *Formal methods for the design of real-time systems*, p. 200-236, Springer.

Börger, E. & Stärk, R., 2003. *Abstract state machines: a method for high-level system design and analysis*, Springer Science & Business Media.

Chapurlat, V., Daclin, N., 2013. Proposition of a guide for investigating, modeling and analyzing system operating modes: OMAG. LGI2P, ENS Mines d'Alès

Harel, D., 1987. Statecharts: A visual formalism for complex systems. *Science of computer programming*, 8(3): p.231-274.

Henzinger, T., 2000. The theory of hybrid automata. *Verification of Digital and Hybrid Systems*, Springer, p.265–292.

Jenney, J., 2011. Define Life Cycle System Modes. (See `http://themanagersguide.blogspot.com`)

Naumenko, A., 2002. *Triune Continuum Paradigm: a paradigm for General System Modeling and its applications for UML and RM-ODP*, PhD., EPFL, Switzerland.

Olver, A. M. & Ryan, M. J., 2014. On a Useful Taxonomy of Phases, Modes, and States in Systems Engineering. *SETE 2014*, Adelaide, Australia.

OMG, 2015. *Unified Modeling Language*, version 2.5. (See `http://www.omg.org`)

OMG, 2017. *Systems Modeling Language*, version 1.5. (See `http://www.omg.org`)

SMC, 2005. *Systems Engineering Primer & Handbook*, United States Air Force SMC.

Voirin, J., 2017. *Model-based System and Architecture Engineering with the Arcadia Method*. Elsevier.

Walden, D. D., Roedler, G. J., Forsberg, K., Hamelin, R. D., Shortell, 2015. *Systems engineering handbook: a guide for system life cycle processes and activities*. INCOSE, Wiley.

Wasson, C.S., 2010. System Phases, Modes, and States Solutions to Controversial Issues. *Wasson Strategics, LLC.*