A Study of Behavior and State Representation Methods in Modern PLM Systems

Petr Mukhachev, Clement Fortin

To cite this version:

Petr Mukhachev, Clement Fortin. A Study of Behavior and State Representation Methods in Modern PLM Systems. 17th IFIP International Conference on Product Lifecycle Management (PLM), Jul 2020, Rapperswil, Switzerland. pp.634-644, 10.1007/978-3-030-62807-9_50 . hal-03753138

HAL Id: hal-03753138
https://inria.hal.science/hal-03753138

Submitted on 17 Aug 2022

A Study Of Behavior And State Representation Methods In Modern PLM Systems

Petr Mukhachev[1][0000−0002−0906−1411]* and Clement Fortin[1]

Skolkovo Institute of Science and Technology, Moscow 121205, Russia, Bolshoy Boulevard 30, bld. 1
petr@mukhachev.me*, c.fortin@skoltech.ru
* Corresponding author

Abstract. Increasing product complexity and the growing importance of software components poses challenges for PLM systems in modern companies and extended enterprises. Today product behavior and its processes drive product value, so companies use sophisticated software to accurately predict it before its manufacture. In this work we argue that states are a natural atomic part of any product, system or process. Then we briefly review and compare current methods for behavior and state representations in domain engineering and systems engineering practices. We show that the support provided to the engineer by PLM systems in managing and sharing behavioral and state product properties across an extended enterprise is insufficient in comparison to their significance for successful product development and its complete life cycle.

1 Introduction

Growing product complexity continues to be one of the hardest challenges facing design and manufacturing today. Many vendors developed systems to help the designer save his time and effort during the design. One of the earliest efforts was to build modeling tools, i.e. tools to aid in representing and viewing the designer's work. In order to support system evaluation, tools for simulation were developed as well as tools for manufacturing support. Today, PLM systems cover many engineering and management domains. However, the introduction of software intensive systems, IoT, Industry 4.0, Cyber-Physical systems concepts set additional challenges to interoperability, modeling and simulation capabilities of the software.

Modern cyber-physical and automated systems blend the distinction between physical and computer systems. The development of these systems and their components requires physical as well as behavioral product information. The physical representation includes typical CAD structures, like form and material. Behavioral product specification is related to its functionality, possible interactions with other systems or their parts in different domains, like structural, electrical or informational. Sophisticated simulation of system behavior at the detail product development phase and its interaction with the environment has become essential for predicting and ensuring commercial product success.

Furthermore, in a highly connected world, companies often need to simulate their target system interacting with other systems across the extended enterprise, so advanced means for change management, collaboration and model reuse between disparate simulation and modeling systems are required.

Despite the diversity of simulation software, any digital process simulation uses some kind of event model: continuous event model, or discrete event model, or a mixed one. In any of these cases, the system of interest could be represented by a set of variables that evolve over time in a continuous or discontinuous way. The domain of these variables is called state space and their values at any particular point in time define the state of a simulated system. This way state is a fundamental concept for systems behavior simulation and analysis.

In spite of their fundamental meaning for product development, system states still could not be effectively reused and handled by software solutions. Geometry modeling and solid object representation serves as a backbone for version control and collaboration, which do not provide sufficient support for product state control.

In this work we will underline the importance of the system state concept for product development. We will concentrate on system state definition as a fundamental block of product representation at all stages of the life cycle. We will discuss with examples current practices for state and behavior representation from the systems engineering and PLM point of view and underline the importance of explicit management of stateful objects for complex system development.

2 Defining Processes And Product States

Fundamentally, the work of a designer is to create the definition of products and systems as objects. An object can be defined as a thing that exists or might exist. Real objects can be transformed in various ways: they can be heated, deformed or wear out during the operations. A process is something that transforms objects: consumes, creates, or modifies its state. The function is the main process of the system, which is designed to deliver value — benefit at cost — to the system beneficiary.

The Webster dictionary [1] proposes the following definitions of “state”, any of which is applicable in the scientific and engineering domain:

> “State”, noun:
> - a condition or stage in the physical being of something
> - any of various conditions characterized by definite quantities (as of energy, angular momentum, or magnetic moment) in which an atomic system may exist

There are several important implications of that definition:

- there are no stateless objects. A CAD solid object has at least a single state: it is an abstraction which does not change in time and can only change its position. Real objects are much more complex and could be potentially decomposed into many states. Thus, a state is a natural part of any physical object in a system, which defines the object at some point in time or in a time frame;

– the notion of state effectively allows us to describe only some of the parameters of a condition in which the system may exist. Indeed, the designer considers only those variables which are essential for product definition, ignoring others. Moreover, he narrows the set of variables to a minimal set that appropriately, i.e. to a degree that allows to solve a problem at hand, describe the product or system. Even the information that does not allow appropriate system description could be useful, e.g. at early project stages, when the system is not yet sufficiently defined;
– a state is a natural primitive constituent of any process within a system. A process in a system can be defined as a change of state over time.

Sometimes the change in system parameters can be insufficient for our senses or instruments to capture the difference with the adjacent state. This idea leads us from a continuous state space to a discrete state space. A discrete state space is also useful to intentionally simplify a system behavior representation.

Valid states for a kettle are: hot and cold; a landing gear of an aircraft can be retracted or extended; a tablecloth can be clean or dirty, etc. These states are characterized by parameters, such as temperature, gear position, or color respectively. Usually the designer defines his or her product for being “good enough” in some state, such as cruise state for an aircraft, or for changing the state of other objects, such as “making the water hot” in case of a kettle.

3 Representations

Knowledge representation and reasoning are deeply intertwined. It was argued by Davis [4] that by choosing a particular representation, one chooses a character of answers and method of reasoning it could provide by its ontology, which inevitably captures only a part of the underlying phenomenon.

Many systems for reasoning about processes and states are used in industry. Even more are developed in academic literature, but none of them surprisingly are explicitly present in PLM systems. The next section briefly reviews some of existing methods for state and process representation.

Discrete event systems and continuous event systems are a part of modeling and simulation theory. It is an active field of research, which formalizes modeling concepts for complex systems. System states play an essential role in the formalism by defining the event as a transition from one state to another. Currently this theory could be considered as a backbone for different simulation engines and softwares used by engineering community [3].

Process Specification Language [8], [11] is a comprehensive ontology developed by NIST, which was designed to provide unified means for software semantic interoperability and formal reasoning about business and manufacturing processes. The ontology is based on extended and modified situation calculus, which provides formal reasoning mechanisms. The state extension supports definitions of states before and after certain activities through their attribution to changeable properties of the objects. Many kinds of properties and their connections to processes and other properties are defined for formal reasoning support.

The SAPPHIRE model was originally proposed by Venkataraman and Chakrabarti [14] to support conceptual product design. In Figure 1 it is shown in the form proposed by McSorley in [10]. The model underlines causal relationships between objects, processes and states. The function of an object could be attributed to an *"affect"* link to components of other models. In other words, it represents the reality that the function of a system affects its environment. The model allows to describe in detail and track product behavior. In this causality model, states have a central role as a causal connector between physical phenomena and other entities in a system.

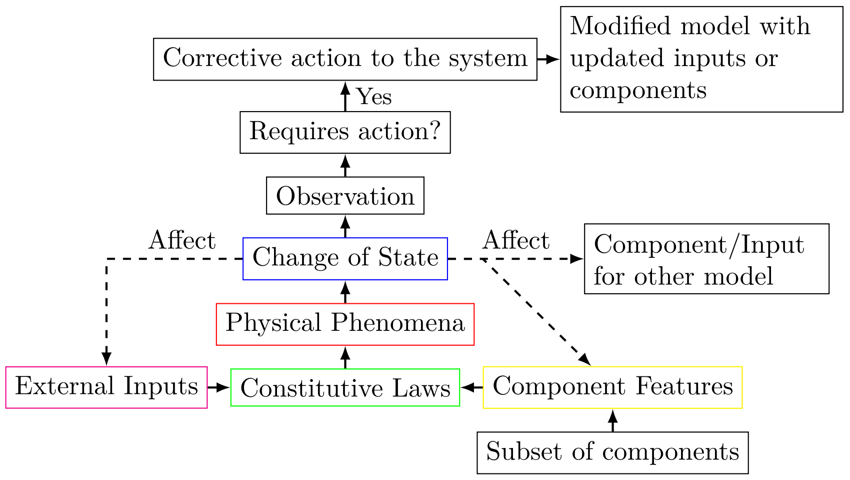

Fig. 1. SAPPHIRE model structure

3.1 Systems Engineering

Systems engineering could be defined as a methodical multi-disciplinary approach for the design, realization, technical management, operations and retirement of a system. A significant overlapping exists between PLM and Systems Engineering. PLM scope of application is larger and can be applied independently of Systems Engineering methods. Systems Engineering is more concerned with engineering methods and processes from the very beginning of the system development, as well as with enhancing communication and interoperability between multidisciplinary teams, as it was described by Vosgien in [15]. So, in contrast to domain modeling and analysis in PLM systems, for systems engineering the semantic representation is essential, especially at the conceptual design stage, when the system may be not yet sufficiently defined to run precise simulations.

From this point of view, states are important to define concept of operations and requirements for a product to accurately convey system functionality description to system's stakeholders. According to NASA Systems Engineering Handbook [12], top-level requirements define constraints and scope for a technical system. The concept of operations and functional requirements describe them on subsequent stages. These documents prescribe how the system will be used and define its functional and behavioral expectations, that is, they loosely define states and transitions of the considered system and product. For example, consider two excerpts from James Webb Space Telescope (JWST) Mission Operations Concept [13], which defines possible states of the telescope and its subsystems and transitions between them:

- MO-44. The JWST Mission will be divided into 5 operational phases: Prelaunch, Launch, Deployment and Trajectory Correction, Cruise and Commissioning, and Normal Operations.
- MO-226. A generic sequence of events in a typical science observation with a JWST instrument would be:
  1. The spacecraft slews to a new target field.
  2. The FGS (Fine Guidance Sensor) performs a guide star acquisition and the spacecraft fine points the Observatory.
  3. etc...

Each mission phase will have different requirements, corresponding to various mission states, and different types of verification procedures for the same physical products and parts. In other words, the designer has to consider that an object of interest has more than one potential state. In early product development stages, the designer doesn't know all the details to precisely specify state variables values, so he is forced to hide some complexity by specifying only high-level system states, in fact representing subsystem states as indistinguishable.

Traditional ways of delivering the information to the stakeholders is to use documents and natural language. However, currently, many graphical and formal notations have appeared to partially substitute natural language in order to make the information more expressive and compact. As a result, Model-Based Systems Engineering (MBSE) concept emerged. One of the earliest attempts to formalize systems engineering was performed by Wymore [17]. He used system state as one of fundamental building blocks in his formalism. However, modern MBSE to a large extent adopted state and process concepts as well as their representations from software engineering practices. Harel state machine is adopted in SysML and very popular among systems engineers. A review of different kinds of state machines is presented in [16] by Wach.

The Object Process Methodology developed by Dori [5] provides a simple, but powerful vocabulary for system representation in a holistic way. It is standardized as ISO19450-2015 and gains popularity in different domains. It has only three fundamental objects: object, process and state, along with different types of relations between them. OPM defines both graphical and textual language representations which can be used to model any complex system and offers zoom-in capabilities in its proposed application OPCloud, to represent any subsystems or components in greater detail particularly for the system architecture definition. Dov Dori proposes [6] to use the term *stateful objects* in OPM to articulate states as a natural constituent of an object. With this definition he effectively integrates changeability into any physical object or system, and proposes that all complex systems representations must include this fundamental characteristic to be appropriately defined.

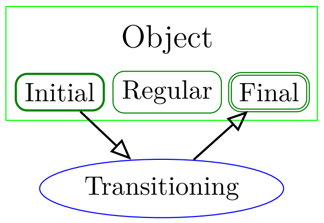

Fig. 2. OPM object, process and state representations

As an atomic structure, the state of the object is defined as a “possible situation or position of an object”. The state of the system is derived as a “snapshot of the system model taken at a certain point in time, which shows all the existing object instances, current states of each stateful object instance, and the process instances, with their elapsed times, executing at the time the snapshot occurs”.

At every point in time, a stateful object is in one of its states or in transition between two of its states as a consequence of a process currently affecting that object. Rounded-corner rectangle with a label placed inside the object denotes an object state. Initial, final and default states could be visually distinguished. The classification of state within a value range is also supported.

OPM authors argue that stateful objects, processes and relationships constitute a minimal universal ontology, meaning that any system in any domain could be modeled using that ontology.

SysML standardized by OMG, is a collection of diagrams for visual systems representation. SysML has a pretty rich tool set for behavior representation that includes statecharts, sequence charts, activity diagrams and functional flow diagrams. SysML was originally designed to replace classic documents and was not meant to provide simulation capabilities. UML and SysML adopt Harel statecharts as one of the means for behavior representation.

First published in 1984, and later adopted in UML and SysML, Harel statecharts [9] are a broad extension of the state machine formalism. They were at first aimed for software systems description, but eventually gained popularity for physical systems and now are adopted in UML2 and SysML languages. They are easy to read, require no prior training and are widely used in different areas.

Statecharts can be concisely described as state diagrams with hierarchy, orthogonality and broadcast communication support. An example of a statechart is shown on Figure 3. Here states B and C are orthogonal, i.e. could coexist simultaneously. They are triggered from state A by event1. State B has one sub-state D, which is entered right after event1. State C has two sub-states E and F, which are switched by event2 and event3.

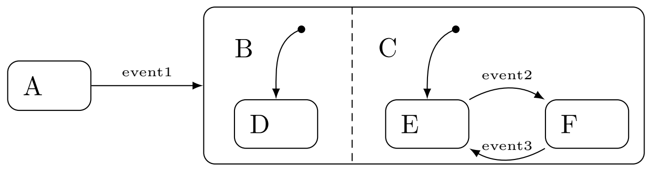

Fig. 3. Statechart example

In statecharts, events are considered to be external inputs and should be provided for execution. Some tools, like MagicDraw use events generated by activity diagrams in order to trigger the transitions. Many tools try to provide some simulation capabilities to SysML, integration with different simulation software, etc. However, modern SysML still remains to be a graphical language with 9 diagrams representing the structure and behavior of a system, rather than its simulation.

3.2 Domain Modeling

Electrical, mechanical, hydraulic and many other mature engineering domains have developed many types of simulation software with readable graphical processes representations and dynamic system simulation support. They are capable to calculate the values of state variables in any particular point in time by solving algebraic and differential equations or using discrete event systems. The heterogeneous and multidisciplinary nature of today's products often demands interoperability with specialized or locally used software as well as other large PLM systems.

In-house solutions are often implemented using general-purpose modeling software like Matlab and Simulink, or programming languages: Python, Java, and C++ are among the most widely used now. For these solutions engineers share system behavior using Excel or plain text "csv" formatted files, because they afford relatively easy to read and write system independent interface. For more complex interactions the Functional Mock Up (FMI) is a powerful tool which was developed by the MODELISTAR consortium and the Modelica Association Project [2], which is now supported by more than 100 tools. Time-continuous as well as time-discrete process descriptions are supported by FMI as well as feedback loops and many more features.

A time series representation can be easily considered as a representation of state evolution in time of discrete state or continuous state system, but for the FMI interface between systems this kind of mapping is not always possible, because it implements direct simulation of a process with its inputs and outputs.

Software and robotics engineers also frequently use state machines for description of logic in their systems. Many variations of state machines are used for computer systems representations, protocol modeling, etc.

Work of the engineer frequently requires the use of simulation software to check whether the requirements for the product are met or not. Some vendors of simulation software afford integrations with requirements management software. As we have seen, requirements are deeply connected with system states. PLM system support for system state could significantly enhance interoperability between these products.

It could be seen that despite the need to explicitly share and manage information about operational states, current PLM software capabilities mostly concentrate on geometry description and its change management. There is little or no support for managing engineering and software changes for these systems.

Space Systems Design Example. In the European Space Agency, a data model for information exchange in concurrent design facilities is described in technical memorandum ECSS-E-TM-10-25A [7]. It is generally designed to facilitate data exchange during the earliest stages in the product lifecycle, thus the standard aims for simplicity of specification with less precise definitions, which are appropriate for this task. Product states are implemented to store several instances of a single variable. This interface is mainly designed to exchange discrete state variables rather than continuous ones and very actively used during space mission design. Typical examples comprise the specification of power consumption of an instrument, different torques of attitude control system, etc. in different operational modes.

4 Discussion

A complex electromechanical system is operating in various states throughout its life cycle and change its state during operations, e.g. for the whole aircraft — taxiing, taking-off, climb, cruise or descent, while for its subsystems it could be retracted/extended gear, retracted/extended flaps or slats, different operation states of the engine, etc. Each system state may be a subject to different requirements that have to be properly implemented in the design, e.g. size of retracted undercarriage might be more constrained than that in an extended position, certain safety requirements may be applied only to engine in operation, etc. A difficulty that can be foreseen in a sold modeler is the coherency of all the states that needs to be maintained with options and version control.

To illustrate the above, in Fig. 4 different deployment states of James Webb Space Telescope are shown. While the spacecraft essentially consist of the same structural elements in all of its states, each of them could be a subject to different requirements and thus diferent simulation scenarios. Different people could be in charge of product design and its functionality in different states, so state coherency may become a big issue.

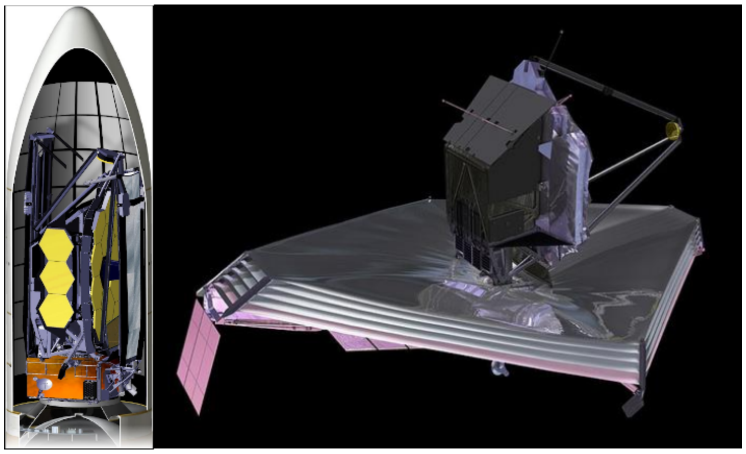

Fig. 4. JWST states: inside rocket fairing(left) and deployed(right)

Cyber-physical systems and Industry 4.0 make the problem of maintaining state coherency even more demanding by increased number of interactions with other systems in an extended enterprise. These systems mix software, electronic, and physical parts making system state management increasingly important.

Systems Engineering has developed many languages with rich vocabulary and expressivity for more formalized description of behavior, concept of operations, requirements, and architecture in comparison to natural language. Besides being machine interpretable, they tend to be stricter and clearer than natural languages. During system development they are primarily used for information exchange between stakeholders, for requirements and functional specifications. In many graphical languages for process specification and representation, states play a key role.

On the other hand, simulation tools that provide extensive support for numerical simulation operate with time series, which can be seen as continuous state change in time. Simulation usually serves as preliminary verification to ensure requirements compliance. In that sense, state representation could help in generating simulation procedures as well as tracing requirement to concrete simulation results and verification procedures. For example, it could be possible to explicitly define single procedure and run it on a number of system states to check system compliance to the requirement in every considered case.

There is an intrinsic connection of state management with configuration management for simulation purposes. For analysis purposes that do not involve state change, an object in some state could be regarded as one of its configurations. For example, an aircraft with retracted landing gear could be treated similarly to one of its configurations. This aspect is quite important for integration with Simulation Data Management software, proper design of experiments and numerical simulation for verification. However, PLM systems have generally evolved from solid modeling tools through Engineering Data Management tools and much of their functionality is still constrained by data structures from solid modeling heritage and thus, single state data structure.

5 Conclusion

The importance of states is shown by their explicit presence in the most significant documents in the project, such as concept of operations and requirements definition documents. During detail design, they are used to define simulation procedures and during the verification and validation stages states are used to define physical and numerical experiments.

Despite the importance of states for product development, product data management systems still heavily rely only on solid models and solid product part representations. Solid object by definition has a single state, and that fact may sufficiently limit the capabilities for cyber-physical and complex systems design support. Objects in a real world are fundamentally stateful and we should represent this feature in design support systems in order to support full product life cycle.

Our study clearly shows that states are explicitly present in software design practices, mission requirements definition, simulation procedures for verification and validation, since behavior is so pervasive in these processes. However, mechanical design and its corresponding PLM systems with their phenomenal solid modeling capabilities have so far maintained implicit state representations apart from their simulation modules; their formal representation of behavior, process and states remains limited, and therefore limits their capability to support the complete development of complex cyber-physical systems.

References

1. Definition of STATE. https://www.merriam-webster.com/dictionary/state
2. Functional Mock-up Interface. https://fmi-standard.org/
3. Bernard Zeigler, Alexandre Muzy, Ernesto Kofman: Theory of Modeling and Simulation. Elsevier (2019). https://doi.org/10.1016/C2016-0-03987-6
4. Davis, R., Shrobe, H., Szolovits, P.: What Is a Knowledge Representation? AI Magazine **14**(1), 17–17 (Mar 1993). https://doi.org/10.1609/aimag.v14i1.1029
5. Dori, D.: Object-Process Methodology. Encyclopedia of Knowledge Management, Second Edition pp. 1208–1220 (2011). https://doi.org/10.4018/978-1-59904-931-1.ch116

6. Dori, D.: Conceptual Modeling: Purpose and Context. In: Dori, D. (ed.) Model-Based Systems Engineering with OPM and SysML, pp. 75–96. Springer, New York, NY (2016). https://doi.org/10.1007/978-1-4939-3295-5_9
7. ESA: ECSS-E-TM-10-25A Engineering Design Model Data Exchange (CDF). ESA Requirements and Standards Division (2010)
8. Gruninger, M., Menzel, C.: The Process Specification Language (PSL) Theory and Applications. AI Magazine **24**(3), 63–63 (Sep 2003). https://doi.org/10.1609/aimag.v24i3.1719
9. Harel, D.: Statecharts: A visual formalism for complex systems. Science of Computer Programming **8**(3), 231–274 (Jun 1987). https://doi.org/10.1016/0167-6423(87)90035-9
10. McSorley, G., Fortin, C., Huet, G.: Modified SAPPHIRE Model As a Framework For Product Lifecycle Management. In: DS 77: Proceedings of the DESIGN 2014 13th International Design Conference (2014)
11. Pouchard, L.C., Cutting-Decelle, A.F., Michel, J.J., Grüninger, M.: ISO 18629 PSL : A Standardized Language For Specifying and Exchanging Process Information. IFAC Proceedings Volumes **38**(1), 37–45 (Jan 2005). https://doi.org/10.3182/20050703-6-CZ-1902.01525
12. Shea, G.: NASA Systems Engineering Handbook Revision 2 (Jun 2017)
13. Space Telescope Science Institute: JWST Mission Operations Concept Document. Tech. Rep. JWST-OPS-002018 (2006)
14. Venkataraman, S., Chakrabarti, A.: Sapphire – an Approach to Analysis and Synthesis. In: DS 58-2: Proceedings of ICED 09, the 17th International Conference on Engineering Design, Vol. 2, Design Theory and Research Methodology, Palo Alto, CA, USA, 24.-27.08.2009 (2009)
15. Vosgien, T.: Model-Based System Engineering Enabling Design-Analysis Data Integration in Digital Design Environments: Application to Collaborative Aeronautics Simulation-Based Design Process and Turbojet Integration Studies. Ph.D. thesis, Paris (2015)
16. Wach, P., Salado, A.: Can Wymore's Mathematical Framework Underpin SysML? An Initial Investigation of State Machines. Procedia Computer Science **153**, 242–249 (Jan 2019). https://doi.org/10.1016/j.procs.2019.05.076
17. Wymore, A.W.: Mathematical Theory of Systems Engineering. John Wiley & Sons (1967)