DESIGNING A MODEL-BASED, MULTI-PERSPECTIVE PROCESS DESIGN ENVIRONMENT

Shaked, Avi;
Reich, Yoram

Tel Aviv University, Systems Engineering Research Initiative

ABSTRACT

The development of engineered systems is a complex process, involving many activities and development artifacts. The challenges introduced by this complexity calls for a model-based design environment. The PROVE methodology is an artifact-centric, model-based development process design methodology. In this paper, we present the first, fully model-based design environment of the PROVE. Furthermore, our environment incorporates additional process design perspectives that are applied in real-time and in a completely consistent manner, based on a formally defined information model. We discuss the details of our environment's design as well as highlight its potential benefits.

Keywords: Process modelling, Design process, Systems Engineering (SE)

Contact:
Reich, Yoram
Tel Aviv University
School of Mechanical Engineering
Israel
yoramr@tauex.tau.ac.il

Cite this article: Shaked, A., Reich, Y. (2021) 'Designing a Model-Based, Multi-Perspective Process Design Environment', in *Proceedings of the International Conference on Engineering Design (ICED21),* Gothenburg, Sweden, 16-20 August 2021. DOI:10.1017/pds.2021.110

1 INTRODUCTION

The development of engineered systems is a complex process, involving many activities and development artefacts. This high complexity introduces a major challenge to the process designers who develop such systems. Designers are often faced with multiple process objectives by different stakeholders – such as regulators, customers and enterprise management – and need to take into account organizational structures, roles and responsibilities. Process designers are also faced with different development approaches and constraints, some of which are a reflection of the various disciplines involved. For example, while software development may be performed in sprints due to its logical nature; hardware development has a physical manifestation that often requires a different, more linear and ordered design approach.

While the aforementioned complexity is demanding in its own, existing modeling environments do not ease the job of a process designer. They tend to incorporate versatile yet often perplexing modeling standards that may only partially fit the modeling objective (Shaked and Reich, 2021a) without making them accessible to the process designer. A recent technical report on model-based systems engineering maturity – based on an extensive, carefully developed survey – confirms that the industrial maturity of using systems engineering related information models in technical processes and their management is very low (McDermott et al, 2020). This often leads to the process designer neglecting formal modeling approaches and turning into free-form alternatives (e.g., model-free diagrams), with the latter being the prominent form of technical design process descriptions in practice (Eckert and Clarkson, 2010; Browning, 2002). In turn, this risks not only the consistency and completeness of the resulting process descriptions, but also the ability to reflect and reuse process designs in future efforts.

In this paper, we present a new development process-modeling environment: PROVE Tool. This environment is based on the previously established model-based development process design methodology PROVE (PRocess Oriented View for Engineering). PROVE encourages an artifact-centric approach to process modeling, which has an advantage over the more traditional activity-centric approach, especially with respect to management of process data and changes and to communicating process designs in collaborative processes (Kunchala et al., 2020). PROVE was shown to contribute to rigorous modeling and to the relaxation of the user's modeling effort (Shaked and Reich, 2021b). The model-based PROVE methodology and several applications are discussed in previous publications (Shaked and Reich, 2019a; 2019b; 2021b; 2021c). While modeling tools are available for using other process modeling methodologies, such as various tools for Business Process Modeling and Notation (BPMN) (Recker, 2010) and Eclipse Process Framework for Software Process Engineering Metamodel (SPEM) (Koudri and Champeau, 2010); there is a gap in a modeling environment that supports designing processes with PROVE.

Here we focus on aspects of the newly developed tool as a versatile, model-based design environment, addressing the aforementioned gap. Additionally, by providing a rigorous, formally-defined modeling environment as an implementation of the PROVE methodology, we further establish the technical validity of the PROVE methodology as a model-based process design approach. Our implementation of the theoretically established PROVE methodology attests to the feasibility of the methodology, and provides process designers with the ability to use PROVE effectively and in its originally intended model-based approach.

Furthermore, our tool development effort emphasizes important aspects of model-based design in general. Specifically, we demonstrate and discuss aspects of incorporating multiple perspectives in formal design tools, by manifesting varied design concepts as representations on top of a cohesive information model. We also discuss our design method, which offers a way to establish consistency between multiple digital representations by design.

In Section 2, we overview relevant background for model-based development process design. In Section 3, we discuss our approach to the development of the PROVE Tool modeling environment as well as the underlying technology. In Section 4, we present the developed modeling environment while showcasing some of its main features. In Section 5, we conclude and highlight some aspects of the modeling environment.

2 BACKGROUND

Model-based design is a design approach that addresses the need to handle the increasing complexity of systems and their development (Ramos et al., 2011; Bernal et al., 2015). Few standards relating to model-based process design exist. We refer to the most notable of these, based on their wide availability for use within modeling tools.

The UML and SysML specifications (OMG, 2017; OMG, 2009) offer some standardized modeling notations to represent processes. However, they are primarily architectural modeling frameworks, and accordingly, focus on depicting processes within the scope and perspective of the technical system design.

Other standards that are more pertinent to this work are BPMN and SPEM. While these provide a process-oriented view, they are extensive standards that were designed to support diverse process modeling scenarios. BPMN is a notation that can be used for modeling processes; it aims to be "readily understandable by all business users, from the business analysts that create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and finally, to the business people who will manage and monitor those processes"; thereby creating "a standardized bridge for the gap between the business process design and process implementation" (OMG, 2011). SPEM is a meta-model, which includes a rich language as well as some methodology-related concepts, and was originally designated to be used in order to "describe a concrete software development process or a family of related software development processes" (OMG, 2002).

While BPMN and SPEM's versatility may promote their wide theoretical applicability, it does not promote sound modeling in practice. Specifically, the selection between different modeling alternative hinders rigorous modeling and the ability to reuse model fragments (Shaked and Reich, 2021a). Recker et al. (2009), for example, established that BPMN is of high complexity, and called for reducing its complexity for practical implementations. Furthermore, Dijkman et al. (2008) reported that BPMN models are prone to semantic errors, and attributed this to the heterogeneity of BPMN constructs and some ambiguous definitions. Similarly, SPEM is widely considered hard to use. The SPEM 2.0 specification itself admits the poor adoption of SPEM v1.X versions, attributing this to ambiguous and hard to understand semantics (OMG, 2008). In a comparison between the two versions of SPEM, the SPEM 2.0 specification was deemed very complex and hard to understand, suggesting that the revision did not address the standard's shortcomings (Bendraou et al., 2010). Furthermore, the SPEM 2.0 specification was evaluated as unclear, and possibly incomplete, as well as requiring extensive implementation efforts. The latter conclusion is supported by a study that claimed that SPEM has insufficient depth of use in industry, and attributed this to the high complexity of SPEM and its low return on investment (Ruiz-Rube et al., 2012). The authors called for user-friendly tools for promoting use of model-based process management. Kuhrmann et al. (2010) also identified SPEM as requiring high expertise, and further mentioned that it has no visual representation that appeals to relevant process stakeholders. The PROVE methodology is a model-based process design methodology that addresses the aforementioned concerns of semantic clarity and complexity by-design and includes an artifact-centric process design perspective (Shaked and Reich, 2019). While PROVE is designed as a model-based design approach, no full model-based implementation of PROVE was provided so far, and its technical validity (i.e., the feasibility of its implementation) has not been established yet.

In addition to the process design perspective, there are other perspectives that may be of value to a process designer. The use of perspectives in modeling is considered beneficial to reducing complexity and to effective communication (Frank, 2014). An artifact lifecycle representation is considered essential in artifact-centric modelling approaches (such as PROVE), capturing the possible states and transitions of the artifact (Vaculín et al., 2011). Design Structure Matrix (DSM) representation can facilitate the exploration of aspects regarding a process design's granularity, such as the level of details of the process design and particularly its hierarchical structure (Maier et al., 2017); and various methods may be employed to improve the design (Browning, 2015).

None of these additional perspectives were shown to be compatible with PROVE until now. Moreover, preserving consistency in multi-perspective modeling implementations is challenging, with some previous work not addressing this despite being fundamental to the model-based design approach (Bork et al., 2015). A method to translate artifact-centric process models (such as PROVE

models) into an artifact state transition representation was previously suggested and demonstrated (Yongchareon et al, 2015). However, this method does not provide support for composite states, which is a principle ontological characteristic of artifacts in development processes, as identified by the PROVE methodology. Furthermore, the method is not automatically synchronized with the process model, which hinders the adoption of a full model-based design approach.
In the next section we present PROVE Tool, which is an easy-to-use model-based process design modeling environment. PROVE Tool features simple yet effective visual, synchronized representations that are designed to provide the development process designer with the various, complementary perspectives.

3 METHOD

PROVE Tool was developed as a domain specific extension for Eclipse Modeling Tools[1]. For its development, we used the Eclipse modeling infrastructure, particularly, ECORE Tools for specifying the metamodel and Sirius as the engine of its graphical representations and domain-specific user interface design.
First, a metamodel was developed using ECORE Tools, to include a formal, technically-valid manifestation of the concepts identified by the PROVE methodology (Shaked and Reich, 2019). The metamodel conforms with the Object Management Group's EMOF formal specification (OMG, 2019). Then, we used Sirius – a software technology for creating modelling workbenches – for creating the representations. A representation can comprise two types of section: a visual representation derived from the model, which we refer to as "diagram"; and a toolbox, which consists of various action tools to manipulate the model and/or diagram. The Sirius specification engine allows establishing representations using graphical building blocks that relate to queries of a model, based on the metamodel definitions. The resulting model-based diagrams remain synchronized with the underlying model. The information model is structured exclusively and formally according to the aforementioned metamodel, and this cannot be overridden by any representation-related action tool; resulting in an information model that remains consistent with respect to its metamodel.
We implemented the PROVE design perspective as the initial, core process design representation. The representation was designed to follow the original "boxes and arrows" PROVE notation as much as possible, with boxes representing the processes and delineating their scope, and with arrows representing artifacts in their states (Shaked and Reich, 2019). Representational layers were designed to augment the process design representation with minor variations, based on layering mechanism available in Sirius. A "design" layer provides the designer with design-related tips about the current process design, and specifically its agreement with PROVE related concepts. A "status" layer provides the designer with the developmental statuses of the artifacts and allow changing an artifact's status; in accordance with the PROVE framework, which considers the development status as the collection of all achieved artifact states. These representational variations used coloring notations to provide the design tips or artifact status. The metamodel was updated throughout the design of the process design representation, in accord with various technical constraints and in support of practical tool implementation. Still, the metamodel was kept true to the concepts of the PROVE methodology (specifically, no new ontological concepts were introduced), reflecting its minimal, domain-specific modeling approach.
Next, additional model-based representations were developed, exhibiting the other perspectives that were deemed as beneficial for process design. We developed an artifact lifecycle representation as a model-based state diagram, for viewing the lifecycle of an engineering artifact. The aforementioned layer mechanism was used to enhance the artifact lifecycle representation with a "status" layer reflecting artifact status using a coloring notation. We also developed a process interface map representation as a model-based table-like representation, inspired by the design structure matrix for the purpose of showing the interaction between the various activities.

[1] The tool is available at https://github.com/TAU-SERI/PROVE, and is open-source and free for use.

4 PROVE TOOL – A DEVELOPMENT PROCESS DESIGN ENVIRONMENT

4.1 PROVE Tool Metamodel

Figure 1 shows the PROVE metamodel for PROVE Tool information models. The PROVE metamodel includes the three main constructs of the PROVE methodology: "Process" (also alternately referred to as "activity" in PROVE), "Artifact" and "Artifact state." An additional element – "Artifact state instance" – is included in the formal metamodel, and this is done in order to support multiple instantiations of artifacts in specific states, based on the "Artifact state" (directly) and the associated "artifact" (indirectly).

Two additional elements – "Shadow node incoming" and "Shadow node outgoing" – are model elements designed strictly to support the tool's representations. These are colored differently in the metamodel, in order to portray their representational function (this does not have any technical implication). These elements were incorporated into the metamodel only due to existing limitations of the modeling infrastructure, as discussed later. These metamodel elements are not to be used by the process designer, and do not hold any semantic value.

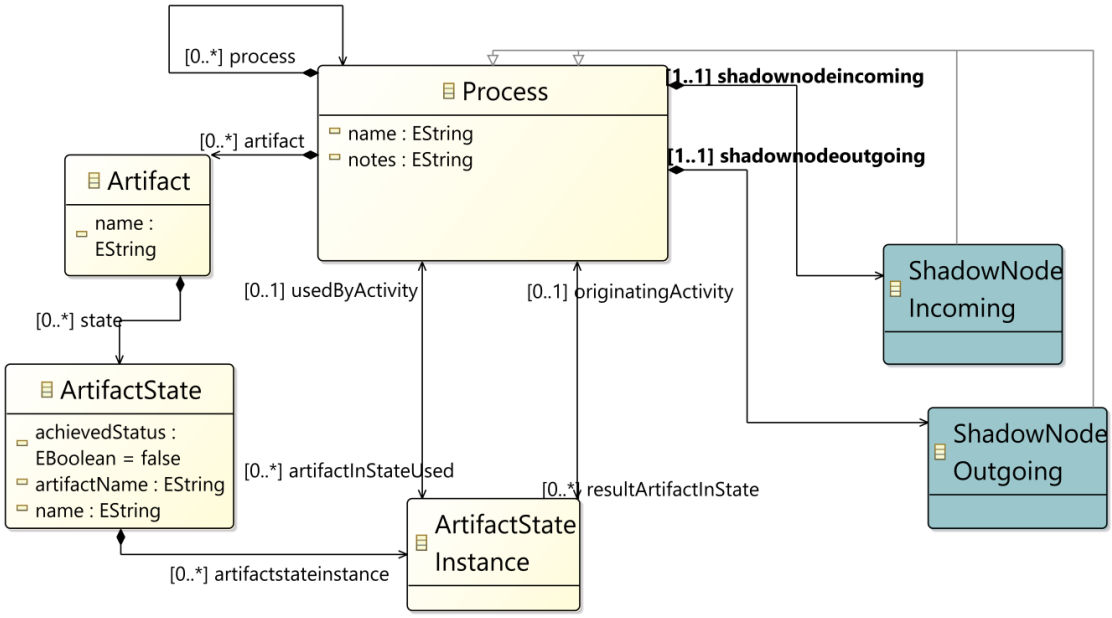

Figure 1: PROVE Tool metamodel

The specification of element attributes is an additional aspect of the implementation of a domain-specific methodology in the form of a formal metamodel, as seen in Figure 1. Specifically, the "name" attribute is a common attribute in our implementation, and is used to uniquely identify elements. The "artifactName" in "Artifact State" was designed to support easier tool development using the Sirius modeling component (and is expected to be in sync with the content of the "name" attribute of the "Artifact" element, as a tool design constraint). Other attributes emerged out of the need to support users from a practical implementation perspective (which was not the focus of the original PROVE conceptual methodology): the "notes" attribute of "Process" allows the process designer to add a natural language comment associated with the element; and the "achievedStatus" attribute of the "Artifact State" was designed to support the development status perspective on top of PROVE. Some uses of these attributes in the representational design of PROVE Tool are mentioned shortly.

In addition to the aforementioned blocks, the metamodel shows the relationships between model elements. These relationships are in the form of reference (directional and bi-directional arrows) and of composition (diamond-headed edges). The multiplicity is depicted explicitly on each of these edges. A type reference (also commonly known as inheritance) is depicted using hollow arrow-headed edges, signifying the inheritance of attributes from the target to the source.

In the following subsection we describe the representations built on the metamodel.

4.2 PROVE Tool Representations

The main representation available in PROVE Tool is the process design representation. This representation is designed in accordance with the PROVE notation and supports detailing the process design in various levels of granularity (hierarchies). Figure 2 shows this representation in the modeling environment. The two main components of this representation are (1) the model-based diagram (the left section in the figure); and (2) the palette (the right section of the figure).

The process design representation's diagram can be created for different process scopes, presenting only the relevant information for the specific process scope. Figure 3 shows such a diagram detailing a specific sub-process – ActO – of the main process (depicted in Figure 2). The diagram associated with the sub-process shows artifact-in-state edges entering and existing the subprocess, without a second specific node, as required by the PROVE notation.

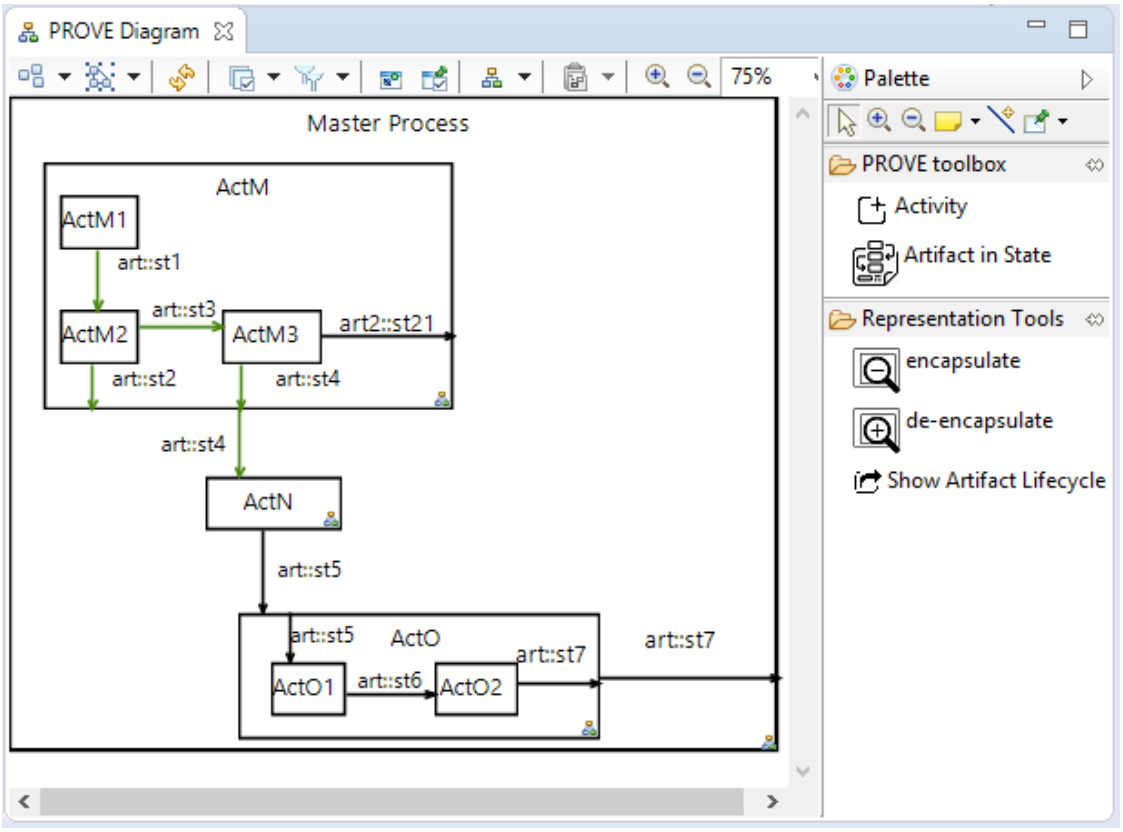

Figure 2: PROVE Tool process design representation (with "status" layer activated)

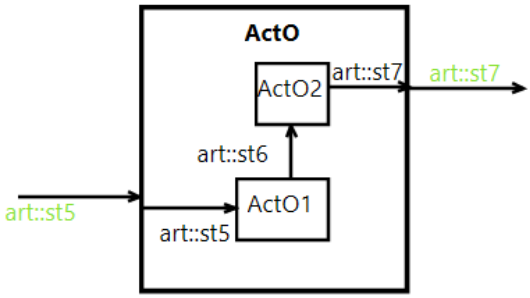

Figure 3: A subprocess design representation (with "analysis" layer activated)

The modeling palette, which appears in Figure 2 as "PROVE toolbox" allows the user to add elements to the information model. This is yet another Sirius mechanism that was used to follow the PROVE methodology. This palette allows selecting an element type – either an activity or an artifact-in-state – for a designated new element, and then applying it to the diagram. The specific chosen element is added to the information model based on the diagrammatic context to which the selection is applied. We stress that – while it may seem otherwise to the user – elements are added to the underlying information model and not directly to the diagram; and the diagram is then automatically updated

based on the model. Accordingly, the palette is only the User Interface (UI) of an elaborate mechanism that is responsible for adding the elements to the model; and in some cases, further options are presented to the user. For example, in order to add an artifact-in-state instance, the user needs to select the "artifact-in-state" tool from the palette, and apply it between two activities by clicking on a source activity and then on a target activity. The tool then prompts the user to enter the designated artifact name and the designated state. Then, an element addition mechanism checks – behind the scenes – for the artifact name in the model. If it does not find it, it adds a new artifact to the model (pending user's confirmation). Similarly, it then looks for the specific designated state for the specific artifact, and, if missing, adds this new state to the model. Checking the model for existing artifacts and existing states before adding new model element assures coherence and helps to avoid redundancy; and a manifestation of this is discussed below, when we describe the status layer. Finally a new instance of "ArtifactStateInstance" is created and associated with the user-selected source as the "originating Activity" and with the user-selected target as the "usedByActivity."

Another component of the representation's palette is the "Representation tools" palette (on the right section of Figure 2, below the "PROVE toolbox" palette). Unlike the previously mentioned modeling palette, this component includes tools that only affect the diagrammatic representation. These tools were implemented for PROVE Tool, but may, in fact, be applicable to other modeling solutions. The "encapsulate" and "de-encapsulate" tools are used in order to hide or show (respectively) the lower-level design of a process; and – corresponding with black-box and white-box engineering design concepts (Cross, 2000) – they allow representing the process with different levels of abstraction based on the hierarchical structure of the underlying model (Maier et al., 2017). The "Show Artifact Lifecycle" is a navigational tool, allowing users to navigate to the artifact lifecycle representation of an artifact (associated with a specfic artifact-in-state).

Figure 4 shows an example of the artifact lifecycle representation. This representation is a model based representation. The representation derives a state machine of a specific artifact, based on the information model (that can be created using the design representation). The engine behind the diagram queries the underlying information model (typically created by using the process design representation) and depicts the available states of the artifact. Furthermore, it exhibits the transitions between states as they are represented in the model, and associates them with the relevant activities. One can identify, for example, that the specific artifact in question – named "art" in the model – can make the transition between state "st3" and state "st4" using activity ActM3, as observable on the process design representation of Figure 2. Also, one can identify two branches of states, designating that the specific artifact may be in more than one state simultaneously (e.g., in "st2" as well as in any of the other states). This demonstrates the "composite state" concept of the PROVE methodology[2]. Since this paper is not of computational nature, we do not provide the details of our real time state machine derivation from the information model; the implementation, however, is fully available in the aforementioned, open-source package of PROVE Tool.

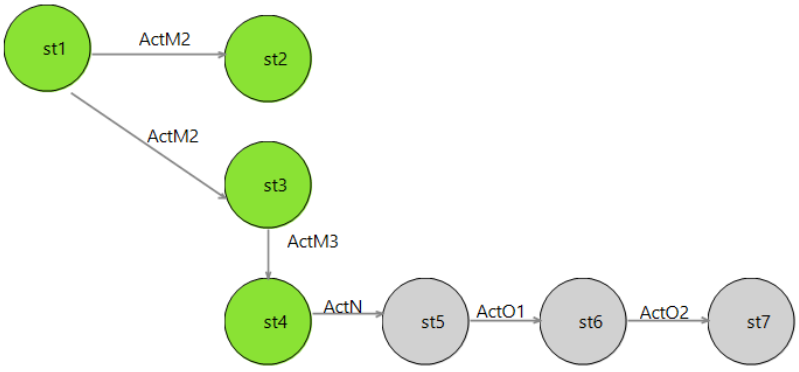

Figure 4: Artifact lifecycle representation (with "status" layer activated)

[2] Formally, according to the PROVE framework, the composite state of the artifact in Figure 4 is the collection of all achieved states (indicated using green nodes, as we discuss in the next subsection of the paper): {st1, st2, st3, st4}. The figure also reflects that – according to the current process design – the artifact needs to achieve the "st4" state before achieving the "st5" state (as opposed to "st2," which is not required in order to achieve "st5").

The process interface map representation details the various activities (row and column elements) in the form of the artifact-in-state elements that are used as their interface (the cells). An example of this representation for the model discussed so far is shown in Figure 5. This representation is dynamic, and prepared in real time using two principle stages. First, the representation mechanism queries the information model in order to establish the table row and columns, based on the activities that are defined in the model (elements of type "Process"). The row representation also demonstrates the ability to present/group activities in hierarchies. Second, the representation mechanism queries the information model in order to identify all artifact-in-states instances and list them in the appropriate table cell.

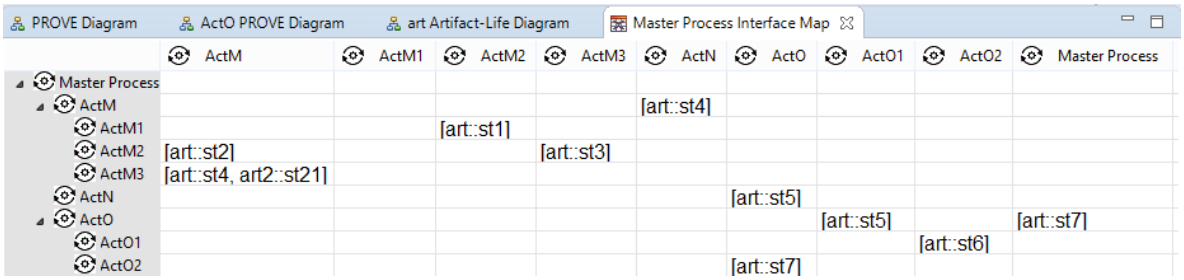

Figure 5: Process interface map representation

4.3 Representation variations using coloring notation

Figure 3 shows an example of the process design representation with the design layer activated. The artifact-in-state instance "art" in "st5" – denoted "art::st5" – is shown with green label to designate that it is indeed used by one or more activities (in our case the activity "ActO1") within the subprocess which receives it ("ActO"). Similarly, the label of the artifact in state instance "art::st7" is green, designating that it was created by a lower level activity ("ActO2") within the "ActO" subprocess.

The second layer is the "status" layer, which aims to reflect the development process status. We allow a user to double click on an artifact-in-state instance to update the "Achieved" status of the associated artifact state model element via a customized form, as shown in the screenshot provided in Figure 6. If a specific artifact state model element is marked as "Achieved" (i.e., its "achievedStatus" Boolean attribute is set to True) then all of the artifact-in-state instances that relate to the specific artifact state appear in green within the diagram. Figure 2 shows the process design with artifact "art" marked as having achieved states "st1," "st2," "st3" and "st4"; and accordingly all relevant edges are marked in green to designated the achieved states.

The "status" layer of the artifact lifecycle representation is shown in Figure 4. Achieved states of the artifact are shown as green nodes, in fully synchronization with the model reflected in Figure 2 (whose status is discussed above).

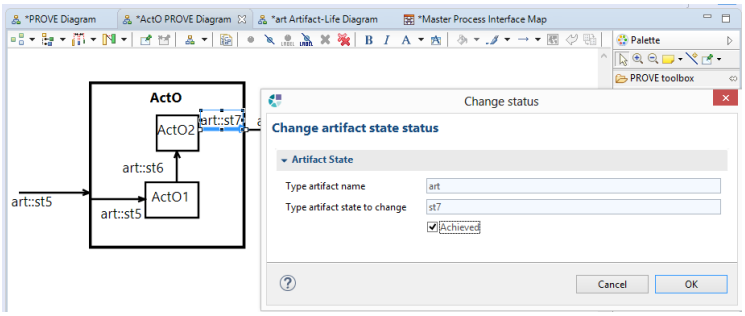

Figure 6: Change artifact state status form

5 DISCUSSION

In this paper, we presented PROVE Tool, a new, multi-perspective modeling environment for development process design. PROVE Tool builds on the theoretical grounds of the PROVE methodology, and is the first true model-based implementation of the methodology. Specifically, it captures PROVE process designs as formal information models, based on a rigorously defined metamodel.

Furthermore, PROVE Tool extends the PROVE methodology with additional modeling perspectives, thereby contributing to the theory-informed application of model-based development process design. Different representations of an information model are applied in real time to represent the information in a fully synchronized manner from different perspectives. These representations reflect concepts of the PROVE methodology (the design representation), of state machines (the artifact lifecycle representation) and of design structure matrix (the activity interface map representation) in concert, and they can be used to model and communicate various aspects of process designs. Such run-time approach to applying the representations was identified as an enabler of implementing versatile software systems (Frank, 2014).

Our formal metamodel for development process design establishes the PROVE methodology as a technically valid and feasible model-based approach. It can also contribute to the understanding of the PROVE as a modeling methodology (Bork et al., 2020). Furthermore, the consistent, multi-perspective modeling implementation atop this metamodel attests to the effectivity of the ontology captured in the PROVE methodology, as other, process-related concepts were implemented using the PROVE compliant metamodel, without violating the concepts of the PROVE methodology.

The multiple perspectives of the development related processes provide a basis for rigorous process design and its improvement. As an illustration, by studying the interface map representation (Figure 5) of the process design that was captured using the PROVE design representation (of Figure 2), interdependencies and clusters may be revealed and further explored in order to improve the process design, e.g., by executing Design Structure Matrix related techniques (see, for example, a review of such methods in (Bronwing, 2015) as well as by exploring aspects of the design's granularity (Maier et al., 2017), such as the level of details of the process design and particularly its hierarchical structure. Similarly, by studying the state transitions of an artifact (Figure 4), the compliance of the process design with enterprise policy or regulation can be enforced. For example, an artifact may not be in state "Released" unless it was first properly inspected for quality issues (indicated by a designated state such as "Quality Assurance Approved"). The "status" layer of the process design representation contributes details about the development status, and this may actually be taken into account as feedback for redesign of the process, considering progress made to a point in time (e.g., in a case where some artifacts did not achieve their desirable status and the process needs to be rethought in order to meet required goals) (Daniel and Daniel, 2018). We intend to further explore aspects of this multi-perspective process design in the future, and possibly even include additional perspectives to support the design, as well as to support perspectives by other process stakeholders (e.g., enterprise resource allocation). Additional effort is planned to integrate the PROVE Tool models with models by other tools (e.g., systems design tools and project management tools), using model to model transformations; thereby promoting interoperability between PROVE Tool and complementary modeling approaches and tools.

Since the design of PROVE Tool, our model-based, multi-perspective design approach has been used successfully to develop another engineering workbench in support of the TRADES methodology for cybersecurity threat and risk assessment (Shaked and Reich, 2021d)[3]. This attests to the generalizability of our design method. We expect future efforts to implement the described method for creating modeling environments that implement other domain-specific methodologies.

REFERENCES

Bendraou, R., Jezequel, J.M., Gervais, M.P. and Blanc, X., 2010. A comparison of six uml-based languages for software process modeling. IEEE Transactions on Software Engineering, 36(5), pp.662-675.

Bernal, M., Haymaker, J.R. and Eastman, C., 2015. On the role of computational support for designers in action. Design Studies, 41, pp.163-182.

Bork, D., Buchmann, R. and Karagiannis, D., 2015, October. Preserving multi-view consistency in diagrammatic knowledge representation. In International Conference on Knowledge Science, Engineering and Management (pp. 177-182). Springer, Cham.

Bork, D., Karagiannis, D. and Pittl, B., 2020. A survey of modeling language specification techniques. Information Systems, 87, p.101425.

Browning, T.R., 2002. Process integration using the design structure matrix. Systems Engineering, 5(3), pp.180-193.

[3] This modeling tool is available at https://github.com/IAI-Cyber/TRADES.

Browning, T.R., 2015. Design structure matrix extensions and innovations: a survey and new opportunities. IEEE Transactions on Engineering Management, 63(1), pp.27-52.
Cross N., 2000. Engineering Design Methods, Strategies for Product Design (Wiley, New York, ed. 4), pp.78-80.
Daniel, P.A. and Daniel, C., 2018. Complexity, uncertainty and mental models: From a paradigm of regulation to a paradigm of emergence in project management. International journal of project management, 36(1), pp.184-197.
Dijkman, R.M., Dumas, M. and Ouyang, C., 2008. Semantics and analysis of business process models in BPMN. Information and Software technology, 50(12), pp.1281-1294.
Eckert, C.M. and Clarkson, P.J., 2010. Planning development processes for complex products. Research in Engineering Design, 21(3), pp.153-171.
Frank, U., 2014. Multi-perspective enterprise modeling: foundational concepts, prospects and future research challenges. Software & Systems Modeling, 13(3), pp.941-962.
Koudri, A. and Champeau, J., 2010, July. MODAL: a SPEM extension to improve co-design process models. In International Conference on Software Process (pp. 248-259). Springer, Berlin, Heidelberg.
Kuhrmann, M., Kalus, G., Wachtel, E. and Broy, M., 2010. Visual process model design using domain-specific languages. In Proceedings of SPLASH Workshop on Flexible Modeling Tools (Vol. 2010).
Kunchala, J., Yu, J., Yongchareon, S. and Liu, C., 2020. An approach to merge collaborating processes of an inter-organizational business process for artifact lifecycle synthesis. Computing, 102(4), pp.951-976.
Maier, J.F., Eckert, C.M. and Clarkson, P.J., 2017. Model granularity in engineering design–concepts and framework. Design Science, 3.
McDermott T. A., Hutchison N., Clifford M., Van Aken E., Salado A., Henderson K., 2020. Benchmarking the Benefits and Current Maturity of Model-Based Systems Engineering across the Enterprise, Technical Report SERC-2020-SR-001, Stevens Institute of Technology, Systems Engineering Research Center.
OMG (Object Management Group), 2017. UML 2.5.1 – Formal Specification.
OMG (Object Management Group), 2009. SysML 1.6 – Formal Specification.
OMG (Object Management Group), 2011. BPMN 2.0 – Formal Specification.
OMG (Object Management Group), 2002. SPEM 1.0 – Forma Specification.
OMG (Object Management Group), 2008. SPEM 2.0 – Formal Specification.
OMG (Object Management Group), 2019. Meta Object Facility (MOF) Core Specification 2.5.1.
Ramos, A.L., Ferreira, J.V. and Barceló, J., 2011. Model-based systems engineering: An emerging approach for modern systems. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 42(1), pp.101-111.
Recker, J.C., zur Muehlen, M., Siau, K., Erickson, J. and Indulska, M., 2009. Measuring method complexity: UML versus BPMN. Association for Information Systems.
Recker, J., 2010. Opportunities and constraints: the current struggle with BPMN. Business Process Management Journal, 16(1), pp.181-201.
Ruiz-Rube, I., Dodero, J.M., Palomo-Duarte, M., Ruiz, M. and Gawn, D., 2012. Uses and applications of spem process models. a systematic mapping study. Journal of Software Maintenance and Evolution: Research and Practice, 1(32), pp.999-1025.
Shaked, A. and Reich, Y., 2019a. Designing development processes related to system of systems using a modeling framework. Systems Engineering, 22(6), pp.561-575.
Shaked, A. and Reich, Y., 2019b. Improving Coordination and Collaboration in Connected and Automated Vehicle Development Projects Using Model Based Process Design. SAE Technical Paper 2019-01-0103, 2019.
Shaked, A. and Reich, Y., 2021a. Requirements for Model-Based Development Process Design and Compliance of Standardized Models. Systems, 9(1), p.3.
Shaked, A. and Reich, Y., 2021b. Using Domain-Specific Models to Facilitate Model-Based Systems-Engineering: Development Process Design Modeling with OPM and PROVE. Applied Sciences, 11(4), p.1532.
Shaked, A. and Reich, Y., 2021c. Improving Process Descriptions in Research by Model-Based Analysis. IEEE Systems Journal, 15(1), pp. 435-444.
Shaked, A. and Reich, Y., 2021d. Model-based Threat and Risk Assessment for Systems Design. In Proceedings of the 7th International Conference on Information Systems Security and Privacy.
Vaculín, R., Hull, R., Heath, T., Cochran, C., Nigam, A. and Sukaviriya, P., 2011, August. Declarative business artifact centric modeling of decision and knowledge intensive business processes. In 2011 IEEE 15th International Enterprise Distributed Object Computing Conference (pp. 151-160). IEEE.
Yongchareon, S., Yu, J. and Zhao, X., 2015. A view framework for modeling and change validation of artifact-centric inter-organizational business processes. Information systems, 47, pp.51-81.