A study on user-friendly formal specification languages for requirements formalization

Pang, Cheng; Pakonen, Antti; Buzhinsky, Igor; Vyatkin, Valeriy

Published in:
Proceedings - 2016 IEEE 14th International Conference on Industrial Informatics, INDIN 2016

DOI:
10.1109/INDIN.2016.7819246

Published: 01/01/2016

Document Version
Peer reviewed version

A Study on User-Friendly Formal Specification Languages for Requirements Formalization

Cheng Pang[1], Antti Pakonen[2], Igor Buzhinsky[1, 3], and Valeriy Vyatkin[1, 4]
[1]Department of Electrical Engineering and Automation, Aalto University, Espoo, Finland
[2]VTT Technical Research Centre of Finland, Espoo, Finland
[3]ITMO University, Saint-Petersburg, Russia
[4]Department of Computer Science, Electrical and Space Engineering, Luleå University of Technology, Sweden
cheng.pang.phd@ieee.org, antti.pakonen@vtt.fi, igor.buzhinskii@aalto.fi, and vyatkin@ieee.org

***Abstract*-Formal methods and languages are used to prove the correctness of various industrial systems, especially mission-critical ones. They can also be viewed as a means to provide safety and correctness demonstration to the stakeholders of such systems. In domains such as nuclear power plant engineering, the benefits from structured safety evidences would seem obvious. However, most stakeholders in nuclear power industry are not even familiar with formal notations. As a result, to promote the applications of formal methods in practice, the first step is to make formal specification languages (FSLs) more accessible. With user-friendly FSLs, users can focus on safety requirements rather than on their sophisticated formalization. This paper, as a preliminary work towards an integrated framework supporting transparent safety demonstration, reviews existing approaches applied to facilitate requirements formalization and formal specifications. Moreover, the common features of user-friendly languages and their tool supports are also summarized.**

Keywords — formal methods; formal specification languages; model checking; visual formalisms; requirements engineering

I. Introduction

Studies on formal methods and their applications in various safety-critical domains have been conducted for decades. However, routine applications of formal methods in practice are still not very pervasive and limited to certain professionals. In nuclear power plant engineering, formal methods have been applied in several stages. For example, formal specification languages (FSLs) can be used to specify the design requirements of instrumentation and control systems in a nuclear power plant. Requirements specified in FSLs are more precise and unambiguous. This helps practitioners to ensure that they have captured and accurately reflected all the required functions and constraints. Thanks to the rigorous mathematical logic, formal requirements can be automatically validated to reveal conflicts and allow consistent communication among stakeholders during the entire system lifecycle. However, formalization of design requirements or system properties into abstract formal logic and the correct understandings of complex mathematical expressions require substantial expertise and practices [1, 2]. Typical industry practitioners and their customers often lack such knowledge and training. Also, due to the subtleties of formalisms and symbolic expressions, even experienced users can easily make mistakes especially when composing complex formal specifications. More dangerously, incorrect formal specifications invalidate the verification results, which is unknown to the users [3].

Moreover, one fundamental challenge in nuclear power industry is to demonstrate the safety of the designed systems. To be effective and efficient, formal requirements and their verification results together with other evidences must be presented in a comprehensive but understandable form for the authority to intuitively assess the safety claims. This requires more user-friendly approaches for writing and reading formal specifications so that people can concentrate on the safety requirements rather than on their formalization. This paper, as a preliminary work towards an integrated framework for safety assessment and transparent safety demonstration in nuclear power industry, investigates existing ways to make FSLs, especially for model checking [4] in our context, more user-friendly. As different FSLs are more appropriate for formalizing specific types of requirements, this paper studies the common techniques to facilitate requirements formalization and formal specification composition, and to improve their overall comprehensibility.

This paper is organized as follows. Section II first summarizes the existing approaches for requirements formalization. Then, Section III focuses on visual formalisms and related graphical methods to facilitate the composition of formal specifications. Section IV discusses the features that must be considered when designing a user-friendly FSL and tools for its support. Finally, Section V concludes this paper and outlooks further research.

II. Requirements Formalization

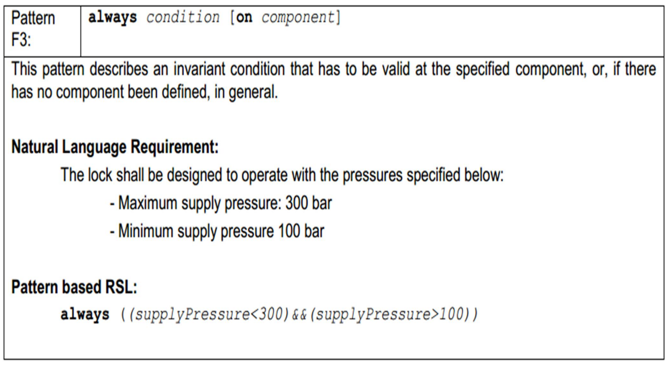
One important application of FSLs is to formalize design requirements written in natural languages. This allows the requirements to be processed by computers for validation, communication, and management unambiguously and consistently. One common approach to assist requirements formalization is to use predefined templates and patterns. In general, templates [5] or boilerplates [6] are pre-formatted textual representations of semi-formal requirements. Each template or boilerplate consists of two parts: fixed keywords and attribute placeholders. When a template or boilerplate is instantiated, its placeholders are substituted by concrete values. For example, the following C-BP16 boilerplate [7] is used to specify a system behavior that shall occur:

C-BP16: <*system*> shall <*action*>,

where <*system*> and <*action*> are placeholders. Moreover, simpler templates and boilerplates can be further combined to specify more complex requirements, which improves their reusability.

The usefulness of templates or boilerplates alone is quite limited. As their number increases, their selection and instantiation become difficult and unmanageable, especially when there are multiple templates or boilerplates that can be used to express the same requirement. Therefore, tools must be developed to guide or automate their selection, combination, and instantiation. For example, the CNL editor [8] provides auto-completion and prompt functions to assist the editing of requirements based on templates. The DODT tool [9] uses boilerplates to partially automate the translation of English requirements into semi-formal requirements. With tools, templates and boilerplates are proved to improve requirements formalization while still maintaining their readability.

Requirements composed using templates and boilerplates are semi-formal and hence cannot be directly processed and analyzed using formal methods. Templates and boilerplates are rather a means to make the structure of requirements more consistent. On the other hand, patterns, in our context, are proven formal textual representations used to specify recurring requirements in a domain. Each pattern has rigorous semantics, which explicitly prescribes its applicability and ensures its consistent interpretation. Similarly, each pattern consists of fixed keywords and attribute placeholders. In the functional pattern exemplified in Fig. 1, keywords are in bold, placeholders are in italic, and square brackets indicate optional phrases. The descriptions instruct the pattern's usage.



Pattern F3:	**always** *condition* [**on** *component*]

This pattern describes an invariant condition that has to be valid at the specified component, or, if there has no component been defined, in general.

Natural Language Requirement:

The lock shall be designed to operate with the pressures specified below:

- Maximum supply pressure: 300 bar
- Minimum supply pressure 100 bar

Pattern based RSL:

always *((supplyPressure<300)&&(supplyPressure>100))*

Fig. 1. Functional pattern example. [7]

Patterns were first used by Dwyer et al. in [10] to facilitate the composition of generic formal specifications for model checking. In [11], Dwyer et al. further refined and evaluated their proposed patterns over a sample of 500 property specifications, which demonstrated the practicality of formal specification patterns. In subsequent research, more specific patterns have been proposed and empirically studied in various domains. For example, in [12] Bitsch first summarized the characteristics of generic safety requirements for industrial automation systems. He then classified the safety requirements into a checklist, which guides users to select the appropriate formal patterns. Similarly, Campos et al. [13, 14] surveyed and proposed formal patterns for automated production systems based on existing literature. Campos et al. also realized that once the number of patterns increases, it is difficult to detect the errors occurred during the process of manual selection and application of patterns. As a result, they have developed the Properties Editor tool, similar to the CNL editor [8], to assist and automate the generation of formal requirements based on formal patterns. It can be concluded that formal patterns have become one useful vehicle for capturing and transmitting knowledge of formal methods. However, to fully leverage formal patterns to simplify the process of requirements formalization, corresponding tools must be developed to organize patterns and guide their selection and instantiation.

Another important technology frequently used to facilitate requirements formalization is domain ontology. A domain ontology is a collection of pre-agreed concepts, terms, relations, and axioms for a specific domain. The combination of a domain ontology and a set of requirements templates forms a simple controlled natural language (CNL), whose vocabulary and grammar are restricted [8]. A well-designed CNL has the right equilibrium between the language's expressiveness and its ability to be processed by computers. Grover et al. [15] exemplified an interactive approach based on CNL to compose formal specifications for model checking. If the requirements are written using a CNL whose vocabulary comes from a domain ontology, the requirements can be reasoned, to certain degree, to check their consistency.

In contrast to manually formalized informal requirements, there are attempts to automate the formalization of informal requirements directly from natural language descriptions. For instance, Miriyala and Harandi have invented an interactive system, called SPECIFIER [16], which takes informal descriptions written in CNL as input to derive formal specifications by using schemas, analogy, and difference-based reasoning. Soeken and Drechsler [17] utilized natural language processing technologies to extract formal models during the specification of informal textual requirements. Both approaches still require human intervention to refine or correct the captured formal specifications.

III. Visual Formalisms

To apply formal verification, system properties must be expressed as formal specifications such as temporal logic [18] before they can be verified. Ideally, these properties can be extracted directly from the formal requirements. However, in reality the requirements are usually written in natural languages. Also, in many cases existing informal requirements adapted from, for example, standards and previous project documents must be reused. For general practitioners, the composition of formal specifications is difficult due to the unfamiliar syntaxes and subtleties in semantics [19]. One common approach to facilitate the composition is to use visual formalisms or other graphical notations that are close to the users' knowledge domains.

It is known that pictorial information is much easier processed and understood by human brains compared to pure texts [20]. Therefore, it is believed that appropriate visual representations can facilitate the comprehension of complex data, information, and notions, such as formal specifications. However, graphical notations alone are not sufficient to illustrate all system properties. In [21], Razal conducted an empirical assessment to evaluate the efficacy of graphical formal methods (GFMs). GFMs unify textual formal symbols with intuitive graphical notations to compose formal specifications. Theoretically, GFMs leverage graphical notations to hide the complexity of mathematical logic and therefore improve the readability of formal specifications. However, arbitrary combination of textual and graphical notations is meaningless. To be useful, the two notations must be complementary and fundamentally compatible. Otherwise, they cannot be used jointly to represent the same information from different perspectives. Fig. 2 illustrates the UML-B GFM proposed in [22] where UML diagrams are integrated with B notations [23]. Due to the formal semantics, GFMs can be transformed directly into formal models as the input for verification tools. If supported by integrated tools, the aforementioned process can be fully automated and hence enhance GFM's accessibility. To evaluate the suitability of GFMs, Razal suggested to use theories such as ontological evaluation [24] to quantify whether the designed notations can effectively convey the users' intentions.

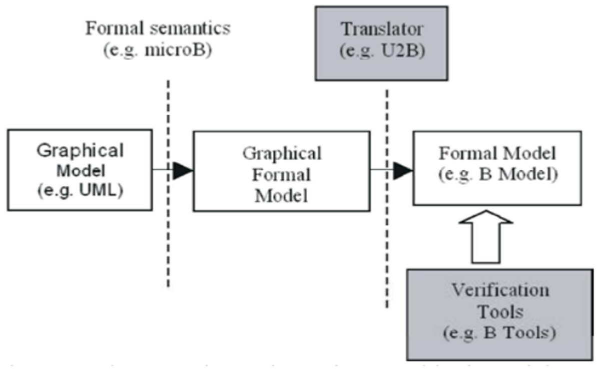


Fig. 2. UML-B graphical formal model. [21]

For GFMs to be effective and user-friendly, comprehensive tool support is compulsory. Amálio and Glodt have further confirmed this in [25], where the UML-like Visual Contract Language (VCL) specifies the visual primitives for graphical modelling of predicates and system dynamics. The Visual Contract Builder (VCB) tool provides comprehensive supports for the editing and consistency checking of VCL diagrams. VCB also automates the transformation of VCL models to Z specifications. The importance of [25] lies in the first empirical evaluation of tool support for GFMs. In particular, a rigorous survey based on statistical hypothesis testing has been designed and conducted. This work provides guidance for quantifying the accessibility and usability of tools for GFMs.

Apart from UML-like visual notations, other graphical documentation languages can also be extended to incorporate formal semantics. For example, in [26], France elaborated an approach to supplement elements of Data Flow Diagram (DFD) with rigorous formal semantics. The extended DFD thus has two aspects: the pictorial representation based on existing DFD notations and the new behavioral semantics defined in algebraic specifications. Lee and Sokolsky [27] proposed a flexible two-level approach to improve the accessibility of the temporal logic L_R. At the first level, experts of formal methods define a set of L_R constructs and related patterns to express properties for a particular domain. Then, at the second level, domain users follow the given patterns to specify concrete system properties as directed acyclic graphs. As illustrated in Fig. 3, nodes of a directed acyclic graph can represent predicates, logical connectives, quantified temporal operators, and modal operators.

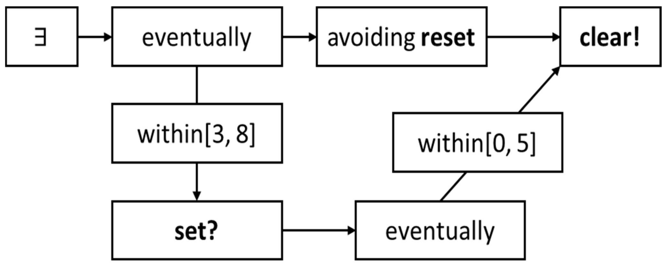


Fig. 3. An example property in L_R. [27]

On the other hand, timing diagrams and alike are frequently adopted to facilitate the property specifications for model checking. This is largely because timing diagrams and temporal logics, especially linear-time ones, are semantically coherent. More importantly, timing diagrams are familiar to engineers. Dietz [28] proposed the Constraint Diagrams for specifying assumption and commitment requirements. As indicated in Fig. 4, it is assumed that if process *P* has been idle for ten seconds, then within one second alarm *A* will be triggered. Constraint Diagrams can be directly compiled into the Duration Calculus interval temporal logic.

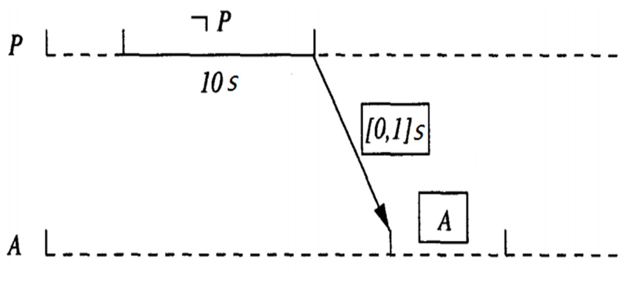


Fig. 4. A watchdog Constraint Diagram. [28]

In the Real-Time Graphical Interval Logic (RTGIL) [29], similar timeline constructs have also been used to depict interleaving events and their duration constraints for concurrent real-time systems. The semantics of RTGIL is based on propositional interval temporal logic. Therefore, with the provided theorem prover and counterexample generator, RTGIL specifications can be formally verified. Fisler [30] proposed a diagrammatic logic, called Timing Diagram Logic (TDL), which has customizable semantics and supports timing constraints with variables. TDL specifications can be converted to Büchi automata for verification. Smith et al. [31] developed the TimeLine Editor to visualize the specifications of event sequences and their causal relations on a timeline. The timeline is also converted to a Büchi automaton and verified in the Spin model checker. In the manufacturing and industrial automation domains, Symbolic Timing Diagrams (STDs) [32] have been applied in several research works to simplify the specification process. For instance, Preusse [33] adapted the standard semantics of STDs to simplify the specifications of production requirements and operation sequences of manufacturing plants. The STD specifications can then be automatically translated into the Computational Temporal Logic (CTL) [34]. In [35] Vyatkin and Bouzon developed a visual specification language, which resembles STDs, to specify partially ordered events in the input and output signals of industrial automation controllers. The graphical specifications can also be translated into CTL for model checking.

The above works are all related to the visual representations of formal specifications. Özcan et al. [36], on the other hand, investigated the possibility of visualizing executable formal specifications via animation. The intention was to promote the use of formal specifications in software prototyping stage to capture and formalize design requirements. Özcan et al. demonstrated their idea using a Water Level Monitoring System (WLMS). Initially, the functional requirements of WLMS were informally described in English. Then, as shown in Fig. 5, the TranZit editor was used to manually formalize the informal requirements using the Z notation [37]. The formal specifications in Z were then translated into an extended LISP format, which can be animated in the ZAL environment. Developers can interact with ZAL to validate properties of the original formal specifications by observing the animation.

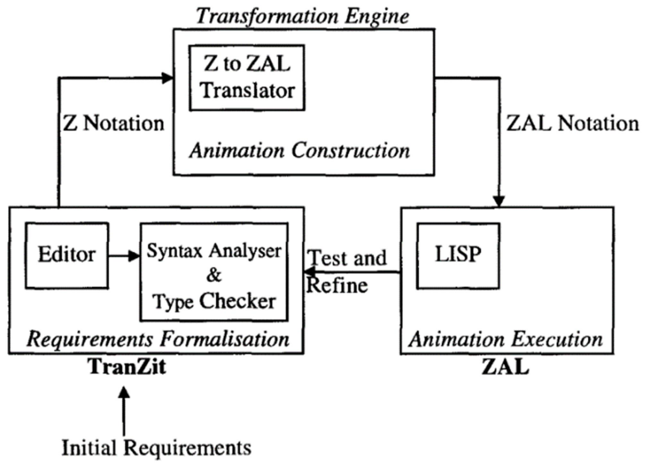


Fig. 5. Validation of requirements in the ZAL system. [36]

The approach of Özcan et al. also revealed some issues. First of all, the validation process starts with visualizing the target system. A visual prototype of the system is built mainly based on terms in the formal specifications. If the terms cannot be easily visualized, i.e. directly mapped to concrete events or objects, the benefits of visualization are limited. Moreover, additional information must also be visualized in order to provide contextual details for the developers to comprehend the animated scenarios and to validate the formal specifications. These contextual details are added according to developers' experiences and understandings about the target system. However, inappropriate contexts may affect the judgements on the validity of the visualized formal specifications. Also, each formal specification only defines a particular application scenario of the target system. The composition of valid fragment scenarios may not result in a valid overall behavior. As a result, Özcan's approach can help developers comprehend individual system requirements in early development stages. The proposed framework is also useful to formally verify the system requirements in late stages with a full formal model of the target system.

Table 1 summarizes the formalisms mentioned in this section with a particular focus on supporting tools. As visible from the table, the majority of formalisms have been supported with tools at least during the time of their development, but currently their availability is doubtful, with several exceptions (e.g. UML-B and VCL).

IV. Usability of User-Friendly Formal Specification Languages and Tools

Designing a user-friendly FSL is difficult as a number of technical and philosophical factors must be taken into account. This section summarizes some essential factors affecting the usability of FSLs according to the literature.

Leveson et al. shared their experiences in the design and development of FSLs for commercial process control systems in [38]. Their work was closely collaborated with industry authorities and domain experts, which makes their lessons very valuable. Leveson et al. have summarized five issues that must be tackled when designing an FSL. The first issue is how to bridge the semantic gap between the users', e.g. industry practitioners, mental model and the model composed using the FSL. It was concluded that the smaller the gap, the better the readability and reviewability of the FSL. A general solution is to make the FSL design user-centered by using syntaxes, notations, design principles, etc. based on the users' domain knowledge. This is demonstrated by the work of Ljungkrantz et al. elaborated later in this section. The second issue is to assist the users to construct black-box requirement models, which only describe the externally visible behavior of a system. This means that details of system implementation and internal design should not be included in the requirement specifications. Therefore, mechanisms and features must be introduced in the FSL to enforce black-box specifications. Fig. 6 indicates how user-centered FSLs and black-box specifications can help reduce the semantic gaps. It can be

TABLE 1. VISUAL FORMALISMS AND TOOLS FOR THEIR SUPPORT

Visual formalism	Basis	Tools
UML-B [22]	UML, B	UML-B plugin for Rodin (Event-B IDE). Available: http://wiki.event-b.org/index.php/UML-B
Visual Contract Language (VCL) [25]	UML	Visual Contract Builder (VCB) [25] – plugin for Eclipse. Available: https://vcl.gforge.uni.lu/download.html
Semantically Extended DFD [26]	Data Flow Diagram (DFD)	A tool is mentioned in [26]; no evidence of current maintenance / availability found
GFL for L_R [27]	L_R temporal logic	Supported in the PARAGON toolset, as mentioned in [27]; no evidence of current maintenance / availability found
Constraint Diagrams [28]	Duration Calculus interval temporal logic	A tool is described in [39]; no evidence of current maintenance / availability found
Real-Time Graphical Interval Logic (RTGIL) [29]	Propositional interval temporal logics	A tool is mentioned in [29]; no evidence of current maintenance / availability found
Timing Diagram Logic (TDL) [30]	Timing diagrams	–
TimeLine Editor [31]	Timing diagrams	The TimeLine Editor tool is described in [31]; no evidence of current maintenance / availability found
Symbolic Timing Diagrams (STDs) [32]	Timing diagrams	Two tools are mentioned in [32], section 2.4; no evidence of current maintenance / availability found
Visual specification language [35]	Timing diagrams, Net condition/event systems (NCES)	Timing Diagram Editor tool is mentioned in [35]; no evidence of current maintenance / availability found
Visualizing executable formal specifications via animation [36]	Z notation	TranZit and ZAL tools are mentioned in [36]; no evidence of current maintenance / availability found

seen that the semantic gap d_4 between a user's mental model and the design specification is much bigger than the semantic gap d_1.

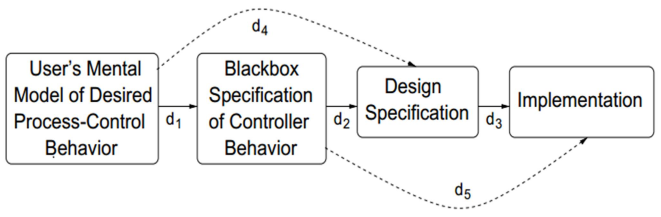

User's Mental Model of Desired Process-Control Behavior
Blackbox Specification of Controller Behavior
Design Specification
Implementation
d_1
d_2
d_3
d_4
d_5

Fig. 6. Schematic semantic gaps.[38]

The third issue regards the prevention of using error-prone features, such as internal broadcast events, in FSLs. A feasible solution is to provide comprehensive guidance when such features are being used. The fourth issue is to increase the reusability of existing formal specifications. To achieve this, means such as macros and functions should be supported by the FSL. Finally, the FSL must facilitate the inspection of incorrect and incomplete requirement specifications. Potential solutions are checklists of formal criteria and the use of language syntax to enforce such constraints. As a subsequent work, a readability assessment of state-based FSLs has been conducted in [40]. In particular, the overall representation of state machine structure, the expression of state transition conditions, and the usage of internal broadcast events, macros, and state hierarchies are empirically evaluated. It is revealed that state machines are most intuitive if presented in graphical or tabular forms, while complex transition conditions are better tabulated. Moreover, macros are helpful for composing specifications but they also exacerbate the difficulty when reading large specifications, especially ones with nested macros. Thus, the use of macros must be facilitated by tools. Finally, hierarchical abstractions are compulsory to enhance scalability of FSLs. Again, tool support must be provided to avoid interpretation errors of hierarchical specifications.

Another way to simplify the writing of formal specifications is to extend existing FSLs with syntaxes and semantics that are close to the users' domain languages. Ljungkrantz et al. [41] proposed an extended linear temporal logic (LTL [18]) called ST-LTL to promote the use of formal methods in the industrial automation domain. ST-LTL is specifically tailored to formally specify control logics of IEC 61131-3 [42] programmable logic controllers (PLCs) written in structured text (ST). Based on the syntax of ST, additional functions, operators, and suffixes are introduced to express LTL properties. This similarity facilitates control engineers to formally specify PLC control logics. Moreover, due to the execution model of PLCs, some semantic extensions to existing LTL have been incorporated in ST-LTL. These extensions allow ST-LTL to specify complex properties involving timer behavior, sequences, and rising/falling edges of variables. It has been formally proved that ST-LTL has the full expressiveness of LTL.

Later, Ljungkrantz et al. conducted an empirical study on the practicality of ST-LTL in [19]. Specifically, 105 input/output related properties of ten IEC 61131-3 function blocks used in manufacturing industry were investigated. The study revealed that logical implications are the specification type most frequently used by control engineers in practice.

In order to be user-friendly, FSLs and patterns must be tailored specifically for the target users to match their knowledge and application domains. On the other hand, tools can significantly improve the user experiences of FSLs. Based on the Cognitive Dimensions of Notations framework [43] and related ISO criteria, Razali and Garratt [44] have conducted a survey to assess the usability of two formal verification tools for the B method. This provides some insights and guidelines for designing tools to further improve the accessibility of FSLs. The tools are evaluated according to three categories of feature properties: interface, utility, and resource management. Regarding tool interface, menus, panes, and dialogues must be well structured and organized so that they can effectively and promptly provide comprehensive information to users. For the formal modelling utilities, apart from routine functions like graphical editing, syntax checking, and verification, features such as model visualization and code generation are also helpful. The resource management facilities are used to support tool execution.

V. Conclusion

To promote the applications of formal methods as part of industrial practitioners' daily works, a user-friendly approach for specifying formal requirements is compulsory. Previous studies on formal specification languages (FSLs) have revealed several features and factors that can enhance the usability of FSLs. For requirement formalization, templates and patterns based on domain ontologies can greatly simplify the composition of FSL formulae. FSLs with syntaxes and notations close to the domain languages used by industry practitioners will facilitate the property formalization process. Moreover, tool support and graphical representations are also compulsory elements to make FSLs more accessible.

In future works, the industrial practices of requirements engineering in the nuclear power plant engineering domain will be investigated. Based on these results, a visual FSL will be proposed that would potentially bridge the gap between specification practices in the nuclear industry and in the formal method research domain.

Acknowledgment

This work was supported, in part, by the Integrated Safety Assessment and Justification of Nuclear Power Plant Automation (SAUNA) project of the Finnish Research Programme on Nuclear Power Plant Safety 2015–2018 (SAFIR 2018) [45] funded by the Nuclear Waste Management Fund (VYR) [46], and by the Government of Russian Federation, Grant 074-U01.

References

[1] K. Finney, "Mathematical Notation in Formal Specification: Too Difficult for the Masses?," *IEEE Transactions on Software Engineering,* vol. 22, pp. 158-159, 1996.

[2] R. Schlör, B. Josko, and D. Werth, "Using a Visual Formalism for Design Verification in Industrial Environments," in *Services and Visualization Towards User-Friendly Design.* vol. 1385, T. Margaria, B. Steffen, R. Rückert, and J. Posegga, Eds., ed: Springer Berlin Heidelberg, 1998, pp. 208-221.

[3] G. J. Holzmann, "The Logic of Bugs," presented at the 10th ACM SIGSOFT symposium on foundations of software engineering, Charleston, SC, US, 2002.

[4] E. M. Clarke, O. Grumberg, and D. Peled, *Model Checking*. Cambridge, Massachusetts, US: The MIT Press, 1999.

[5] T. Tommila and A. Pakonen, "Controlled natural language requirements in the design and analysis of safety critical I&C systems," VTT Technical Research Centre, Finland, Research Report VTT-R-01067-14, 2014.

[6] *CESAR - Cost-efficient Methods and Processes for Safety-relevant Embedded Systems*: Springer Vienna, 2013.

[7] CESAR. (2011). *Improved Definition of RSL and RMM (1.0 ed.)* [Online]. Available: http://www.cesarproject.eu/fileadmin/user_upload/CESAR_D_SP2_R2.3_M3_v1.000_PU.pdf

[8] T. Tommila, A. Pakonen, and J. Valkonen, "Ontology-Driven Natural Language Requirement Templates for Model Checking I&C Functions," in *Enlarged Halden programme group meeting (EHPG 2013)*, Storefjell, Norway, 2013.

[9] S. Farfeleder, T. Moser, A. Krall, T. Stålhane, H. Zojer, and C. Panis, "DODT: Increasing Requirements Formalism using Domain Ontologies for Improved Embedded Systems Development," in *14th IEEE International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS 2011)*, Cottbus, Germany, 2011, pp. 271-274.

[10] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, "Property Specification Patterns for Finite-State Verification," in *2nd Workshop on Formal Methods in Software Practice*, Clearwater Beach, FL, US, 1998, pp. 7-15.

[11] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, "Patterns in Property Specifications for Finite-State Verification," in *21st International Conference on Software Engineering (ICSE 1999)*, Los Angeles, CA, US, 1999, pp. 411-420.

[12] F. Bitsch, "Safety Patterns — The Key to Formal Specification of Safety Requirements," in *Computer Safety, Reliability and Security*. vol. 2187, U. Voges, Ed., ed: Springer Berlin Heidelberg, 2001, pp. 176-189.

[13] J. C. Campos, J. Machado, and E. Seabra, "Property Patterns for the Formal Verification of Automated Production Systems," in *17th IFAC World Congress*, Seoul, Korea, 2008, pp. 5107-5112.

[14] J. C. Campos and J. Machado, "A Specification Patterns System for Discrete Event Systems Analysis," *International Journal of Advanced Robotic Systems,* vol. 10, 315, pp. 1-13, 2013.

[15] C. Grover, A. Holt, E. Klein, and M. Moens, "Designing a Controlled Language for Interactive Model Checking," presented at the 3rd International Workshop on Controlled Language Applications, Seattle, WA, US, 2000.

[16] K. Miriyala and M. T. Harandi, "Automatic Derivation of Formal Software Specifications From Informal Descriptions," *IEEE Transactions on Software Engineering,* vol. 17, pp. 1126-1142, 1991.

[17] M. Soeken and R. Drechsler, *Formal Specification Level: Concepts, Methods, and Algorithms*: Springer International Publishing, 2015.

[18] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science*, Providence, RI, US, 1977, pp. 46-57.

[19] O. Ljungkrantz, K. Åkesson, M. Fabian, and A. Ebrahimi, "An empirical study of control logic specifications for programmable logic controllers," *Empirical Software Engineering,* vol. 19, pp. 655-677, 2014.

[20] B. A. Myers, R. Chandhok, and A. Sareen, "Automatic Data Visualization for Novice Pascal Programmers," in *IEEE*

Workshop on Visual Languages, Pittsburgh, PA, US, 1988, pp. 192-198.

[21] R. Razal, "Understanding the Efficacy of Graphical Formal Methods-Empirical Assessment," *World Applied Sciences Journal,* vol. 31, pp. 889-903, 2014.

[22] C. Snook and M. Butler, "UML-B: Formal modeling and design aided by UML," *ACM Transactions on Software Engineering and Methodology (TOSEM),* vol. 15, pp. 92-122, 2006.

[23] J.-R. Abrial, *The B-Book: Assigning Programs to Meanings*: Cambridge University Press, 2005.

[24] R. Weber, "Conceptual Modelling and Ontology: Possibilities and Pitfalls," *Journal of Database Management,* vol. 14, pp. 1-20, 2003.

[25] N. Amálio and C. Glodt, "A tool for visual and formal modelling of software designs," *Science of Computer Programming,* vol. 98, Part 1, pp. 52-79, 2015.

[26] R. B. France, "Semantically Extended Data Flow Diagrams: A Formal Specification Tool," *IEEE Transactions on Software Engineering,* vol. 18, pp. 329-346, 1992.

[27] I. Lee and O. Sokolsky, "A Graphical Property Specification Language," in *High-Assurance Systems Engineering Workshop*, Washington, DC, US, 1997, pp. 42-47.

[28] C. Dietz, "Graphical Formalization of Real-Time Requirements," in *Formal Techniques in Real-Time and Fault-Tolerant Systems*. vol. 1135, B. Jonsson and J. Parrow, Eds., ed: Springer Berlin Heidelberg, 1996, pp. 366-384.

[29] L. E. Moser, Y. S. Ramakrishna, G. Kutty, P. M. Melliar-Smith, and L. K. Dillon, "A Graphical Environment for the Design of Concurrent Real-Time Systems," *ACM Transactions on Software Engineering and Methodology (TOSEM),* vol. 6, pp. 31-79, 1997.

[30] K. Fisler, "Timing Diagrams: Formalization and Algorithmic Verification," *Journal of Logic, Language and Information,* vol. 8, pp. 323-361, 1999.

[31] M. H. Smith, G. J. Holzmann, and K. Etessami, "Events and Constraints: A Graphical Editor for Capturing Logic Requirements of Programs," in *5th IEEE International Symposium on Requirements Engineering*, Toronto, Canada, 2001, pp. 14-22.

[32] R. C. Schlör, "Symbolic Timing Diagrams: A Visual Formalism for Model Verification," Doctoral thesis, Fachbereich Informatik, Carl-vonOssietzky Universitat Oldenburg, Germany, 2001.

[33] S. Preusse, *Technologies for Engineering Manufacturing Systems Control in Closed Loop*: Logos Verlag Berlin, 2013.

[34] E. A. Emerson and E. Clarke, "Characterizing correctness properties of parallel programs using fixpoints," in *Automata, Languages and Programming*. vol. 85, J. Bakker and J. Leeuwen, Eds., ed: Springer Berlin Heidelberg, 1980, pp. 169-181.

[35] V. Vyatkin and G. Bouzon, "Using Visual Specifications in Verification of Industrial Automation Controllers," *EURASIP Journal on Embedded Systems,* vol. 2008, pp. 1-9, 2008.

[36] M. B. Özcan, P. W. Parry, I. C. Morrey, and J. I. Siddiqi, "Visualisation of Executable Formal Specifications for User Validation," in *Services and Visualization Towards User-Friendly Design*. vol. 1385, T. Margaria, B. Steffen, R. Rückert, and J. Posegga, Eds., ed: Springer Berlin Heidelberg, 1998, pp. 142-157.

[37] ISO/IEC Standard 13568, "Information Technology — Z Formal Specification Notation — Syntax, Type System and Semantics," ed, 2002.

[38] N. G. Leveson, M. P. E. Heimdahl, and J. D. Reese, "Designing Specification Languages for Process Control Systems: Lessons Learned and Steps to the Future?," in *Software Engineering — ESEC/FSE '99*. vol. 1687, O. Nierstrasz and M. Lemoine, Eds., ed: Springer Berlin Heidelberg, 1999, pp. 127-146.

[39] E.-R. Olderog and H. Dierks, "Moby/RT: A tool for specification and verification of real-time systems," *J. UCS,* vol. 9, pp. 88-105, 2003.

[40] M. K. Zimmerman, "Investigating the Readability of Formal Specification Languages," Master thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Massachusetts, US, 2002.

[41] O. Ljungkrantz, K. Akesson, M. Fabian, and Y. Chengyin, "A Formal Specification Language for PLC-based Control Logic," in *8th IEEE International Conference on Industrial Informatics (INDIN 2010)*, Osaka, Japan, 2010, pp. 1067-1072.

[42] IEC Standard 61131-3, "Programmable controllers — Part 3: Programming languages," ed, 2013.

[43] T. R. G. Green, "Cognitive Dimensions of Notations," in *People and Computers V*, A. Sutcliffe and L. Macaulay, Eds., ed Cambridge, UK: Cambridge University Press, 1989, pp. 443-460.

[44] R. Razali and P. Garratt, "Usability Requirements of Formal Verification Tools: A Survey " *Journal of Computer Science,* vol. 6, pp. 1189-1198, 2010.

[45] *The Finnish Research Programme on Nuclear Power Plant Safety 2015 - 2018 (SAFIR2018)* [Online]. Available: http://safir2018.vtt.fi/

[46] *Nuclear Waste Management Fund* [Online]. Available: https://www.tem.fi/en/energy/nuclear_energy/nuclear_energy_administration/nuclear_waste_management_fund