Design of formal languages and interfaces: "formal" does not mean "unreadable".

Spichkova, Maria

Spichkova. (2014). Design of formal languages and interfaces: “formal” does not mean “unreadable”. In Emerging Research and Trends in Interactivity and the Human-Computer Interface (pp. 301–313). IGI Global. https://doi.org/10.4018/978-1-4666-4623-0.ch015
Document Version: Published Version

Published Version: https://doi.org/10.4018/978-1-4666-4623-0.ch015

Emerging Research and Trends in Interactivity and the Human-Computer Interface

Katherine Blashki
Noroff University College, Norway

Pedro Isaias
Portuguese Open University, Portugal

A volume in the Advances in Human and Social Aspects of Technology Book Series (AHSAT) Book Series

Information Science REFERENCE
An Imprint of IGI Global

Managing Director: Lindsay Johnston
Editorial Director: Myla Merkel
Production Manager: Jennifer Yoder
Publishing Systems Analyst: Adrienne Freeland
Development Editor: Allyson Gard
Acquisitions Editor: Kayla Wolfe
Typesetter: Christina Barkanic
Cover Design: Jason Mull

Published in the United States of America by
Information Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com

Library of Congress Cataloging-in-Publication Data

Emerging research and trends in interactivity and the human-computer interface /Katherine Blashki and Pedro Isaias, editors.
pages cm
Includes bibliographical references and index.
Summary: “This book addresses the main issues of interest within the culture and design of interaction between humans and computers,exploring the emerging aspects of design, development, and implementation of interfaces”-- Provided by publisher.
ISBN 978-1-4666-4623-0 (hardcover) -- ISBN 978-1-4666-4624-7 (ebook) -- ISBN 978-1-4666-4625-4 (print & perpetual access) 1. Human-computer interaction. 2. Human-computer interaction--Research. I. Blashki, Kathy, 1961- II. Isaias, Pedro.
QA76.9.H85E479 2014
004.01’9--dc23
2013025032

This book is published in the IGI Global book series Advances in Human and Social Aspects of Technology (AHSAT) (ISSN: 2328-1316; eISSN: 2328-1324)

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Chapter 15
Design of Formal Languages and Interfaces:
"Formal" Does Not Mean "Unreadable"

Maria Spichkova
RMIT University, Australia

ABSTRACT

This chapter provides an introduction to a work that aims to apply the achievements of engineering psychology to the area of formal methods, focusing on the specification phase of a system development process. Formal methods often assume that only two factors should be satisfied: the method must be sound and give such a representation, which is concise and beautiful from the mathematical point of view, without taking into account any question of readability, usability, or tool support. This leads to the fact that formal methods are treated by most engineers as something that is theoretically important but practically too hard to understand and to use, where even some small changes of a formal method can make it more understandable and usable for an average engineer.

INTRODUCTION

There are many definitions of human factors, however most of them are solely oriented on human-machine operations in terms of system and program usability, i.e. on those parts that are seen by the (end-)user, but not by the requirements, specification and verification engineers. Nevertheless, many problems during the engineering phase are completely the same as by using the final version of a system just because of a simple fact that many people sometimes forget: engineers, even those who are working on verification or formal specification, are humans too and have the same human abilities and weaknesses as people working in any other areas, from arts to construction. Moreover, developing safety-critical systems using formal methods means much harder constraints and stress than using a completed version of software application (e.g., using an entertainment software, typing a personal e-mail using a smartphone, etc.) because of consequences of

DOI: 10.4018/978-1-4666-4623-0.ch015

any mistake: a typo in an e-mail can lead to misunderstanding which is easy to clear up, where a specification or verification error by developing of a safety-critical system, like a fly-by-wire system for airlines or pre-crash safety functionality for vehicles, can cost many human lives.

Nowadays, the research of human factors and of Human Computer Interface (HCI) mostly concentrates on the development of entertainment or every-day applications, but it was initiated and elaborated exactly because of mistakes in usage and development of safety-critical systems. For example, one of the widely cited HCI-related accidents in safety-critical systems are the accidents involved massive radiation overdoses by the Therac-25 (a radiation therapy machine used in curing cancer) that lead to deaths and serious injuries of patients which received thousand times the normal dose of radiation (Miller, 1987; Leveson & Turner, 1993). The causes of these accidents were software failures as well as problems with the system interface.

The Therac-25 was an extension of the two previous models, the Therac-6 and the Therac-20, but the upgrade was unsafe: the software was not correctly updated and adapted to the elaborated extensions in the system architecture. In this model, in comparison to the previous ones, the company tried to mix two system modes, a low-energy mode and a high-energy mode, together. In the high-energy mode the filter plate must be placed between the patients and the X-ray machine, so that a radiation beam is used in a correct way. Because of some software failures the high-energy mode was used in the Therac-25 without the filter plate. This kind of failures occurred also in the old models, but it did not lead to overdosed accidents due to hardware interlocks. In the Therac-25 the company replaced the hardware interlocks with software checks, this result in a deathly overdosed treatment.

The HCI-related problem with this machine was that the Therac-25 in some cases displayed system states incorrectly and showed just some error codes instead of full warning or error messages, and, moreover, these codes were not even well documented. As the result, the operator of the Therac-25 was not able to recognise a dangerous error situation and continued the treatment even after the machine showed warning messages, which did not look like a warning or a signal to stop the treatment. Together with very little training, this caused the operators not aware of the importance of keeping the safety guideline and as a result, they violated many of the safety guidelines. In some case, the operators conducted the treatment even when the video and audio monitoring, which were the only method to observe the patient in separated room, were not working. These accidents have shown that studying the human errors and their causation should be a significant part of software and system engineering at least in the case of safety-critical systems.

An appropriate system interface which allows a correct human computer interaction is just as important as correct, errorfree behaviour of the developed system: even if the system we develop behaves in an ideal correct way, this does not help much in the case the system interface is unclear to the user or is too complicated to be used in a proper way. According to statistics presented in (Dhillon, 2004), the human is responsible for 30% to 60% the total errors which directly or indirectly lead to the accidents, and in the case of aviation and traffic accidents, 80% to 90% of the errors were due to human. Thus, it is necessary to take human factors into account by developing safety-critical systems.

The fundamental goal of human factors engineering, as claimed in (Wickens, Hollands 2000), is to reduce errors, increase productivity and safety when the human interacts with a system. Engineering psychology applies psychological perspective to the problems of system design and focuses on the information-processing capacities of humans. The goals of formal methods are almost the same: to reduce errors, increase productivity and safety of the developed systems, however,

the formal methods does not focus on the user of the system and the interface between the user and the system – they focus on the system itself, but only in very rare cases they take into account information-processing capacities of engineers.

In our approach *Human Factors of Formal Methods*, HF^{2}M, we focus on human factors in formal methods used within formal specification phase of a system development process (Feilkas et al., 2011; Feilkas et al., 2009): on (formal) requirements specification and on the developing of a system architecture that builds a bridge between requirements and the corresponding system.

The main ideas of the approach are language and framework independent, but for a better readability and for better understanding of these ideas we show them based on formal specification presented in the Focus (Broy & Stølen, 2001), a framework for formal specifications and development of interactive systems.[1] We can also see this methodology as an extension of the approach "Focus on Isabelle" (Spichkova, 2007) integrated into a seamless development process, which covers both specification and verification, starts from informal specification and finishes by the corresponding verified C code (Hölzl et al., 2010; Spichkova et al., 2012).

BACKGROUND

There are many applications of formal methods to analyse human computer interaction and to construct user interfaces, e.g., (Shackel & Richardson, 1991; Følstad et al., 2012), as well as a number of approaches on the integrating human interface engineering with software engineering, e.g., (Volpert, 1991; Heumann, 2002; Constantine, 2003), but the field of application of human factors to the analysis and to the optimization of *formal methods* area is still almost unexplored. To our best knowledge there are no other works on this field, the nearest area is only the application of human factors to the development of engineering tools, however, there are many achievements in the HCI research that could be applicable within the formal languages as well as verification and specification engineering tools, for example, the ideas of the usage-centered approach for presentation and interaction design of software and Web-based applications were introduced in (Constantine & Lockwood, 1999; Constantine & Lockwood, 2002).

Speaking of any kind of science and research, one can say that a lot of new ideas are just well forgotten old ones, and a lot of newly developed methodologies are, in fact, the reinvention of the wheel. Leaving the research results solely in the area they are introduced, or just forgetting them does not have any benefit, vice versa, application of old ideas on a new field brings them to a new level and gives them new power to improve safety or, even more general, living standards.

Unfortunately, we should acknowledge that dealing with formal methods often assumes that only two factors must be satisfied: the method must be sound and give such a representation, which is concise and beautiful just from the mathematical point of view, without taking into account any question of readability, usability, or tool support. This leads to the fact that formal methods are treated by most engineers as "something that is theoretically important but practically too hard to understand and to use", and, moreover, the term "formal" is for many people just some kind of synonym for "unreadable", however, even small syntactical changes of a formal method can make it more understandable and usable for an average engineer.

Looking on the matter from a different standpoint, we can see that most of programming languages have a formal background, even if this is not mentioned to programmers and engineers explicitly. For example, the Structured Query Language (SQL) is nowadays a standard for managing data in relational database management systems, however it is originally based upon relational algebra and relational calculus – this side of the

programming language is generally unimportant to the SQL-programmers, being an important feature for the developers of the language itself.

Using natural languages, e.g., English, to specify a system we profit by their flexibility and power, we do not have any special learning efforts, because we can write the specification directly, without encoding. These advantages sounds very attractive, but considering a specification of a safety-critical system, their all are exceeded by the disadvantage that a natural language is ambiguous, vague, and imprecise. A formal language, even if it requires an initial learning effort and uses a notation unfamiliar to an average engineer, is unambiguous and precise, and, moreover, due a predefined syntax and semantics a formal language is machine processible, i.e. using such a specification we could do some development steps (semi-)automatically.

Because of this image of formal methods, some approaches try to cover the fact they have formal background and "simulate" the appearances of informal representation to look user-friendly. In some cases it implies that the approach becomes semiformal or introduces extra specification ambiguity. For example, controlled natural languages (CNL) try to avoid disadvantages of both natural and formal languages and being a subset of a natural language have a well-defined syntax and semantics (Macias & Pulman,1993; Fuchs & Schwitter,1995). Their syntax is unambiguous, but engineers can interpret the semantics of some sentences in wrong way just because the language looks like a natural one and this gives a feeling it can be also used according to all rules of the natural language, i.e. the restriction can be ignored through lack of attention which is "provoked" by the visual similarity to the natural language.

A famous example of the misinterpretation is the sentence "I see the girl with the telescope". In English, this sentence allows not only the interpretation "I see the girl via the telescope" but also the interpretation "I see the girl which has a telescope". Which one should be correct in the case of CNL? If we want to have an unambiguous syntax, we should take a choice. E.g., in Attempto Controlled English (Kuhn, 2010; Fuchs & Schwitter, 2007) only the first interpretation is allowed, but reading such a specification it is very easy to forget this rule. Moreover, looking at the specification in controlled language, an engineer can consider that he does not need to know *any* rules, because he *consider* he can understand the specification without spending time on any additional training, whereas he misunderstand it.

Specifying safety-critical systems, it is not enough to use controlled languages and semiformal languages – the precise formal specification is essential to ensure that the safety properties of the system really hold. Speaking about human factors according to the safety-critical systems we focus mostly on technical aspects; this idea, applied to the formal methods, is often called *Engineering Error Paradigm* (Redmill & Rajan, 1997). Human factors that are targeted by the Engineering Error Paradigm typically include the design of HCI as well as the corresponding automatization: by this paradigm humans are seen as they are almost equivalent to software and hardware components in the sense of operation with data and other components, but at the same time humans are seen as "the most unreliable component" of the total system. This implies also that designing humans out of the main system actions through automatization of some system design steps is considered as a proposal for reducing risk. In the case of design of safety-critical systems, this means automatic translation from one representation kind to another one, e.g., between two formal languages or between two internal representation within some tools.

Another important view of the Engineering Error Paradigm is that human errors often occur as a result of mismatch in HCI and overestimation of physical capabilities of a person. With other words, human performance and reliability need to be considered in the design process (Klare, 2000); in our case, we have to focus on clearness – up to obviousness – and readability of formal specifications. For these reasons we have to analyse the

achievements of HCI approaches to apply their ideas on another kind of HCI – interface between (verification, software) engineer and the applied formal method or tool. *The Individual Error Paradigm* (Redmill & Rajan, 1997) focuses on understanding the reasons why people make mistakes or commit unsafe acts, and then tries to eliminate those reasons. The same idea should be applied to analyse the syntax of a formal method: Which kind of specification mistakes and misreading is prevailing? How can we prevent them? Can we do it automatically or, at least, semi-automatically?

HUMAN FACTORS + FORMAL METHODS = HF²M

One of the common mistakes by writing a system specification, particularly writing a requirement specification, is the omission of assumptions about the system's environment. The concentration on the question *"What we want our system can do?"* is very natural, but it leads to the point that the question *"Under which constraints the correct work of the system can be ensured?"* is ignored, however, the answer to this question gives us a crucial property of the system. To make this kind of mistake is even easier if we have additional efforts through concentration on a formal syntax, but it is also even more disappointing in this case, after devoting much effort in the precise and unambiguous specification. However, the solution to this problem can be really simple and uses the same principle as enriching an email client by an alert like "The attachment keyword is found. Do you want to add the attachment now or should we remind you later?"

Specifying a system formally we should have special alerts that remind us to cover this part of the system description. In the case of the Focus specification language this means to restrict all the specification styles (both textual and graphical) to the variants using the Assumption/Guarantee representation, where a component is specified in terms of an assumption and a guarantee: whenever input from the environment behaves in accordance with the assumption, the specified component is required to fulfil the guarantee. Thus, it will be impossible to overlook the question about the necessary properties of the environment, and if the system does not have any constraints under which it provides the correct functionality, the corresponding field of the specification should be filled out by the constraint "true" representing the property that the system should work correctly in any environment. The probability that an engineer signs this property without checking the corresponding system constraints is much smaller than in the case the engineer do not get any reminder to check these constraints.

As mentioned in our previous work (Spichkova, 2007), during requirements specification phase and the phase of a system architecture development we need to care about later phases (modelling, simulation, testing, formal verification, implementation) already doing the formal or, even, semiformal specification of a system – that is, choosing an appropriate abstraction and modelling technique. A crucial question is here how we can optimize the formal representation and formal methods with respect to human factors. In our approach we focus on the following aspects:

- Representation of the formal specification in more readable way, optimisation of the specification layout/formatting.
- Unification of the representation of different specification views and artefacts by using an integrated specification language.
- Automatization of several aspects of the specification and verification process.

Let discuss these issues in more detail.

Layout/Formatting of a Formal Specification

The main aspect of HF²M is the representation, i.e. layout/formatting and visualization including graphical representation, of formal specification.

The first results of visual optimization of specifications are presented in (Spichkova, 2011b). That work covers all specification styles of the Focus framework – from textual to graphical representation, also covering on the timing aspects of the specification. The notion of time takes central stage for many kinds of safety-critical systems, especially in the case of embedded real-time ones: abstracting from the timing aspects we may lose the core properties of a system we represent, e.g. the causality property. To help an engineer to concentrate on the timing properties of the system to be specified and verified, we introduced so-called timed state transition diagrams (TSTDs). Specifying system behaviour by TSTD we can use three specification styles: classical diagram (automaton), table and also textual style. Inter alia, we suggest to simplify the timed specification in the way to get shorter specifications that are more readable and clear: specifying a component we have often such a case where for some time intervals both conditions hold: local variables are still unchanged and there is no output. This can occur, e.g., if at this time interval the component gets no input or if some preconditions don't hold. In classical Focus, as well as in Isabelle/HOL, we need to specify such cases explicitly otherwise we get an underspecified component that has no information how to act in these cases.

In many cases even not very complicated optimization changes of a specification method can make it more understandable and usable. Moreover, taking into account the Individual Error Paradigm, we can extend specification templates in order to get not only more readable, but also more correct specification, e.g. by introducing an obligatory assumption-part of the specification.

The simplest optimization steps are often overlooked just because of their obviousness, and it would be wrong to ignore the possibility to optimize the language without much effort. For example, simply adding an enumeration to the formulas in a large formal specification as well as extending the specification template by general rules makes its validation on the level of specification and discussion with co-operating experts much easier.

Figure 1 presents an example of this kind of optimisation. The first (basic) specification layout leads to the situation where even a very short specification is hardly readable. In the example we have a specification which guarantee-part consists of just six properties, where even in middle-size case studies an average size of the guarantee-part is at least thirty properties; it is easy to imagine how unreadable could be a large formal specification written using this kind of layout. The second specification has only tiny modifications in formatting vs. the first one, but even adding empty lines between properties of different kind makes the guarantee-part of the specification more readable. In the third specification we number all properties in the guarantee-part with the aim not only to improve the readability but also to make the discussion of the specification more concrete and free of misunderstandings.

1. Basic specification layout.
2. Specification layout with tiny optimizations.
3. Optimized specification layout.

In the HF^{2}M approach, we see a formal specification as a ground to the discussion of the system properties, requirements, and structure, therefore the specification itself plays here a role of an interface between engineers of different disciplines (e.g., software and electrical engineers) and dealing with requirements, system, software, architecture, verification and many other aspects of the development. Thus, applying one of the basic design rules to a formal specification we get very similar results as in the case of development of webpages, interfaces, newspapers, etc. because of the nature of the problem that we are aiming to solve: problems in the information representation are very similar in any area, and the solutions from one area could be adopted to another one.

Figure 1. Comparing different specifications layouts

Unification of the Representation

Another point, which is seen as obvious if we are speaking about interfaces and interaction, is the unification of the representation of any information we are dealing with (cf. also Figure 2). Specifying components and system in a formal language is helpful to have a possibility to change the view on the system or the kind of its description to cover several problem areas by a single specification language: this helps to simplify representation of different views on a system as well as to switch between them. However, it does not make any sense to extend the core of a (formal) language/framework, because this can decrease readability of a specification – an overflow of additional information, which is not really needed to specify a concrete system on a concrete level, can distract

from the important properties and aspects. Thus, we need another solution of this problem. Instead of the reinvention of existing approaches, it is more sufficient to reuse within the formal methods some successful ideas from other areas. Analysing the similar problems within general software engineering, we can see that one of the effective ways is an extension of the core framework by a number of several add-ons covering different application areas and different functionality. According to this idea, we made the following "add-ons like" extension of the Focus formal language:

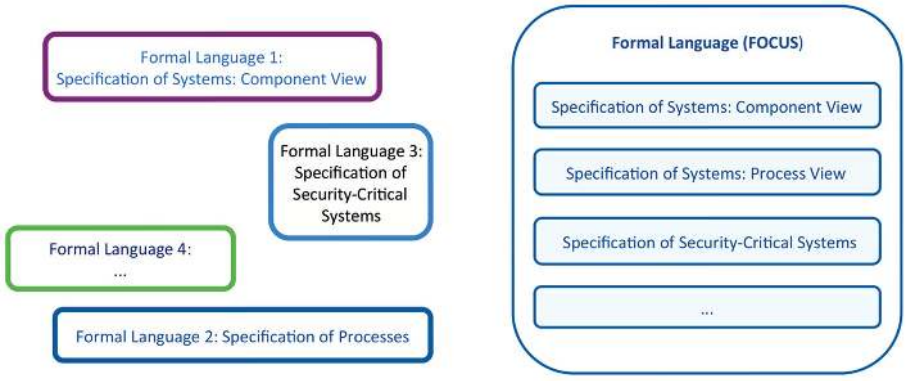
- Specification of processes and matching to the representation of components.
- Specification of security-critical systems with respect to secrecy properties.

Specifying systems in a formal language, we often need to present not only components but also processes within the system. Even if the common practice to describe system parts is to use a component view, the representation of system processes becomes more and more important: nowadays the process view and the data flow representation are a typical part of the development of interactive or reactive systems. Specifying both components and processes within the same language, without changing the framework, we not only increase the readability of a system specification but also can easier ensure consistency among these different views on a system: this extension of the language functionality allows us to have more precise and at the same time more flexible representation of the system.

For these reasons we extend the formal language Focus by the theory of processes described in (Leuxner, 2010). A process is understood there as "an observable activity executed by one or several actors, which might be persons, components, technical systems, or combinations thereof". Each process has one *entry (activation, start) point* and one *exit (end) point*. An entry point is a special kind of input signal/channel that activates the process, while an exit point is a special kind of output signal/channel that is used to indicate that the process is finished. We treat a process as a special kind of a Focus component having additionally two channels (one input and one output channel) of a special kind. These channels represent the entry and exit points of the process.

Dealing with *security-critical systems* we have another question in the foreground: how we can combine system components that each enforce a particular security requirement in a way that allows us to predict which properties the combined system will have (Apostolopoulos et al., 1999). Formal verification of software systems and es-

Figure 2. Unification of the information representation on the level of languages



pecially reasoning about compositional properties is a challenge in particular important in the area of security-critical systems: combining system components which have a number of security/ secrecy properties, the most important and the most difficult question is to predict which of these properties the composed system will have. For this purpose we introduced in (Spichkova, 2012) a representation methodology for crypto-based software, such as cryptographic protocols, and their composition properties. Having such a formal representation, one can argue about the protocol properties as well as the composition properties of different cryptographic protocols in a methodological way and make a formal proof of them using a theorem prover.

Using these extensions, on the one hand, we do not need to switch between languages, the representation is unified to make the communication between different development team easier and the accurate specifications of different system's parts more understandable, on the other hand, if, for example, the representation of cryptographic properties is irrelevant for the system we specify and verify, the engineer do not need to study the aspects of the formal language related to the security-critical systems. Extending the formal language in the add-ons manner we increase the specifications' readability without the rapid increase of learning effort required by the formal language.

Automatization

Last but not least point in of HF^{2}M is an appropriate automatization of a number of steps within the specification and verification process, because the automatization not only saves time but also excludes (at least partially) the human element as the most "unreliable" in failure, according to the *Engineering Error Paradigm* (Redmill & Rajan, 1997). As the next step of or research, we are currently proving all the theoretical ideas of HF^{2}M practically, using the AutoFocus CASE tool.

AutoFocus is a scientific prototype[2] implementing on top of the Eclipse platform[3] a modelling language based on a graphical notation. This prototype uses a restricted version of the formal semantics of the Focus specification and modelling language (Schätz, 2004; Schätz & Huber, 1999; Huber et al., 1996). Specifying a system in AutoFocus, we obtain an executable mode, which can be validated by means of the AutoFocus simulator to get a first impression of the system under development and possibly find implementation errors that we introduced during the transformation of the requirements into an AutoFocus model.

The following extensions of the AutoFocus CASE tool are in progress (Spichkova et al., 2013): the add-ons that allow

- To generate formal Focus specification from the CASE tool representation.
- To edit in the user-friendly[4] way a (generated) Focus specification represented in LaTeX.
- To write a specification using the predefined templates.

The Focus generator produces a specification of the model by representing the formal specification in LaTeX according to the predefined templates restricting all specification styles to the Assumption/Guarantee variant to exclude the loss of the constrains on the system's environment. Using this generator we can, on the one hand, get a readable formal specification developed according the suggested optimisations, on the other hand, apply the HCI development methods within the common application area, development of the tools, focusing this time on the formal methods are „hidden" by the modelling tool.

Even a readable formal specification is hard to keep up to date if the system model is frequently changing during the modelling phase of the development. This causes the situation where the system documentation is often outdate and does not describe the latest version of the system: system requirements documents and the general systems description are not updated according to the system's or model's modifications, sometimes because this update is overseen, sometimes on purpose, because of the timing or costs constraints on the project. This problem could be also be solved by using this add-on: we simply generate new (updated) formal specification from the model. The current version of the editor inherits the most of the functions an open source plugin TeXlipse[5] (e.g., the syntax check of the specification as well as syntax highlighting, code folding, etc.), and is extended by additional features such as

- Focus operators as well as the main Focus frames: component and function specification.
- Several specification tables.
- Predefined data types and streams.
- Tool box for the predefined Focus operators, which allows a quick access to the most important features of the formal language.

This add-on is oriented on the features of the Focus language, but it does not require any special sophisticated knowledge, and this point leads us to the next step of our research: how can we represent the element of formal language in such a way that the language learning effort is minimized.

FUTURE RESEARCH DIRECTIONS

As mentioned in the previous section, one of the future research directions is to investigate the possibilities of formal language optimization in order not only to increase the readability of the specification, but also to minimize the learning effort needed to be fluent using the formal language.

Another interesting direction is the tool-support of the methodology "Focus on Isabelle" (Spichkova, 2007). This methodology allows verifying properties of the system using the semi-automatical theorem prover Isabelle/HOL. Using "Focus on Isabelle" we can influence the complexity of proofs and their reusability already during the specification phase, because the specification and verification/validation methodologies are treated here as a single joint methodology with the main focus on the specification part. Moreover, using it we can perform automatic correctness proofs of syntactic interfaces for specified system components. Having an automatic translation of formal specifications from Focus to Isabelle/HOL we can apply the methodology not only in theory but also in practice.

CONCLUSION

In our work "Human Factors of Formal Methods" we aim to apply the engineering psychology achievements to the design of formal methods, focusing on the specification phase of a system development process. The main ideas discussed in this chapter are language independent, but for better readability and for better understanding of these ideas we show them on the base of formal specifications presented in the Focus specification framework.

According to the Engineering Error Paradigm we optimize representation of formal specification, which corresponds to the classical HCI design, as well as add a corresponding automatization of a number specification and verification steps of system design. This approach demonstrates that even small changes within a formal method can make it much more understandable, usable, and also safe. Moreover, in many cases it is sufficient to reuse within the formal methods some successful ideas from other areas where the similar representation or design problems were already solved.

REFERENCES

Apostolopoulos, G., Peris, V., & Saha, D. (1999). Transport layer security: How much does it really cost? In *Proceedings of the Conf. on Computer Communications (IEEE Infocom)*. IEEE Computer Society.

Broy, M., & Stølen, K. (2001). *Specification and development of interactive systems: Focus on streams, interfaces, and refinement*. Berlin: Springer. doi:10.1007/978-1-4613-0091-5.

Constantine, L. (2003). Canonical abstract prototypes for abstract visual and interaction design. In *Proceedings of Interactive Systems: Design, Specification, and Verification* (Vol. 2844). Berlin: Springer. doi:10.1007/978-3-540-39929-2_1.

Constantine, L. L., & Lockwood, L. A. D. (1999). *Software for use: A practical guide to the models and methods of usage-centered design*. Reading, MA: Addison-Wesley.

Constantine, L. L., & Lockwood, L. A. D. (2002). Usage-centered engineering for web applications. In *Proceedings of IEEE Software*. IEEE.

Dhillon, B. (2004). *Engineering usability: Fundamentals, applications, human factors, and human error*. American Scientific Publishers.

Feilkas, M., Hölzl, F., Pfaller, C., Rittmann, S., Schätz, B., Schwitzer, W., et al. (2011). *A refined top-down methodology for the development of automotive software systems – The keylessentry system case study* (Technical Report TUM-I1103). Munich, Germany: TU München.

Feilkas, M., Fleischmann, A., Hölzl, F., Pfaller, C., Scheidemann, K., Spichkova, M., & Trachtenherz, D. (2009). *A top-down methodology for the development of automotive software* (Technical Report TUM-I0902). Munich, Germany: TU München.

Følstad, A., Law, E., & Hornbæk, K. (2012). Analysis in practical usability evaluation: A survey study. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM.

Fuchs, E. N., & Schwitter, R. (1995). Specifying logic programs in controlled natural language. In *Proceedings CLNLP 95, ELSNET/COMPULOG-NET/EAGLES Workshop on Computational Logic for Natural Language Processing*. Edinburgh, UK: University of Edinburgh.

Fuchs, E. N., & Schwitter, R. (2007). Web-annotations for humans and machines. In *Proceedings of the 4th European Semantic Web Conference (ESWC 2007)*, (LNCS). Berlin: Springer.

Heumann, J. (2002). Use cases, usability requirements, and user interfaces. In *Proceedings of Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2002)*. ACM.

Huber, F., Schätz, B., Schmidt, A., & Spies, K. (1996). AutoFocus - A tool for distributed systems specification. In *Proceedings of of FTRTFT'96, (LNCS)* (Vol. 1135). Berlin: Springer.

Hölzl, F., & Spichkova, M. & Trachtenherz. D. (2010). *Safety-critical system development methodology* (Technical Report TUM-I1020). Munich, Germany: TU München.

Klare, G. R. (2000). Readable computer documentation. *ACM Journal of Computer Documentation*, *24*(3), 148–168. doi:10.1145/344599.344645.

Kuhn, T. (2010). *Controlled English for knowledge representation*. (PhD Thesis). University of Zurich, Zurich, Switzerland.

Leuxner, C., Sitou, W., & Spanfelner, B. (2010). A formal model for work flows. In *Proceedings of the 8th IEEE International Conference on Software Engineering and Formal Method* (pp. 135–144). IEEE.

Leveson, N. G., & Turner, C. S. (1993). An investigation of the therac-25 accidents. *IEEE Computer*, *26*(7), 18–41. doi:10.1109/MC.1993.274940.

Macias, B., & Pulman, S. (1993). Natural language processing for requirements specifications. In F. Redmill, & T. Anderson (Eds.), *Safety-Critical Systems, Current Issues, Techniques and Standards*. London: Chapman & Hall.

Miller, E. (1987). The therac-25 experience. In *Proceedings of the Conf. State Radiation Control Program Directors.* IEEE.

Redmill, F., & Rajan, J. (1997). *Human factors in safety-critical systems*. London: Butterworth-Heinemann.

Schätz, B. (2004). Mastering the complexity of reactive systems: The AUTOFOCUS approach. In *Formal Methods for Embedded Distributed Systems: How to Master the Complexity* (pp. 215–258). Boston: Kluwer Academic Publishers. doi:10.1007/1-4020-7997-4_7.

Shackel, B., & Richardson, S. J. (1991). *Human factors for informatics usability*. Cambridge, UK: Cambridge University Press.

Spichkova, M. (2007). *Specification and seamless verification of embedded real-time systems: Focus on isabelle.* (PhD Thesis). Munich, Germany: TU München.

Spichkova, M. (2011). Architecture: Requirements + decomposition + refinement. *Softwaretechnik-Trends, 32*(4).

Spichkova, M. (2011). *Focus on processes* (Technical Report TUM-I1115). Munich, Germany: TU München.

Spichkova, M. (2012). *Component composition: Formal specification and verification of cryptographic properties* (Technical Report TUM-I124). Munich, Germany: TU München.

Spichkova, M., Hölzl, F., & Trachtenherz, D. (2012). Verified system development with the autofocus tool chain. In *Proceedings of the 2nd Workshop on Formal Methods in the Development of Software (WS-FMDS).* WS-FMDS.

Spichkova, M., Zhu, X., & Mou, D. (2013). Do we really need to write documentation for a system? CASE tool add-ons: Generator+editor for a precise documentation. In *Proceedings of the International Conference on Model-Driven Engineering and Software Development (MODELSWARD'13).* MODELSWARD.

Volpert, W. (1991). Work design for human development. In C. Floyd et al. (Eds.), *Software Development and Reality Construction*. Berlin: Springer-Verlag.

Wickens, C. D., & Hollands, J. G. (2000). *Engineering psychology and human performance*. Englewood Cliffs, NJ: Prentice Hall.

ADDITIONAL READING

Abrial, J.-R. (1996). *The b-book: Assigning programs to meanings*. Cambridge, UK: Cambridge University Press. doi:10.1017/CBO9780511624162.

Broy, M. (1991). Towards a formal foundation of the specification and description language SDL. *Formal Aspects of Computing*, *3*(1), 21–57. doi:10.1007/BF01211434.

Broy, M., Fox, J., Hölzl, F., Koss, D., Kuhrmann, M., Meisinger, M., & Wild, D. (2007). Service-oriented modeling CoCoME with focus/autofocus. In *The Common Component Modeling Example: Comparing Software Component Models (LNCS)* (Vol. 5153, pp. 177–206). Berlin: Springer. doi:10.1007/978-3-540-85289-6_8.

Bauer, V., Broy, M., Irlbeck, M., Leuxner, C., Spichkova, M., Dahlweid, M., & Santen, T. (2013). *Survey of modeling and engineering aspects of self-adapting & self-optimizing systems* (Technical Report TUM-I130307). Munich, Germany: TU München.

Campetelli, A., Hölzl, F., & Neubeck, P. (2011). User-friendly model checking integration in model-based development. In *Proceedings of International Conference on Computer Applications in Industry and Engineering*. IEEE.

Dekker, S. (2006). *The field guide to understanding human error* (2nd ed.). Lund, Sweden: Lund University School of Aviation.

Dekker, S. (2011). *Drift into failure*. London: Ashgate.

DeMarco, T. (1979). *Structured analysis and system specification*. Englewood Cliffs, NJ: Prentice Hall.

Dhillon, B. S. (2009). *Human reliability, error, and human factors in engineering maintenance: with reference to aviation and power generation*. Boca Raton, FL: CRC Press. doi:10.1201/9781439803844.

Dhillon, B. S. (2012). *Safety and human error in engineering systems*. Boca Raton, FL: CRC Press. doi:10.1201/b12534.

Fuchs, N. E., & Schwertel, U. (2003). Reasoning in attempt to controlled English. In F. Bry, N. Henze, & J. Maluszynski (Eds.), *Principles and Practice of Semantic Web Reasoning, International Workshop PPSWR* (LNCS), (Vol. 2901). Berlin: Springer.

Kaljurand, K. (2009). Paraphrasing controlled english texts. In *Proceedings of the CEUR Workshop.* CEUR.

Klein, C., Prehofer, C., & Rumpe, B. (1997). Feature specification and refinement with state transition diagrams. In P. Dini (Ed.), *Fourth IEEE Workshop on Feature Interactions in Telecommunications Networks and Distributed Systems*. Boca Raton, FL: IOS-Press.

Kuhn, T. Royer, L., Fuchs, N. E., & Schroeder, M. (2006). Improving text mining with controlled natural language: A case study for protein interactions. In U. Leser, B. Eckman, & F. Naumann (Eds.), *Proceedings of the 3rd International Workshop on Data Integration in the Life Sciences 2006 (DILS'06)*, (LNBI). Berlin: Springer.

Kuhn, T. (2010). Codeco: A grammar notation for controlled natural language in predictive editors. In *Proceedings CEUR Workshop*. CEUR.

Kuhn, T. (2013). A principled approach to grammars for controlled natural languages and predictive editors. *Journal of Logic Language and Information*, *22*(1). doi:10.1007/s10849-012-9167-z.

Morgan, C. (1994). *Programming from specifications*. Englewood Cliffs, NJ: Prentice Hall.

Mosses, P. D. (1997). CoFI: The common framework initiative for algebraic specification and development. In M. Bidoit & M. Dauchet (Eds.), *Theory and Practice of Software Development (TAPSOFT '97), 7th International Jiont Conference CAAP/FASE* (LNCS), (Vol. 1214). Berlin: Springer.

Potter, B., Sinclair, J., & Till, D. (1996). *An introduction to formal specification and Z*. Englewood Cliffs, NJ: Prentice Hall.

Ratiu, D. (2009). *Intentional meaning of programs*. (PhD Thesis). Munich, Germany: TU München.

Schätz, B., & Giese, H. (2007). Models of reactive systems: communication, concurrency, and causality. In H. Giese, G. Karsai, E. Lee, B. Rumpe, & B. Schätz (Eds.), *Proceedings of the 2007 International Dagstuhl conference on Model-based engineering of embedded real-time systems (MBEERTS'07)*. Berlin: Springer.

Schätz, B. (2011). 10 years model-driven - What did we achieve? In *Proceedings of the 2011 Second Eastern European Regional Conference on the Engineering of Computer Based Systems* (ECBS-EERC '11). IEEE Computer Society.

Schwitter, R. (1998). *Kontrolliertes Englisch für anforderungsspezifikationen*. (PhD thesis). University of Zurich, Zurich, Switzerland.

Spichkova, M. (2012). Towards focus on time. In *Proceedings of the 12th International Workshop on Automated Verification of Critical Systems (AVoCS'12).* AVoCS.

Spichkova, M. (Ed.). (2012). *Seminar: Human factors in software engineering* (Technical Report TUM-I1216). Munich, Germany: TU München.

Spivey, M. (1988). *Understanding Z: A specification language and its formal semantics*. Cambridge, UK: Cambridge University Press.

KEY TERMS AND DEFINITIONS

Controlled Natural Language (CNL): A subsets of a natural language, obtained by restricting its grammar and vocabulary in order to eliminate (or, at least, to reduce) ambiguity and complexity of the specification written in this language.

Engineering Error Paradigm: A particular kind of Human Error Paradigms, which focuses on the technical aspect of the system and interact between the human factor and the system. This paradigm sees the human factor as one part of the system.

Focus on Isabelle: A specification and verification framework, which is the result of the coupling of the formal specification framework Focus in the generic theorem prover Isabelle/HOL.

Formal Method: A particular kind of techniques (based on logic and mathematics) for the specification, development and verification of software and hardware systems.

Human Computer Interface (HCI): An interface between a user and a (software and/or hardware) system.

Readability: The ease in which text can be read and understood without ambiguity and misinterpretation.

Safety-Critical System: A system which failure could result in loss of human life or damage to the environment or valuable objects.

Specification: A system's description representing the set of requirements to be satisfied by the system.

ENDNOTES

1 See http://focus.in.tum.de.
2 http://af3.fortiss.org
3 http://www.eclipse.org
4 A "user" means here a "software engineer".
5 http://texlipse.sourceforge.net