Towards Readability Aspects of Probabilistic Mode Automata

Heinz Schmidt and Maria Spichkova
School of Science, RMIT University, Melbourne, Australia

Keywords: Software Engineering, Formal Methods, Petri Nets.

Abstract: This paper presents a new approach and design model targeting hybrid designer- and operator-defined performance budgets for timing and energy consumption. The approach is based on Petri Nets formalism. As the cognitive load is typically high while using formal methods, this increases the chances of mistakes. Our approach is focused on the readability aspects and aims to decrease the cognitive load of developers. We illustrate the proposed approach on example of a sample embedded multi-media system, a modern digital camera.

1 INTRODUCTION

In the domain of embedded systems, the trend to enhance more and more system functionalities through software solution is constantly increasing. This makes the design of these systems and the corresponding quality assurance more and more challenging Sangiovanni-Vincentelli and Martin (2001). Real-time and dependability constrains provide additional challenges, which also lead to necessity of probabilistic analysis within the phases of design and verification of these systems. Also, some constraints within embedded systems are mutually dependent, for example, timing and energy consumption constraints cannot be analysed independently of each other, see Mudge (2001); Saxe (2010); Wolf et al. (2008).

One of the successfully applied paradigms is Component-based software development, which was initially introduced many decades ago (CBSD, see Adler (1995); Clements (1995)). However, CBSD cannot solve directly issues related to the constraints on safety, timing, energy consumption, etc. Henzinger and Sifakis (2006); Spichkova et al. (2012), but can provide a solid basis for extended approaches.

In recent work Peake and Schmidt (2011) we have extended our rich architecture definition language (RADL, see Schmidt (2003)) and underlying theory Schmidt et al. (2003) to meet such industrial requirements, aiming at a scalable and compositional (component-based) approach to soft dependability guarantees: with probability, guarantee risk, execution time, cost etc. Industrial practice requires the capability to compose a variety of heterogeneous models and components, specified and designed using different methods and frameworks. Many real-world engineering environments are not locked into a single model, single framework or single-language environment. While we abstract from the programming languages underlying such an heterogeneous software engineering approach, we hope to show that, and how, our design-oriented model-based approach links with concrete programming by means of elementary modelling blocks providing abstractions directly for code blocks. This is natural and perhaps more appropriate in design of embedded systems than in other fields, as component models in this context often use architectural elements to abstract from software and hardware blocks at the same time. However we expect that this approach carries across to other domains.

In our current work, we targeting hybrid designer- and operator-defined performance budgets for timing and energy consumption. We propose an approach that is on Petri Nets formalism. Our approach is focused on the readability aspects and aims to decrease the cognitive load of developers, as having high cognitive load increases the chances of mistakes in system design and quality assurance process. We also aim to keep the method lightweight, following the classification presented in Zamansky et al. (2018).

To illustrate the proposed approach, we use an example of a sample embedded multi-media system, a modern digital camera. This allows us to demonstrate how the time (and the ensuing synchronisation) and energy constraints can be analysed taking into account their mutual dependencies. We propose that extra-functional properties have to be considered from early performance requirement specification through to model-based testing and run-time

Schmidt, H. and Spichkova, M.
Towards Readability Aspects of Probabilistic Mode Automata.
DOI: 10.5220/0007781005550562
In *Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2019)*, pages 555-562
ISBN: 978-989-758-375-9

verification. Beside the compositional approach to reasoning about and testing such properties in a hybrid modelling environment, our contribution is in the separation of concern of different aspects of modelling and in context-dependent methods of reasoning about such properties. Notably we have developed methods which allow automated contextual resource allocation strategies, under dynamically varying, and suitably parameterised, architectural configurations.

2 EXAMPLE: DIGITAL CAMERA

Consider the design of a modern digital camera from the perspective of different types of use:

Scenario 1: A busy professional sports photographer requires the ability to capture many hundreds or thousands of high quality images rapidly, with minimal shutter lag, in rapid bursts of up to 100 photos. Within the given price point afforded by budget, she is prepared to sacrifice "convenience" features, accepting shorter battery life and fewer shots per memory card while carrying extra battery packs, memory cards or even a laptop for frequent uploads, as well as extra lenses, and manage reconfiguration as needed.

Scenario 2: One weekend a family member is getting married, and as the *de facto* camera expert she has agreed to act as a semi-official or backup photographer for the wedding. In this capacity she aims for simplicity and convenience, so she can still enjoy the day and mingle without being conspicuous or weighed down by equipment. The couple insist they prefer photos in a standard compressed consumer format (JPEG), which at least eliminates extra effort later at her workstation, and maximises memory card capacity. She selects what she can carry easily—a single camera body and lens and perhaps a single additional memory card, but no extra battery pack. She is unwilling to spend anything like her usual time and effort on camera configuration, instead often (perhaps not always) relying on camera to automatically select exposure, focus and aperture. Occasionally, for particularly important shots she takes full control again. In this second case, battery life is paramount.

The specific challenge is to design a camera which is capable of flexible reconfiguration to suit multiple contexts, including for example these. The generic challenge is to:

(i) *Characterise context* in terms of user configuration choices, usage (e.g. selected modes/operations/functions) and user-visible desired properties.

(ii) *Reason in a context-sensitive way about system properties* and manage internal configuration to ensure consistency between configuration/profiles and desired properties. For example, to make the camera battery last longer, the camera must somehow sacrifice quality and/or performance in an acceptable way.

However the true usage context is often hard to predict. What exactly are the user's requirements and intentions? Even the user may not know exactly what she intends beyond the immediate moment. Contextual uncertainty extends to environmental conditions, which may have a non-trivial impact on performance. For example ambient temperature may affect performance (e.g., energy consumption) of key camera components significantly, including batteries Rao et al. (2003), sensors and actuators such as lenses. This has implications for the design not only of embedded systems, but also at a macroscopic level. Thus, large-scale computing centres have significant inter-dependency on their local environment; such facilities are already planned with environmental conditions such as temperature in mind to be able to maximise performance and performance per cost while minimising cooling and energy consumption.

We extend the camera design presented by Lee Lee (2006). In our example, the camera has the following logical components:

- a general purpose processor (GPP),
- a digital signal processor (DSP),
- actuators to control, e.g., mirror and shutter curtain, lens focus and aperture,
- sensors, e.g., for auto-focus,
- a buffer to store images temporarily, and
- a flash memory as a long-term storage media.

To keep the example small enough for a conference paper, we abstract from other typical functions such as USB driver for photo download, LCD user interface, camera flashbulb, and various advanced settings.

In high performance scenarios a dedicated GPP-flash memory link is possible. We focus on the interplay of functionality relevant for taking a range of different shots involving real-time physical control, as well as selecting tradeoffs between timing and energy consumption.

As presented in Figure 1, the system has three modes, each with different resource requirements:

- *IDLE* mode covers waiting for shutter half/full press and pre-focusing.
- In *single frame (SF)* mode, the camera returns to the idle mode after shooting is completed, while
- in *multi frame (MF)* mode, shooting is continued as long as the shutter release button is kept pressed.

MF contains two sub-modes, *high-speed (HS)* and *low-speed (LS)*. *MF* starts with *HS* and switches to *LS* if/when the image buffer gets full, where shooting of the consequent frame is delayed until enough space is freed in the buffer by writing to the flash memory. With these mode abstractions in mind, from a design perspective it is expected that refinements to components used in these modes may enable new features (for example smart/continuous save in HS at a performance penalty).

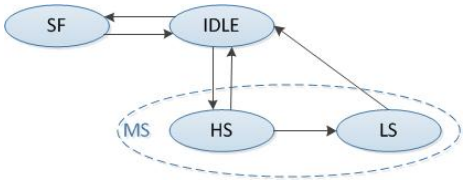

Figure 1: Digital Camera: Modes.

Furthermore, in each mode the user can select lens focusing and exposure metering to be performed automatically or manually, i.e. each mode has four submodes. More precisely, in the case of multi frame shooting, each of the *MF* submodes, *HS* and *LS*, has four further submodes:

- *FE*: automatic operations are fully enabled: both autofocus *AF* and automatic exposure *AE* are enabled;
- *F:* only the autofocus *AF* operation is enabled;
- *E:* only the automatic exposure *AE* operation is enabled;
- 0: neither autofocus *AF* nor automatic exposure *AE* are enabled.

In the *IDLE* mode the user may perform *AF*, *AE* or both, while composing a picture. During this time DSP cannot be activated and *AF* and *AE* operations are performed on GPP to reduce energy consumption. When the user presses the shutter release button, first, *AF* and *AE* operations that are being executed are completed, then the idle mode is terminated and the system switches to *SF* or *MF* depending on the user selection.

Another way to represent system modes (which can be related to the same submodes hierarchy as introduced in Figure 2) is to work parallel with on mode variables, because the choice to activate *AF* and *AE* operations is highly independent of whether the camera is in the *IDLE*, *SF*or one of the multi frame modes. Let call them *CameraMode* and *AutoMode* defined over enumeration types

$$\{IDLE,\ SF,\ HS,\ LS\}$$

and

$$\{FE,\ F,\ E,\ 0\}$$

respectively. We can also see this as a feature composition/interaction, see e.g., Calder and Magill (2000); Apel et al. (2010); Broy (2010). Thus, one feature is responsible for the choice of the current value of *CameraMode* and for the processes in the corresponding mode, where the second feature solely deals with the *AF* and *AE* operations.

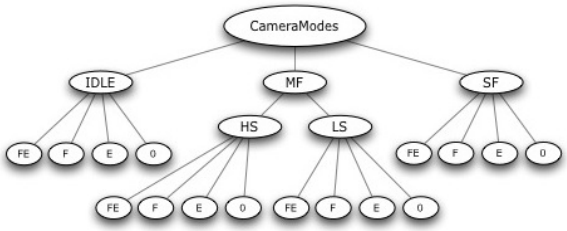

Figure 2: Digital Camera: Submodes Hierarchy.

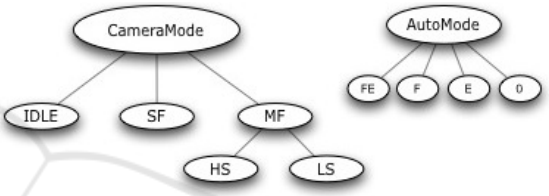

Figure 3: Digital Camera: Parallel Model for the Submodes Hierarchy.

Table 1 lists some of the relevant software components, their descriptions and their implementation platform (GPP/DSP). Some components are implemented in both processors to allow dynamic reconfiguration of the system in order to provide optimal resource usage. Within these constraints, a key challenge is allocating computing resources for the software elements to best suit partly predictable usage conditions. The DSP is especially suited to image processing operations, yet the DSP has significant energy overheads. We characterise the main design problems for the camera as follows. (i) Given an overall objective (e.g. minimise time consumption), satisfy that objective at run time. (ii) Given a usage profile, minimise energy and time consumption at run time.

3 PROPOSED VISUALISATION APPROACH

One of the problems using formal representation is that often only two factors are considered as important: 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, etc.,

Table 1: Software Components.

	Description	GPP	DSP
Operatoions			
AF	AutoFocus: Automatic lens focusing	✓	✓
AE	Automatic Exposure metering	✓	✓
IP	Image Processing on local buffer, red-eye reduction, etc.	-	✓
IB	Image Buffering: Transfer image from sensor to local buffer	-	✓
IS	Image Storage: Transfer images from buffer to flash card	✓	-
AS	Activate Shutter etc. (e.g. aperture adjust)	✓	✓
BC	Buffer check: Check if buffer is full	-	✓
Modes			
IDLE	Idle mode	-	✓
SF	Single-Frame shutter	✓	✓
MF	Multi-Frame shutter	✓	✓
Submodes			
FE	AF and AE enabled	✓	✓
F	AF only enabled	✓	✓
E	AE only enabled	✓	✓
0	AF&AE disabled	✓	✓

but even small syntactical changes of a method can make it more understandable and usable for engineers Constantine (2003); Dhillon (2004); Klare (2000); Spichkova (2013). Figure 4 presents an the example of Petri net specifying HS mode details for the digital camera, which provides a typical representation of a coloured Petri net. Within our approach, we propose the following enhancements: To make representation more readable, first of all we should take into account the human factor. Thus, if a path (in this case a colour marked path, green or red) starts on the left/right of the net, we should proceed to draw it on the same side if possible and avoid cross moving the paths without any important reason.

Thus, on Figure 4 two paths are switched after the operation *do AS*, which can confuse some readers. Then, we can try to find a solution to avoid a lot of crossing arrows having different meanings: the blue and maroon arrows indicate synchronisation of the counters, and we can replace them by visual grouping of operations on the same counter. As result we obtain an optimised coloured Petri net presented in Figure 5, which is semantically equivalent to the representation in Figure 4. This optimisation increases ease of use by human readers (designers, testers etc.) without decreasing simplicity for machine readability and semi-automated support or expressiveness/power (for the domain or domains of choice).

Usability derives from the following aspects:

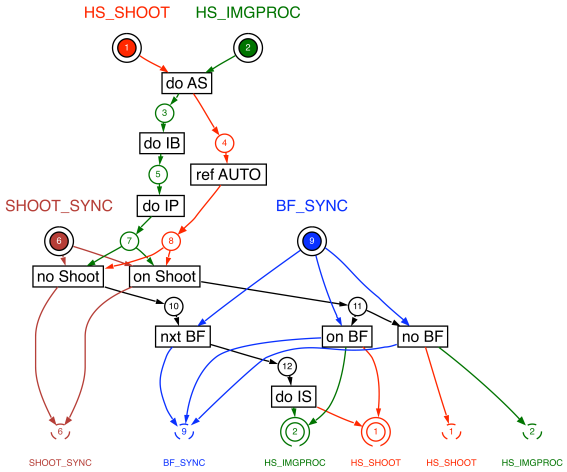

Figure 4: *HS* mode details presented as a coloured Petri net.

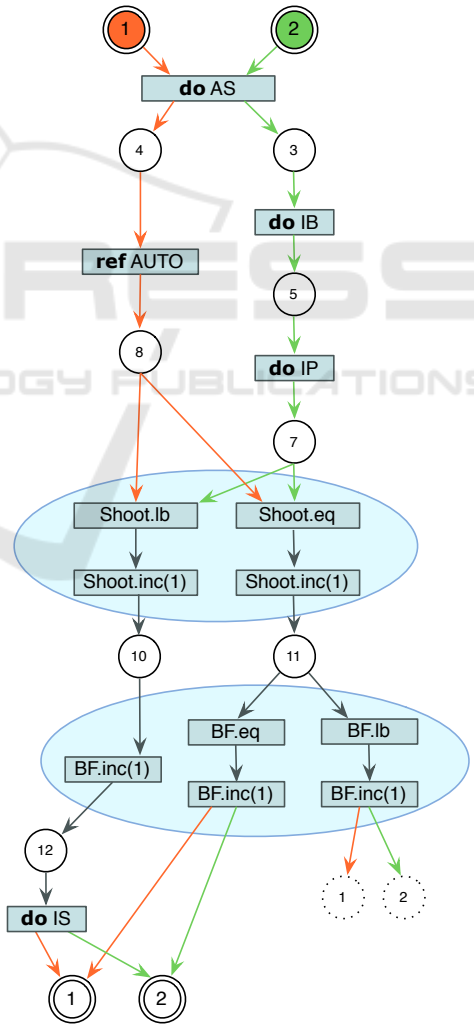

Figure 5: HS mode details: Another Visualisation.

- Lowering the barrier between the simplified and expressive language for the machine support and that of the domain languages of the user(s) and associated with the purpose, e.g., by using controlled natural languages that try to avoid disadvantages of both natural and formal languages and being a subset of a natural language with a well-defined syntax and semantics, see Kuhn (2010); Fuchs and Schwitter (1995); Macias and Pulman (1993).
- Applying an appropriate automatisation of a number of steps within the modelling and verification process: this not only saves human time and allows to get results much faster then humans can produce manually, but also (partially) excludes the human element as the most "unreliable" in failure, see Redmill and Rajan (1996); Spichkova and Zamansky (2016). For example, a formal specification can be generated from the corresponding CASE tool representation which can be edited in a more readable way also using predefined templates, see Spichkova et al. (2013); Vo and Spichkova (2016).
- Supporting directly common and standard abstractions that are well-established (and hence part of the software engineering training), e.g. *Message Sequence Charts* Genest and Muscholl (2008); Harel and Thiagarajan (2003)), or defined in standards (such as UML, IEC-61131, etc):
- Unification of the representation of any information we are dealing with (see, e.g., Spichkova (2013));
- Easing the use of novel compositional principles and high-level tools, that are opening novel and powerful methods to users of formal specification or specification-based/model-driven methods.

Having a representation like presented on Figure 5, we can easily transform a Petri net to a *hierarchical MSC*. In the case a component-based specification of the system is need in addition to the process representation, an MSC can be schematically translated to the corresponding formal specification as shown in Spichkova et al. (2012); Spichkova (2010). Let us also shortly discuss translation/representation of the following modelling artefacts: (global) parameters, local, time and counter variables.

Local Variable: use can be translated into state and transition label expansions for NF purposes Katoen (2005), but can also be intuitively understood in data types and data structures that capture state. However for Petri net normal form used in our approach and compositionality considerations restrictions need to be designed along with such capabilities to limit the scopes of these variables appropriately, viz. to FSM components of nets, in terms of their use in guards and assignments associated with transitions and states.

Global Parameter: use are of a similar nature with respect to normalisation but needs to be limited to achieve compositionally. For example, we could say the global parameters may occur locally in guards (i.e., they are read-only) as well as in initialisation expressions (for the initial states when FSM objects are created) or with re-assignments limited to higher-level FSMs (such as mode automata) when submodes are entered and before these branch out into rational parallel processes. Another common example is the use of iterator and bounded loop process constructs that have a very structured use of local *counter* variables which never serve synchronisation but are providing a reasoning tool for local termination and performance approximation, based on an interplay of local (loop) invariants and loop control variables, which implies strict monotonicity and boundedness.

Time Variables (Clocks): are a further example and in some sense a special case of counter control above – in the sense that all practical approaches to timed automata and synchronous time models discretise an infinite number of real-time points into a real-time intervals with integer bounds and then solve a linear convex hull problem to determine feasibility and/or optimal schedules that meet time constraints. There is also a significant difference here, that needs to be considered, relative to counters. In general, counter processes can be explained as a macro structure based on sequence and choice, and hence are lower-level automata (or process expressions) themselves, and hence they do not add 'new' semantics but can be explained in terms of existing semantics. For example if we are in rational parallel processes, they are just a syntactic sugar extension that does not take us out of this class. Likewise with other classes of processes (such as pushdown automata). In contrast, timed extension are true semantic extensions, in that they define a different class of behaviours and automata, because the define *what* the legitimate processes (occurrence nets) are that are traces of the give language (net system or process expression).

4 RELATED WORK

Component-based software engineering utilises a well-defined composition theory to enable the prediction of such properties. as performance and reliability. This is one of the largest fields of software and system engineering, there are many approaches on component-based design (CBD) cover-

ing different aspects and focusing on requirements, quality, timing properties etc. (see e.g., CBS (2003); Broy (2010, 1999); Broy et al. (2008)). Several component-based prediction approaches, e.g. Palladio Kapova et al. (2010); Martens et al. (2008); Becker et al. (2007), CB-SPE Bertolino and Mirandola (2004), ROBOCOP Bondarev et al. (2004) (see also a survey in Becker et al. (2006)) derive the benefits of reusing well-documented component specifications. In our approach we focus on the questions of resource-awareness and adaptivity of systems as well as on the readability aspects of the formalism.

Mode automata have a long history motivated by real-time design practices and methods used in industry in connection with statecharts. Maraninchi et al. Maraninchi and Rémond (2003) capture the notion of modes formally for a practical extension of the real-time synchronous language Lustre and include elements of the well-known I/O-automata. Mode automata define synchronous mode automata as a hybrid between data-flow and transition systems. Talpin et al. Talpin et al. (2006) extend this work to so-called polychronous mode automata to work with the multi-clock data-flow formalism SIGNAL. Both these types of automata are non-deterministic and do not deal with probabilities. The (bisimulation) equivalence and therefore compositional reasoning for mode automata is undecidable. However, Maranichi et al. introduce a synchronised (lock-step) parallel product for modes in which shared symbols (intersection of alphabets nonempty) are synchronised while local symbols (the symmetric difference of the alphabets) are independent. While the modes of a single automaton are mutually exclusive in their approach, and the behaviour of these mode automata is fully abstract wrt. probabilistic testing, the automata product suffers from combinatorial explosion (state space explosion), due to the aim of allowing arbitrary shared variables and interference of parallel processes.

Cheung et al. Cheung et al. (2012) describe an architecture-level method, SHARP, for predicting reliability (and timing) of concurrent systems. Whereas SHARP is specifically designed for reliability and timing prediction, our method is intended to be generic thus also catering, e.g., for energy consumption. SHARP models involve *scenarios* which are either *basic* (similar to message sequence charts) or hierarchical, involving sequential, conditional or concurrent composition. SHARP supports concurrent composition of finite numbers of instances of a particular scenario, corresponding to symmetrically replicated components. SHARP derives completion time and reliability predictions from scenarios for use at higher levels of abstraction. For each basic scenario, SHARP requires transition rates for all individual actions, then calculates a single continuous-time Markov model model from which completion time and reliability are derived. For an hierarchical scenario a system level CTMC is constructed using abstraction techniques such as queuing networks and abstraction of sequential components into single global states. In contrast, our approach requires probabilities/rates at the system level only. Our approach seeks to avoid or defer calculation of monolithic models.

Our cost estimation is inspired by Valiant's bulk synchronous-parallel model Valiant (1990) of parallel computing where global *strong synchronisation* conservatively approximates systems which may in reality use more fine grained synchronisation and indeed may allow for more asynchrony than the above approximation would suggest. In performance benchmarks reported in Yusuf et al. (2011), Yusuf et al. demonstrated that such conservative predictions may still be accurate enough if there is enough WCET variation and a large enough number of activities/tasks scheduled on individual processing elements. Thus adjacent modes may be assumed to be strongly separated in the global model while in fact such modes are partially interleaved with respect each other (subject to restrictions on repetition such as boundedness for message sequence graphs as described by Alur Alur and Yannakakis (1999) and star-connectivity in trace languages). For conservative cost estimation purposes this seems reasonable. We expect that (with diminishing returns) such models can be refined selectively, to bound costs of adjacent sequences of overlapping modes, in a context-dependent way.

An interesting approach on integration of synchronous and asynchronous communication was presented by Hennicker et al. Hennicker et al. (2010); Hennicker and Knapp (2011). In this approach, I/O-transition systems were used as the formal background for modelling of system behaviour. As result, a refinement relation was defined, which is compositional w.r.t. synchronous and asynchronous connections of components and which preserves connection-safety, and next existing interface theories for modal I/O-transition systems were extended to support assemblies, (greybox) assembly refinement and assembly encapsulation, also showing that communication-safety is preserved by assembly refinement, that black-box refinement of component interfaces is compositional w.r.t. grey-box refinement of assemblies and, conversely, that assembly encapsulation maps grey-box to black-box refinement.

5 CONCLUSIONS

In this paper, we proposed a Petri-Nets-based approach targeting hybrid designer- and operator-defined performance budgets for timing and energy consumption. The core focus of this approach is on decreasing the cognitive load of the designers to decrease the chances of design mistakes. To achieve better readability, we extended the coloured Petri Nets formalism. To illustrate the proposed solution, we presented an example of a sample embedded multimedia system, a modern digital camera.

Future Work: We are going to integrate the presented approach with the results of our prior work, a probabilistic global behaviour analysis approach developed for reliability and fault-tolerance studies (including fault injection) and a parallelism/concurrency focused framework centred on partially ordered traces, Petri nets and timing/energy costs.

REFERENCES

(2003). In Cechich, A., Piattini, M., and Vallecillo, A., editors, *Component-Based Software Quality: Methods and Techniques*, volume 2693 of *LNCS*. Springer.

Adler, R. (1995). Emerging standards for component software. *IEEE Computer*, 28(3):68 –77.

Alur, R. and Yannakakis, M. (1999). Model checking of message sequence charts. In Baeten, J. C. M. and Mauw, S., editors, *CONCUR'99*, volume 1664 of *LNCS*, pages 114–129. Springer.

Apel, S., Lengauer, C., Möller, B., and Kästner, C. (2010). An algebraic foundation for automatic feature-based program synthesis. *Science of Computer Programming*, 75(11):1022 – 1047. Special Section on the Programming Languages Track at the 23rd ACM Symposium on Applied Computing.

Becker, S., Grunske, L., Mirandola, R., and Overhage, S. (2006). Performance prediction of component-based systems: A survey from an engineering perspective. In *Architecting Systems with Trustworthy Components*, volume 3938 of *LNCS*, pages 169–192. Springer.

Becker, S., Koziolek, H., and Reussner, R. (2007). Model-based performance prediction with the palladio component model. In *6th international workshop on Software and performance*, pages 54–65. ACM.

Bertolino, A. and Mirandola, R. (2004). Cb-spe tool: Putting component-based performance engineering into practice. In Crnkovic, I., Stafford, J., Schmidt, H., and Wallnau, K., editors, *Component-Based Software Engineering*, volume 3054 of *LNCS*, pages 233–248. Springer.

Bondarev, E., de With, P., and Chaudron, M. (2004). Predicting real-time properties of component-based applications. In *In Proc. of the 30the EUROMICRO conference*, pages 40–47.

Broy, M. (1999). A logical basis for component-based systems engineering. In *Calculational System Design. IOS*. Press.

Broy, M. (2010). Multifunctional software systems: Structured modeling and specification of functional requirements. *Sci. Comput. Program.*, 75(12):1193–1214.

Broy, M., Fox, J., Hölzl, F., Koss, D., Kuhrmann, M., Meisinger, M., Penzenstadler, B., Rittmann, S., Schätz, B., Spichkova, M., and Wild, D. (2008). Service-Oriented Modeling of CoCoME with Focus and AutoFocus. In *The Common Component Modeling Example: Comparing Software Component Models*, pages 177–206. Springer.

Calder, M. and Magill, E., editors (2000). *Feature Interactions in Telecommunications and Software Systems*. IOS Press.

Cheung, L., andd Leana Golubchik, I. K., and Medvidovic, N. (2012). Architecture-level reliability prediction of concurrent systems. In *ICPE'12*. ACM.

Clements, P. C. (1995). From subroutines to subsystems: Component-based software development. *The American Programmer*, 8(11).

Constantine, L. L. (2003). Canonical abstract prototypes for abstract visual and interaction design. In Jorge, J. A., Jardim Nunes, N., and Falcão e Cunha, J. a., editors, *Interactive Systems. Design, Specification, and Verification*, volume 2844 of *LNCS*, pages 1–15. Springer.

Dhillon, B. S., editor (2004). *Engineering Usability: Fundamentals, Applications, Human Factors, and Human Error*. American Scientific Publishers.

Fuchs, N. E. and Schwitter, R. (1995). Specifying logic programs in controlled natural language. In *Proceedings of the Workshop on Computational Logic for Natural Language Processing*, pages 3–5.

Genest, B. and Muscholl, A. (2008). Pattern matching and membership for hierarchical message sequence charts. *Theory of Computing Systems*, 42(4):536–567.

Harel, D. and Thiagarajan, P. S. (2003). Message Sequence Charts. In Lavagno, L., Martin, G., and Selic, B., editors, *UML for Real: Design of Embedded Real-Time Systems*, pages 77–105. Kluwer Academic Publishers.

Hennicker, R., Janisch, S., and Knapp, A. (2010). Refinement of components in connection-safe assemblies with synchronous and asynchronous communication. In *Foundations of Computer Software: future Trends and Techniques for Development*, Monterey'08, pages 154–180. Springer.

Hennicker, R. and Knapp, A. (2011). Modal interface theories for communication-safe component assemblies. In *8th international conference on Theoretical aspects of computing*, ICTAC'11, pages 135–153. Springer.

Henzinger, T. A. and Sifakis, J. (2006). The embedded systems design challenge. In *FM*, pages 1–15.

Kapova, L., Buhnova, B., Martens, A., Happe, J., and Reussner, R. (2010). State dependence in performance evaluation of component-based software systems. In *International conference on Performance engineering*, WOSP/SIPEW '10, pages 37–48. ACM.

Katoen, J.-P. (2005). Labelled transition systems. In Broy, M., Jonsson, B., Katoen, J.-P., Leucker, M., and

Pretschner, A., editors, *Model-Based Testing of Reactive Systems*, volume 3472 of *Lecture Notes in Computer Science*, pages 615–616. Springer.

Klare, G. R. (2000). Readable computer documentation. *ACM J. Comput. Doc.*, 24(3):148–168.

Kuhn, T. (2010). *Controlled English for Knowledge Representation*. PhD thesis, Faculty of Economics, Business Administration and Information Technology of the University of Zurich.

Lee, B. (2006). Optimizing heterogeneous architectures. *EDN*.

Macias, B. and Pulman, S. G. (1993). *Natural Language Processing for Requirements Specification*, pages 67–89. Chapman and Hall Ltd., London.

Maraninchi, F. and Rémond, Y. (2003). Mode-automata: a new domain-specific construct for the development of safe critical systems. *Sci. Comput. Program.*, 46(3):219–254.

Martens, A., Becker, S., Koziolek, H., and Reussner, R. (2008). An empirical investigation of the effort of creating reusable, component-based models for performance prediction. In *Component-Based Software Engineering*, pages 16–31. Springer.

Mudge, T. N. (2001). Power: A first-class architectural design constraint. *IEEE Computer*, 34(4):52–58.

Peake, I. D. and Schmidt, H. W. (2011). Systematic simplicity-accuracy tradeoffs in parameterised contract models. In *Seventh International ACM Sigsoft Conference on the Quality of Software Architectures (QoSA)*, Boulder, Colorado, USA.

Rao, R., Vrudhula, S., and Rakhmatov, D. (2003). Battery modeling for energy aware system design. *IEEE Computer*, 36(12):77 – 87.

Redmill, F. and Rajan, J. (1996). *Human Factors in Safety-Critical Systems*. Butterworth-Heinemann.

Sangiovanni-Vincentelli, A. L. and Martin, G. (2001). Platform-based design and software design methodology for embedded systems. *IEEE Design & Test of Computers*, 18(6):23–33.

Saxe, E. (2010). Power-efficient software. *Communications of the ACM*, 53(2).

Schmidt, H. W. (2003). Trustworthy components - compositionality and prediction. *Journal of Systems and Software*, 65(3):215–225.

Schmidt, H. W., Peake, I. D., Xie, J., Thomas, I. E., Krämer, B. J., Fay, A., and Bort, P. (2003). Modelling Predictable Component-Based Distributed Control Architectures. In *Object-Oriented Real-Time Dependable Systems*, pages 339–346.

Spichkova, M. (2010). From Semiformal Requirements To Formal Specifications via MSCs. Technical Report TUM-I1019, TU München.

Spichkova, M. (2013). Design of formal languages and interfaces: "formal" does not mean "unreadable". In Blashki, K. and Isaias, P., editors, *Emerging Research and Trends in Interactivity and the Human-Computer Interface*. IGI Global.

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

Spichkova, M. and Zamansky, A. (2016). A human-centred framework for combinatorial test design. In *11th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*, pages 228–233. SciTePress.

Spichkova, M., Zhu, X., and Mou, D. (2013). Do we really need to write documentation for a system? In *International Conference on Model-Driven Engineering and Software Development (MODELSWARD'13)*.

Talpin, J.-P., Brunette, C., Gautier, T., and Gamatié, A. (2006). Polychronous mode automata. In *Proceedings of the 6th ACM & IEEE International conference on Embedded software*, EMSOFT '06, pages 83–92, New York, NY, USA. ACM.

Valiant, L. G. (1990). A Bridging Model for Parallel Computation. *Communications of the ACM*, 33(8).

Vo, P. T. N. and Spichkova, M. (2016). Model-based generation of natural language specifications. In *Federation of International Conferences on Software Technologies: Applications and Foundations*, pages 221–231. Springer.

Wolf, W., Jerraya, A. A., and Martin, G. (2008). Multiprocessor system-on-chip (MPSoc) technology. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(10).

Yusuf, I. I., Schmidt, H. W., and Peake, I. D. (2011). Architecture-based fault tolerance support for grid applications. In *Quality of Software Architectures*, QoSA'11, pages 177–181. ACM.

Zamansky, A., Spichkova, M., Rodriguez-Navas, G., Herrmann, P., and Blech, J. O. (2018). Towards classification of lightweight formal methods. In *13th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*, pages 305–313. SciTePress.