Towards Early Emergent Property Understanding

Merging Behavior Space Exploration and Model-Based Software Engineering

Georg Hackenberg
Technische Universität München
Boltzmannstrasse 3
85748 Garching b. München, Germany
hackenbe@in.tum.de

Denis Bytschkow
fortiss GmbH
Guerickestrasse 25
80805 München, Germany
denis.bytschkow@fortiss.org

ABSTRACT

During early phases of complex systems engineering typically many structural and behavioral aspects are unclear. In particular, when it comes to constraints on the result of interactions between distributed components (emergent properties) current software engineering approaches provide limited support. Therefore, we propose an extension to current software models for describing the goals of interaction rather than the underlying decision logic. Further, we propose a generic algorithm for obtaining goal-oriented behavior. Finally, the concepts are evaluated in a case study.

Categories and Subject Descriptors

D.2.2 [**Software Eng.**]: Design Tools and Techniques

General Terms

Model-based software engineering, behavior space exploration, emergent property design

1. INTRODUCTION

Software engineering is a difficult task, especially when it comes to the control of large scale and distributed cyber physical systems. During early phases of development typically many aspects of system structure and behavior are uncertain. For example, in the case of smart grids [16] most of the physical correlations as well as constraints on physical properties are well-understood. However, it is an unsolved question how the smartness in terms of sensors, actuators and respective software components could look like inside the different system parts.

One central and critical challenge is the systematic, targeted and efficient engineering of emergent properties. Hereby emergent properties refer to the result of primitive interactions between components in a complex system [20]. The balance between energy generation and consumption at all times is such an emergent property for the smart grid domain. The primitive interactions are the control decisions taken by local (software) components, which might be deployed to household devices or in an office environment.

In case of the smart grid, evidence exists that formal methods are suitable to tackle the problem of engineering a dependable control system [11]. However, due to the recency of this application domain current approaches only provide limited understanding of the structure and behavior of the respective software landscape. The problem of structure has already been addressed by specialized approaches for defining, implementing and testing according architectures [12]. The problem of behavior is more difficult to solve due to the emergent nature of primitive interactions.

Formally speaking, emergent property engineering deals with finding suitable component behavior efficiently. Today, most commonly used model-based engineering frameworks provide some form of mathematical logic to express component behavior [18]. The purpose of the logic is to determine the behavior with respect to relevant environmental conditions. Consequently, to determine the behavior the relevant environmental conditions have to be identified and their relationship has to be formalized.

Problem statement. Due to the complexity of the system as well as due to the distributed nature of primitive decisions it is difficult to really determine the behavior with respect to the emergent properties. On the other hand, if the behavior of system components is underspecified it is difficult to provide relevant insights into property-driven behavior. Therefore, the challenge is to enrich today's models with as little information as possible to enable behavioral analysis already at early stages of development. Secondly, suitable analysis methods have to be found to utilize this information for generating meaningful analysis results.

Contribution. In this work we propose an extension to current model-based software engineering methods [3] to encode primitive interactions and their goals. In particular, the goals of primitive interactions are modeled in terms of (possibly) non-linear minimization problems with respect to non-deterministic control decisions over time. Further, an algorithm is proposed, which can be used to generate traces of optimal control decisions with respect to scenario-like input. The algorithm is analyzed in terms of the validity of the result as well as the performance of execution.

2. RELATED WORK

As argued in the introduction (Section 1) model-based software engineering provides key quality characteristics with respect to a number of engineering problems related to complex system development. Probably the most widely used approach is the UML [18] providing a wide range of views onto the system under development. Regarding critical behavioral properties, more formal approaches [3] are used to ensure a high level of system dependability. However, a strong link between current approaches and emergent property engineering is missing.

Considering the evaluation of non-deterministic models – in particular at early stages of development projects – a number of analysis techniques exist. Key examples are nondeterministic [10], bounded [6], and probabilistic [14] model checkers. Typically, their focus is on proving properties about the reachable state space of a system under predefined environmental conditions. Though delivering useful results, their prover characteristics might, however, be too strong when considering a high degree of uncertainty at early stages of system development.

An alternative to proof-oriented model checking is the idea of behavior space exploration limiting model analysis to heursitic and/or random results. A number of approaches exist varying in details of the exploration algorithm [5, 1, 17] as well as the underlying system model [7, 8]. The characteristics of the approaches are well suited to early problem understanding with respect to the modeling effort as well as the analysis performance. For operationalization one key problem with behavior space exploration is to define a tight integration with established software models and model-based software engineering methods.

Assuming a mature problem understanding (typically found at later stages of development projects) the solution to achieving emergent property through primitive interactions can, for example, be seen in the domain of robot planning. The approaches typically employ techniques like dynamic programming [19] or distributed search [9] using among other things probabilistic Markov models [21, 22]. The respective results provide a good understanding of how respective architectures and behaviors might look like, but their transfer to other problem domains remains a challenge. This in particular holds when considering that robot planning is typically concerned with achieving global motion characteristics.

Finally, some approaches applicable to our problem domain are, in particular, dedicated to the issue of distributed control [2, 4]. One important contribution from this field of research is the concept of model predictive control, where models of the system are used at runtime to derive control decisions for each time step. Again, these ideas already provide a solution-oriented view onto the engineering of complex systems with emergent properties. Overall, it remains a challenge to define the link between early problem understanding as promoted by model-based software engineering and concrete solution strategies found in many domains.

In this work we target the challenge by integrating the ideas of behavior space exploration and software models.

3. FRAMEWORK

The proposed framework is derived from the FOCUS theory [3] through simplification. The following concepts are removed: Interfaces, ports, channels, and automata. Instead, observations and typed expressions are used as explained in the following section.

3.1 System Theory

A system is defined by a set of components $\mathbb{C}$ and a set of observations $\mathbb{O}$. Components are defined in Equation 1.

$$C = (\mathbf{O}, \mathbf{C}), \mathbf{O} \subseteq \mathbb{O}, \mathbf{C} \subseteq \mathbb{C} \tag{1}$$

Observations are defined as a mapping from the time domain T to some data domain $D \in \mathbb{D}$ (see in Equation 2). Our prototypical implementation currently supports the data domains $\mathbb{R}$ ($java.lang.Double$), $\mathbb{B}$ ($java.lang.Boolean$) and $\mathbb{S}$ ($java.lang.String$) as well as the discrete time domain $\mathbb{N}$ ($java.lang.Integer$).

$$O : T \rightarrow D \tag{2}$$

If an observation O is (a) deterministic, a mapping rule is provided. A mapping rule is a typed expression with argument $t \in T$ as shown in Equation 3.

$$O(t) = Expression_D(t), D \in \mathbb{D} \tag{3}$$

If an observation O is (b) non-deterministic no mapping rule is supplied. Instead, the framework explores the behavior space to find good solutions according to some optimization criteria. The optimization criteria are specified by means of annotating respective observations. Therefore an annotation mapping is defined from the set of observations $\mathbb{O}$ to the powerset of annotations $\mathbb{A}$ as shown in Equation 4.

$$A : \mathbb{O} \rightarrow \mathcal{P}(\mathbb{A}) \tag{4}$$

The set of currently supported annotations $\mathbb{A}$ is defined in Equation 5. The meaning of the annotations is explained in Section 3.2.

$$\mathbb{A} = \{require, equals, minimize, maximize, cost\} \tag{5}$$

3.2 Exploration Algorithm

The exploration algorithm serves to find good choices for non-deterministic observations according to the annotations. For each consecutive time point $t \in T$ the following sequence of steps is executed:

1. **Generate** all variants for non-deterministic observations O without $O(t)$.

2. **Calculate** the respective values for the remaining observations O with $O(t)$.
3. **Verify** the required boolean observations (i.e. constraints) $O(t) = true$ where $require \in A(O)$.
4. **Prune** the dominated variants according to *equals*, *minimize* and *maximize* annotations (see below).
5. **Sort** the variants according to the single observation O with $cost \in A(O)$.

In step 4 (pruning) the verified variants are compared to each other with the goal to limit the following algorithm iterations to those variants that are guaranteed to lead to better results than the ones pruned. Therefore a dominance relationship between variants is introduced by means of the annotations *equals*, *minimize* and *maximize*. According to this relationship variant V_1 is dominated by variant $V2$ if $\forall O \in \mathbb{O}$ the following equation holds:

$$\begin{array}{rcl} equals \in A(O) & \Rightarrow & O_{V_1}(t) = O_{V_2}(t) \\ minimize \in A(O) & \Rightarrow & O_{V_1}(t) \geq O_{V_2}(t) \\ maximize \in A(O) & \Rightarrow & O_{V_1}(t) \leq O_{V_2}(t) \end{array} \quad (6)$$

The equation states that some observations have to be equal for variants to be comparable. Then, for some observations it is better to follow variants with lower values, and for some observations it is better to follow variants with higher value. Typically, the *minimize* annotation is used on the *cost* observation to indicate, that less cost is preferable. Overall, the pruning strategy allows to reduce the search space drastically depending on the system model and the dynamic programming annotations.

4. CASE STUDY

To evaluate the approach we investigate the behavior of a refrigerator and an energy storage with respect to available sun power. As emergent property we define minimum cumulated difference between produced and consumed energy (i.e. minimum load on the connection point to the grid) as an indicator for operational autonomy of the sub-system under investigation. In [11] we motivated the use of formal model-based engineering in the energy system domain due to its criticality. Here we extend the existing model with exploration constraints to obtain cost-oriented component behavior using our dynamic programming algorithm.

4.1 System Model

With respect to performance analysis two variants of the model are compared: (M1) A single $refrigerator$ and (M2) a combination of $refrigerator$ and $storage$.

The refrigerator model (M1) is depicted in Figure 1. Seven observations are provided: Sun power, refrigerator temperature, refrigerator power, refrigerator input (from the controller), refrigerator constraint, balance and cost (for minimization). The sun power O_{Sun} provides scenario-like input for the exploration. The values are read from a CSV file.

The refrigerator input O_{IR} represents a primitive interaction and, therefore, is modeled non-deterministically.

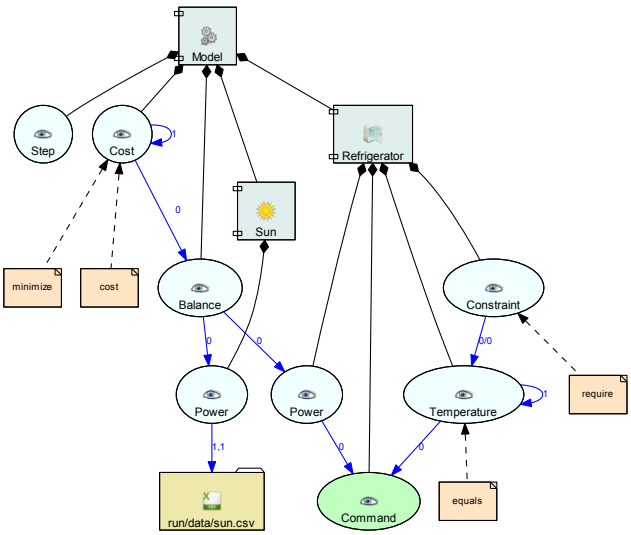

Figure 1: Refrigerator model

$$O_{IR} : T \rightarrow \{0, 1\} \quad (7)$$

The refrigerator power O_{PR} depends on the refrigerator input O_{IR}, i.e. whether the cooler is activated or not.

$$O_{PR}(t) = 200W * O_{IR}(t) \quad (8)$$

The refrigerator temperature O_{TR} is calculated from the temperature of the previous time point, a heat-up function f_{heat}, a cool-down function f_{cool} and the non-deterministic controller input O_{IR}. For simplicity, in this work the functions f_{heat} and f_{cool} are constant. For more precise physical behavior, however, the theory also supports more complex equations.

$$O_{TR}(t) = O_{TR}(t-1) + f_{heat}(t) - f_{cool}(t) * O_{IR}(t) \quad (9)$$

A typical refrigerator constraint O_{CR} then tests whether the temperature O_{TR} lies inside a predefined temperature band between T_{min} and T_{max}.

$$O_{CR}(t) = T_{min} \leq O_{TR}(t) \leq T_{max} \quad (10)$$

The balance calculates the difference between produced and consumed power at the current time point. Note that produced and consumed power vary in sign [11].

$$O_{Balance}(t) = O_{Sun}(t) + O_{PR}(t) \quad (11)$$

Finally, the cost O_{Cost} is derived from the cost of the previous time point and the absolute value of the current balance $O_{Balance}$.

$$O_{Cost}(t) = O_{Cost}(t-1) + |O_{Balance}(t)| \tag{12}$$

The combined model (M2) reuses the balance and cost observations from the refrigerator model. Additionally, observations for storage input (by the controller), storage power, storage level and storage contraint are defined. Again, the input O_{IS} is modeled non-deterministically.

$$O_{IS} : T \to \{-1, \pm 0, +1\} \tag{13}$$

Further, the storage power O_{PS} depends the input O_{IS}, i.e. whether the storage is off, loads up the level, or unloads the level.

$$O_{PS}(t) = 200W * O_{IS}(t) \tag{14}$$

The storage level O_{LS} is then calculated from the level of the previous time point, a loss factor e and a load constant which is depending on the input O_{IS}.

$$O_{LS}(t) = O_{LS}(t-1) * e + \begin{cases} -200 & \text{if } O_{IS}(t) = -1 \\ \pm 0 & \text{if } O_{IS}(t) = \pm 0 \\ +100 & \text{if } O_{IS}(t) = +1 \end{cases} \tag{15}$$

Finally, the storage constraint O_{CS} is simply defined as the level O_{LS} being in the interval between 0 and L_{max}.

$$O_{CS}(t) = 0 \leq O_{LS}(t) \leq L_{max} \tag{16}$$

The combined model (M2) then redefines the balance observation $O_{Balance}$ by using the sum of the powers O_{PS}, O_{PR} and O_{PS}.

$$O_{Balance}(t) = O_{Sun}(t) + O_{PR}(t) + O_{PS}(t) \tag{17}$$

To guide the exploration algorithm, first the cost observation is defined. Additionally, a *minimize* constraint is added for pruning.

$$A(O_{Cost}) = \{minimize, cost\} \tag{18}$$

Further, the constraint observations are defined to be required, i.e. they have to evaluate to *true* to survive the *verify* phase.

$$A(O_{CR/CS}) = \{require\} \tag{19}$$

Finally, to make sure the options are not pruned too aggressively the refrigerator temperature and storage level are required to be equal.

$$A(O_{TR/LS}) = \{equals\} \tag{20}$$

4.2 Exploration Results

The optimal behavior according to the optimization constraints is shown for both cases in Figure 2.

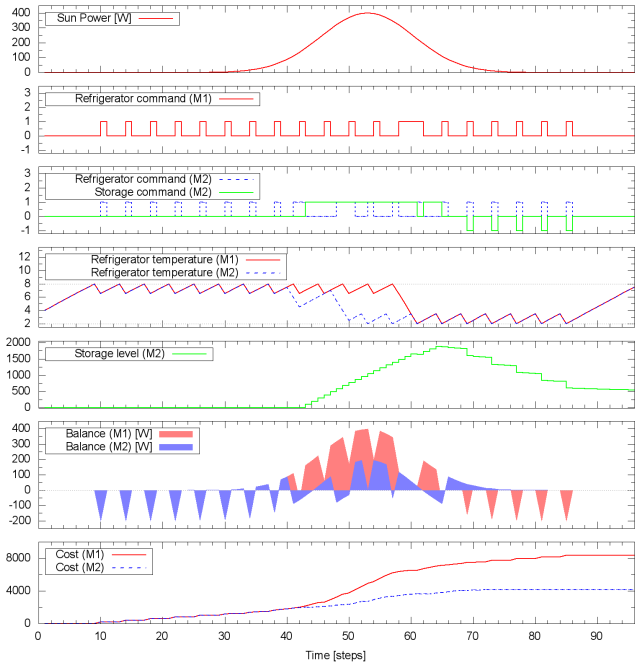

Figure 2: Cost-optimized behavior after 96 time steps for all three cases.

The intuition behind the refrigerator (M1) control strategy is to use energy during those periods when the sun energy production is suitable. Minimizing costs according the function (12) is the optimization criteria. Temperature has the equality dominance. This means, if two behavior traces have equal temperature, but one of them has lower costs, the algorithms, will prune the trace with higher costs and follow the behavior with lower cost only. The desired temperature band between $T_{min} = 2\,\mathrm{C}$ and $T_{max} = 8\,\mathrm{C}$ acts as a constraint, which must be fulfilled every time step.

As can be seen in Figure 2, the found behavior strategy of (M1) is keeping the refrigerator at high temperature to save costs and cool only to keep the required temperature band. Only at time step 57 the refrigerator uses the energy provided by the sun to cool down to the lowest level possible reducing the overall costs and having thermal buffer. Afterwards the refrigerator activates cooling 5 time steps more and uses the thermal buffer to reach an optimal solution. Please note that even if only one illustration of the behavior is shown, also other behavior traces could potentially yield the same final costs value.

The combined model (M2) includes an additional storage

component. The optimization criteria is not changed. The storage possibility reduces the cost of the system by loading the storage from time step 43 to 65 and unloading it during the time steps, when cooling energy is required to keep the temperature band. Storage behavior influences also the refrigerator, shifting the cooling to other time steps, during the sunny period. Consequently, storage reduces the cost function by a factor of 2.

4.3 Exploration Performance

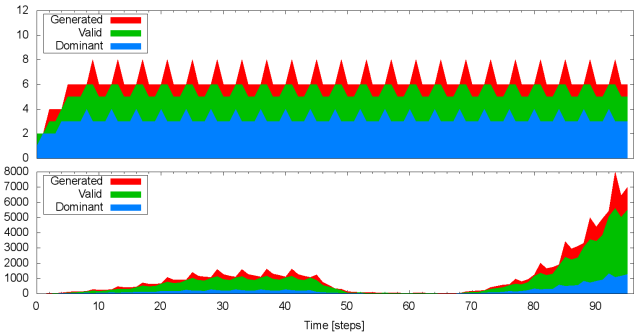

Figure 3: Explored behavior after 96 time steps for all three cases.

The exploration performance is analyzed in terms of generated, valid and dominant (partial) behavior per time step as illustrated in Figure 3. For the refrigerator model (M1) the found complexity seems to be constant due to a maximum of 8 generated alternatives per time step. Some generated traces do not fulfill the constraints and, therefore, do not pass the verify step (see Section 3.2). From the remaining traces some are pruned according the dominance annotations. Only few traces are kept for further exploration steps.

For the extended model (M2) initially a stronger increase of the number of alternatives can be observed. The reason is that the model defines an *equals* dominance on the storage level. During the sunny period the number of traces reduces significantly since the cost minimum dominance strongly prunes alternatives. When no sun power is provided, the number of traces increases again.

In all cases pruning discards at least 50% of alternatives in each time step. This reduces the calculation effort greatly. In contrast verification of required boolean properties has less influence on search space reduction.

5. DISCUSSION

The case study shows that the combination of behavior space exploration and traditional formal and model-based software engineering has the potential to reduce modeling effort in early project phases while providing useful insights on system performance. In particular, the goals of primitive decisions (and interaction between components) can be modeled rather than the decision process underlying each component. This allows the developer to obtain goal-oriented behavior quickly providing fertile ground for discussion, understanding and refinement in the following iterations of the project. Therefore, the approach is well-suited for use in early phases when problem understanding is limited and stakeholder requirements are uncertain.

Currently, the model extensions are limited to numerical goals only, i.e. minimization of a certain *cost* observation. Further, behavior space exploration can only be guided by means of a limited notion of dominance between variants consisting of *equals*, *minimize* and *maximize* annotations. For the problem investigated in section 4 these extensions are already sufficient to obtain the desired exploration results. It is, however, unclear how well the current approach fits to other problem domains and larger problem spaces. One challenge will be the integration of qualitative observations as opposed to quantitative (i.e. numerical) observations and their handling during pruning and sorting.

Regarding the exploration algorithm, optimal results can be expected assuming unlimited memory and time. On the downside, in particular larger problem spaces typically cause the problem of state space explosion [15], therefore practically preventing an exhaustive exploration. To deal with this problem, some optimization can be made to generate as few variants a possible through verifying and pruning as early as possible. However, with ever increasing problem spaces the potential for algorithm optimization is rather limited demanding heuristic approaches such as genetic programming [13]. Essentially, a respective engineering tool would support a range of different exploration strategies each providing varying performance on a given problem.

As the proposed approach also tries to provide first steps towards a model-based emergent property engineering method, the question has to be answered how the obtained exploration results can be used to guide development decisions. Typically, some sort of model refinement is used to assure a systematic way from the initial problem understanding to precise notions of requirements to solutions accepted by the stakeholders. As such, the question has to be answered how the combination of structural, behavioral, non-deterministic and annotational model concepts can be tailored iteratively, while ensuring high process efficiency. A current direction is to go from completely non-deterministic models over probabilistic models to entirely determinisic models.

Finally, the current approach is limited to a very specific notion of the *system* including static structure, discrete time and a limited observation type space (extensible through custom enumeration types). It seems that discrete time is sufficient for system models at early phases of the project, where the main goals are problem understanding and requirements specification. Deployable solutions, however, have to cope with continuous physical effects requiring more precise and complex behavioral models of the system components. In contrast, dynamic system structure is a real blocker for some application domains such as systems with high component mobility.

6. CONCLUSION

We took a first step towards engineering emergent properties through integrating behavior space exploration into traditional model-based software engineering. The approach deliberates the developer from specifying the logic behind the primitive interactions of components and rather focusing on understanding and modeling the goal of interaction. In a case study we showed how to use the approach for modeling the problem of refrigerator and storage control in an

energy system with respect to a price curve.

Current limitations are due to the strong numerical focus of the goals as well as the annotations for guiding the exploration algorithm. Also, the algorithm performance is limited due to the exhaustive exploration of arbitrary problems, where there might not be enough potential for pruning the search space. Finally, the work only focuses on the technical realization of and experimentation with the combination of exploration and today's software models.

In future work we plan to scale the approach to larger problem spaces including hundreds of distributed, independent, but collaborating households and work environments. Further we target investigation of these models with respect to engineering activities in particular at early stages of development projects. The long-term goal of this research is to provide the right models, tools and methods for systematically engineering large-scale distributed systems with critical behavioral requirements.

7. REFERENCES

[1] R. Alur, R. Brayton, T. Henzinger, S. Qadeer, and S. Rajamani. Partial-order reduction in symbolic state-space exploration. *Formal Methods in System Design*, 18:97–116, 2001.

[2] B. Bamieh, F. Paganini, and M. Dahleh. Distributed control of spatially invariant systems. *Automatic Control, IEEE Transactions on*, 47(7):1091 –1107, jul 2002.

[3] M. Broy and K. Stølen. *Specification and development of interactive systems: focus on streams, interfaces, and refinement*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.

[4] E. Camponogara, D. Jia, B. Krogh, and S. Talukdar. Distributed model predictive control. *Control Systems, IEEE*, 22(1):44 –52, feb 2002.

[5] S. Christensen, L. Kristensen, and T. Mailund. A sweep-line method for state space exploration. In T. Margaria and W. Yi, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 2031 of *Lecture Notes in Computer Science*, pages 450–464. Springer Berlin / Heidelberg, 2001.

[6] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. Nusmv 2: An opensource tool for symbolic model checking. In E. Brinksma and K. Larsen, editors, *Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, pages 241–268. Springer Berlin / Heidelberg, 2002.

[7] E. de Souza e Silva and P. M. Ochoa. State space exploration in markov models. *SIGMETRICS Perform. Eval. Rev.*, 20(1):152–166, June 1992.

[8] S. Derisavi, P. Kemper, and W. H. Sanders. Symbolic state-space exploration and numerical analysis of state-sharing composed models. *Linear Algebra and its Applications*, 386(0):137 – 166, 2004.

[9] E. Durfee and T. Montgomery. Coordination as distributed search in a hierarchical behavior space. *Systems, Man and Cybernetics, IEEE Transactions on*, 21(6):1363 –1378, nov/dec 1991.

[10] E. Emerson and K. Namjoshi. On model checking for non-deterministic infinite-state systems. In *Logic in Computer Science, 1998. Proceedings. Thirteenth Annual IEEE Symposium on*, pages 70 –80, jun 1998.

[11] G. Hackenberg, M. Irlbeck, V. Koutsoumpas, and D. Bytschkow. Applying formal software engineering techniques to smart grids. In *Proceedings of the 1st International Workshop on Software Engineering Challenges for the Smart Grid*, SE4SG '2012, New York, NY, USA, 2012. ACM.

[12] D. Koss, F. Sellmayr, S. Bauereiss, D. Bytschkow, P. Gupta, and B. Schätz. Establishing a smart grid node architecture and demonstrator in an office environment using the soa approach. In *Proceedings of the 1st International Workshop on Software Engineering Challenges for the Smart Grid*, SE4SG '2012, New York, NY, USA, 2012. ACM.

[13] J. Koza and R. Poli. Genetic programming. In E. K. Burke and G. Kendall, editors, *Search Methodologies*, pages 127–164. Springer US, 2005.

[14] M. Kwiatkowska, G. Norman, and D. Parker. Prism: Probabilistic symbolic model checker. In T. Field, P. Harrison, J. Bradley, and U. Harder, editors, *Computer Performance Evaluation: Modelling Techniques and Tools*, volume 2324 of *Lecture Notes in Computer Science*, pages 113–140. Springer Berlin / Heidelberg, 2002.

[15] F. J. Lin, P. M. Chu, and M. T. Liu. Protocol verification using reachability analysis: the state space explosion problem and relief strategies. *SIGCOMM Comput. Commun. Rev.*, 17(5):126–135, Aug. 1987.

[16] S. Massoud Amin and B. Wollenberg. Toward a smart grid: power delivery for the 21st century. *Power and Energy Magazine, IEEE*, 3(5):34 – 41, sept.-oct. 2005.

[17] R. Pelánek, T. Hanžl, I. Černá, and L. Brim. Enhancing random walk state space exploration. In *Proceedings of the 10th international workshop on Formal methods for industrial critical systems*, FMICS '05, pages 98–105, New York, NY, USA, 2005. ACM.

[18] B. Selic. On the semantic foundations of standard uml 2.0. In M. Bernardo and F. Corradini, editors, *Formal Methods for the Design of Real-Time Systems*, volume 3185 of *Lecture Notes in Computer Science*, pages 75–76. Springer Berlin / Heidelberg, 2004.

[19] K. Shin and N. McKay. A dynamic programming approach to trajectory planning of robotic manipulators. *Automatic Control, IEEE Transactions on*, 31(6):491 – 500, jun 1986.

[20] L. Steels. Towards a theory of emergent functionality. In *Proceedings of the first international conference on simulation of adaptive behavior on From animals to animats*, pages 451–461, Cambridge, MA, USA, 1990. MIT Press.

[21] P. Svestka and M. Overmars. Coordinated motion planning for multiple car-like robots using probabilistic roadmaps. In *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on*, volume 2, pages 1631 –1636 vol.2, may 1995.

[22] R. Washington. Incremental markov-model planning. In *Tools with Artificial Intelligence, 1996., Proceedings Eighth IEEE International Conference on*, pages 41 – 47, nov. 1996.