Proc. IEEE Int. Systems Conference (SysCon 2015), Vancouver, Canada, April 2015, pp. 660-667.

# System Architecture Optimization With Runtime Reconfiguration of Simulation Models

Alexander Wichmann, Sven Jäger, Tino Jungebloud, Ralph Maschotta, and Armin Zimmermann
Systems and Software Engineering Group
Computer Science and Automation Department
Technische Universität Ilmenau
Ilmenau, Germany
Contact: see http://www.tu-ilmenau.de/sse

***Abstract*—The development of complex systems often leads to delays in integration and roll-out, massive cost overruns, and in many cases to sub-optimal solutions regarding system performance and robustness. In order to overcome these issues, model-based systems engineering methodologies can be employed. The application of simulation models in addition to system specifications increase the engineers' understanding of complex system behavior. The approach also allows the detection of incomplete or inconsistent functional requirements as well as other design failures.**

**With increasing complexity of system models it is necessary to optimize parameters to achieve a resource-efficient system. Indirect simulation is a well-known tool for this task, and there are numerous heuristics that can control the search in the parameter space. However, this task is much harder if the design parameters are not only numerical values from a predefined interval. In many cases the design issue (and optimization task) involves decisions about how a system architecture should be composed out of modules and interfaces / connections. In such cases, already the generation of feasible design variants may be hard to automate as a part of an optimization algorithm. Domain-specific knowledge and technical constraints have to be taken into account, and the non-numerical parameters have to be encoded such that they become accessible by the controlling heuristic.**

**This paper presents a model-based methodology to optimize the architecture of a wireless sensor network that is designed for future avionic applications. We show how dynamically reconfigurable simulation models can be used to simplify an optimization of architecture or topology of a system. The system modeling and design tool MLDesigner is used for model implementation and simulations. In addition to that, the heuristic optimization process is described in such a model, thus controlling on a meta level the actual optimization. It can be changed or adapted easily, because it is not programmed but described with a high-level model.**

***Index Terms*—Performance evaluation, global optimization, discrete event simulation, probabilistic metaheuristic, simulated annealing, model reconfiguration**

## I. Introduction

Early system design phases are of particular importance and decisive for the success of a project. Errors in this phase lead to very costly changes later on. Wrong design decisions and insufficient implemented functional requirements result in poor system solutions which remain undetected until final testing. During later project phases such as component integration, sub-system or overall-system tests, failures are discovered too late. These circumstances force designers to reiterate earlier design phases, resulting in roll out delays and cost overruns.

Model-based system engineering can reduce the cost of the development process and increase the ability to detect costly design failures during early design phases. By using executable models (i.e., models that can be simulated and their performance evaluated), a validation of the behavior of the final system is possible. These models can be thought of as an executable *specification* instead of a text document which cannot be technically validated.

Resource efficiency of a complex system often relies on the collaboration of different components, which is difficult to estimate during the design phase. Moreover, decisions of module designers may improve the local behavior, but nevertheless degrade the overall properties of the system. The effect on the global system properties is often very hard to predict. The architecture of these systems is thus often designed with tolerance margins. To achieve a more resource-efficient system, these margins have to be reduced. Experiments have to be created, in which solution, subsystems, or even the entire system should be checked, executed, and analyzed manually. This is a time-consuming task and requires a good amount of expert knowledge. Another approach is to use the executable specification together with simulation-based optimization heuristics to reduce the effort of finding a more efficient architecture of the system.

The aim of this type of optimization is to design a system or its architecture such that its key features are optimally set. This may include low production cost, low running costs, size, weight, reliability, robustness, and so on. For the optimization it is important that each feature is mathematically expressible, such that an optimization algorithm can estimate and compare it. The function of a system to be optimized usually contains several conflicting of non-functional requirements. Weights have to be set to define the relative importance to find the best trade-off: a compromise has to be found such that the overall benefit will be optimal.

The optimization of systems based on simulation models is an important and widely covered research area [1], [2], [3]. For our application domain of avionics and wireless sensors, Sghairi et al. [4] describe how a flight control system can be constructed in terms of architecture optimization. Of particular importance is the extremely high reliability of the system. The aim of the optimization is to build an architecture which

is less expensive and contains little redundancy, but is still safe enough. The solution consists in an incremental modeling approach, where gradually more modules are added on a basic level based on the requirements.

The work of Azami et al. [5] discusses energy and lifetime issues of wireless sensors, which have their own finite power supply and different tasks to solve. The optimization goal is the selection of sensor nodes, which completely fulfill their task and thereby require as little energy as possible to have a long life time. This problem was solved with a selection algorithm based on simulated annealing [6]. Further work on optimization of wireless sensor networks include [7], [8], [9].

Automatic optimization of complex system architecture models requires a domain-specific tool chain. In the work of Leeuwen et al. [10], a closed-loop design optimization method was shown. A model described in a domain-specific modeling language similar to UML/MARTE or SysML is transformed to executable code for a discrete-event simulator called DynAA. The architectural optimization is done by MatLab.

There are numerous adaptable parameters and choices when selecting and using a heuristic for indirect optimization. Programming a tool and integrating it with a simulator for such a task would thus require to change parts and recompile the tool often. It is thus necessary to increase the level of technical detail and reduce the amount of tools to be integrated, to ease the task for a system designer and modeler. One promising option is to specify the optimization process itself with a specific model that will control the actual optimization run without the necessity of programming. If it is possible to use the same tool as for the system model itself, this would simplify the task even more.

The contribution of this paper is to show how the multi-domain simulation tool MLDesigner [11] can be successfully applied both for run-time system model construction and simulation, as well as on the meta level of the optimization process. The different necessary generic building blocks for optimization sub tasks are developed, forming a library in this tool. This allows to easily switch between optimization algorithms and configure or test them, or even to develop and analyze new variants. As a proof-of-concept, we present a simplified optimization of the architecture of an aircraft wireless sensor network. The architecture can be described as a set of parameters by using a generic approach for the domain model. Parameters for the initial generic model are detected such that at the end of the optimization an approximation for the best possible architecture is determined.

The paper is structured as follows: The subsequent section describes the simulation environment as well as some of its underlying modeling concepts and techniques required for the later system model and optimization. Section III covers the wireless sensor network (WSN) model as well as the model of the optimization loop and its application to the WSN architectural optimization. Results for the architecture optimization of the underlying model are given in Section IV.

## II. Simulation Environment

The network simulation model of this paper was developed using the modeling and simulation framework MLDesigner [11]. MLDesigner is a multi-/hybrid-domain modeling and simulation environment available for all major operating systems. It provides a graphical user interface to develop hierarchical models, execute and debug simulations, and to analyze results. The simulation capabilities comprise different computation domains such as discrete event, synchronous data flow, continuous time/discrete event as well as hierarchical finite state machines.

For our model, we used the discrete event domain [12] for the sensor network and the lifetime model, and finite state machines for the energy consumption description of sensor nodes. To achieve high scalability and runtime reconfiguration, i.e., independence of the model size and structure from the number of similar objects in the system, the tool provides the feature of generating instances of a formal model during a simulation. This capability is referred to as *dynamic instance support* and detailed subsequently.

### A. Dynamic Instances

Modeling of complex networked systems often requires flexibility on how components, elements, and processes are integrated and how they interact. The creation of dynamic model instances in the discrete event domain assists those objectives. Using dynamic model instances, building blocks can be created and deleted while a simulation of the same model is running.

Turning model elements into dynamic instances is straightforward in the chosen tool. From a users point of view, the graphical modeling environment adds several control elements to the existing interface automatically. The model instance now has auxiliary ports. The block diagram structure of an example model is shown in Figure 1 before and after a transformation.

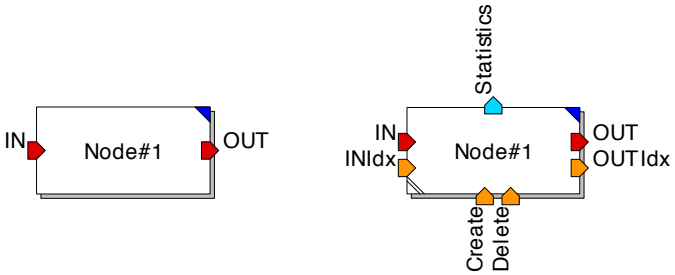


Fig. 1. Exemplary MLDesigner module instance *Node#1*. Left: Normal editor shape; Right: Module with enabled dynamic instance support.

Each input port such as `IN` is associated with a newly created port, the index control port of type integer with the same name and suffix `Idx`. Those control ports specify the number (index) of the runtime instance of the model element that should receive data from this port. If there is no instance known with the given index yet, it will be added automatically.

In addition to the formal input interface, two auxiliary ports named `Create` and `Delete` of type integer are added. These

ports are used to create or delete an instance with the index specified by the value placed on the port. Inputs on these ports are ignored under the following circumstances: (a) Create requests for an already existing index; (b) Delete requests on non-existing instances.

Each output port such as `OUT` is associated with a newly created port, denoting the index release port of type integer with the same name and suffix `Idx`. The port specifies which instance releases data on the associated output port.

## III. Model Design and Methodology

We reuse the wireless sensor model introduced in [13] in the following, to illustrate the method of architecture optimization.

### A. Wireless Sensor Network Model

The schema of the top level model is depicted in Figure 2. It represents an embedded sensor network including wired and wireless networks [14], [15].

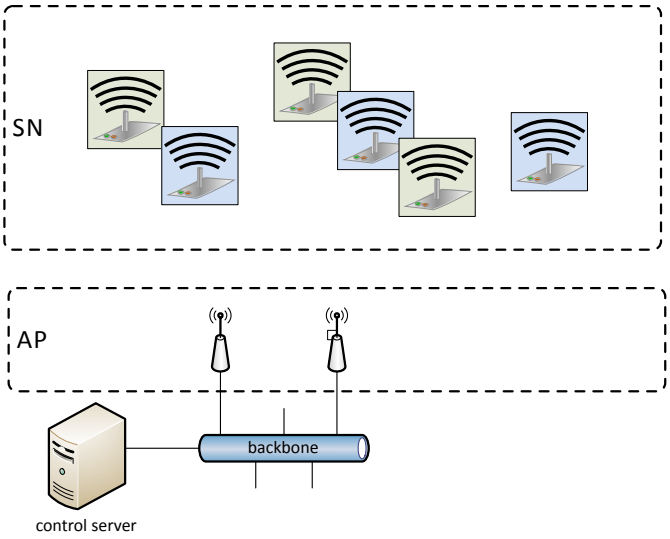


Fig. 2. Top-level view of WSN model [16]

Sensor nodes are a major part of this network. They may exist in a configurable number with variable properties. Each sensor node can be assigned any number of predefined applications. The applications are configurable and generate specific data which will be sent over a shared medium. Each access point, who is in transmission range of a sending sensor node, receives the data. Completely received data is processed by the access point and relayed via a backbone to the server. This shared medium may work with different protocols. In addition to the deterministic protocol, a stochastic protocol and a hybrid one are available. The deterministic protocol is used in the following, where every sensor node and access point has its own time slot to send. Thus collisions are nearly excluded.

This model was implemented with MLDesigner using dynamic instances. Thus it does not contain configuration details, but describes system entities generically. During the execution of the elements, which were described by a dynamic instance,

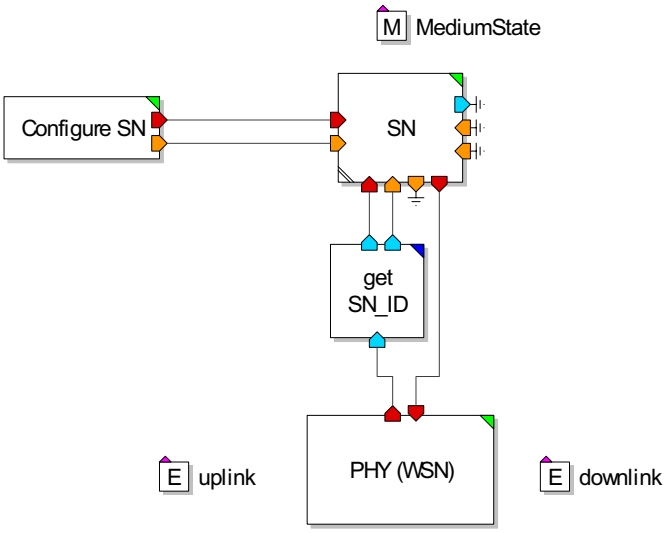


Fig. 3. Model of sensor node environment

they are dynamically generated (instantiated) based on stored template entries and thus made usable.

Dynamic instances were used for the description of sensor nodes, applications, shared medium, and access points. The fundamental operating principle is shown for the example of the sensor nodes. Figure 3 presents the model of the sensor node environment. The module with label *SN* represents the sensor node realized by dynamic instances. The configurations of the sensor node are stored persistently.

The following properties of a sensor node are configurable: the position of the sensor node, the radio frequency to be used, and the slot number for sending data over the shared medium. During execution of the model this configuration is read from the persistent storage using the module *Configure SN*. Then the sensor nodes are dynamically generated with the appropriate properties. The module *PHY (WSN)* provides the connection to the shared medium. It can receive message packets from the medium and transfers it to the sensor node or sends packets from the sensor node over the shared medium.

The refinement of the sensor node is depicted in Figure 4. Each consists of different major modules: The *MAC* module includes the protocol for the communication. The *Simple NET* module provides the connection from the *MAC* layer to the *Applications* layer. The top module represents the applications assigned to a sensor node. As this dynamic instance acts as a template, it is duplicated for each sensor node configuration when the simulation starts. Each copy will get the configuration for a sensor node, stores this configuration in the data model *node_config*, and thus represents an individual sensor node.

It is even possible to use dynamic instances hierarchically. For the sensor nodes such a hierarchical use of dynamic instances is applied (c.f. Figure 5). Each sensor node is assigned to applications, which are modeled using dynamic instances again. The configuration of these applications are thus also saved in a persistent storage place. Furthermore, the

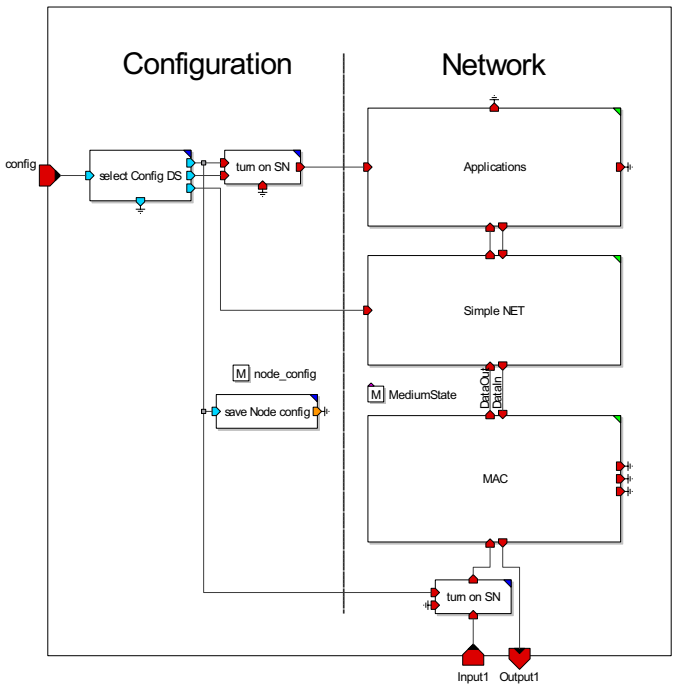


Fig. 4. Internal structure of a sensor node

allocation specifying which application runs on which sensor node is stored there as well. During the execution of the model the current sensor node setup is specified. Thus the configuration for the application is clearly selectable and can be generated dynamically.

Just like the sensor nodes, also the access point is realized using dynamic instances. Figure 6 presents the external structure. The configuration of the access node is again stored persistently, read out via the *AP Config* module, and deposited in the data model *vec_uids*. The access point can be dynamically generated with this information during the execution. The access point has both a connection to the backbone network

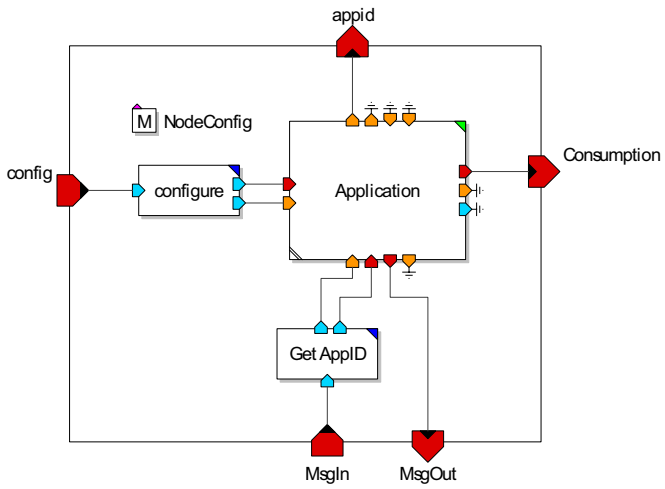


Fig. 5. External structure of an application

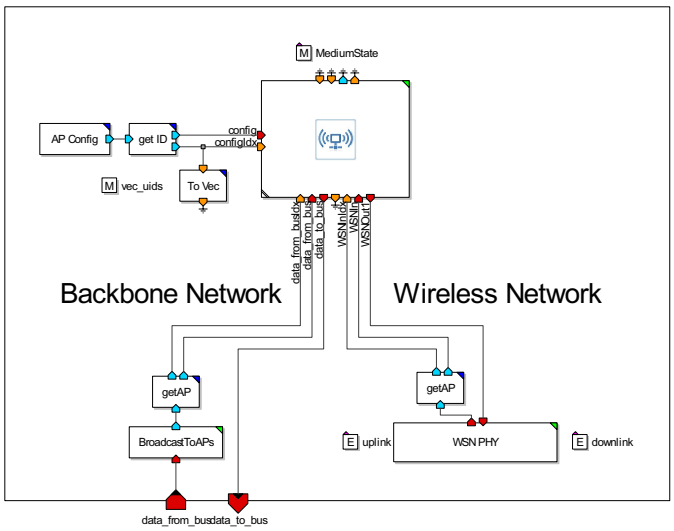


Fig. 6. Access point environment

as well as to the wireless network, such that the access point can forward the received data to the other medium.

### B. Optimization Loop

In the following the general method of architecture optimization is described in detail. The top-level design of this method is represented by a simple simulation loop model shown in Figure 7. It resembles the standard indirect optimization approach [3], [2] and operates as follows.

The loop starts with the module *Heuristic Design Space Exploration* containing the heuristic optimization algorithm to be used. The algorithm decides if an additional loop iteration with another parameter set should be started (i.e., the optimization has not yet finished), or if the process should be stopped with the current best solution as a sufficiently good approximation of the optimal solution. If a further simulation is to be executed, the optimization parameters are recalculated and sent to the next module. For the first pass the optimization parameters are set depending on the initialization definition of the optimization algorithm.

The module *Topology Generation* creates an architecture of the system depending on the optimization parameters sent from the previous module. In principle all parameters of the model and the configuration of the dynamic instances can be changed. Which configurations and parameters are subject to change and how this change takes place has to be defined by the refinement of the module. At the end of the module execution, the generated architecture data is saved in a persistent storage (data base for instance).

The simulation of the model is triggered in the module *Simulation* afterwards, and uses the topology data from the storage. The results of the simulation are stored persistently, such that they can be evaluated after the end of the simulation run. The final element of the simulation loop is the module *Objective Function Calculation*. It includes the objective function which is necessary for the evaluation of the current

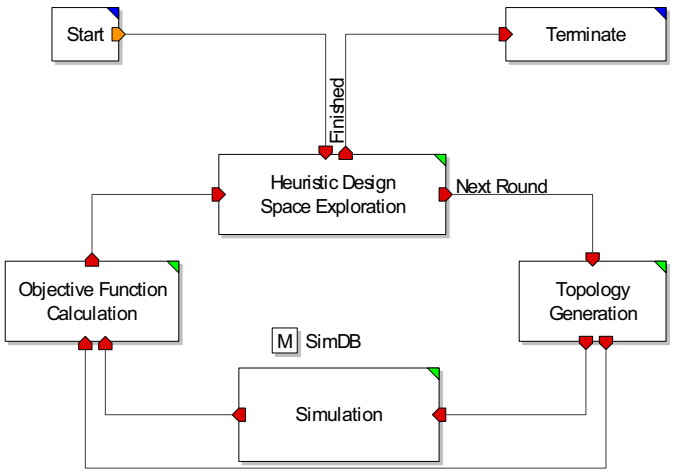


Fig. 7. Model of the simulation loop

architecture. The results from the simulation are analyzed, and the value of the objective function is calculated based on them for the current architecture. Finally, the module *Heuristic Design Space Exploration* is executed again, using the new result value.

### C. Indirect Optimization Model for the Wireless Sensor Network

In order to demonstrate the feasibility of the described optimization loop model, an exemplary configuration of it is described in this section. The module *Heuristic Design Space Exploration* depends on the heuristic optimization algorithm, *Topology Generation* depends on the actual model and system to be analyzed, while the *Objective Function Calculation* specifies the optimization goal. All these modules have to be refined accordingly but help to keep these different aspects of the optimization task independent.

*1) Description of the Heuristic:* It should be noted that this paper is not about optimization heuristic development or evaluation, but presents the model-based technology to implement arbitrary ones. Thus we decided for simplicity to present two initial variants of a heuristic, namely (1) a full exploration ("brute force") of a promising part of the design space, and (2) a random selection of parameters that may span the full design space. This helps in validating the implemented method for the moment and will be later extended to more meaningful heuristics.

The module *Heuristic Design Space Exploration* is thus instantiated with a method where all possible architectures are configured, simulated, and finally analyzed first. At the end the architecture with the best value of the objective function is chosen as the best solution and the optimization ends.

This approach would guarantee to find the real optimum, but is in practice no applicable because of its necessary CPU time. For the validation purpose of this paper, we restricted the full analysis on solutions with only three access points. With less than three there will be always at least one sensor node without connection. On the other hand, more access points are expected to lead to a worse optimization function value (see below), and thus the full coverage of this part was carried out.

However, to check also for other configuration settings not covered completely, random parameter selection was implemented as a second variant of the heuristic module. This will span a larger area of the design space, but will not guarantee finding the optimum. It thus serves as a validation and may give an insight also into how much time can be saved in optimization steps without losing too much result quality. As this heuristic has no obvious stopping condition unlike the first method, we decided to let the algorithm spend similarly long time in the two runs with the heuristics.

*2) Parameter Encoding and Topology Generation:* The architecture is configured as follows: The number and the positions of the access points are changeable, while the possible positions for an access point are preselected. The number of these positions is denoted by $m$. It follows that the number of access points $c$ is limited to be between $0$ and $m$. The storage of both parameters is done in a coded bit vector called $v$. The bit vector $v$ has a size of $m$ entries, each entry contains either a $0$ or a $1$. Each vector can be represented as an integer number $n$. The $i$-th position of vector $v$ indicates whether an access point is placed at the i-th position or not. The number $c$ of access points results from the number of entries with the value $1$.

For the brute force method the determination of this parameter is implemented with a loop: All combinatory possible positions are generated for three access points. Without the restriction of the preselected three access points, an outer loop would simply iterate through the number of access points.

For the random method, the number of access points to be created is generated first with a uniform distribution, and then one possible position vector is constructed randomly.

The module *Topology Generation* then receives the value $n$ and computes the vector corresponding to it. After that the links between sensor nodes and the access points are created in the model. Every access point gets links to each sensor node located within a radius of 8 meters, based on previous measurements of radio link quality in our environment. More distant nodes are outside the transmission range of the sending node and thus cannot receive data. Finally it is checked whether an architecture is valid, which means that all sensor nodes must be connected to at least one access point. For valid architectures, the simulation is started, otherwise a new architecture has to be determined by the heuristic.

*3) Optimization Function:* Now the optimization goal and the corresponding calculation rule have to be defined in the module *Objective Function Calculation*. For our example the objective is to minimize the number of access points taking into consideration the following constraints. The maximum load of an access point is considered to be ideal at around 80 percent. Furthermore, the load of all existing access points should be similar to balance system load and avoid bottlenecks. For these conditions on the optimization objective the following objective function was defined:

$$f(n) = \frac{\prod_{i=1}^{c} g(load_i^{max})}{stdDev(load_i^{avg})} \quad \text{, where}$$

$$n = \text{value of the position vector } v$$
$$c = \text{number of access points}$$
$$g(.) = \text{Gaussian function with } \sigma = 1.0 \text{ and } \mu = 0.8$$
$$load_i^{max} = \text{maximum load of access point } i$$
$$stdDev(.) = \text{standard deviation}$$
$$load_i^{avg} = \text{average load of access point } i$$

where the value of the function $f$ should be maximized. The variable $c$ represents the number of existing access points. Function $g$ is a Gaussian function with variance 1.0 and an expected value of 0.8. $load_i^{max}$ is the maximum load of the $i$-th access point, and $load_i^{avg}$ its average for each $i$. The values for the Gaussian function $f$ result from the above condition on the maximum load. The function $stdDev(load_i^{avg})$ calculates the standard deviation of the average loads of all access points for a penalty on unbalanced solutions.

## IV. Results and Evaluation

The presented method of the architecture optimization of Section III-B was carried out using the model from Section III-A. This section presents and discusses the obtained results.

Technically, the simulations were run in our group's server cluster, dedicating two parallel virtual machines to each of the two (pseudo-) heuristic methods (brute force and random).

### A. Full Exploration of the Design Space for 3 Access Points

Figure 8 shows the results of the first experiment (full exploration for three access points). On the horizontal axis, the bit vector value $v$ is plotted, encoding the used positions of the access nodes. The value range of the $x$ axis is in the interval $(2^{52}, 2^{72})$. This is due to the validation condition for the architecture, the positions of the sensor nodes, and the calculation of the bit vector value: For example, to have radio access to the righmost sensor, an access point needs to be placed between the positions 53 and 72. Thus, the smallest possible value for the bit vector of a valid architecture configuration must be greater than $2^{52}$.

The vertical axis shows the number of used access nodes. Each color on the graph reflects the value of the function $f$ from the previous section for a valid system configuration which was simulated (red = high values, blue = low values). A total of 59 640 architecture configurations were investigated, of which only 3 268 were valid and thus actually simulated. The time effort to compute these values was approximately one week. The results are visualized in Figure 8. The values between the valid sampling points are interpolated linearly to improve readability, even though the design space actually does not completely fill the area.

Figure 8 shows that there are several small local peaks located in the bottom area. The global maximum is located in the bottom left area and has a value of $f() = 23.32$, which

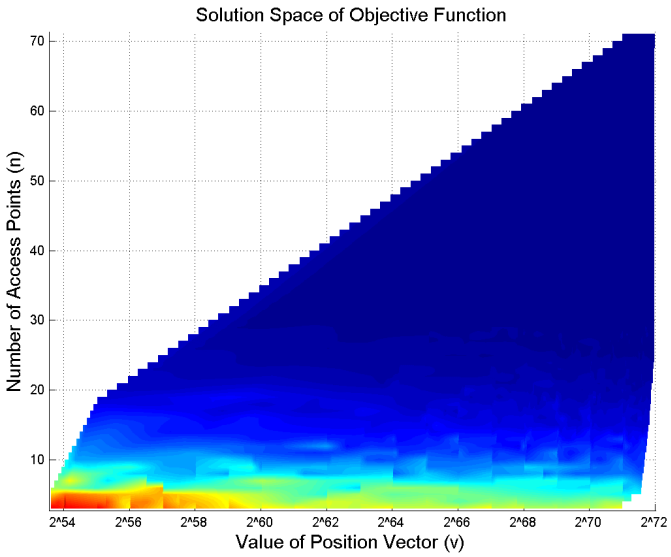


Fig. 8. Result of Design Space Exploration

is reached with an architecture configuration containing three access points at positions 14, 41 and 54.

Whether this is the actual maximum value for any number of access points, cannot be said with certainty based only on this experiment. This is because not all possible configurations are investigated.

### B. Random Exploration of the Full Design Space

The random method was run in parallel to the first method, and stopped after approximately six weeks of run time. It generated 3 053 configurations, out of which 2 818 were valid and simulated. The significant difference in the time effort compared to the first experiment is due to the run time of the modules *Simulation* and *Objective Function Calculation* for the current model implementation. Their computation delay was between five minutes and two hours depending on the configuration. The reason is that for configurations with more access points, there are more sensor nodes within radio link reach of several access points, leading to many more packet receptions for the same number of generated packets. For example, the simulation of a configuration with three access points may lead to approximately 22 000 events/samples. A simulation with 72 access points, however, may generate up to 300 000 events. Generation, storage, and analysis of this data leads to a significant difference in the run time of the modules. Thus for the first experiment with a restriction to only three access points, the average simulation time is much shorter.

Figure 9 shows the optimization function results for the investigated architecture configurations, including both the results for the full (green crosses) and the random method (red dots). The $x$ axis depicts the used positions of the access points encoded in the $v$ vector, while the $y$ axis shows the number of access nodes in the configuration. The $z$ axis depicts the value of the objective function to be maximized. The results

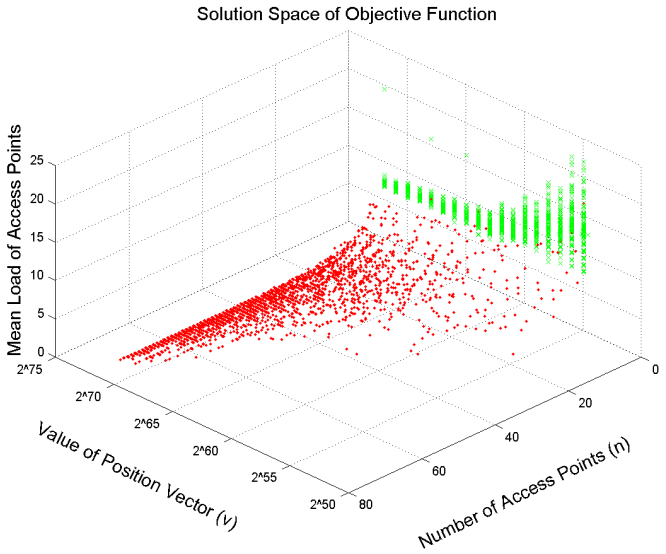


Fig. 9. Distribution of the results of both optimization methods

from the brute force method are distributed as expected: The first valid result can be found with three access points. All possible configuration with three access points are generated and analyzed systematically. It can be clearly seen that the optimization strategies only found a small number of values of the whole design space in the vertical plane $n = 3$.

It should be noted that the number of examined configurations per each number of access points is uniformly distributed. For the valid configurations, this is however not the case: Here, the number of valid configurations increases with increasing number of access points, because invalid configurations are more likely if only a few access points can be placed. With only a few access points positioned randomly there is a high chance that at least one sensor node is not within the range of any access point.

The brute force optimization found the global optimum with an objective function value of 23.23 for an architecture configuration using three access points and the positions 14, 41 and 54. The optimum in the result set of the random method is also an architecture configuration with three access points, with very close positions 12, 39 and 54. This validates the expected result nicely. The objective function value is 17.91.

It may be noted that dots representing results appear more often in the upper-left area of Figure 9 (i.e., large values of $v$). This is expected and justified by the position vector $v$ encoding. The magnitude of the value is determined by the highest index of an access point location. The positions of the access points are selected randomly. If a large number of access points is selected, the probability is high that there is at the least one among them with a high index value, and thus the numerical value of the bit vector is large. In addition, the minimum $v$ value for an increasing number of access points is increasing as well. Thus, there are fewer valid configurations for smaller access point numbers and vice versa.

In conclusion our confidence is high that the partial brute force method has found the actual optimum. The random method has not found the exact optimum in the same run-time, but came very close, and provided a good overview of the solution space in addition. The results support the expectation that the maximum found is also the global maximum.

The goal of this work, however, was not to find a good optimization method, but to demonstrate the functionality of the method for system architecture optimization with run-time reconfiguration of simulation models. This has been shown with the example.

## V. Conclusion

We have investigated the automatic derivation and evaluation of architecture versions using simulations with the MLDesigner modeling and simulation framework. The methodology is proposed to find the optimal design of aerospace wireless sensor networks based on an evaluation of simulation results.

The paper presented a method for the model-based optimization of architectures with the use of dynamic instances. Based on generalized sub models as well as their external configuration description it is possible to simulate different configuration variants. The optimization process itself is also modeled and executed by simulating it, thus allowing to configure and change the methodology without programming changes in a tool. We showed example numerical results for the optimization of a wireless sensor system in an aircraft. In the future we will investigate optimization methods suitable for system architectures in wireless sensor networks.

## Acknowledgment

This paper is based on work funded by the Federal Ministry for Education and Research of Germany with grant number 01S13031A.

## References

[1] L. Liberti and N. Maculan, *Global Optimization: From Theory to Implementation*. Springer Verlag, 2006.

[2] M. C. Fu, "A tutorial overview of optimization via discrete-event simulation," in *11th Int. Conf. on Analysis and Optimization of Systems*, ser. Lecture Notes in Control and Information Sciences, G. Cohen and J.-P. Quadrat, Eds., vol. 199. Sophia-Antipolis: Springer-Verlag, 1994, pp. 409–418.

[3] Y. Carson and A. Maria, "Simulation optimization: Methods and applications," in *Proc. of the 29th Winter Simulation Conference*, ser. WSC '97, 1997, pp. 118–126.

[4] M. Sghairi, A. de Bonneval, Y. Crouzet, J.-J. Aubert, and P. Brot, "Architecture optimization based on incremental approach for airplane digital distributed flight control system," *Advances in Electrical and Electronics Engineering - IAENG Special Edition of the World Congress on Engineering and Computer Science*, 2008.

[5] M. Azami, M. Ranjbar, A. S. Rostami, and A. J. Amiri, "Increasing the network life time by simulated annealing algorithm in WSN with point coverage," *Int. J. of Ad Hoc, Sensor & Ubiquitous Computing*, vol. 4, no. 2, 2013.

[6] C. R. Reeves, *Modern Heuristic Techniques for Combinatorial Problems*. John Wiley & Sons Inc, 1993.

[7] S. H. Chagas, J. B. Martins, and L. L. de Oliveira, "Genetic algorithms and simulated annealing optimization methods in wireless sensor networks localization using artificial neural networks," *IEEE 55th International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2012.

[8] Y. Huang, S. Zheng, P. Zhani, and X. Qinl, "Adaptive genetic simulated annealing algorithm for WSNs," *2nd International Conference on Computer Engineering and Technology (ICCET)*, 2010.

[9] R. C. Abreu, J. E. C. Arroyo, A. G. dos Santos, and V. de Oliveira Matos, "Evolutionary optimization algorithms for topology control in wireless sensor networks," in *Proc. of the 2012 World Congress in Computer Science, Computer Engineering, and Applied Computing*, 2012.

[10] C. van Leeuwen, J. de Gier, J. O. de Filho, and Z. Papp, "Model-based architecture optimization for self-adaptive networked signal processing systems," in *SASO 2014 - 8th IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, 2014.

[11] MLDesign Technologies, "MLDesigner." [Online]. Available: http://www.mldesigner.com

[12] B. P. Zeigler, T. G. Kim, and H. Praehofer, *Theory of Modeling and Simulation*, 2nd ed. Orlando, FL, USA: Academic Press, Inc., 2000.

[13] S. Jäger, T. Jungebloud, R. Maschotta, and A. Zimmermann, "Model-based QoS evaluation and validation for embedded wireless sensor networks," *Systems Journal, IEEE (accepted for publication)*, 2014.

[14] J. Zheng and A. Jamalipour, *Wireless Sensor Networks: A Networking Perspective*. John Wiley & Sons, 2009.

[15] L. Girod, T. Stathopoulos, N. Ramanathan, J. Elson, E. Osterweil, T. Schoellhammer, and D. Estrin, "A system for simulation, emulation, and deployment of heterogeneous sensor networks," in *In Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems*. ACM Press, 2004, pp. 201–213.

[16] S. Jäger, T. Jungebloud, R. Maschotta, and A. Zimmermann, "Model-based QoS evaluation for embedded wireless sensor networks," in *Proc. IEEE International Systems Conference (SYSCON'13)*, 2013.