10th Int. IEEE Systems Conference (SysCon 2016), April 2016, Orlando, Florida

Specification and Execution of System Optimization Processes with UML Activity Diagrams

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*—Designing complex systems requires domain knowledge as well as tool-supported modeling and analysis techniques. General-purpose as well as domain-specific tools can be used for this task. The latter has the advantage of not requiring low-level model knowledge from systems designers that are domain experts, but is only possible with specialized tools that have to be programmed for a certain purpose or field by software engineers. The gap between general-purpose tools and domain-specific applications can be bridged by a (meta-)model-based description of structure and behavior of domain objects and the subsequent generation of a software tool. This approach has been successfully demonstrated in earlier work and the result is termed *simulation-based application* (SBA).**

One of the main applications of such tools is to find the best solution for design decisions, which can be done by manual evaluations of design ideas or automatically if parameters and design space are well understood. Such an automatic indirect optimization method should be adapted to the specific domain, which would require programming effort for the SBA. A logical extension to the usual *system description* is to apply the model-based paradigm to such a *method description* as well.

This paper proposes an approach to model optimization processes (i.e., heuristics) graphically with UML activity diagrams that describe the data and control flow of such an algorithm. The resulting models are transformed into an executable algorithm automatically and control the work flow of an optimization with the software tool that has been designed for this task. An example of a heuristic optimization process for a wireless sensor network setup is presented.

***Index Terms*—system modeling, system optimization, heuristic optimization, activity diagram, UML, fUML, model-to-text transformation**

I. Introduction

System modeling includes the model-based description of structure and behavior of a system. During system creation, a large number of decisions is necessary with the goal of obtaining an optimal system. This variability of systems engineering solutions (the design space [1]) can be expressed in the system model by parameters which may be configurable from outside the model. Simulation-based applications (SBA, [2], [3]) allow domain experts or system designers, which are often not modeling experts, to work on design decisions without having to deal with a detailed model of the system specified in a general-purpose model class. A concrete system model is then constructed by an SBA automatically based on a predefined object-oriented domain description of structure and behavior of typical elements, and by using parameters specified by the system designer. Such parameters act then as the interface between the otherwise black-box model and the user.

Based on the number of parameters and their possible values, the number of model configurations (i.e., the size of the design space) can be very high. A simple approach is to search for a good system configuration by trial and error or based on experience and design ideas, which is manually executed by the domain expert. This approach may help to understand the system better, but is too cumbersome and time-consuming for a real search for an optimal solution.

Another possibility is to find the solution using a closed-loop approach, i.e., by a fully automatic optimization algorithm [4]. Because of the complexity of the underlying system behavior, a direct optimization such as for linear programming problems is not possible, but an indirect optimization has to be applied [5]. In order to find the optimal system configuration, heuristic techniques of system optimization can be used, which is an important and widely covered research area [6], [7], [8].

In [9] we have proposed a closed-loop indirect optimization approach, where possible configurations are determined by heuristic optimization methods and evaluated by simulating the model with the configurations iteratively. Figure 1 depicts the general workflow of such an approach. The optimization process starts with an initial run of a heuristic optimization method contained in the action *run heuristic optimization*. Here, a first configuration (parameter setting) is determined and sent to a topology generator. This component reconfigures the system model based on the received configuration and validates the configuration against existing system model constraints. The performance of the system model (if it passed the test) is evaluated by a simulation tool afterwards. A predefined objective function is applied to the simulation results. Finally, the optimization method is activated again with the result of the simulation run. If the termination criterion is satisfied, the loop is stopped and the configuration with the best value is

returned as the overall result. Otherwise, a new configuration is calculated and the optimization loop is executed again.

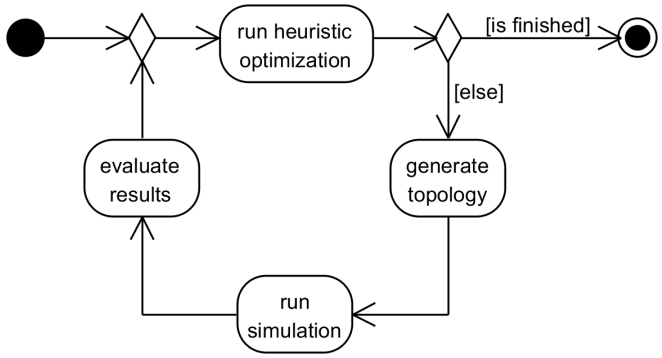

Fig. 1. Top level optimization loop [9]

This optimization process could be realized by using the system modeling tool, using an SBA, or based on a separate optimization tool. A separate tool would require to develop a specialized tool for each optimization purpose.

Using the system modeling and simulation tool to model the optimization process has the advantage that the modeling expert just has to work with one tool. However, from the point of view of the system engineers it is more difficult, because the domain expert is usually not a modeling expert and thus does not have a deep knowledge of the used system modeling and simulation tool. Therefore, it is a difficult task for a domain expert to configure or change an optimization model in a system modeling and simulation tool.

The Simulation Based Application is a specialized tool for domain experts to configure and control simulations of system models. Hence, it is possible to extend these tools with the functions to optimize parameters and system structure by using the underlying system model. The used SBA follows a plug-in approach to allow a simple implementation of plug-in extensions, such as graphical editors to configure domain specific models, or to start a simulation tool for model execution and analyzing simulation results [2], [3].

There are several possible approaches to implement an optimization heuristic for an SBA. One possibility is the classic implementation by using a standard programming language. In previous work, this has already been done in SBA tool in the programming language C++, and is described at the beginning of Section II. This implementation is the basis for a model-based approach similar to existing tools used in many applications. As an example, in [10] an MDD approach is used to optimize non-functional constraints of wireless sensor networks based on structural system descriptions.

To model an optimization process completely, both the structure and behavior have to be described. The Eclipse Modeling Project (EMP) can be used for model-based development of domain-specific applications. In [11] a special Story-Diagram, which is an enhancement on an Activity diagram, is used to model the behavior of UML class diagrams.

The Object Management Group (OMG) defines the FUML (Semantics Of A Foundational Subset For Executable UML Models, [12]) to realize models with executable behavior. The authors of [13] present a special action language for FUML Activity diagrams. In another related work, the Action Language for Foundational UML (ALF) [14] is used to describe the behavior inside FUML models of cyber-physical systems [15].

In this paper standard UML class diagrams and activity diagrams [16] are used to model the structure and behavior of optimization processes of a system, which should be integrated into the existing C++-based Simulation Based Application. A C++ representation of the models are generated based on these models using a UML4CPP generator which presented in [17]. These classes can be used to execute the defined optimization process by using a C++ FUML conform execution engine, which is defined in a model-based way and automatically generated too [18], [19].

The aim of this approach is a complete model-based specification of executable optimization processes of system models by using domain-specific languages of the domain expert only. As a first step, this paper describes the whole workflow to model and execute activity diagrams of system optimization processes in Section II. An activity diagram is used because it is a major standard description of software and system behaviors [16]. To realize this approach, the Eclipse modeling project and the new Sirius project is used which enable a more effective realization of domain-specific languages than other approaches [20], [21]. Moreover, opposed to existing methods we use a code generator which generates executable C++ code.

An example for model-based optimization of systems is shown in Section III, covering the optimization of a wireless sensor network example [3], [9]. Further steps should provide executable domain-specific languages [22].

II. Methodology

This section describes the proposed steps to derive a model-based, executable optimization process based on an existing implementation.

The structure of a C++ implementation for general optimization processes is described first. This implementation is used as a basic implementation for model-based optimization processes. Therefore, the implementation is moved into the class diagram to provide the code generation based on these class diagrams. The implementation of the general optimization loop is replaced by a model-based and executable description of the same behavior, which is described in Subsection II-C. This step could be repeated to refine the implemented behavior by using executable models of the same behavior. The model-based description of optimization heuristic will be done in further developments as an example. The aim of this methodology is the complete model-based and executable specification of optimization processes. A key technology for this methodology is a UML4CPP code generation which is briefly described in Subsection II-D.

A. Foundation of system optimization

Figure 2 presents the class diagram of an implemented optimization plug-in. The main class is *OptimizationPlugin*, the plug-in functionality for registering this plug-in in the SBA. Furthermore, this class contains a function called *optimize(...)* implementing the top-level functionality of an optimization approach presented in Figure 1. This function has an input object *OptimizationObject* used for data exchange between the following classes. In particular this object includes a current configuration as well as the evaluation result and information about optimized state for the configuration. Additionally, *optimize* has access to the following class instances implementing several parts of the optimization loop.

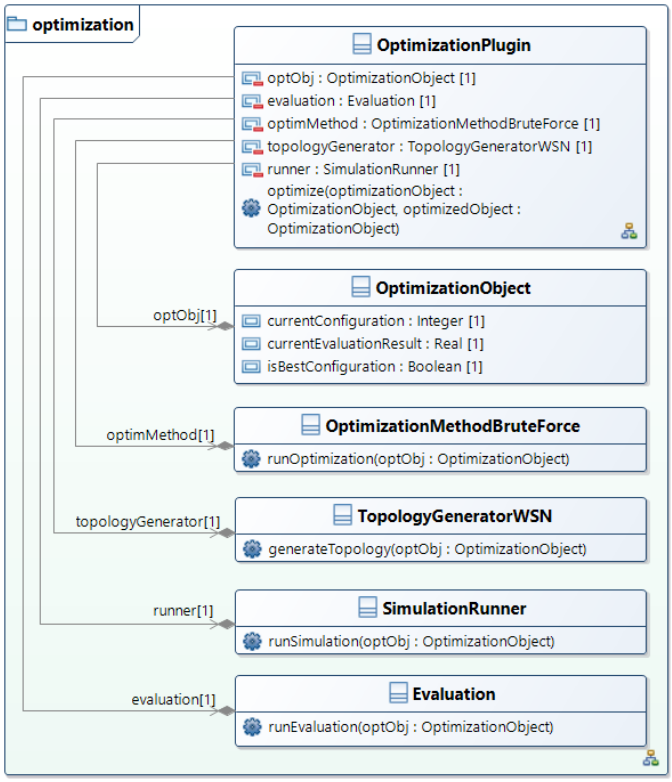

Fig. 2. Classes of implemented optimization plug-in

The class *OptimizationMethodBruteForce* implements the behavior of a brute force algorithm and searches for the best configuration by evaluating all possible configurations incrementally. The class *TopologyGeneratorWSN* configures a wireless sensor model which is already described in [3] and is also used for optimization in [9]. Furthermore, a validation of the configured model is executed in this class. The valid model is simulated in class *SimulationRunner* afterwards. Invalid models are not simulated but passed without optimization affecting results. In the last class called *evaluation* an objective function is calculated based on current simulation results, which is used to evaluate the configuration. The objective function depends on the optimization goal to be defined before each optimization.

After starting the execution of *optimize(..)*, the function *runOptimization(..)* of class *OptimizationMethodBruteForce* is running and receives an *OptimizationObject* with information about current configuration and about termination constraints of the used heuristic optimization method. If the termination constraint is fulfilled, the function is finished and the result object is returned. Otherwise, *generateTopology(..)* of class *TopologyGeneratorWSN* is executed. If the currently configured model is valid, the corresponding simulation is run using class *SimulationRunner*. An evaluation of the current configuration is started afterwards. The *runOptimization(..)* function is executed again at the end of the optimization loop.

B. Modeling the structure of system optimization

An indirect optimization approach [9] and the implemented realization of the SBA plug-in from Section I are analyzed. The following UML diagrams are useful for modeling system optimization. The class diagram is suitable for the structural description of the optimization approach. Here, the components including their attributes and behavior declarations can be modeled. Furthermore, the dependencies between several classes realizing parts of the optimization approach and the main class including the optimization loop execution are expressed.

To improve the replaceability of each optimization component, an interface layer is added, which includes important function declarations. Each class is derived by one interface and thus implements its behavior using predefined interface functions. The advantage is that classes derived from the same interface can be exchanged easily without modifications in other classes that use it. For instance, it is possible to exchange the optimization class easily.

The resulting class diagram is depicted in Figure 3. It is subdivided into an interface layer and a class layer.

The interfaces provides basic functions and property declarations, which are required for top-level optimization loop control. The interface *XOptimizationObject* serves as data transmission between optimization components and includes basic property access operations. In particular, the current configuration is contained as well as the evaluation result of it and an information if the configuration is the optimal one.

XOptimizationMethod is an interface from which each class implementing an optimization method should be derived. The interface includes the function *runOptimization(...)* with input parameter *XOptimizationObject*. The termination criterion is checked first. If this criterion is satisfied, the *isBestConfiguration(...)* property of the parameter has to be enabled. In the negative case, a new configuration has to be calculated and assigned as the outgoing object.

The interface *XTopologyGenerator* provides a function called *generateTopology(..)*, also using the object *XOptimizationObject* as inout parameter. The optimizing system model is configured based on the configuration contained in *XOptimizationObject* and validation constraints of the system model are checked.

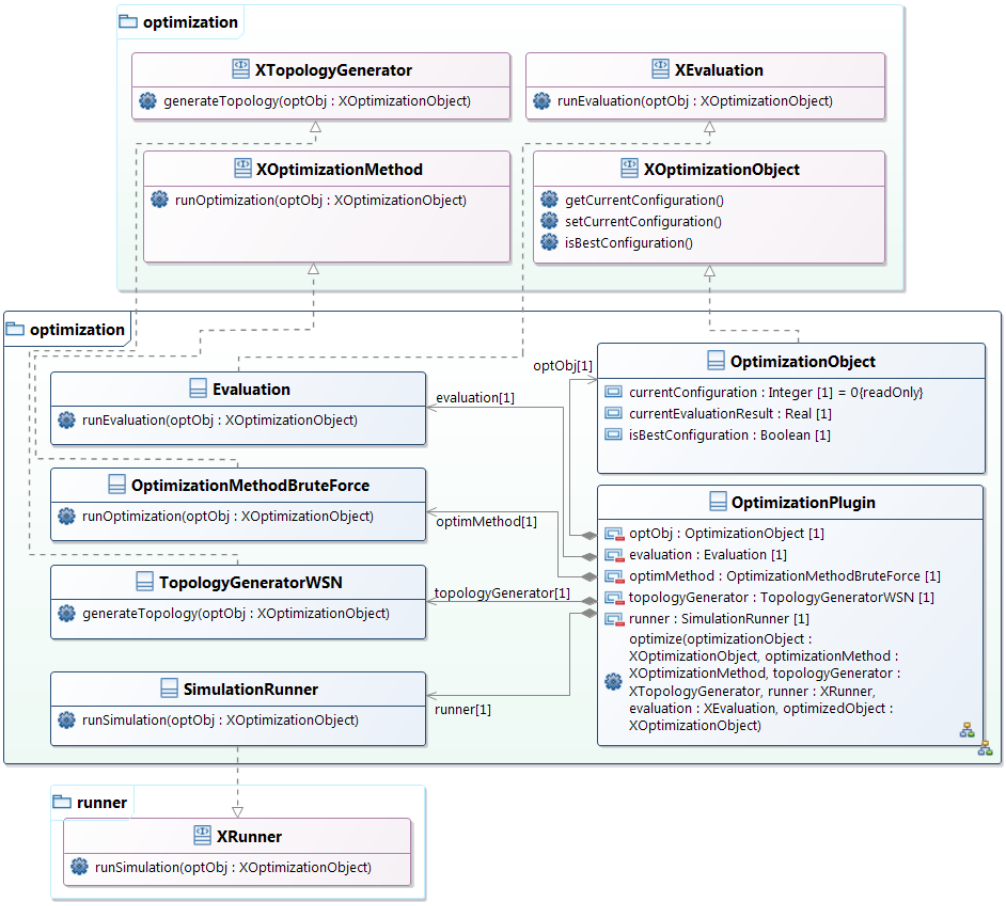

Fig. 3. Class diagram for optimization structure

Classes implementing interface *XRunner* provide the functionality to start external programs and wait until their execution is finished. The last interface is called *XEvaluation* and provides the valuation of current configuration based on simulation results and a predefined objective function describing the optimization goal.

The package *optimization* includes all interfaces except interface *XRunner*. This interface is placed in package *runner*, because its functionality can also be used in the tool outside of the optimization domain.

The class package contains six classes. The main class of the optimization process is named *OptimizationPlugin* and includes the function *optimize(...)*, which is implementing the optimization process behavior. Starting with the execution of *runOptimization(...)*, the functions *generateTopology(...)*, *runSimulation(...)* and *runEvaluation(...)* are executed cyclically until the termination criterion is satisfied. Instances of the corresponding classes are required as input parameters. Furthermore, *optimize(...)* provides the mechanism for registering this process as plug-in in the SBA.

All other classes are derived from interfaces and are implemented in functions, whose interface has to be derived by at least one class. It is possible that an interface is derived by more than one class. For instance the class *OptimizationBruteForce* presented in Figure 3 implements the functions from interface *XOptimizationMethod* using the optimization method brute force. Another class derived from *XOptimizationMethod* may implement a simulated annealing method. The decision, which optimization method should be used for optimization, is determined by the parameter *optimizationMethod* of the function *optimize(...)*.

C. Modeling the behavior of system optimization

UML activity diagrams are proposed to model the steps of an optimization process. In Figure 4, the behavior of function *optimize(...)* is specified. The parameters of class function definitions are depicted as so-called activity parameters and presented in the left and right side. They serve as input and output of objects. Actions, control nodes and flows are used to describe behavior in activity diagrams.

The activity diagram supports different types of actions. *call operation action* references to a class function and executes behavior assigned to the referenced function. It contains input and output pins corresponding to the parameters of the called class function. Additionally, there is a further pin connected with an activity parameter, which includes the instance of called class function.

Another action is named *call behavior action* and references

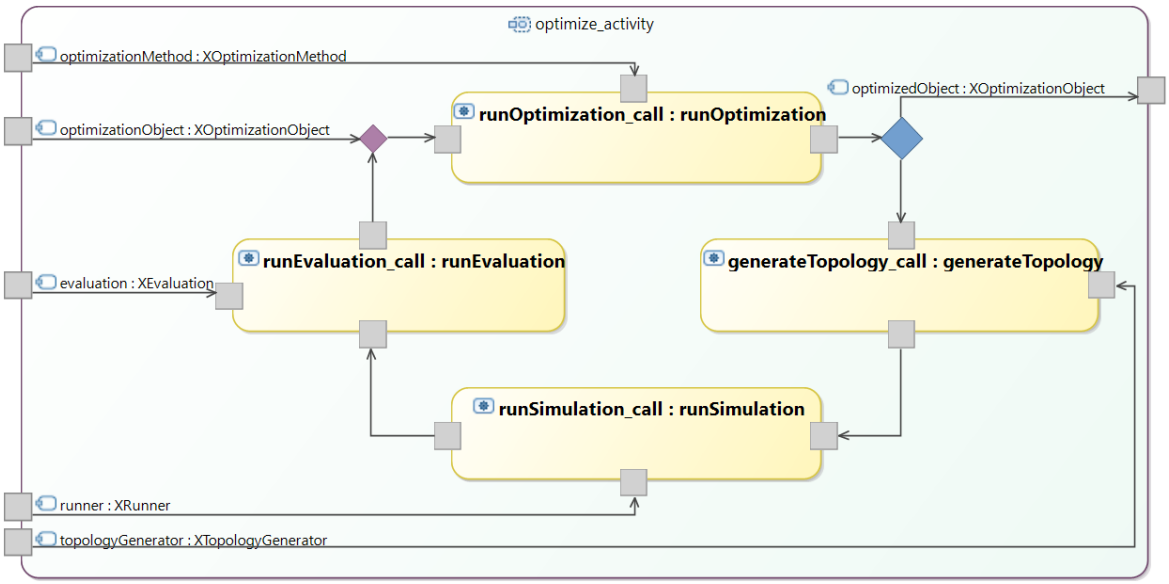

Fig. 4. Central optimization behavior modeled with UML activity diagram

an already existing behavior model in the entire model. This can be any UML element derived from a behavioral element like actions inside the activity diagram or other behavior diagrams such as state chart diagrams. Thus, the difference to the first action is, that a behavior element is referenced. In contrast, the *call operation action* is a reference to a structural operation element.

Flows could be control flow or data flow. Control flows are used to define the order of behavior executions. On the other hand, data flows are required to model data streams. They are connected with actions using pins, whereby control flows are connected to actions directly.

Control nodes include decide- and merge-nodes as well as fork- and join-nodes. They are used to influence the flows between actions. On decide nodes, the flow has the choice to take exactly one outgoing flow depending on the guards, which are assigned to outgoing flows. The flow of a confirmed guard is used. On the contrary, fork nodes duplicate the flow and use all outgoing flows parallel. The merge node combines different flows to one flow whereby just one ingoing flow has to be used. In contrast, the join node synchronizes all incoming flows to one outgoing flow.

Figure 4 presents the modeled behavior of the class function *optimize(...)*. Here, *call operation actions* are used to model the behavior, because it should be referenced to structurally defined functions from Section II-B. Data flows are used in order to model the data stream.

The first executed action is called *runOptimization_call* and is referenced to function *runOptimization(...)* of interface *XOptimizationMethod*. This action has two input pins and one output bin. The top positioned input pin is reserved for an *XOptimizationMethod* instance. An *XOptimizationObject* object is received on the other input pin. The applied optimization method behavior is deposited - checking if the termination criterion is satisfied and in negative case calculating a new configuration to evaluate. The *XOptimizationObject* object is published on the outpin pin on the right side afterwards. This pin is connected with a decide element. Here, the decision is done depending on the termination state of the data object *XOptimizationObject*. If this property is true, the top edge connected with the activity parameter *optimizedObject* is chosen. Otherwise, the bottom flow is used. This flow is connected with the *call operation action generateToplogy_call* references the behavior assigned to *XTopologyGenerator::generateTopology()*. The system model is reconfigured based on the configuration, which is received from *runOptimization_call*. Reconfigured models are simulation using corresponding simulation tools, which are started by the action *runSimulation_call*. The last action is calculating the objective function using simulation results. Finally, the first action is executed again.

In the current state of our work, the behavior of actions is defined by using implemented code fragment, which are added to the corresponding actions. In general, the hierarchical refinement of the behavior modeling can be continued until basic behaviors can be used and refinements not helpful any more. However, basic behavior is described by using code fragments of the target programming language.

D. Realization of executable models

Based on UML class diagram and activity diagram a model for system optimization is realized in Sections II-B and II-C. To make these models executable, a generator is required to transform UML models to executable code. However, the executable model should run within the Simulation Based Application written in C++. Thus, a UML4CPP generator [17], [23] and fUML generator [18] are used.

The UML4CPP generator supports the building of C++ source and header files as well as compiling instruction file.

The general building process works is described as follows: An interface file and implementation files are created for each modeled class. The advantage of separation implementation and interface description is well known. The interfaces define classes inclusive a declaration of derived class as well as all properties and functions provided by this class. The implementation files are composed by one header file and one source file implementing the corresponding interface file.

Additionally, the generator provides the creation of a factory class and a package class. Package classes providing access to the meta information of the classes during run time. Corresponding to the factory pattern, the creation of instances are supported by factory classes.

Regarding the activity diagram, the UML4CPP generator inserts the meta information of activities to the package class. Actions consisted by activities are generates as an executable class, which represents a single behavior as a function. The order of action execution inside an activity is already defined by the meta information inserted to the package class. The assignment of this functions to the class functions is still missing, which is done by the fUML generator. According to defined actions the generator creates links between class function calls and the behavior function execution.

Finally, the generated files are compilable using the generated instruction files. The executable model is then ready for simulation by the domain expert.

III. Example

The presented method of modeling executable activity diagrams from Section II was carried out using the optimization problem from [9]. The system which has to be optimized is presented in Figure 5. It represents a wireless sensor network developed for avionic applications. This systems consists of sensor nodes, a shared medium, access points and a control server. The sensor nodes produce data and send them to access points using a shared medium. The access points receive this data and forward the data to a control server using a wired medium. The detailed description of the system is contained in earlier papers ([9], [24]) and will not be repeated here.

The goal of the optimization problem is to minimize the number of access points under the following constraints. The load of all access points should be balanced and considered to be ideal at 80 percent utilization in order to avoid bottlenecks. Additionally, the transmission range of the sensor nodes is limited to 8 meters. To ensure that all data can be transmitted from the sensor nodes to the control server, the access points should be placed in range of the sensor nodes. Based on the optimization goal and constraints an objective function is used which is described in detail in [9].

Number and positions of access points can be varied as a design parameter, under the restriction that the possible positions are predefined and limited to 11 here. The number of used access points and their positions is encoded in a bit vector consisting of entries numbered from 0 to 10. Each entry represents a position where an access point could be placed,

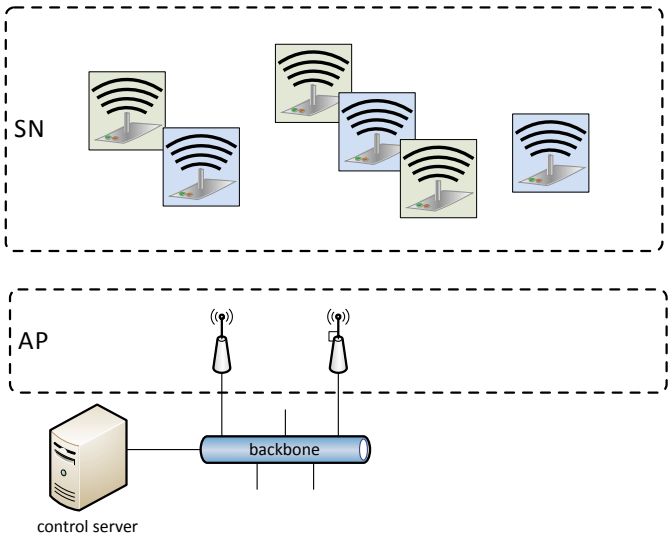

Fig. 5. Top-level view of WSN model [24]

and value 1 means that an access point is placed (similar to a bit vector). Otherwise, the position is not used.

This optimization process is modeled and generated by using the method described in Section II. The resulting executable optimization plug-in was executed on a server cluster consisting of four virtual machines. Each machine executes the full optimization process. The results of each configuration are averaged over the results to avoid statistical problems.

The design space for this optimization problem consists of 2 048 configurations. Because each sensor has to be in range of at least one access node, 1 179 out of the 2 048 configurations are invalid. Thus, 869 valid configuration are simulated and evaluated.

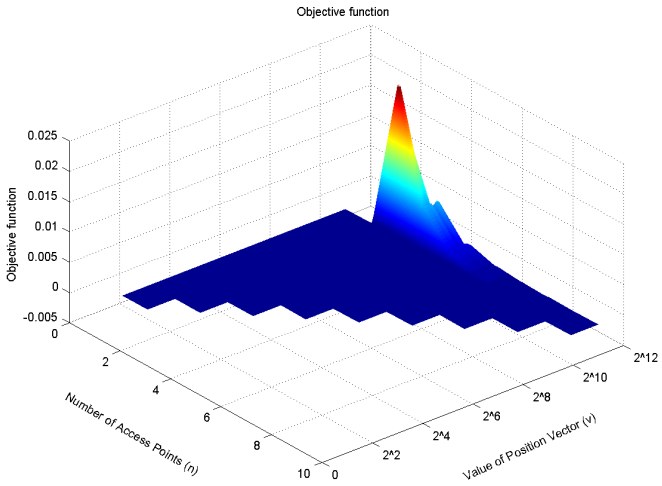

Fig. 6. Result of design space exploration

The results are shown in Figure 6. The number of used access points is shown on the x axis. The y axis depicts the vector encoding the position used by the access points. The corresponding values of the objective function are presented

on the z axis. Additionally, the objective functions values are colored: blue represents low values and red depicts high values.

The results are distributed as expected. All configurations are calculated incrementally, while only the valid configurations are simulated and evaluated. The optimal system configuration uses three access points at positions 1, 5 and 10. The best value of the objective function is 0.0249.

It may seem strange at first that the valid configurations appear in bit vector values from (2^{10}) till (2^{11}). This is expected because the positions of the sensor nodes and the possible positions for the access points are configured in the way, that position 10 has to be used for connecting all sensor nodes with at least one access point.

In conclusion the brute force optimization has obviously found the global optimum. The result will be used to compare it with real heuristics that optimize the model with fewer simulations, but a lower probability of finding the optimum. Finding a good optimization method was not the goal of this work, but to demonstrate the workflow of modeling executable activity diagram.

IV. Conclusion

This paper presented an approach of model-based specification of executable system optimization processes based on activity diagrams. A workflow for modeling and executing activity diagrams is shown using a UML4CPP generator and C++ fUML-conform execution engine. Thus, it allows to create executable tools based on modeled behavior without implementation work. The results of executed activity diagram describing the optimization of an avionic application example is shown.

Future steps include the model-based refinement of the presented optimization behavior. Example objective functions and a set of heuristic optimization processes will be defined using executable models.

Acknowledgment

This paper is based on work funded by the Federal Ministry for Economic Affairs and Energy of Germany under grant number 20K1306D.

References

[1] T. Taghavi and A. Pimentel, "Visualization of multi-objective design space exploration for embedded systems," in *Digital System Design: Architectures, Methods and Tools (DSD), 2010 13th Euromicro Conference on*, Sept 2010, pp. 11–20.

[2] R. Maschotta, S. Jäger, T. Jungebloud, and A. Zimmermann, "A framework for agile development of simulation-based system design tools," in *Proc. IEEE International Systems Conference (SYSCON'13)*, 2013.

[3] 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.

[4] 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.

[5] J. Sobieszczanski-Sobieski, A. Morris, and M. van Tooren, *Multidisciplinary Design Optimization Supported by Knowledge Based Engineering*. John Wiley, 2015.

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

[7] 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.

[8] 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.

[9] A. Wichmann, S. Jäger, T. Jungebloud, R. Maschotta, and A. Zimmermann, "System architecture optimization with runtime reconfiguration of simulation models," in *IEEE International Systems Conference (SYSCON15)*, 2015.

[10] N. X. Thang and K. Geihs, "Model-driven development with optimization of non-functional constraints in sensor network," in *Proceedings of the 2010 ICSE Workshop on Software Engineering for Sensor Network Applications*, ser. SESENA '10. New York, NY, USA: ACM, 2010, pp. 61–65. [Online]. Available: http://doi.acm.org/10.1145/1809111.1809128

[11] H. Giese, S. Hildebrandt, and A. Seibel, "Improved flexibility and scalability by interpreting story diagrams," in *Proceedings of the Eighth International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2009)*, T. Magaria, J. Padberg, and G. Taentzer, Eds., vol. 18. Electronic Communications of the EASST, 0 2009.

[12] OMG, "Semantics of a Foundational Subset for Executable UML Models," Object Management Group, Tech. Rep., 2013.

[13] C.-L. Lazar, I. Lazar, B. Parv, S. Motogna, and I.-G. Czibula, "Tool Support for fUML Models," in *Int. J. of Computers, Communications & Control*, 2010.

[14] OMG, "Concrete Syntax for a UML Action Language: Action Language for Foundational UML," Object Management Group, Tech. Rep., 2013.

[15] A. Gerlinger Romero, K. Schneider, and M. Goncalves Vieira Ferreira, "Towards the applicability of alf to model cyber-physical systems," in *Computer Science and Information Systems (FedCSIS), 2013 Federated Conference on*, Sept 2013, pp. 1427–1434.

[16] OMG, "Unified Modeling Language TM (OMG UML), Version 2.5," Object Management Group, Tech. Rep., 2013. [Online]. Available: http://www.omg.org/spec/UML/2.5/Beta2/PDF

[17] Sven Jäeger, Ralph Maschotta, Tino Jungebloud, Alexander Wichmann, and Armin Zimmermann, "An EMF-like UML Generator for C++," 2016, submitted.

[18] Ramyashree, "Model Driven Development of fUML based Execution Engine," Master's thesis, Technische Universität Ilmenau, 2015.

[19] F. Bedini, "Design and implementation of executable uml activity diagrams for c++," 2016, unpublished.

[20] Eclipse, "Sirius," 2014. [Online]. Available: http://wiki.eclipse.org/Sirius

[21] V. Viyovic, M. Maksimovic, and B. Perisic, "Sirius: A rapid development of dsm graphical editor," in *Intelligent Engineering Systems (INES), 2014 18th International Conference on*, July 2014, pp. 233–238.

[22] S. Jäger, R. Maschotta, T. Jungebloud, A. Wichmann, and A. Zimmermann, "Model-driven development of simulation-based system design tools," 2016, submitted.

[23] A. Chandrasekaran, "Model Driven Development for Design and Optimization of UML based Simulation Processes," Master's thesis, Technische Universität Ilmenau, 2015.

[24] 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.