Model-Based Architecture Optimization for Self-adaptive Networked Signal Processing Systems

C.J. van Leeuwen*, J.M. de Gier†, J.A. Oliveira de Filho‡ and Z. Papp§
TNO Technical Sciences
The Hague, The Netherlands
Email: coen.vanleeuwen@tno.nl*, jan.degier@tno.nl†, julio.oliveiradefilho@tno.nl‡ and zoltan.papp@tno.nl§

***Abstract*—This short paper introduces a closed-loop design optimization method for self-organizing and self-optimizing networked systems with a focus on signal processing and control. The design process starts with creating graph-based model of the system using a dedicated modeling language.The design is exported and converted to executable code in order to obtain the properties of the runtime behavior of the system using a simulation environment. The embedding optimization loop iteratively invokes the evaluation and searches for optimal architectures and parameterization in the user defined design space. A distinguishing feature of the tool is that it allows for runtime changes in the models, i.e. it is capable of evaluating runtime reconfigurable architectures. The design space is split into two disjunct sub-spaces: one of them defines the runtime reconfigurability (the self-* capabilities), the other defines the region of design time optimization. The tool is demonstrated via a real-time monitoring application.**

I. Introduction

Self-organizing and self-adaptive networked systems have to be realized in order to cope with changing operational conditions, changing user needs and other unforeseen influences. System architects and designers face difficult challenges when designing such runtime reconfigurable networked systems. In order to help designers to make informed decisions, a toolchain was developed in which a designer can model the architecture of the system, define its components' behavior, evaluate the system level performance, and optimize the design.

The proposed method uses model-based engineering [1] as the methodology to build and evaluate the system. The designer uses a Domain Specific Modeling Language (DSML) with primitives such as Tasks, Nodes and Connections to describe the system at hand. When the modeling is completed (on a certain level of fidelity), the designer can generate code compatible with the evaluation environment. Finally the designer defines the design space of the model which he wishes to optimize using the optimization tool. The approach will be discussed in more detail section II.

In this paper the methods are introduced and explained using an example. In the example a network of four sensor nodes monitor the temperature in a cargo container in order to determine the cargo's health state. The four nodes sample the environment with a certain frequency, process the samples for estimating the local state (temperature), and communicate with each other to refine the state estimation via state fusion. The example will be discussed in section III.

This short paper is accompanied by a video, which can be found via the following URL: http://youtu.be/ZP6q9J5wX4k.

II. Approach

A. Modeling Language

A DSML was developed for the toolchain providing both textual and visual representation[1]. Conceptually the modeling language is similar to system description languages such as UML/MARTE or SysML, but less extensive, yet having some unique features specifically for dynamic architectures: the modeling language has a set of special language primitives to describe adaptivity, required to model systems that are self-adaptive. The modeling concept and the underlying language are described in detail in [2]. In this language the system is defined using the following five different aspects:

- The functional model describes the tasks making up the data flow of the system. Tasks are connected through ports, and denote the concurrent activities in the system.
- The behavioral model (similar to the UML state diagram), is used to determine the sequential behavior of tasks, by indicating the time-consuming operational segments.
- The physical model specifies the nodes and communication channels in the system, and how they are connected.
- The node definition model specifies the nodes' hardware. This defines the available resources, albeit computational, communication or energy-wise.
- The mapping model describes what tasks are running on what nodes, and which communication devices are used for which ports.

B. Design evaluation

In order to derive the system level KPIs (Key Performance Indicators) of the design (e.g. expected lifetime, throughput, response time, etc.) the code generator tool delivers Matlab code compatible with the DynAA [2] simulation framework, which was built specifically for runtime reconfigurable networked systems. At the core of this framework is a discrete event simulator (implemented in Java), on top of which is a model layer implementing the language primitives that describes the entities such as Tasks, Connections and Nodes. The generated code builds on top of these existing components to build the user-specified model.

[1]The graphical modeler tool was developed using the MetaEdit+ metamodeling tool (http://www.metacase.com/mep/).

The DynAA simulation tool is unique in that every component may fail, disappear, reappear or be modified during simulation, the system itself can transform itself structurally (i.e. realizing self-* capabilities) and the simulator will cope with the changes accordingly. In other simulation tools a designer can trick the system into disabling or enabling certain components based on a switch mechanism, but this can become a unmanageable issue when dealing with self-organizing systems in which architectural changes are fundamental and frequent.

C. Optimization

The DynAA Design Space Exploration (DyDSE) tool is the playground for design-time optimization, and puts the simulation in an optimization loop. In each iteration, the user-defined model is parameterized, built, simulated and evaluated under user-defined scenarios. This process is iterated until the optimization algorithm finishes, and the resulting model is returned as the output. The DyDSE tool is written so that it can use user specified (e.g. case specific) optimizers, including those from the Matlab Optimization Toolbox such as the grid-point, interior-point, sequential quadratic programming or a genetic algorithm approach.

Besides the system model, the optimization tool also requires a design space definition which provides the tool with the exploration settings. This approach enables the fine tuning of the *runtime - design-time tradeoff*, which is a critical issue in real-time reconfigurable systems [3]. The design space definition contains the following information:

- A list of the parameters to change in the model, including their domain.
- The set of allowed structural transformations (e.g. task reallocation, activation of hardware resources, task instantiation, etc.).
- The loggers that will gather the KPIs from the simulation.
- A reference to the constraint function and the objective function. These are custom user-written functions that operate on the information from the loggers in order to determine what the value of a potential solution is.
- The optimization algorithm to use, and its settings.

III. Cargo Monitoring Example

A small real-life example was worked out to demonstrate the toolchain. In the functional model there are two types of tasks: sampling tasks (four instances) and signal processing tasks (amount will be determined by the optimizer). The outputs of the sampling tasks are connected to the input of the processing task, but what sampling tasks are connected to what processing tasks is to be decided by the optimizer. There are three possible structures: 1-to-1, all-to-all or 2-to-1 (any arbitrary connection matrix could be used, but the search space was limited to these options for pragmatic reasons). All processing tasks communicate with each other, so that information is always shared to obtain a state estimation based on all sensors.

TABLE I. Model parameters and their possible values

Parameter	**Values**			
Communication period	12	16	20	24
No. of processing tasks	1	2	3	4
Sampling to Processing	$1 \rightarrow 1$	$* \rightarrow *$	$2 \rightarrow 1$	
Processing to Node	$1 \rightarrow 1$	$* \rightarrow 1$	$2 \rightarrow 1$	

As defined in the behavioral model, the sampling task writes the sensor information to its output port, and then delays for 10 seconds (sampling time). The processing task will read data from the sampling task, and from other potential processing tasks and processes the input accordingly. The processing time is dependent on the amount of data read, and after calculation the result is communicated to the other processing tasks. The delay before repeating (communication period) is to be decided by the optimizer.

In the physical and mapping model there are four sensor nodes, communicating wirelessly with every other. There is one sampling task mapped to each sensor node, and they all use the same communication device. The allocation of the processing tasks to the sensor nodes is determined by the optimizer using different structural combinations (1-to-1, all-to-1 or 2-to-1). All model parameters are repeated in Table I.

The model performance, which is calculated by the objective function of the optimization loop, is a balance of the uncertainty of the state estimation and the total amount of energy used. The uncertainty is obtained using statistical models of the noise of the sensors, combined with the time since the last update, the combining of multiple measurements and the extrapolation distance. The total amount of power used is obtained from the simulation results.

After performing an exhaustive grid search over the different parameter values, the optimizer returns that the optimal solution will be using a communication period of 12 seconds, using two processing tasks. The processing tasks are mapped to two different nodes, and two sampling tasks communicate to one processing task.

IV. Conclusion

In this paper we introduced a design optimization toolchain for self-organizing networked signal processing systems. It features a graphical interface for modeling the system, a simulation environment for obtaining KPIs, and uses an optimization framework to optimize the design architecture and parameters. We have shown that our optimization framework is capable of searching through different designs and delivers non-trivial answers, which outperforms any of the alternatives.

References

[1] T. Saxena, A. Dubey, D. Balasubramanian, and G. Karsai, "Enabling self-management by using model-based design space exploration," in *Engineering of Autonomic and Autonomous Systems (EASe), 2010 Seventh IEEE International Conference and Workshops on*. IEEE, 2010, pp. 137–144.

[2] J. Oliveira de Filho, Z. Papp, R. Djapic, and J. Oostveen, "Model-based Design of Self-adapting Networked Signal Processing Systems," *2013 IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems*, pp. 41–50, Sep. 2013.

[3] E. Kang, E. Jackson, and W. Schulte, "An approach for effective design space exploration," *Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems*, pp. 33–54, 2011.