An Architectural Style for Optimizing System Qualities in Adaptive Embedded Systems using Multi-Objective Optimization

Arjan de Roo, Hasan Sözer, Mehmet Akşit
Software Engineering Group, University of Twente, Enschede, The Netherlands
{*roo,sozerh,aksit*}*@ewi.utwente.nl*

Abstract

Customers of today's complex embedded systems demand the optimization of multiple system qualities under varying operational conditions. To be able to influence the system qualities, the system must have parameters that can be adapted. Constraints may be defined on the value of these parameters. Optimizing multiple system qualities under the given set of parameters and constraints is called Multi-Objective Optimization (MOO). This is a well-known mathematical problem, for which numerous solutions have been proposed. The application of an MOO solution in an embedded system involves specific design decisions. It is preferable that these design decisions are documented in the architectural description. Therefore, this paper presents an architectural style, which specializes the Component-and-Connector viewtype, to enable the analysis and design of an architecture from an MOO point of view. A case study from industry is used to demonstrate the usage of this style.

1. Introduction

A current trend in embedded systems is towards optimal system qualities under varying circumstances, e.g. environmental conditions, user needs and input. For example, in the printing system domain customers demand more productivity while energy consumption must be minimized. A solution to obtain optimal qualities is first to make the system adaptable, so that the system qualities can be influenced. Second, use this adaptability to control the system to function optimally under the given circumstances. How to design and control adaptable systems is the goal of the Octopus project [1]. This project is a joint effort in a consortium of both industrial and academic partners: Océ-Technologies B.V. (one of the world's leading manufacturers of printer and copier systems), the Embedded Systems Institute and four Dutch universities.

To optimize the system qualities (objectives) of a system, the right value for the control parameters (decision variables) has to be chosen. Examples of parameters are the power given to a component and the speed of the system. The range of possible values is often subject to constraints. For example, a component needs a certain amount of power under a given speed. This is an optimization problem known as multi-objective optimization (MOO) [2]. The realization of MOO in embedded systems is usually distributed at several parts of the system, possibly implemented by different engineers. Therefore, this realization should be documented in the architectural description to support communication, analysis and verification, and to guide design, implementation and reuse. In this paper we introduce a style to document the software architecture from the MOO point of view and we report on our ongoing work. The next section gives an introduction in MOO. This is followed by a presentation of the style in section 3. Section 4 applies this style to an industrial case study. In section 5 future work is explained. Finally, related work is discussed and conclusions are provided.

2. Multi-Objective Optimization

Multi-objective optimization algorithms try to optimize a given set of objective functions for a given set of decision variables. This type of problem was first introduced in works on decision making in economy by Edgeworth [3] in the late nineteenth century. Pareto extended the work with the concept of Pareto optimality [4].

Definition 1 (Multi-Objective Optimization Problem [5]): A multi-objective optimization problem can be represented by the tuple $\langle \vec{u}(\vec{x}), \vec{g}(\vec{x}), \vec{h}(\vec{x}) \rangle$, where

- $\vec{x} \in \mathbb{R}^n$ represents the value of the n decision variables.
- $\vec{u}(\vec{x}) \in \mathbb{R}^n \rightarrow \mathbb{R}^o$ is a vector of o objective functions.
- $\vec{g}(\vec{x}) \in \mathbb{R}^n \rightarrow \mathbb{R}^p$ is a vector of p inequality constraints.
- $\vec{h}(\vec{x}) \in \mathbb{R}^n \rightarrow \mathbb{R}^q$ is a vector of q equality constraints[1].

The solution to this problem is the minimum of $\vec{u}(\vec{x})$ under the constraints $\vec{g}(\vec{x}) \leq 0$ and $\vec{h}(\vec{x}) = 0$

Note that, if there is more than one objective function, there might not be a single optimal solution for this problem. Instead, the solution is a set of Pareto optimal points [4].

1. In the rest of this paper we will use the identifier c to represent a set of constraints, containing both the equality as well as the inequality constraints.

3. Multi-Objective Optimization Style

This section describes an architectural style to document MOO in the architectural description. This style is called the Multi-Objective Optimization style (MO2 style). The MO2 style is a specialization of the Component-and-Connector (C&C) viewtype as described in [6]. The following gives a brief description of this style. The different *elements* are described in further detail in following subsections.

- *Elements*: adaptable component (AC), multi-objective optimization component (MOO-C), transformation component (TC);
- *Relations*: Same as in the C&C viewtype;
- *Properties of elements*:
  - *Adaptable component*: ACs provide information about objectives, related decision variables and constraints on the value of the decision variables;
  - *Multi-objective optimization component*: The MOO-C receives information about objectives, decision variables and constraints from ACs and provides the solution to this MOO problem;
  - *Transformation component*: The TCs apply a straightforward mathematical transformation to their input to provide a data output of a different type.
- *Properties of relations*: Same as in the C&C viewtype;
- *Topology*: Same as in the C&C viewtype.

3.1. Adaptable Components (AC)

In an embedded system, the architecture is usually decomposed into several components. Each of these components has certain control parameters, which make the component adaptable. These control parameters form the decision variables that are used in MOO. The components might also have objectives and constraints expressed in the decision variables. In the MO2 style, such a component is referred to as Adaptable Component (AC). Note that there does not need to be a one-to-one mapping between components in other views and ACs in an MO2 view.

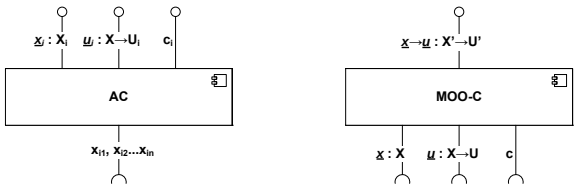

Figure 1. AC and MOO-C interface

As shown in Figure 1, an AC has three interfaces that provide information:

- $\underline{x_i} : X_i$, the set of decision variables for this component. Both the names ($\underline{x_i}$) as well as the defined value range (X_i) is provided;
- $\underline{u_i} : X \rightarrow U_i$, a set of objective functions, named $\underline{u_i}$, providing a transformation from X (the total decision space) to U_i (the value range of the objective functions);
- c_i, a set constraints.

An AC also has several interfaces for required information: $x_{i1}, x_{i2} \ldots x_{in}$ (schematically drawn as one interface). These are used to set the value of the n decision variables.

3.2. Optimization Component (MOO-C)

Figure 1 also shows the interfaces of the MOO-C. This generic component implements an MOO solution.

The MOO-C requires the different inputs of the optimization problem: decision variables ($\underline{x} : X$), objectives ($\underline{u} : X \rightarrow U$) and constraints ($c$). It provides the Pareto optimal set [4] of decision values, as a relation between the decision value and the corresponding objective value. The result also contains the names of the decision variables and objective functions (hence the '$\underline{x} \rightarrow \underline{u}$' part of the interface).

This result can be further processed to find a single decision value, by using the transformation components explained in the next section.

3.3. Transformation Components (TC)

To compose ACs and MOO-Cs, we also need certain reusable components that transform types of interfaces, by applying straightforward mathematical operations on the data. Figure 2 shows a number of TCs that can be used to transform the information between interfaces.

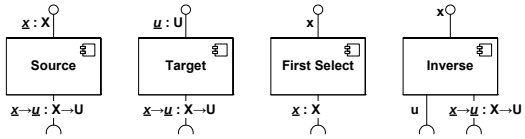

Figure 2. Transformation Components

Source provides the source set of a relation. Similarly, *Target* provides the target set of a relation. *First Select* selects the first value from a set of values. *Inverse* provides a source value from a relationship, given the specific target value and the relationship.

4. Industrial Case Study

In this section we present a view of an architecture that is documented using the MO2 style. A printing system is taken as a case study[2].

Figure 3 shows the ACs in this example case. These are components from the existing architecture that together

2. Due to confidentiality, we present a simplified, yet representative case study.

expose the different decision variables, constraints and objective functions. Note that the figure also shows examples of values on the *Provides* interfaces (decision variables, objectives and constraints).

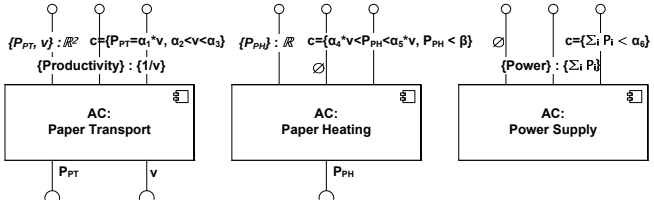

Figure 3. The adaptable components

As shown in the figure, there are three ACs. The first is the *Paper Transport* component. This component has two decision variables: the power to the component (P_{PT}) and the speed (v). It provides one objective: maximizing productivity. It provides two constraints, one that expresses the power-speed relationship ($P_{PT} = \alpha_1 * v$) and one that expresses the allowed speed values ($\alpha_2 \leq v \leq \alpha_3$).

The second component is the *Paper Heating* component. This component has one decision variable: the power to the component (P_{PH}). It provides no objectives. It provides two constraints: one that defines the power-speed relationship ($\alpha_4 * v \leq P_{PH} \leq \alpha_5 * v$) and one that limits the power to the maximum allowable power by design ($P_{PH} \leq \beta$).

The third component is the *Power Supply* component. It has no decision variables. It provides one objective: minimizing power consumption, expressed as the sum of value of the power decision variables of all components. It provides one constraint, which limits the total power consumption to the available power.

Figure 4 shows the MO2 view of the architecture, composing ACs, MOO-Cs and TCs. Note that the ACs are grouped to keep the figure simple.

The optimization functionality is divided into two steps:

1) In the first step, the Pareto space for the objectives is calculated.
2) In the second step, a utility function on the objectives is provided by a trade-off component (also modeled as AC) in the architecture. This utility function is used to find the single best Pareto point for the multiple objectives. This is again done with an MOO-C, in which the decision variables are the utility functions of the first step. The objective is the trade-off function. There are no constraints.

The TCs are used to compose the ACs and MOO-Cs and to transform the resulting Pareto optimal solution back to a specific control decision provided to the ACs.

By documenting the architecture using the MO2 style, we have made explicit where decision variables, constraints and objectives originate from and how they are used to find an optimal value for the decision variables. Now this view has been documented, it can be utilized for analysis and to support implementation. Hereby, MOO-Cs and TCs provide generic solutions and can therefore be reused in different systems.

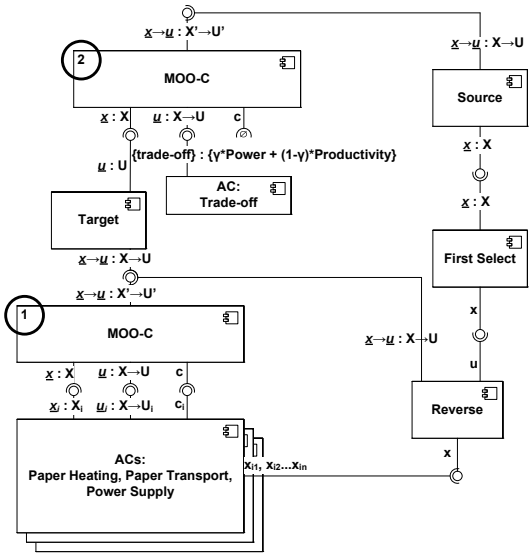

Figure 4. The MO2 view of the architecture

5. Future Work

As future work, we are going to enhance the style to cope with hierarchical composition of MOO solutions. Geilen et al. have developed an algebra to compose the solutions of smaller MOO problems into a solution of a larger, composed MOO problem [7]. We will investigate how these techniques can be fit into the style.

Currently, the style only gives a static view of the optimization process. We are going to investigate how the style can be extended with a dynamic view.

We also want to extend the style to include probabilistic distributions as constraints, as the relationships between decision variables might not be exactly known.

We will create a tool that can be used to specify architectures using the MO2 style. We are planning to use the tool ArchStudio [8] as a basis and extend the xADL architectural description language [9] to be able to express the MO2 style.

There are several possibilities for static analysis. For example, it can be checked whether an AC actually has the interfaces to adapt the decision variables it claims it has (through the *Provides* interface $\underline{x_i} : X_i$)[3]. The static analysis techniques will be incorporated into ArchStudio.

The MOO-C elements contain generic algorithms to solve MOO problems. Therefore, they can be reused between different applications of the MO2 style. We are planning to

3. Note that in this case, the provided decision variables should be statically defined and they cannot be changed dynamically

create an API containing MOO-C elements implementing different algorithms. We will start with linear programming algorithms. Existing implementations are available, for example the Gnu Linear Programming Kit [10] or lp_solve [11]. We will use these libraries to create reusable MOO-Cs.

Code generation and verification are further extensions of the tool; the description of the architecture can be used to generate skeleton code. Furthermore, existing code can be checked for compliance with the architectural description.

Ultimately, we want to develop a design methodology that describes how an engineer can apply MOO in an embedded system. It will contain the steps needed to design a specific solution using the MO2 style, including guidance to select the appropriate MOO-C for a specific type of optimization problem.

We will apply the MO2 architectural style to several case studies provided by our industrial partner in the context of printing systems.

6. Related Work

The Views&Beyond approach makes a distinction between *viewtypes*, *styles* and *views* [6]. Viewtypes, like the Component-and-Connector viewtype, are broad categories of views. Styles are specializations of viewtypes that define recurring instantiations of the viewtype. Examples of C&C styles are the Client-Server style and the Pipe-and-Filter style [6]. Views are instantiations of a viewtype, possibly adhering to one or more styles, for a specific system. In this paper, we have introduced a specific style for documenting MOO solutions for embedded systems.

The IEEE 1471 standard describes that an architecture consists of a set of views [12]. Each of these views conforms to a certain viewpoint. Thus, in the parlance of IEEE 1471, the MO2 style can be regarded as a viewpoint.

Specific styles for control software have been developed before, like the Process Control styles described in [13]. But these styles take a more general view on control; they describe the system in terms of a controlled system, sensors, controllers and actuators. The MO2 style is applicable to a more specific type of functionality in control software, multi-objective optimization, and therefore can exploit specific properties of this functionality.

Many algorithms have been developed to solve multi-objective optimization problems. These algorithms can be broadly divided into classical mathematical algorithms and evolutionary algorithms [14]. The MO2 style now provides the means to document the utilization of these solutions in the architectural design.

7. Conclusion

We have presented an architectural style to document decision making with multiple objectives, called the MO2 style. We have described its base properties and outlined directions for future work. This style makes explicit in the architectural description how MOO is used in the system. In this way, a designer can explicitly document the implementation of multi-objective optimization at the architectural design level, to support analysis, verification, reuse and implementation.

Acknowledgment

This work has been carried out as part of the OCTOPUS project under the responsibility of the Embedded Systems Institute. This project is partially supported by the Netherlands Ministry of Economic Affairs under the Embedded Systems Institute program. We thank Océ's team of software architects and Jacques Verriet from ESI for reviewing this paper and providing useful feedback.

References

[1] Embedded System Institute, "Octopus project website," 2007. [Online]. Available: www.esi.nl/octopus

[2] R. L. Keeney and H. Raiffa, *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*, New York, 1976.

[3] F. Y. Edgeworth, *Mathematical Psychics: An Essay on the Application of Mathematics to the Moral Sciences*, 1881.

[4] V. Pareto, *Cours D'Économie Politique*, 1896.

[5] Y. Collette and P. Siarry, *Multiobjective Optimization: Principles and Case Studies*, 1st ed., 2003.

[6] P. Clements et al., *Documenting Software Architectures: Views and Beyond*, 2002.

[7] M. Geilen et al., "An algebra of pareto points," *Fundamenta Informaticae*, vol. 78, no. 1, pp. 35–74, 2007.

[8] E. Dashofy et al., "Archstudio 4: An architecture-based meta-modeling environment," in *ICSE'07*, 2007, pp. 67–68.

[9] E. M. Dashofy, A. V. d. Hoek, and R. N. Taylor, "A highly-extensible, xml-based architecture description language," in *WICSA '01*, 2001, p. 103.

[10] "Gnu linear programming kit." [Online]. Available: http://www.gnu.org/software/glpk/

[11] "lp_solve." [Online]. Available: http://lpsolve.sourceforge.net/

[12] M. W. Maier, D. Emery, and R. Hilliard, "Software architecture: Introducing IEEE Standard 1471," *IEEE Computer*, vol. 34, no. 4, pp. 107–109, 2001.

[13] M. Shaw and D. Garlan, *Software architecture: perspectives on an emerging discipline*, 1996.

[14] J. Branke et al., "04461 summary – practical approaches to multi-criterion optimization," in *Practical Approaches to Multi-Objective Optimization*, ser. Dagstuhl Seminar Proceedings, 2005.