ArcheOpterix: An Extendable Tool for Architecture Optimization of AADL Models

Aldeida Aleti Stefan Björnander Lars Grunske Indika Meedeniya

Faculty of ICT, Swinburne University of Technology
Hawthorn, VIC 3122, Australia, Tel: +61 3 9214 4758
E-mail: {aaleti,sbjornander,lgrunske,imeedeniya}@swin.edu.au

Abstract

For embedded systems quality requirements are equally if not even more important than functional requirements. The foundation for the fulfillment of these quality requirements has to be set in the architecture design phase. However, finding a suitable architecture design is a difficult task for software and system architects. The reasons for this are an ever-increasing complexity of today's systems, strict design constraints and conflicting quality requirements just to name a few. To simplify the task, this paper presents an extendable Eclipse-based tool, called ArcheOpterix, which provides a framework to implement evaluation techniques and optimization heuristics for AADL specifications. Currently, evolutionary strategies have been implemented to identify optimal and near optimal deployment architectures with respect to multiple quality objectives and design constraints. Experiments with a set of initial deployment architectures provide evidence that the tool can successfully find solution architectures with better quality.

Keywords: ArcheOpterix, Architecture Optimization, AADL, Pareto Optimization, Evolutionary Algorithms

1 Introduction

The quality of the architectural design is critical for the successful development of an embedded system. Two commonly mentioned reasons are:

1. the architecture design sets the foundation for the successful achievement of quality requirements and fulfillment of limited resource budgets [2, 17, 24] and,
2. the architecture design helps to deal with the ever increasing complexity of today's embedded systems [30].

The implications of the software and system architecture on quality characteristics such as safety, availability, reliability, maintainability and temporal correctness just to name a few are well documented in the research literature [2, 17] and affirmed in industrial practice. Decisions made in the architecture design phase have a very large impact on the cost and quality of the final system. An additional difficulty is that quality requirements can often conflict with one another and with economic constraints.

The growing complexity of today's embedded systems is mainly induced by customers requiring more and more functionality. However, another factor is that the design of embedded systems such as vehicle control systems is presently moving from standalone and partitioned systems to functionally integrated architectures [4, 15], which are characterized by extensive sharing of information and hardware resources. In such architectures, shared processors and communication channels allow a large number of configuration options at design time and a large number of reconfigurations options at runtime.

However, when a number of architectures can potentially deliver the desired functionality of a system, designers are faced with a difficult optimization problem. In the rare case let us assume that it is technically possible to fulfill all quality requirements, than the software engineer must find the architecture that entails minimal development and other lifecycle costs. In most cases, fulfilling all quality requirements is infeasible due to conflicting requirements and consequently one must find the architecture or architectures that achieve the best possible tradeoffs among quality attributes and cost. It is widely accepted that the various formulations of the above represents a hard, combinatorial multi-objective optimization problems that can only be approached systematically with the aid of optimization techniques and computerized algorithms that can effectively search for optimal solutions in large potential design spaces [16, 29].

This paper presents a tool called ArcheOpterix[1] that aims to help software architects with this difficult task. ArcheOpterix is an Eclipse plug-in that provides a platform to implement different architecture evaluation and optimization algorithms. The design allows extending the tool with different quality evaluation algorithms and metrics. These evaluation algorithms should follow the principles of model-driven engineering [12] in that they allow to reason about the quality attributes based on an abstract architectural model. Furthermore the optimization engine can be exchanged, allowing to experiment with different optimization heuristics and different optimization problems. In this paper we specifically focus on component deployment problems, however the tool is not limited to these problems.

Since the application domain of ArcheOpterix are embedded and pervasive systems we have chosen AADL (Architecture Analysis and Description Language) [10] as the underlying architecture description language. AADL has been evolved based on the foundation of the architecture description language MetaH [3] and the goal of AADL is to specifically support model-based quality analysis and specification of software and system architectures for complex embedded systems. It has gained increasing attention by the industry in this domain, especially in companies developing automotive and avionic systems. The language itself has been standardized by Society of Automotive Engineers (SAE) in the standard AS5506. Due to the use of AADL, the design of ArcheOpterix is closely aligned with AADL's development environment OSATE (Open Source AADL Tool Environment) [9] and Eclipse [35].

In summary this paper contains the following contributions:

- a high-level description of the tool ArcheOpterix that can be used to optimize AADL specifications,
- detailed information about implementations of some early quality evaluation and architecture optimization plug-ins, and
- results from a set of initial experiments on a set of specific deployment problems that show the suitability of the tool to solve these problems.

The rest of the paper is organized as follows: Section 2 gives and overview of the architecture and goals of ArcheOpterix. Its basic capabilities and some current implementations will be described. Section 3 presents early results of several experiments that have been performed to test the capabilities ArcheOpterix. The tool ArcheOpterix as well as the presented architecture optimization approach based on AADL specifications is compared with related work in Section 4 and we conclude with an outlook to future work in Section 5.

[1]The name ArcheOpterix is derived from the species and fossil *Archaeopteryx* that lived in the Jurassic period. Archaeopteryx is transitional species which represents evolutionary link between dinosaurs and birds. Since the tool aims at architecture optimization and currently mostly evolutionary algorithms have been implemented, we thought that ArcheOpterix would be a suitable name for the tool

2 Architecture and Tool Description

2.1 Overview of the Tool

The ArcheOpterix tool has been developed with Java and Eclipse [35] and can be directly used as a plug-in for the Open Source AADL Tool Environment; OSATE [9]. The architecture of the framework is presented in figure 1 followed by a presentation of its major elements and their interactions;

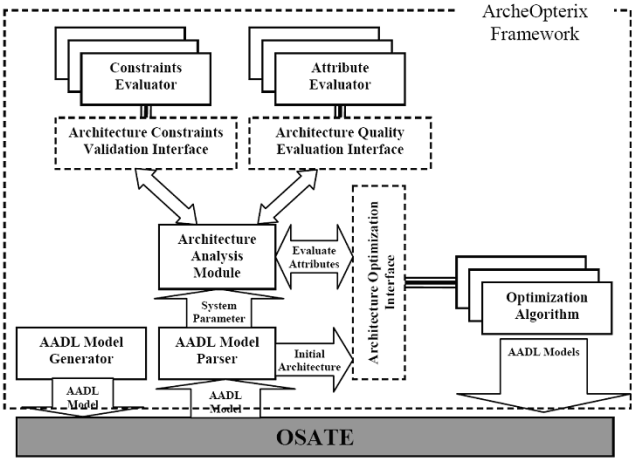

Figure 1. Architecture of ArcheOpterix

AADL Model Parser (AMP). The principle responsibility of this element of the ArcheOpterix is to interpret and extract model abstractions from an AADL specification. The AADL Model Parser takes an AADL model as input (.aaxl file) and accesses the root data structure of the internal model representation. The AMP is capable of capturing AADL standard elements such as processors, processes, networks etc, and further can be extended for more specific elements and domain specific parameters. The AADL Parser acts as a mediator and supports the union of parameter extraction functions required for the Architecture Analysis Module.

Architecture Analysis Module (AAM). This element of the ArcheOpterix framework encompasses the abstraction of model parameters independently from the specification language and application domain. Intuitively, the AAM acts as the central database in the framework containing all relevant information taken from the model speci-

fication in AADL and, acts as the hub with a common interface for application specific implementation of other modules in the framework. The AAM contains a *Context* object, which encapsulates all relevant parameters extracted from the AADL file. For example, in the application of the tool for deployment architecture optimization, the Context object will be populated with hosts, components, networks and interaction parameters. Quality evaluation of an architecture may use different techniques and may be interested in different parameters in the model. In ArcheOpterix, these quality evaluation functions are represented by *AttributeEvaluator* modules. An AAM may contain an arbitrary number of extensible *AttributeEvaluator*s, where they communicate with the module using a generic *Quality Evaluation Interface*. Each AttributeEvaluator implements an *Evaluate(Architecture,Context)* method and provides metrics for a given architecture in the context of the AAM. Apart from the custom implementations of attribute evaluators, the AAM can also be used to interact with OSATE itself to obtain in-built quality quality evaluation capabilities. This can be done by wrapping the OSATE in-built function and interface them with the Quality Evaluation Interface. In addition to the quality evaluation, a given architecture need to be also validated with respect to the constraints specified in the model specification. The *Architecture Constraint Validation Interface* provides plug-in point for modules called *Constraint Evaluators* that check a given architecture for a specific constraint in a given context. The presented two interfaces for Attribute and Constraint Evaluators are technically implemented with the Strategy design pattern [14] that enables different implementations enforcing the common functions.

Architecture Optimization Interface (AOI). The main contribution of ArcheOpterix is to enable the applicability sophisticated algorithms which are abstract and domain independent by nature, for the purpose of architecture optimization. The AOI is the link proposed to cater this objective, coupling architecture evaluation together with architecture unaware optimization algorithms. Knowledge of the application context and the system attributes will be hidden from the algorithms using the AOI by enforcing two generic functions to communicate with; namely *Evaluate(Architecture,Context)* and *Validate(Architecture, Context)*. The interface is also implemented with a standard Strategy design pattern [14] and therefore different algorithmic optimization strategies can be easily linked with the framework by plugging them into the AOI. Different algorithms may achieve optimization in their own approaches. But in common, any algorithm needs to check whether the newly generated architecture is valid with respect to its constraints and fulfills its quality requirements. To achieving these two objectives, AOI communicates with AAM using the aforementioned two generic functions. The output will be a set of deployments that can be optimal, near optimal, Pareto optimal or near Pareto optimal depending on the optimization technique. ArcheOpterix provides the feature of transforming the solutions back into application domain as AADL specifications. For example, if the tool has been used for optimizing deployment architectures, the output will be a set of AADL specifications with deployment relevant code; and as an example an extract of the AADL deployment code is given in listing 1.

Listing 1 A set of AADL specifications.

```
...
properties
Actual_Processor_Binding ⇒ reference host1 applies to comp1;
Actual_Processor_Binding ⇒ reference host1 applies to comp3;
Actual_Processor_Binding ⇒ reference host2 applies to comp4;
Actual_Processor_Binding ⇒ reference host2 applies to comp2;
...
```

AADL Model Generator (AMG). This feature has been developed for the support of testing and rapid model generation purposes. AMG enables to generate AADL models from a given set of input files that consists of different system configurations. This module can be also used for generating discrete test cases during the validation of the tool and convergence of optimization strategies.

2.2 The OSATE Interface: AADL Model Parser (AMP)

The AADL Model Parser (AMP) accepts an AADL model saved in the .aaxl file format. AMP reads the file and extracts its significant parts. In the current implementation the AMP reads the Main property sets as well as the names of the properties defining the network and interaction parameters (default `Host::LocalisationList` and `Component::CoLocalisationList`).

The Network, Interaction, Host, and Component property sets define the attribute of interest for the evaluation of the model. The hosts and components are included in the main system as subcomponents. The hosts are modeled as AADL processors and the components are modeled as processes. The communication channels are modeled as connections. The subcomponents as well as the connections are adorned with properties defining their attributes. Finally, the distribution of components on hosts is defined by the Actual_Processor_Binding property.

2.3 Architecture Evaluation: Architecture Analysis Module (AAM)

The AAM element of the presented ArcheOpterix framework is the common abstraction for system parameter containers, system models and architecture evaluation/validation support. In the initial implementation, the AAM is used

in the domain of deployment architectures. The *Context* of the AAM will be populated with host, component, network and interaction parameters extracted from the AADL parser. The *Context* object in the AAM is implemented as a set of property maps where the keys of the maps are the AADL properties. For example, host in a context contains a map of host parameters as specified in AADL host definition. This has been used in order to grant extensibility and flexibility of the usage of tool for different AADL modeling approaches. The AAM concepts has been implemented supporting extensibility of the ArcheOpterix framework for multiple architecture analysis purposes. Different techniques used in quality evaluation of architectures are encapsulated by AttributeEvaluator modules. The AAM has a pluggable interface for AttributeEvaluator modules and keeps a list of them. The AttributeEvaluators implement a common *Evaluate(Architecture)* interface function in the context specified at AAM's Context object. Via the Architecture Optimization Interface optimization algorithms may request an architecture optimization providing an architecture as an argument, in return a list of attributes calculated for the architecture using AttributeEvaluators linked to AAM will be provided. In the example of deployment architecture optimization, the architecture will be represented by a *Deployment*(an assignment of software components to hardware nodes) and AttributeEvaluators are *Deployment Metric*s (measurements to evaluate a deployment). For the initial experiments we have implemented two Attribute Evaluators: Data Transmission Reliability and Communication Overhead to measure the goodness of a given deployment. These two attributes are evaluated using the schemes presented by Malek [25] and Medvidovic et. al [26]:

Data Transmission Reliability (DTR)

This deployment dependent metric represents to what extent the total data transmission for a given architecture is reliable. For the evaluation of the metric, Sam Malek [25] (this metric has been named as Availability in his thesis) presents following formula, which represents the sum of the product of component interaction frequency and network connection reliability for all component interactions.

$$DTR := \sum_{i=1}^{n} \sum_{j=1}^{n} IP\left(freq, c_i, c_j\right) NP\left(rel, H_{ci}, H_{cj}\right)$$

Communication Overhead (CO)

As a different network and deployment dependent metric the overall communication overhead of the system is used. First part of the equation to calculate the overall communication overhead is the impact of network delay to component interactions and the second part of the equation describes the impact of collision/retransmission enforced by network parameters such as the bandwidth to the overall communication.

$$CO := \sum_{i=1}^{n} \sum_{j=1}^{n} IP\left(freq, c_i, c_j\right) NP\left(td, H_{ci}, H_{cj}\right) + \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{IP\left(freq, c_i, c_j\right) IP\left(evtsize, c_i, c_j\right)}{NP\left(bw, H_{ci}, H_{cj}\right) HP\left(rel, H_{ci}, H_{cj}\right)}$$

Where:

n = Number of components.

$IP(freq, c_i, c_j)$ = Frequency of interaction between component i and Component j.

$IP(evtsize, c_i, c_j)$ = Message size of interaction between component i and Component j.

$NP(rel, H_{ci}, H_{cj})$ = Reliability of network link between hosts of component i and j.

$NP(td, H_{ci}, H_{cj})$ = Network delay in the network link between hosts of component i and j.

$NP(bw, H_{ci}, H_{cj})$ = Bandwidth in the network link between hosts of component i and j.

Since the Architecture Analysis Module contains the union of system parameters together with component configuration information, more sophisticated attribute evaluation schemes such as Fault Tree analysis [28], or the use of Markov models for performance evaluation [33], energy consumption [31], error propagation and fault containment [21] etc. can be implemented in this module. Different constraint evaluators can be also plugged into the AAM using the common *Architecture Constraint Validation Interface*. In the current implementation of ArcheOpterix in the deployment architecture optimization context, three deployment constraint evaluators have been implemented; namely *localization*, *colocalisation* and *memory* constraints. The AAM will check for the satisfaction of all these constraints in the event of validation requests from the optimization algorithms though the AOI.

2.4 Architecture Optimization: Architecture Optimization Module (AOM)

The development of architectures for embedded and pervasive systems requires consideration of multiple quality objectives and several technical and economic constraints. As a result, finding suitable architectures becomes a multi-objective, multi constraint optimization problem. Traditional methods, which deal with single objective optimization and find a single optimal solution for the problem, are not useful in this case. To solve multi-objective optimization problems *Evolutionary Algorithms* are commonly used [6, 11, 20, 34, 38]. These evolutionary algorithms implement mechanisms are inspired by biological concepts such as reproduction, mutation, recombination, natural selection and survival of the fittest, to find a final set of solutions taking in consideration all of the objectives and constraints.

In this section, first the basic principles of evolutionary algorithms will be described and afterwards specific implemented strategies that are used in ArcheOpterix will be introduced. Evolutionary algorithms are a robust optimization strategy that can be used to optimize complex problems with multiple objectives [5]. Commonly an evolutionary algorithm implements the following major steps:

1. Generating of the initial population and evaluate its ranking based on their fitness functions
2. Selecting of the parents for the recombination process
3. Generating offspring solutions by applying a set of genetic operators (e.g. Mutation or Genetic Crossover)
4. Inserting the new solutions to the population and ranking them.
5. Repeating from the second step until reaching the termination criteria (e.g. a certain goal quality or a finite number of iterations)

The ranking of the population is done with regard to all objectives. The evolutionary algorithm implemented in ArcheOpterix uses two approaches for the ranking of the solutions:

- Mapping of all objectives into a single objective function.
- Finding a near Pareto front for the non-dominated solutions.

The mapping of the objectives into a single objective function is done by using a weighted sum function:

$f := \sum_{i=0}^{k} w_i a_i$

Where a_i is the fitness value for the i^{th} objective and w_i is its weight such that $\sum_{i=0}^{k} w_i = 1$. The different weight combinations represent different customer preferences. The second approach uses the concept of non-dominance [23], where a solution is called non-dominant when all its objectives are not dominated by an other solution's objectives and the solution has at least one objective whose value exceeds the other solutions. The Pareto front [23] is the collection of non-dominated solutions plotted in the objective space [5]. For complex problems it is almost impossible to find all possible solutions of the Pareto front [23]. ArcheOpterix draws the near Pareto front line, which contains all the non-dominated solutions found by the Evolutionary Algorithm. Furthermore, ArcheOpterix tries to preserve the diversity of the population, which increases the chances to find a sufficiently large set of [near] Pareto optimal solutions.

2.5 Test Case Generation: AADL Model Generator (AMG)

The test case generation in the ArcheOpterix tool is performed with the AADL Model Generator. This model generator can be used to generate AADL specification with a predefined set of software components and hardware host. The parameter for the quality characteristics and constraints are chosen randomly within certain predefined bounds.

Alternatively, the AADL Model Generator (AMG) can generate AADL models in accordance with an input file. The grammar of the input file is given by listing 2, see listing 3 for an example. The host and component lists define the memory required by each host and component. The network matrix defines the communication attributes between the hosts. The interaction matrix defines the communication attributes between the components. The reserved word **nil** is used to represent absence of communication links.

Listing 2 The Input File Grammar.

input_file	⇒	*hosts components networks interactions localisations colocalisations*
hosts	⇒	**host :** [*host_list*]
host_list	⇒	*host*
	|	*host_list* , *host*
host	⇒	*host_memory*
host_memory	⇒	**integer_constant**
components	⇒	**components :** [*component_list*]
component_list	⇒	*component*
	|	*component_list* , *component*
component	⇒	*component_memory*
component_memory	⇒	**integer_constant**
networks	⇒	**networks :** [*network_matrix*]
network_matrix	⇒	[*network_row*]
	|	*network_matrix* ,
network_row	⇒	*network*
	|	*network_row* , *network*
network	⇒	(*relability* , *delay* , *bandwidth*)
	|	**nil**
relability	⇒	**real_constant**
delay	⇒	**real_constant**
bandwidth	⇒	**real_constant**
interactions	⇒	**interactions :** [*interaction_matrix*]
interaction_matrix	⇒	[*interaction_row*]
	|	*interaction_matrix* , [*interaction_row*]
interaction_row	⇒	*interaction*
	|	*interaction_row* , *interaction*
interaction	⇒	(*frequency* , *event_size*)
	|	**nil**
frequency	⇒	**real_constant**
event_size	⇒	**real_constant**
localisations	⇒	**localisations :** [localisation_matrix]
localisation_matrix	⇒	[*localisation_row*]
	|	*localisation_matrix* , [*localisation_row*]
localisation_row	⇒	*localisation*
	|	*localisation_row* , *localisation*
localisation	⇒	**0** | **1**
colocalisations	⇒	**colocalisations :** [*colocalisation_matrix*]
colocalisation_matrix	⇒	[*colocalisation_row*]
	|	*colocalisation_matrix* , [*colocalisation_row*]
colocalisation_row	⇒	*colocalisation*
	|	*colocalisation_row* , *colocalisation*
colocalisation	⇒	**-1** | **0** | **1**

The network, interaction, and co-localisation matrices are square. AMG checks that the network matrix dimension equals the number of hosts, and that the interaction

Listing 3 An Example of an Input File.

```
hosts: [64,128]
components: [8,16,32,64]

networks:
[[nil, (0.8,2.3,3.4)],
[(0.6,1.2,2.3), nil]]

interactions:
[[nil, (1.2,2.3), nil, (5.6,6.7)],
[(7.8,8.9), nil, (9.0,0.9), nil],
[(7.6,6.5), (5.4,4.3), nil, (3.2,2.1)],
[nil, (3.5,5.7), (7.9,2.4), nil]]

localisations:
[[1, 0, 1, 0],
[0, 1, 0, 1]]

colocalisations:
[[1, -1, 0, 0], [0, 1, -1, 0],
[1, 0, 1, -1], [1, -1, 0, 1]]
```

and co-localisation matrix dimensions equal the number of components. In the localisation matrix, the rows represent hosts and the columns represent components. AMG checks that the number or rows equals the number of hosts and that the number of columns equals the number of components.

The localization matrix restricts the way components can be distributed to hosts. The value 0 in row i and column j prohibits the j^{th} component to be placed in the i^{th} host. The value 1 allows (but do not forces) component j to be placed in host i.

The co-localisation matrix restricts the way two components can be distributed to the same host. The value -1 in row i and column j prohibits component i and j to be distributed to the same host. The value 1 forces the component to be distributed to the same host, and the value 0 allows the component to be distributed to the same host or to two different hosts.

3 First Experimental Results

This section presents a set of experiments and discusses the results. For these experiments a diverse set of systems has been generated to evaluate the ArcheOpterix tool. The number of host and components are changed for different test cases. The network and interaction parameters are randomly generated and range in $[0, 1]$. The component parameters are the required memory, which ranges in $[2^3 - 2^4]$ kilobytes and random colocalization list. Host parameters are the provided memory, which ranges in $[2^4 - 2^7]$ kilobytes and random localization list.

After running the tool for the different test cases, we made some observations in the following perspectives which will be presented in the following:

- The convergence of the optimization algorithm and ability to produce near Pareto-optimal solutions
- The convergence of the optimization algorithm with respect to different population sizes
- The convergence of the optimization algorithm with respect to different mutation rates
- The impact of constraints on the optimization results
- The performance impact of different population sizes and mutation rates

As a first test case, 4 hosts and 10 components have been chosen. The first population has been generated by randomly assigning components to hosts. In figure 2, it can be observed that the average fitness of the population improves distinguishingly during the iterations. The optimization of the average fitness increases until some point and then flattens out. This is an indication that we have obtained a local or global optimum (e.g. the [near] Pareto-front line). Figure 3 shows the final Pareto set for the trade-off between the Data Transmission Reliability and the inverse of Communication Overhead.

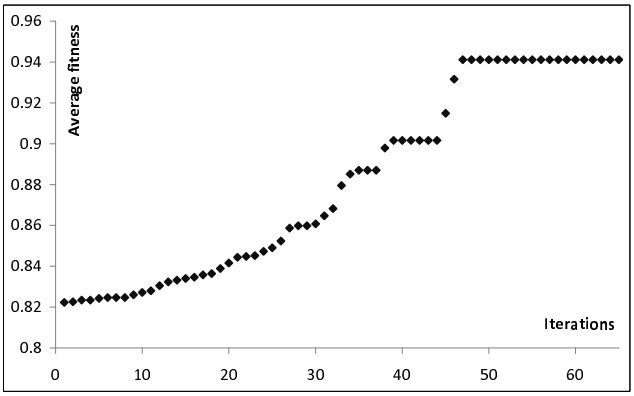

Figure 2. Convergence of the average fitness.

Each of the solutions in Figure 3 is non-dominated by the others. Consequently, a human decision maker selects the solutions which suites him best. ArcheOpterix draws the near Pareto front line, which contains all the non-dominated solutions found by the Evolutionary Algorithm. The distance of the resulting non-dominated front to the Pareto-optimal front should be minimized.

Different population sizes have an effect on the capability to converge towards the Pareto-optimal front. In the next experiment, three different population sizes have been taken and their ability to converge to the final fittest population are compared. In Figure 4, it can be observed that, if the population size is increased, the fitness of population converges faster. When a higher number of individuals is generated for each population, not only the slope of the average fitness to

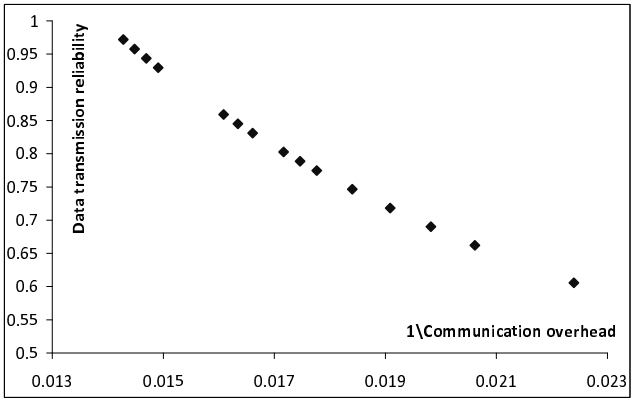

Figure 3. Near Pareto optimal solutions.

iteration line, but also the average fitness of the last population is better. As it can be observed in Figure 4, when a population of size 20 is used, the average fitness converges to a lower bound than when we use a population of size 40 and 60.

To provide enough diversity among the individuals, a large population size should be chosen. It can be observed in Figure 4, where the increase of population from 20 individuals to 40 does not have a distinguishable impact on the performance of the algorithm. The increase of average fitness of the population gets better when we put the population size to 60. However, these results cannot be generalized, as different runs of the evolutionary algorithm may result in a different quality increases, due to the random nature of evolutionary algorithms.

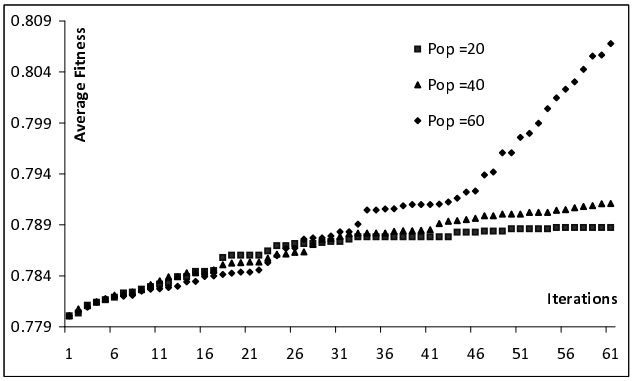

Figure 4. Impact of the population size on the optimization of the average fitness.

The final outcome of the architecture optimization module is the near Pareto-front solutions, which is shown in Figure 5. It can be noted that similar non-dominated solutions exist in each case. Even though the average fitness growth is significantly different in different populations as observed in Figure 4, it is evident from Figure 5 that there are similar final non-dominated solutions for each case.

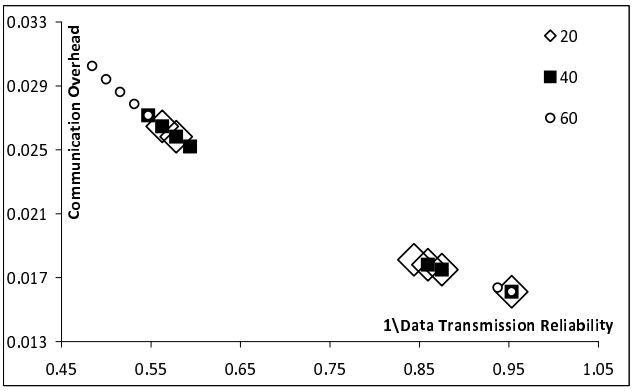

Figure 5. Near Pareto-front solutions for different population sizes.

Similar to the population size impact, in Figure 6, it can be observed that when the number of offspring to be simulated (mutation rate) is increased, the algorithm converges faster to a fitter population. The mutation factor helps in keeping the diversity of the population and producing new fitter solutions. The mutation creates the necessary diversity in the population and thereby facilitates novelty, while selection acts as a force increasing the quality.

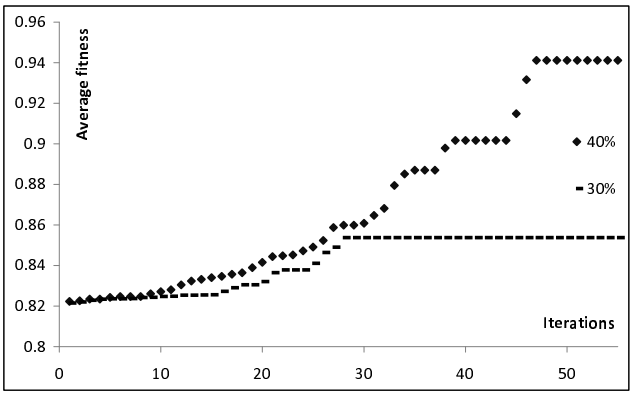

Figure 6. Impact of the mutation rate on convergence of the algorithm.

During the experiments, the effect of the constraints in the development of the fitness line has been measured. In Figure 7, it can be observed that we get fitter population when the checking of constraints is deactivated by the op-

timization algorithm. This can be explained with the fact that the lack of constraints may allow some solutions which are not acceptable but have a high fitness value. Moreover, the usage of constraints causes a slow convergence of the algorithm.

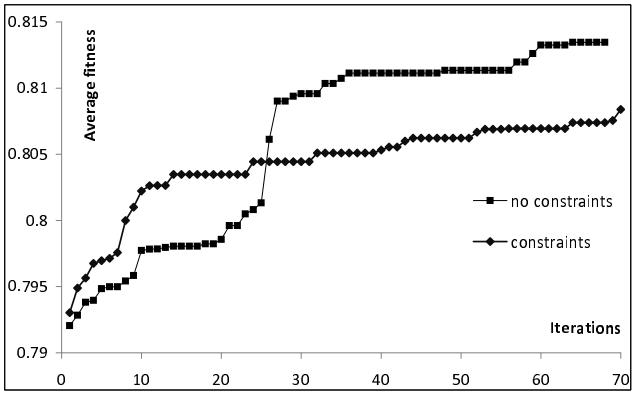

Figure 7. Impact of the constraints on convergence of the algorithm.

During the experiments, an additional concern was the performance of the ArcheOpterix tool and its optimization algorithm. In Figure 8 and 9, it can be seen how the execution time of the algorithm increases by the increase of the population size and mutation rate. It can be clearly observed that the increasing of the mutation rate and population size have a linear impact on the execution time. Due to this fact, it can be assumed that to increase the performance of the algorithm, a trade off should be made for relatively low mutation rate in a reasonably large population.

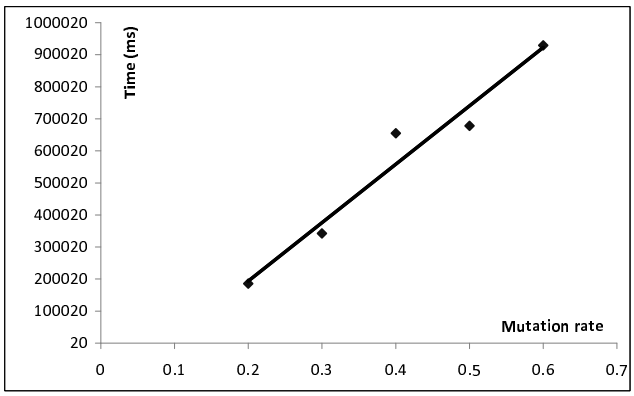

Figure 8. Impact of the population size on the execution speed of the algorithm.

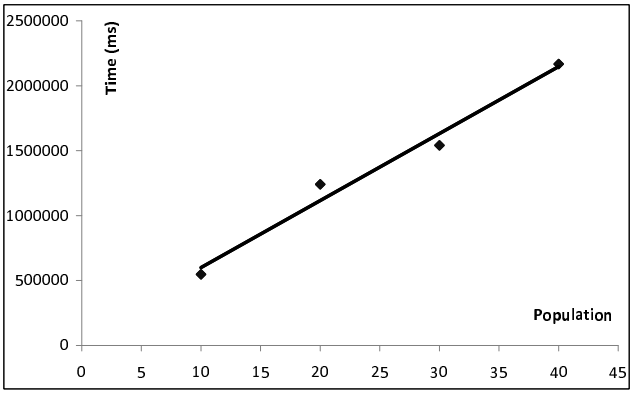

Figure 9. Impact of the mutation rate on the execution speed of the algorithm.

4 Related Work

A considerable number of approaches have been developed over the past decade to tackle the problem of finding optimal and near optimal architectures with respect to different non-function attributes. In this section we would like to compare ArcheOpterix in two categories: (a) tools that are publically available for optimization of architectures and (b) architecture optimization methodologies in general.

In the tool category, ArcheOpterix should be compared with thye tools DeSi [27], ArchE [8], DAnCE [7] and the RACE framework [32]. The DeSi tool, developed by Mikic-Rakic et al. [27] presents an tailorable environment for specification, manipulation, visualization and attribute evaluation of deployment architectures. The tool has been developed as a stand-alone eclipse application and allows runtime deployment optimization via monitoring of live systems. The tool is very mature, but requires model specifications in a special format. ArcheOpterix currently implements similar analysis capabilities; however the future goal of ArcheOpterix is to extends the coverage of use in context of architecture optimization beyond deployment decision making. The ArchE tool [1] is a design assistant that helps software architect to make decision with respect to relevant quality attributes. ArchE contains a rule-based expert systems that search the design space with so called reasoning frameworks. Each reasoning framework support on quality domain and currently a reasoning framework is implemented for modifiability [1]. To allow the support of other quality domains ArchE provides a well defined interface to also generate reasoning frameworks for other quality attributes [8]. The RACE framework presented by Shankaran et al. [32] and the DAnCE implementation by Deng et al. [7] addresses the needs of deployment decisions at real-time. Both the above approaches have been

well succeeded in industry, but limitations exist for applicability for static architecture optimization domain such as they require implementations of specific system models and lack of support for complex model analysis. All these tool provide comparable capabilities to ArcheOpterix. However, these tools need specialized formats for the input architectures and its quality annotations. Often these input formats are really restricted. In contrast ArcheOpterix uses with the AADL an established architecture description language.

In the second category ArcheOpterix is compared with general architecture optimization methodologies. Most of these approaches have also implemented tools, however they are either just for demonstration and experiment purposes or they are publically not available. Papadopoulos and Grante [28] provide a novel technique for safety and reliability analysis in automotive software. Their approach consist of system modeling in Matlab Simulink [36], the approach combines fault tree analysis and genetic algorithms to support the decisions of "whether" and "when" redundancies are needed. Being able to link the system with Matlab Simulink, they grant the opportunity for standard analysis capabilities. Our work presented in this paper, gains the similar advantage from the modeling perspective by using the standard AADL. Since our tool offers the attribute evaluation module in the same tool, the presented approach extends the [28] work from a semi-automated process into one integrated, automated process.

Fredriksson et al. [13] presents a framework for allocating components to real time tasks, focusing on minimizing the resource usage such as CPU time and memory. They have formalized a model for components and tasks, derive memory consumption, CPU overhead for task deployment and propose to use existing scheduling and optimization algorithms with real-time analysis to ensure feasible allocation. While this approach works in a different domain, ArcheOpterix can draw from this research and be extended in the future to provide similar capabilities.

Deployment architectural decision making for highly constrained environments, has been addressed by Kichkalyo et al. [22] and their approach is to use AI planning techniques to find a feasible solution. The formalized general model of a Component Placement Problem (CPP) has been analyzed by their own algorithm called Sekitei. A major novelty of ArcheOpterix in relation to [13], [22] is ability to provide a set of non-dominated architectural solutions instead of obtaining just one feasible solution, which is common for optimization approaches that just use a weighted sum function to translate a multi-objective optimization problem into one with just a single objective.

5 Conclusions

This paper has presented a novel tool called ArcheOpterix for optimization of architectures of embedded systems. This tool uses the AADL as the underlying architecture description language and provides plug-in mechanisms to replace the optimization engine, the quality evaluation algorithms and the constraints checking. To validate the tool a specific multi-objective, multi-constrain component deployment problem has been used. For this problem, similar to [27], two standard quality metrics (data transmission reliability and communication overhead) and three constraints (component location, component collocation and memory consumption) have been used. The results gained from an early implementation of an evolutionary algorithm [11] are encouraging; however there is still room for future improvements and there are several interesting research questions to be solved.

In the future, ArcheOpterix should be extended by the design team and other researcher with additional quality evaluation procedures. Especially, procedures that evaluate quality attributes in quality domains that are relevant for embedded systems, like safety, reliability, security, performance, timeliness and resource consumption [18].

Since ArcheOpterix should also serve as an experiment platform for optimization algorithms the tool will be also extended with additional optimization heuristics. The performance of these heuristics can then be compared based on their efficiency for benchmark problems. Furthermore, optimization algorithms should be implemented that can find a good variety of solutions. Consequently, diversity improving and preserving factors have to be implemented as well in these optimization algorithms.

Finally, ArcheOpterix has been currently implemented as an independent Eclipse plug-in. However, to improve the tool performance and to have access to larger set of evaluation methods a tight integration with OSATE would be beneficial.

References

[1] F. Bachmann, L. J. Bass, M. Klein, and C. P. Shelton. Experience using an expert system to assist an architect in designing for modifiability. In *4th Working IEEE / IFIP Conference on Software Architecture (WICSA 2004), 12-15 June 2004, Oslo, Norway*, pages 281–284. IEEE Computer Society, 2004.

[2] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. AddisonWesley, second edition, 2003.

[3] P. Binns, M. Englehart, M. Jackson, and S. Vestal. Domain-specific software architectures for guidance, navigation and control. *International Journal of Software Engineering and Knowledge Engineering*, 6(2):201–227, 1996.

[4] M. Broy. Challenges in automotive software engineering. In L. J. Osterweil, H. D. Rombach, and M. L. Soffa, editors, *28th International Conference on Software Engineering (ICSE 2006), Shanghai, China, May 20-28, 2006*, pages 33–42. ACM, 2006.

[5] C. A. C. Coello, D. A. V. Veldhuizen, and G. B. Lamont. *Evolutionary Algorithms for Solving Multi-objective Problems*. Springer, 2002.

[6] K. Deb, S. Agrawal, A. Pratab, and T. Meyarivan. A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. KanGAL report 200001, Indian Institute of Technology, Kanpur, India, 2000.

[7] G. Deng, J. Balasubramanian, W. Otte, D. C. Schmidt, and A. S. Gokhale. Dance: A qos-enabled component deployment and configuration engine. In *Component Deployment*, volume 3798 of *Lecture Notes in Computer Science*, pages 67–82. Springer, 2005.

[8] A. Díaz Pace, H. Kim, L. Bass, P. Bianco, and F. Bachmann. Integrating quality-attribute reasoning frameworks in the ArchE design assistant. In S. Becker, F. Plasil, and R. Reussner, editors, *Quality of Software Architectures. Models and Architectures, 4th International Conference on the Quality of Software-Architectures, QoSA 2008, Karlsruhe, Germany, October 14-17, 2008. Proceedings*, volume 5281 of *Lecture Notes in Computer Science*, pages 171–188. Springer, 2008.

[9] P. Feiler and A. Greenhouse. *Plug-in Development for the Open Source AADL Tool Environment*. Available from http://la.sei.cmu.edu/aadlinfosite/OSATEPlugInDevelopmentPresentationSerie.html#Topic19, 2005.

[10] P. H. Feiler, D. P. Gluch, and J. J. Hudak. The Architecture Analysis and Design Language (AADL): An Introduction. Technical report, CMU/SEI-2006-TN-011, 2006.

[11] C. M. Fonseca and P. J. Fleming. Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 416–423, San Mateo, California, 1993. University of Illinois at Urbana-Champaign, Morgan Kauffman Publishers.

[12] R. France and B. Rumpe. Model-driven development of complex software: A research roadmap. In *FOSE '07: 2007 Future of Software Engineering*, pages 37–54. IEEE Computer Society, 2007.

[13] J. Fredriksson, K. Sandström, and M. Åkerholm. Optimizing resource usage in component-based real-time systems. In *CBSE:Component-Based Software Engineering, 8th International Symposium, CBSE 2005, St. Louis, MO, USA, May 14-15, 2005, Proceedings*, volume 3489 of *Lecture Notes in Computer Science*, pages 49–65. Springer, 2005.

[14] E. Gamma. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley, 2007.

[15] K. Grimm. Software technology in an automotive company - major challenges. In *Proceedings of the 25th International Conference on Software Engineering (ICSE), May 3-10, 2003, Portland, Oregon, USA*, pages 498–505. IEEE Computer Society, 2003.

[16] L. Grunske. Identifying "good" architectural design alternatives with multi-objective optimization strategies. In L. J. Osterweil, H. D. Rombach, and M. L. Soffa, editors, *28th International Conference on Software Engineering (ICSE 2006), Shanghai, China, May 20-28, 2006*, pages 849–852. ACM, 2006.

[17] L. Grunske. Early quality prediction of component-based systems - A generic framework. *Journal of Systems and Software*, 80(5):678–686, 2007.

[18] L. Grunske. Specification patterns for probabilistic quality properties. In Robby, editor, *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008*, pages 31–40. ACM, 2008.

[19] A. Hertz and D. Kobler. A framework for the description of evolutionary algorithms. *European Journal of Operational Research*, 126(1):1–12, October 2000.

[20] J. Horn, N. Nafpliotis, and D. E.Goldberg. A niched pareto genetic algorithm for multiobjective optimization. In *International Conference on Evolutionary Computation*, pages 82–87, 1994.

[21] A. Jhumka, M. Hiller, and N. Suri. Assessing inter-modular error propagation in distributed software. In *Proceedings of the 20th Symposium on Reliable Distributed Systems (20th SRDS'01)*, pages 152–161, 2001.

[22] T. Kichkaylo, A.-A. Ivan, and V. Karamcheti. Constrained component deployment in wide-area networks using ai planning techniques. In *IPDPS:17th International Parallel and Distributed Processing Symposium (IPDPS 2003), 22-26 April 2003, Nice, France, CD-ROM/Abstracts Proceedings*, page 3, 2003.

[23] J. D. Knowles and D. Corne. Approximating the nondominated front using the pareto archived evolution strategy. *Evolutionary Computation*, 8(2):149–172, 2000.

[24] N. G. Leveson. An approach to designing safe embedded software. In *EMSOFT '02: Proceedings of the Second International Conference on Embedded Software*, pages 15–29. Springer-Verlag, 2002.

[25] S. Malek. *A User-Centric Approach For Improving A Distributed Software Systems Deployment Architecture*. PhD in computer science, Faculty of The Graduate School University of Southern California, 2007.

[26] N. Medvidovic and S. Malek. Software deployment architecture and quality-of-service in pervasive environments. In *Proceedings of the 2007 International Workshop on Engineering of Software Services for Pervasive Environments, ESSPE 2007, Dubrovnik, Croatia, September 4, 2007*, pages 47–51. ACM, 2007.

[27] M. Mikic-Rakic, S. Malek, N. Beckman, and N. Medvidovic. A tailorable environment for assessing the quality of deployment architectures in highly distributed settings. In *Component Deployment, Second International Working Conference, CD 2004, Edinburgh, UK, May 20-21, 2004, Proceedings*, volume 3083 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2004.

[28] Y. Papadopoulos and C. Grante. Techniques and tools for automated safety analysis & decision support for redundancy allocation in automotive systems. In *COMPSAC*, pages 105–110. IEEE Computer Society, 2003.

[29] Y. Papadopoulos and C. Grante. Evolving car designs using model-based automated safety analysis and optimisation

techniques. *Journal of Systems and Software*, 76(1):77–89, 2005.

[30] A. Pretschner, M. Broy, I. H. Kruger, and T. Stauner. Software engineering for automotive systems: A roadmap. In *FOSE '07: 2007 Future of Software Engineering*, pages 55–71. IEEE Computer Society, 2007.

[31] C. Seo, G. Edwards, S. Malek, and N. Medvidovic. A framework for estimating the impact of a distributed software system's architectural style on its energy consumption. In *WICSA*, pages 277–280. IEEE Computer Society, 2008.

[32] N. Shankaran, J. Balasubramanian, D. C. Schmidt, G. Biswas, P. J. Lardieri, E. Mulholland, and T. Damiano. A framework for (re)deploying components in distributed real-time and embedded systems. In *Proceedings of the 2006 ACM Symposium on Applied Computing (SAC), Dijon, France, April 23-27, 2006*, pages 737–738, 2006.

[33] V. S. Sharma and P. Jalote. Deploying software components for performance. In *CBSE:Component-Based Software Engineering, 11th International Symposium, CBSE 2008, Karlsruhe, Germany, October 14-17, 2008. Proceedings*, volume 5282 of *Lecture Notes in Computer Science*, pages 32–47, 2008.

[34] N. Srinivas and K. Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1995.

[35] The Eclipse Foundation. *Eclipse Ganymede Documentation*. Available from http://help.eclipse.org/ganymede/index.jsp.

[36] The MathWorks. *Simulink - Simulation and Model-Based Design*. Available from http://www.mathworks.com/.

[37] T. Weise. *Global Optimization Algorithms Theory and Application*. http://www.it-weise.de/projects/book.pdf, 2008.

[38] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. In *EUROGEN 2001. Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems*, pages 95–100, Athens, Greece, 2002.