Automatic optimisation of system architectures using EAST-ADL

Martin Walker, Mark-Oliver Reiser, Sara Tucci-Piergiovanni, Yiannis Papadopoulos, Henrik Lönn, Chokri Mraidha, David Parker, Dejiu Chen, David Servat

The Journal of Systems and Software 86 (2013) 2467–2487

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Automatic optimisation of system architectures using EAST-ADL

Martin Walker[a,*], Mark-Oliver Reiser[b], Sara Tucci-Piergiovanni[c], Yiannis Papadopoulos[a], Henrik Lönn[d], Chokri Mraidha[c], David Parker[a], DeJiu Chen[e], David Servat[c]

[a] University of Hull, Cottingham Road, Hull HU6 7RX, UK
[b] Technische Universität Berlin, Ernst-Reuter-Platz 7, D-10587 Berlin, Germany
[c] CEA Saclay Nano-INNOV, Point Courrier No. 174, 91191 Gif sur Yvette cedex, France
[d] Volvo Technology, Dept 6260, M2.7, 405 08 Gothenburg, Sweden
[e] KTH Royal Institute of Technology, Brinellvägen 83, 100 44 Stockholm, Sweden

ARTICLE INFO

Article history:
Received 31 July 2012
Received in revised form 28 March 2013
Accepted 1 April 2013
Available online 8 April 2013

Keywords:
Multi-objective optimisation
Architectural description languages
Dependability Analysis
Timing Analysis

ABSTRACT

There are many challenges which face designers of complex system architectures, particularly safety–critical or real-time systems. The introduction of Architecture Description Languages (ADLs) has helped to meet these challenges by consolidating information about a system and providing a platform for modelling and analysis capabilities. However, managing this wealth of information can still be problematic, and evaluation of potential design decisions is still often performed manually. Automatic architectural optimisation can be used to assist this decision process, enabling designers to rapidly explore many different options and evaluate them according to specific criteria. In this paper, we present a multi-objective optimisation approach based on EAST-ADL, an ADL in the automotive domain, with the goal of combining the advantages of ADLs and architectural optimisation. The approach is designed to be extensible and leverages the capabilities of EAST-ADL to provide support for evaluation according to different factors, including dependability, timing/performance, and cost. The technique is applied to an illustrative example system featuring both hardware and software perspectives, demonstrating the potential benefits of this concept to the design of embedded system architectures.

1. Introduction

Achieving quality attributes such as dependability (e.g. measured in terms of reliability, availability, and safety) and performance (e.g. measured in terms of throughput of information and response times) in complex systems is a challenging task. It is beneficial to consider these attributes throughout the whole design process, factoring them into early decisions and thereby ensuring that quality attributes are controlled from the early stages rather than left to emerge (or not) at the end, when any required changes would incur larger costs and delays. However, balancing the many demands on the system design can be difficult, and the task of exploring potentially huge design spaces for optimal configurations that maximise quality (i.e., dependability and performance) and minimise cost poses even more challenges.

Recent work on model-based development has looked into how progressively refined models of requirements and design can be used to drive the development and verification of complex systems. This work has resulted in methods like UML and SysML which enable system modelling that encompasses description of structure, behaviour and allocation of functions to hardware resources. More recently, ADLs such as AADL and the automotive EAST-ADL have emerged as potential future standards for model-based design of embedded systems in the transport and space industries.

Beyond modelling of nominal behaviour, these languages also incorporate error modelling semantics that enable dependability analysis. Early work has demonstrated that dependability analysis of EAST-ADL models is possible via HiP-HOPS (Chen et al., 2008) while dependability analysis of AADL error models is possible via conversion to fault trees (Joshi et al., 2007) and Generalised Stochastic Petri Nets (GSPN) (Feiler and Rugina, 2007). MARTE (www.omgmarte.org) is another significant development, implemented as a UML profile for the design of real-time embedded systems, enabling prediction of other quality attributes, including performance and schedulability.

Abbreviations: ADL, Architecture Description Language; COE, Central Optimisation Engine; FAA, Functional Analysis Architecture; FDA, Functional Design Architecture; HDA, Hardware Design Architecture; OSDM, Optimisation Space Definition Module; VRM, Variability Resolution Mechanism.

* Corresponding author. Tel.: +44 (0)1482 465994.

E-mail addresses: martin.walker@hull.ac.uk (M. Walker), mark-oliver.reiser@tu-berlin.de (M.-O. Reiser), sara.tucci@cea.fr (S. Tucci-Piergiovanni), y.i.papadopoulos@hull.ac.uk (Y. Papadopoulos), henrik.lonn@volvo.com (H. Lönn), chokri.mraidha@cea.fr (C. Mraidha), d.j.parker@hull.ac.uk (D. Parker), chen@md.kth.se (D. Chen), david.servat@cea.fr (D. Servat).

0164-1212/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2013.04.001

New ADLs are likely to influence the design of future dependable systems. These languages can achieve transparency and consistency in a model-based process but are not yet sufficiently developed to guarantee effective assessment and satisfaction of quality attributes. We can identify two challenges that need to be tackled from the early stages of design:

(a) Ensuring effective prediction of quality attributes such as dependability and performance, via use of advanced, scalable, automated model-based analysis techniques.
(b) Enabling effective exploration of potentially huge design spaces for design solutions that achieve better or optimal trade-offs among dependability, performance and cost. We note that this capability is particularly relevant where little prior design experience exists to guide engineers in the design of a system, e.g. in case of innovative designs or when relatively immature technologies are employed.

1.1. Analysis of architectural models

Over the last fifteen years, in order to tackle the first of these challenges, work on model-based dependability analysis has resulted in new approaches that partly automate and thereby simplify the synthesis of dependability evaluation models. In several techniques, predictive system failure models such as fault trees and FMEAs (Failure Modes and Effects Analysis) are constructed from the topology of the system and component failure models using a process of composition. In this approach, automation and reuse of component failure models across applications become possible to some extent. Compositionality and careful reuse are expected to bring benefits in dependability analysis similar to those introduced by reuse of trusted software components in software engineering. Techniques that follow this approach include HiP-HOPS (Papadopoulos et al., 2001), Component and State-Event Fault Trees (Grunske et al., 2005) and the Failure Propagation and Transformation Calculus (Wallace, 2005). Another body of research with similar objectives has developed in the area of formal verification, focusing on automated dependability analysis of systems represented as state automata (e.g. see work on Altarica (Bieber et al., 2004) and FSAP-NuSMV (Bozzano and Villafiorita, 2007)). In these approaches, model-checking is used to verify the satisfaction of dependability requirements or detect violations of requirements in normal or faulty conditions.

Tools have been developed to support these dependability and performance evaluation techniques and complex case studies have been reported that demonstrate benefits. However, relatively little work has been applied in the context of model-based design (especially in combination with ADLs). In the ATESST2 and MAENAD EU projects, it has been demonstrated that scalable automated dependability analysis can be achieved via harmonisation of EAST-ADL with HiP-HOPS. This work has shown benefits from the ability to produce complex fault trees and multiple failure FMEAs from designs expressed in an ADL (Chen et al., 2008). However, there is still scope for improvement in this work that could extend the state-of-the-art in model-based dependability analysis. In principle, AADL models—effectively state automata showing transitions from normal to degraded and failed states—can be converted to GSPNs and then analysed for dependability (Feiler and Rugina, 2007). However, in this approach, it is not possible to perform qualitative analysis, i.e., establishment of direct causal relationships between causes and effects of failure as in FMEA, which is important when probabilistic data are not available, e.g. at early stages of design. An alternative approach has been demonstrated in (Joshi et al., 2007) via conversion of AADL error models to fault trees. The main difficulty we have identified here is that the temporal semantics of the AADL error model (a form of state machine) are lost in the translation to combinatorial fault trees and this potentially causes serious errors. To correct this conceptual flaw and enable true temporal dependability analysis, we propose to enable analysis of AADL error models via HiP-HOPS, where a recently integrated temporal logic called Pandora can be used to achieve automated synthesis and analysis of dynamic fault trees, making it possible to capture the often significant effects of sequencing of faults. The problem has been discussed in (Walker and Papadopoulos, 2009) and a general solution was proposed; however, this solution still needs to be adapted in the context of particular modelling languages.

Another issue, often ignored in modern dependability analysis techniques, is that of separation of concerns in design. Most model-based dependability analysis techniques assume that the model of the system (whether in the form of a block diagram or a state machine) is a functional model, effectively a joint representation of hardware and software. However, this ignores the separation of concerns that typically takes place in practical modelling, where people tend to develop separate models for hardware and software and link those models with allocation relationships. Such models are not directly analysable by the present state-of-the-art. In HiP-HOPS, a concept has been developed that enables integrated dependability analysis of designs that are represented in more than one perspective (e.g. hardware, software, middleware) via assessment of the fault propagation through allocation relationships.

In the context of timing analysis, many model-based approaches have been recently developed for the automotive domain. Projects such as ATESST and ATESST2 (www.atesst.org), TIMMO (www.timmo.org), and EDONA (www.edona.fr) have been carried out to provide concepts, tools and methodologies for the description of automotive architectures and timing properties.

Among the wide range of timing analyses, schedulability analysis (Selic, 2000) is considered a good candidate for analysing timing properties at the design stage. A model-based framework for schedulability analysis has been developed along these lines in the context of the EDONA project. This framework enables model-based schedulability analysis at the EAST-ADL design level. The EAST-ADL design level includes functional architecture, hardware architecture, and an allocation model stipulating the mapping of functions to hardware nodes. The schedulability framework is based on supplementing EAST-ADL with concepts from the modelling language MARTE following the procedure presented by Anssi et al. (2010). In this framework, schedulability analysis is performed after a manual transformation of EAST-ADL models towards a refined architecture: the MARTE task model. The manual transformation establishes the mapping of functions on tasks (including task priorities and periods). Then the framework provides an automatic transformation of MARTE task models to an academic scheduling analysis tool called MAST (http://mast.unican.es).

It should be noted that in the EDONA approach, schedulability analysis verifies the task model, and does not directly verify the design-level architecture.[1] Schedulability results therefore have to be manually ‘fed-back’ to the design level by interpreting schedulability properties obtained from the MARTE task model. Furthermore, extrapolating feedback to improve the design-level architecture is cumbersome or even impossible for non-schedulability experts. The reason for this lies in the fact that schedulability results depend not only on design-level architecture properties but also *on the choices made during the refinement towards the task model* (e.g. priority assignment, task periods, scheduling policy, etc.).

[1] It is worth noting that EAST-ADL lacks implementation-like concepts, such as the task concept, although other languages (e.g. AUTOSAR) can be used to overcome this later in the system development process.

The ATESST2 and MAENAD projects made improvements to model-based timing analysis for EAST-ADL models in order to produce automatic feedback on the EAST-ADL architecture. Simple feedback can be obtained by analysing architecture properties which do not need refinement towards a task model, such as resource utilisation, which can be computed only knowing the allocation of functions/signals to hardware resources. For more advanced indicators, such as response time analysis of function activation chains that traverse several shared hardware resources, full schedulability analysis is still needed, which necessitates refinement towards a task model to configure schedulability analysis tools. However, by applying *controlled* refinements, it is possible to produce automatic feedback at design level. In this paper we adopt one of these possible controlled refinements, which is detailed in Section 2.4.

1.2. Optimisation of architectural models

Model-based analysis and verification technologies can enrich a model-driven development process by answering important questions regarding the quality of individual design proposals. In complex distributed systems, however, rich functionalities and their distribution across shared hardware and communication channels allow a large number of configuration options at design time and a large number of reconfiguration options at runtime. This creates difficulties in design because, as potential design spaces expand, their exploration for suitable or optimal designs becomes increasingly difficult. When a number of different architectural configurations can potentially deliver the functions of a system, designers are faced with a difficult optimisation problem. Assuming that it is technically and economically possible to fulfil all quality requirements, they must find an architecture that entails minimal development and other lifecycle costs. On the other hand, if fulfilling or optimising all quality requirements is infeasible, then they must find the architecture or architectures that achieve the best possible tradeoffs among quality attributes and cost. The problem is compounded by the fact that quality attributes are often conflicting, e.g. improving safety often means not only increasing costs but also reducing availability. The various formulations of the above represent hard, multi-objective optimisation problems that can only be approached systematically with the aid of optimisation algorithms that can efficiently search large potential design spaces.

Whilst many design problems can only be tackled effectively by the human intellect, it is clear that, as potential design spaces expand, their exploration for suitable or optimal designs (e.g. in terms of quality and cost) becomes increasingly difficult, and some automation is needed. Modelling languages and emerging ADLs could therefore benefit from concepts and technological support that enable this type of optimisation, while still benefiting from the support for multiple analysis and evaluation functions that an ADL offers.

Some work has recently been done in this field from the direction of safety and reliability analysis, though not in the context of ADLs. Classical dependability models like Reliability Block Diagrams (RBDs) (Konak et al., 2006) and, more recently, advanced compositional dependability analysis techniques such as HiP-HOPS have been combined with meta-heuristics (Pareto-based Genetic Algorithms) to assist in the automatic evolution of design models that can meet dependability and cost requirements. HiP-HOPS has contributed to this area by enabling the optimisation of systems that have a networked architecture (i.e. they are not necessarily in parallel/series configurations as in RBDs) and by overcoming the traditional assumption made in RBDs that a component or system either works or fails in a single failure mode (Papadopoulos and Grante, 2005; Papadopoulos et al., 2011; Adachi et al., 2011).

Recent work that has focused on enabling multi-objective optimisation of software architectures has led to the development of new tools that also offer a blend of analysis and optimisation capabilities. One such tool is PerOpteryx, which is based on the Palladio modelling environment (Martens et al., 2010; Koziolek and Reussner, 2011). PerOpteryx allows for the automatic optimisation of software architecture models, developed with Palladio using the Palladio Component Model (PCM), on the basis of four main quality dimensions: cost, reliability, maintainability, and performance. One particular advantage of PerOpteryx is its ability to take advantage of domain specific knowledge, such as performance tactics, to enhance the optimisation (Koziolek et al., 2011).

Another tool is AQOSA (Automated Quality-driven Optimisation of Software Architecture), which uses model transformation technology to convert input models (e.g. from AADL or a general UML2 model) into an intermediate format (AQOSA-IR) that can be used as the basis of the optimisation process (Etemaadi and Chaudron, 2012). Different candidates are provided by a repository of possible components, and a set of external objective function plugins provides the evaluation that drives the process. AQOSA is designed to be independent of any given domain specific language (DSL) or ADL, but therefore relies on a correct model transformation to its own intermediate model to perform the optimisation.

Finally, other work by Grunske et al. has shown the potential of using various meta-heuristics for dependability versus cost optimisation of architectural designs. This work has been applied to AADL models and also looks at other optimisation approaches beyond genetic algorithms, e.g. ant trail algorithms (Aleti et al., 2009a,b; Meedeniya et al., 2010, 2011; Meedeniya and Grunske, 2010). An introduction to the model-based optimisation field can also be found in Grunske et al. (2007), and a wider survey of literature on architectural optimisation techniques can be found in Aleti et al. (2012).

All of the approaches mentioned above exploit meta-heuristics to search a design space for optimal solutions. However, they operate on different models (EAST-ADL, PCM, AQOSA-IR etc.) and use different means of defining the design space (e.g. variability mechanisms). They also address different objectives, which they define in different ways, and use different techniques to evaluate those objectives. The aim of this paper is to contribute to the state of the art by developing an approach which brings a unique combination of objective evaluation techniques (e.g. HiP-HOPS, MAST) derived from information obtained from the variability capabilities and quality attributes provided by EAST-ADL. Therefore, while there are both commonalities and differences across the optimisation approaches mentioned above, we believe our approach offers a unique configuration that combines the benefits of external analysis tools and the all-in-one modelling capabilities offered by EAST-ADL.

1.3. Contributions of the paper

In this paper, we show how earlier work on design optimisation, e.g. work developed in the context of HiP-HOPS, can be transferred to model-based design—specifically in the context of the EAST-ADL language. The novel contributions of the paper are:

- The development of a general method for automatic optimisation of EAST-ADL models via genetic algorithms. The method so far enables evaluation of unavailability, simple cost metrics, and schedulability of candidate designs to drive the optimisation, but could be extended in the future to other functional and non-functional attributes as well.
- The use (in the context of optimisation) of advanced mechanisms which allow adequate parameterisation of models and

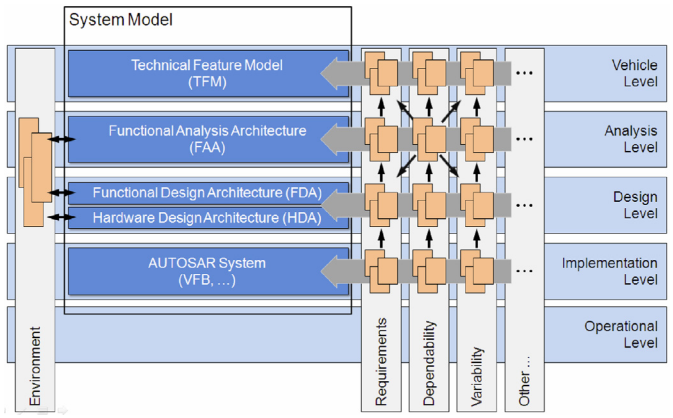

System Model
Technical Feature Model (TFM)
Functional Analysis Architecture (FAA)
Functional Design Architecture (FDA)
Hardware Design Architecture (HDA)
AUTOSAR System (VFB, ...)
Vehicle Level
Analysis Level
Design Level
Implementation Level
Operational Level
Environment
Requirements
Dependability
Variability
Other ...

Fig. 1. The EAST-ADL abstraction levels (http://www.east-adl.info/).

expression of design variability in the language where the model is expressed (i.e., EAST-ADL).
- Applicability of the approach on architectural models which are represented in multiple perspectives, i.e., separate hardware and software design diagrams which are linked with allocations of software to hardware. This is a non-trivial matter as most approaches to dependability and optimisation assume a single functional model of the system as basis for any analysis.
- The use of annotations of a core structural model to formalise timing, cost, energy consumption, etc. for the purpose of architecture evaluation as a part of optimisation. The modular approach makes it possible to provide annotations consistent with variants, and corresponding to the input needs for fitness functions of the intended optimisation. Whilst not all of these annotations are currently exploited, the optimisation architecture described in the paper is extensible and could tap further into the wealth of design information captured within an EAST-ADL model.

In the next section, we shall describe EAST-ADL, its major features, and its capabilities for linking with external analysis tools to perform safety and timing analyses (amongst others). In Section 3, we describe the multi-objective optimisation concept and the tool architecture being developed to fulfil this concept, and in Section 4 we apply the approach to a simple example system to illustrate many of the advances mentioned above (multiple perspectives, variability representation, timing and dependability information etc.). Finally we present our conclusions in Section 5.

2. Representing system architectures with EAST-ADL

2.1. Introduction to EAST-ADL

EAST-ADL is an Architecture Description Language (ADL) initially defined in the European ITEA EAST-EEA project and subsequently refined and aligned with the more recent AUTOSAR automotive standard (www.autosar.org). Currently, it is maintained and evolved by the EAST-ADL Association (www.east-adl.info).

EAST-ADL is an approach for defining automotive electronic systems by way of a comprehensive information model that captures engineering information in a standardised form. Aspects covered include vehicle features, functions, requirements, variability, software components, hardware components and communication.

As a guiding principle, EAST-ADL defines multiple abstraction levels and distributes all development information across these levels (see Fig. 1). Thus, software- and electronics-based functionality of the vehicle is described at different levels of abstraction during the development from early analysis to implementation. The proposed abstraction levels are tailored to provide a separation of concerns and an implicit style for using the modelling elements. The embedded system is complete on each abstraction level, and parts of the model are linked with various traceability relations. This makes it possible to trace an entity from feature level down to components in hardware and software.

The features in the main technical feature model at vehicle level represent the major functionalities and characteristics of the complete system, i.e., the entire vehicle from a top-level perspective, without exposing any realisation details. It is possible to manage the content of each individual vehicle and entire product lines in a systematic manner. In addition to this main feature model with its technical perspective, other feature models may be defined that provide views on the main feature model, e.g. a customer/marketing-oriented view defining models and packages of optional equipment.

A complete representation of the electronic functionality in an abstract form is modelled in the Functional Analysis Architecture (FAA). The purpose is to define and structure the system's functionality from a problem domain perspective, similar to a traditional functional analysis. Entities of the FAA, so-called analysis functions, capture the principal interfaces and behaviour of the vehicle's subsystems.

The solution domain aspects are introduced while defining the design level, which comprises the Functional Design Architecture (FDA) and the Hardware Design Architecture (HDA). The FDA defines a function architecture that takes into account hardware allocation, efficiency, legacy and reuse, commercial-off-the-shelf (COTS) availability, and other architectural qualities. The function structure is such that one or more functions can be subsequently realised by one or several AUTOSAR software components (SW-C). The external interfaces of such components correspond to the interfaces of the realised functions. On the other hand, the HDA defines the execution environment in which the software will be deployed. Its main entities are sensors, actuators, units of execution, their interfaces, and communication links between all these. Finally, the allocation joins the FDA and HDA by defining for each software entity in the FDA on which unit of execution, as defined in the HDA, it will reside.

On the lowest, most concrete abstraction level, the implementation level, EAST-ADL does not provide its own modelling entities; instead, this level is entirely defined by models from the AUTOSAR standard. In this respect, EAST-ADL can be thought of as an extension to AUTOSAR providing support for modelling on higher abstraction levels during earlier development phases. However, traceability is supported from implementation level elements (AUTOSAR) to FDA/HDA elements and, from there, further up to vehicle level elements.

Verifying and validating a feature across all abstraction levels, by using simulation or formal techniques, requires an environment model from the early stages of the design process. This "plant model" captures the behaviour of the vehicle dynamics, driver, etc. The core part of the environment model can be the same for all abstraction levels, which means that, for the purpose of simulation, both FAA and FDA/HDA are attached to the same environment model.

2.2. Variability modelling in EAST-ADL

EAST-ADL provides extensive support for managing variability, which is a key element of the EAST-ADL-based optimisation approach presented in this article. First, variability modelling is used to define the optimisation space (also referred to as the "architectural degrees of freedom" in Aleti et al. (2012)) and then configuration and variability resolution are applied to produce the individual optimisation candidates to be evaluated with respect to the optimisation objectives. Before going into more detail on

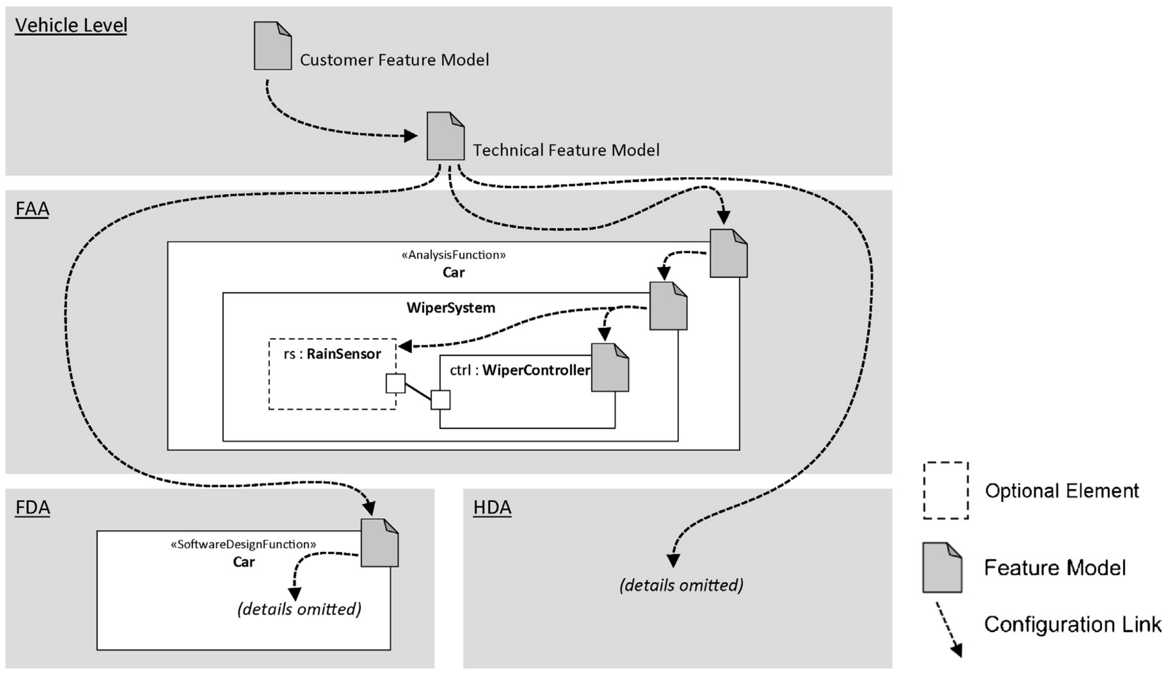


Fig. 2. Feature models on different EAST-ADL levels and the configuration links connecting them.

the optimisation approach in Section 3, we therefore describe the EAST-ADL variability management concepts here.

Modelling of variability in EAST-ADL starts on the vehicle level, where model range features and variability are represented. At this point, the purpose of variability management is to provide a highly abstract overview of the variability in the complete system together with dependencies between these "variabilities". A *variability* in this sense is a certain aspect of the complete system that changes from one variant of the complete system to another. In this abstract overview, the idea is not to specify how the system varies with respect to an individual variability but only that the system shows such variability. For example, the front wiper may or may not have an automatic mode. At vehicle level, the impact of this variability on the design is not defined; only the fact that such variability exists is defined by introducing an optional feature named "RainControlled-Wiping". This is subsequently validated and refined during analysis and design.

One or more feature models may be defined on the vehicle level: the core Technical Feature Model—the main feature model in an EAST-ADL system model—is used to define the complete system's variability on a global level from a technical perspective, and one or more optional Product Feature Models can be used to define views on this technical variability which can be tailored to a particular view-point or purpose, e.g. the marketing-oriented end customer perspective. An example of this is shown at the top of Fig. 2.

While the details of how variability is actually realised in the system are largely suppressed at the vehicle level, they are the focus of attention when managing variability in the FAA, FDA and HDA. In fact, a specific variability may lead to modifications in any development artefact, such as requirements specifications and functional models. Here, describing that a specific variability occurs is not sufficient; it is necessary to describe how each variation affects and modifies the corresponding artefact.

The purpose of feature modelling is to define the commonalities and variabilities of the product variants within the scope of a product line. Feature models are normally used on a high level of abstraction, as described above for vehicle level variability. However, in EAST-ADL, they are also used on analysis and design levels and acquire a much more concrete meaning there: they can be attached to analysis and design functions and are then used to expose the variability within this function only. Fig. 2 illustrates this for the FAA: the top-level analysis function "Car" contains a subfunction "WiperSystem" which in turn consists of the subfunction "RainSensor" and "WiperController"; all the analysis functions "Car", "WiperSystem" and "WiperController" have a feature model attached.

In addition to their feature model, analysis and design functions may contain variation points defining how the function's structure varies from one system configuration to another. In EAST-ADL, such structural variation can be represented by marking structural entities—e.g. subfunctions, ports, connectors—as optional, which is denoted graphically by a dashed line. Fig. 2 presents an example: the "WiperSystem" has an optional subfunction "rs" of type "Rain-Sensor". All actual system variation on analysis and design level is defined using this straightforward concept of optional structural entities.

Configuration decision modelling is another key variability management concept in EAST-ADL: the configuration of a target feature model FM_T—i.e. the selection and deselection of its features—is defined in terms of the configuration of another feature model FM_S, called source feature model. A configuration decision model can thus be seen as a directed relation from FM_S to FM_T that allows us to derive a configuration of FM_T from any given configuration of FM_S. In EAST-ADL, this mechanism is used to define how a certain configuration on a higher level affects the binding of variability on lower abstraction levels and in lower-level components. Again, Fig. 2 provides an example: configuration links are depicted as dashed arrows and are defined between all the feature models; the one from the "Technical Feature Model" to the feature model attached to "Car", for instance, defines how to configure the "Car" analysis function depending on a given configuration of the core technical feature model on vehicle level; note how this configuration link crosses abstraction layers while the links from the "Car" to the "WiperSystem" feature model crosses containment hierarchies within the analysis architecture (the same applies to the FDA and HDA, but this was omitted from the figure). Similarly, configuration links are used to define how the structural variability within analysis and design functions is resolved depending on a given configuration of the function's feature model. For example, the configuration link in the "WiperSystem" function of Fig. 2 states how the optional subfunction "rs" will be selected or deselected depending on the configuration of the wiper system's feature model.

Variability management on analysis and design level is driven by the variability identified on the vehicle level. This means that the main driver for variability definition and also variability instantiation is the vehicle-level feature model. Variability specification in FAA and FDA/HDA essentially consists of the definition of *variation points* within the analysis and design functions (in the form of optional subfunctions, connectors and ports). As mentioned above, feature models can be attached to functions in order to expose the variability within these functions and hide the actual structuring, representation and binding of this function-internal variability. This way, the benefits of information hiding can be applied to the variability representation and variability binding within the containment hierarchy of functions in the FAA and FDA/HDA of EAST-ADL.

In summary, EAST-ADL provides the means to define variant-rich systems and to organise this information such that an entire, fully resolved system configuration can be derived automatically from the configuration of a single feature model on vehicle level. For the purpose of this article, this means that we can define the optimisation space, i.e. the set of systems to be evaluated, by way of EAST-ADL's variability modelling concepts and then produce individual candidates within this optimisation space by applying EAST-ADL's configuration and variability resolution mechanisms.

2.3. Dependability analysis of EAST-ADL models with HiP-HOPS

EAST-ADL offers powerful fault modelling capabilities centred on its ErrorModel concepts. The ErrorModel in EAST-ADL is a separate modelling view, parallel to the nominal system models on each level, and allows system designers to specify how the system elements can generate failures and how those failures can propagate to other parts of the system.

As with other parts of EAST-ADL, the ErrorModel only stores information; analysis capabilities have to be provided by an external tool. As part of the ATESST2 and MAENAD projects, significant effort has gone into enabling the HiP-HOPS (Hierarchically-Performed Hazard Origin and Propagation studies) safety analysis tool to analyse EAST-ADL models by means of model transformation technology. This has entailed some harmonisation of the error modelling concepts in both EAST-ADL and HiP-HOPS and the result is a powerful dependability analysis capability for EAST-ADL models.

A HiP-HOPS analysis has three main stages:

- Fault modelling and failure annotation
- Synthesis of fault trees to model the propagation of failure through the system
- Fault Tree Analysis (FTA) and synthesis of Failure Modes and Effects Analysis (FMEA) tables

The first phase is manually performed and consists of annotating the elements of the system model (or in this case, the EAST-ADL error model) with logical expressions that describe their local failure behaviour, i.e., what output failures they propagate (known as *output deviations*) and how those output failures are caused by a mixture of input faults and internal failure modes. In their basic form, these expressions take the form of a failure class (e.g. commission, omission) and the name of the port and component where the deviation occurs and combines them with logical operators such as AND and OR. For example:

```
Omission-Actuator.out = Omission-Actuator.in OR
                         ActuatorStuck
```

indicates that an omission of the 'out' port of an actuator component is caused by either a similar omission received at the 'in' port of the Actuator or an internal failure mode called "ActuatorStuck", which may also have probabilistic failure data (e.g. failure rate, repair rate, MTTF etc.) defined. Such expressions can take more complex forms to express generalised patterns of component failure behaviour.

The second and third phases are conducted automatically by HiP-HOPS. First, a network of interconnected fault trees is synthesised by combining the logical expressions, thereby showing the relationship between system hazards and combinations of individual component or function failures. In the third phase, the fault trees are analysed using FTA reduction algorithms to obtain the minimal causes of system failures together with estimates for the probabilities of those failures. This information is used to construct FMEA tables which show not only the effects of each individual failure mode, but also the effects of multiple failure modes occurring in conjunction.

2.3.1. Multi-perspective analysis in HiP-HOPS

One of the biggest differences between the HiP-HOPS error model and the EAST-ADL approach is that EAST-ADL does not limit the system modeller to a single architecture, as HiP-HOPS does. Not only are the ErrorModel and nominal model separate, but EAST-ADL provides multiple layers or levels of potential modelling as well as different views or *perspectives* of a model. HiP-HOPS can be applied to EAST-ADL models at both the analysis and design levels, each of which may contain a number of different perspectives. For example, at the analysis level, the primary modelling perspective is the Functional Analysis Architecture (FAA), which provides an abstract view of the functions of the system and which will have a similarly abstract Error Model attached. Later, at the design level, the model may be separated into the Functional Design Architecture (FDA), which represents the concrete functional perspective, and the Hardware Design Architecture (HDA), representing the hardware perspective, and each of these may have a separate—but interrelated—Error Model.

In HiP-HOPS, software and hardware are usually modelled together as part of the same architecture, with software functions being defined as subcomponents of hardware elements. This can be emulated in EAST-ADL by combining everything into a single perspective, but the advantages of having multiple perspectives are then lost. Instead, in order to make it compatible with EAST-ADL and to allow for the propagation of failures from software to hardware or vice versa, HiP-HOPS has been extended with native multi-perspective capabilities, as shown in Fig. 3.

In EAST-ADL, hardware and software entities can be in separate architectures (e.g. HDA and FDA) and the relation from one perspective to the other is accomplished by means of *allocation* relationships, in which software functions are allocated to the hardware components that execute them. Failures of the hardware components will propagate to the software functions allocated to them. HiP-HOPS now provides the same capabilities, and in addition to an output deviation being caused by an input deviation or an internal failure mode, it may now also be caused by a failure propagated from the component the current function is allocated to. For example:

```
Omission-Function.out = Omission-Function.in OR
                        FromAllocation
                        (PowerFailure)
```

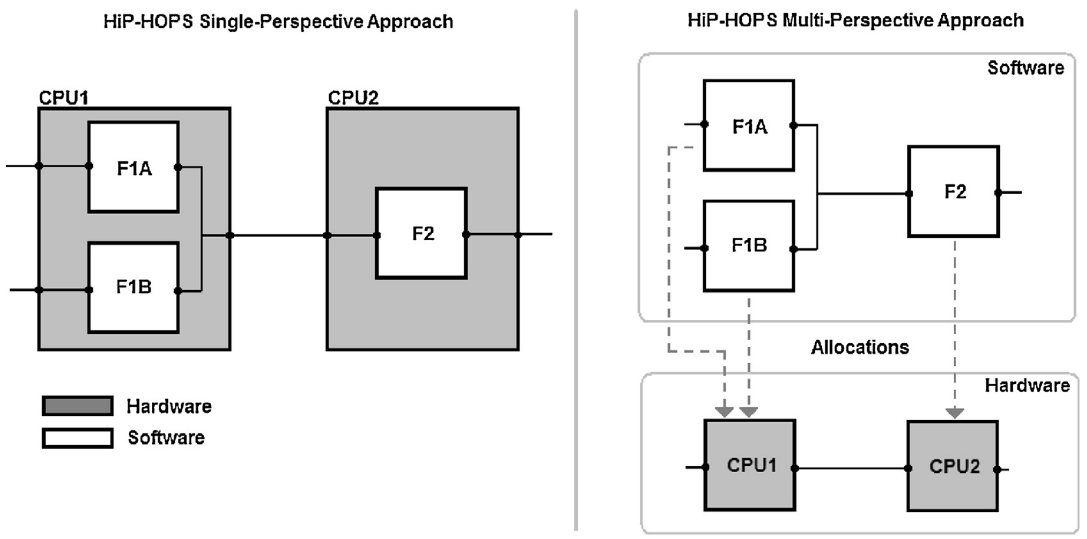


Fig. 3. Multi-perspective modelling in HiP-HOPS.

This specifies that the omission of output from a software function is caused either by a lack of input or a particular failure (power failure in this case) being propagated from the hardware component that the function is allocated to.

Furthermore, in both EAST-ADL and now in HiP-HOPS, it is possible to define *more* than one possible allocation, e.g. using variability constructs in EAST-ADL. This allows scope for potential optimisation of the model on the basis of changing the allocation of software functions to different hardware components.

The introduction of multiple perspectives to HiP-HOPS has necessitated a change to the semantics of common cause failures, which HiP-HOPS previously treated as global failures. Whereas before common cause failures (CCFs) were globally accessible throughout the model, and could cause a failure of any component, now CCFs are defined per modelling perspective, and thus a CCF defined for the hardware perspective (e.g. flooding, fire) would not be accessible from a function in the software perspective, for instance.

For any rare cases where arbitrary cross-perspective propagation is required, HiP-HOPS provides a 'Goto' declaration. This allows failures to propagate from one component in one perspective to another component in a different perspective (or, for that matter, in the same perspective), even if those components do not share an allocation relationship.

Because these new connections (allocations and gotos) connect to existing HiP-HOPS constructs (namely, output deviations and the normal local failure logic), they fit relatively seamlessly into the HiP-HOPS fault tree synthesis process. Once the fault trees are generated, they can be analysed as normal without any further distinction between one perspective and another.

This harmonisation of EAST-ADL and HiP-HOPS error modelling concepts provides a powerful dependability analysis capability for EAST-ADL models, allowing fault propagation to be traced across several different modelling views. This also enables optimisation of EAST-ADL models using dependability characteristics (e.g. safety, unavailability) as objectives, as shown in Section 4.

2.4. Timing analysis of EAST-ADL models

EAST-ADL, and specifically its Timing Extension, provides a rich set of concepts to support timing analysis. The term "timing analysis", however, encompasses a wide range of analyses applicable to different models (and levels of abstraction) produced during the development process. High-level analyses, such as using model-checking for the verification of the timing properties of the functional model, can usually be employed from the very beginning of the development process, as soon as functional behaviours are defined. Conversely, low-level analyses, such as task level schedulability analysis, compute timing properties of functions when conceptually executed on detailed software/hardware resource models. These resource models include information about processor/bus schedulers and task/message configurations and are in general produced at later stages of the development process.

In this paper we are primarily interested in the design level of EAST-ADL, where the allocation of functions on hardware nodes is defined and timing analysis can be applied in conjunction with dependability analysis. Specifically, the design-level concepts used by the analysis are:

(1) activation chains of functions, the rate of their activation, worst case execution times of functions and end-to-end deadlines;
(2) the topology of the hardware network in terms of processors and the buses that connect processors;
(3) the allocation of functions on processors.

These concepts come from EAST-ADL's FunctionModelling, HardwareModelling and Timing aspects, and are specified in the FDA, the HDA, and the associated Timing View respectively.

Timing analysis at this stage aims to evaluate the impact that hardware resources may have on the execution of functions, although it makes use of more abstract resources than those used by schedulability analysis (no tasks, schedulers, etc.). To give a simple example on the importance of evaluating the resource impact on function execution, let us consider the architecture in Fig. 3. In this example, Functions F1A and F1B do not have precedence dependencies, so they can be logically executed in parallel. As a comparison, assume that the two functions are allocated on the same CPU, meaning that F1A and F1B are executed sequentially. Now let us consider that Function F2 must produce a value for each cycle of duration T (deadline). In each cycle F2 must consume at least one value produced by F1A and one produced by F1B in the same cycle. It is clear that if the two functions could be executed in parallel and start at the beginning of the cycle, input values for F2 will be available after $max(C1, C2)$, where $C1$ and $C2$ are estimated worst case execution times. On the other hand, in case of sequential execution, the input values for F2 will be available only after $sum(C1, C2)$. These situations must be carefully analysed as

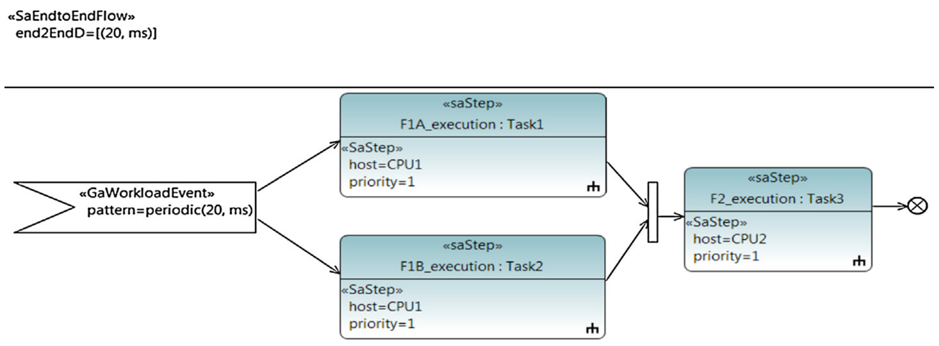


Fig. 4. MARTE task model.

the effect of accessing hardware resources may lead to response times violating deadlines.

To this end we propose an approach that provides a connection to the MAST analysis tool and gives automatic feedback to the EAST-ADL model. As stated in Section 1.1, we need to solve the mismatch between the schedulability analysis level (where the MAST tool works) and the EAST-ADL design level. To this end, we devised a controlled refinement towards a task model that ensures the MAST results provide valuable insight into the EAST-ADL level. It should be noted that this refinement may be embedded in a model transformation that is transparent to the user.

The controlled refinement performs a 1:1 mapping of EAST-ADL functions to MARTE tasks. A task must be configured in terms of periods and priorities; the controlled refinement assigns to a task the period of the function it is mapped to, and all priorities are fixed to the same value. The effect of assigning the same level of priorities for all the tasks (functions) allows for the computing of the upper bound on worst case response times, i.e., any other priority assignment will improve response times. This effect is explained by the way in which response times are computed by MAST when priorities are at the same level.

To clarify this point, consider the example in Fig. 3 again. Applying our controlled refinement to this model means obtaining a MARTE model as shown in Fig. 4. Each function is here represented by a task executing the function itself at the rate of 20 ms, expressed by an external trigger. Priorities and allocations to CPUs are also shown. The deadline of 20 ms for the entire cycle is also specified.

Once the EAST-ADL model is transformed into a MARTE model, MAST can be applied to compute the response time. The response time is obtained by considering worst case execution times and the additional delay caused by sequencing F1A and F1B on CPU1 (for sake of simplicity we do not consider communication delays in the example). In fixed-priority schemes, the additional delay due to sequencing executions is called *pre-emption time*. Remember that under ordinary circumstances, the highest priority task will access the CPU first and lower priority tasks will access after higher priority tasks terminate their execution. The time spent by a task waiting for higher priority tasks to execute is the pre-emption time. The global pre-emption time is the sum of all pre-emption times for all tasks. Obviously, to improve response time, the global pre-emption time should be minimised.[2] But what happens if priorities are set to the same level, as in our case? MAST in this case considers that each task is pre-empted by all other tasks. In case of Fig. 4 the global pre-emption time on CPU1 will be $sum(C1, C2)$. Any other assignment considering different levels of priority will give a lower value for the global pre-emption time on CPU1 ($C1$ or $C2$).

[2] It should be noted that the worst case, in which all the tasks are activated at the same time, is assumed by the analysis, so that the pre-emption time is always included in the response time.

The global pre-emption time could be decreased by reducing the number of tasks. In the example, an alternative mapping with the two functions on a single task would mean reducing the pre-emption time on CPU1 to 0, as only one task runs in CPU1. Indeed, the 1:1 mapping is another source of pessimism that gives us the worst response time, independent of improvements that can be achieved later by applying smarter refinements towards alternative task models.

Once it has been established that the response time can only be improved by subsequent refinements, it can be correctly interpreted in EAST-ADL as a sufficient condition. A response time that meets deadlines at the task level implies that the EAST-ADL architecture can be considered valid. On the other hand, a negative result does not necessarily imply a design error for schedulability. Experience suggests that this does not present a problem if the designer is aware of this characteristic. Furthermore, a complementary and less conservative analysis could be made where a perfect priority assignment is assumed. This would serve to assess feasibility of the architecture.

The response time computed with MAST under the proposed controlled refinement allows us to compare different architectures from a timing point of view, enabling architecture optimisation as described in Section 4.

3. Optimisation of EAST-ADL architectures

Model-based systems analysis techniques allow a great deal of information to be obtained about a system, including its dependability and timing characteristics, amongst others. This is especially true when these analysis techniques are combined with ADLs like EAST-ADL, which serve to centralise all the knowledge about a particular system, since it is no longer necessary to produce a specific error model tailored for a particular dependability analysis tool, or a separate timing model solely for timing analysis etc.

However, as mentioned earlier, this wealth of information can also be difficult to manage. Achieving a balance between the different attribute requirements during the development of complex systems becomes problematic, both because there is a vast number of possible design alterations that could be made and because the interrelationships between the attributes—and therefore the effects of any alterations—are not necessarily clear. This is particularly true when tradeoffs between multiple conflicting attributes (like safety versus cost, or energy consumption versus performance) are involved, as improving one attribute may lead to a worsening of another.

The increasing automation of analysis techniques helps overcome this to an extent, assuming the automated tools are compatible with the model in question. Such tools allow models to be rapidly evaluated according to different criteria, enabling designers to quickly see the effects of any changes to the design architecture and informing design decisions as part of an iterative process.

Even with this automation, evolving the design manually is still a complex and time-consuming process, relying on manual experimentation with different design options, and may still be difficult to fulfil the competing requirements on the system—particularly as the time and effort involved means that typically only a small number of potential design options can be investigated in this way.

In such situations, automatic architectural optimisation can potentially be applied. Automatic optimisation allows a much more efficient search of the design space (the total set of possible design variations), covering a much greater number of alternatives than could be considered manually. However, to be effective, the optimisation must be guided in some way, which requires some form of heuristic evaluation. It is here that the use of ADLs such as EAST-ADL for representing the design model pays dividends, because such a model contains all of the information necessary for analysis to take place with respect to each system attribute being considered. For example, if the objectives of the optimisation were to maximise safety and performance while minimising cost and unavailability, it has to be possible to evaluate each of the candidate designs being explored according to those four characteristics by means of some form of analysis. By making changes to the design model automatically as part of the optimisation process and allowing the analysis tools to access the same common model, it then becomes possible to evaluate the new variant directly and seamlessly, facilitating a rapid, efficient search of the design space.

3.1. Multi-objective optimisation

Taken together, the problem of optimising a large potential design space to obtain one or more good solutions that feature a desirable balance of attributes is known as a *multi-objective optimisation problem*. There are a variety of different automated algorithms that can be employed to solve such problems, but in general they all aim to quickly find viable solutions (i.e., valid design variants in this case) that offer optimal or near-optimal attributes without needing to investigate all possible designs. It should be noted that in a multi-objective optimisation problem, there is not typically a single optimum solution, because the different objectives may be mutually exclusive; instead, the goal is often to produce a set of 'optimal' solutions that feature a range of attributes that balance the different objectives in different ways—in other words, each providing a different tradeoff between the objectives whilst still meeting any constraints.

These are known as the *Pareto solutions* and are based on the concept of *dominance*. A dominant solution is one that is better in at least one objective than every other solution yet found, and no worse in the other objectives. The set of dominant solutions is known as the *Pareto set*. When plotted on the graph, they tend to form a curve known as the *Pareto frontier*, which shows the current progress of the optimisation; new solutions beyond the Pareto frontier are new optimal solutions and will dominate older solutions, whereas solutions within the frontier are dominated and non-optimal. This is shown in Fig. 5, where the Pareto frontier consists of the solutions marked as black dots, whilst dominated solutions are white dots.

In Fig. 5, solutions are shown as dots plotted against two axes, each representing different objective attributes (e.g. unavailability and cost). In this case, the goal is to minimise each attribute, so lower values are better. Thus A dominates B because it has lower values for each attribute; it is better in both respects. A also dominates C, because although they have the same value for one objective (e.g. unavailability), A has a lower value for the other (e.g. it is cheaper). Conversely D is a non-dominated solution because it is better than A in one objective but worse in another; for example, D may be cheap but not very reliable, whereas A is more reliable but also more expensive—both are equally valid solutions.

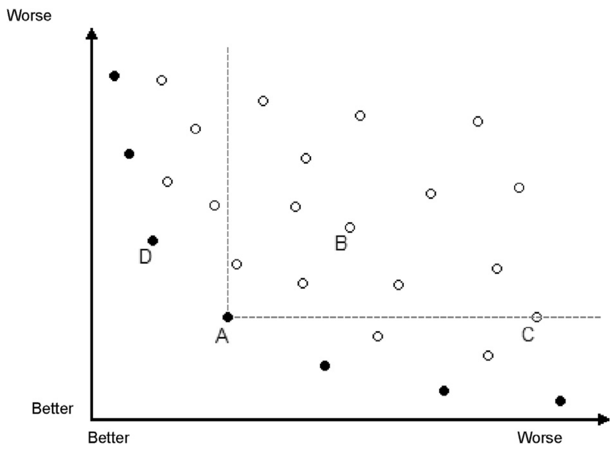


Fig. 5. Pareto frontier.

Although there are many different multi-objective optimisation algorithms, e.g. tabu search (Kulturel-Konak et al., 2003), ant colonies (Liang and Smith, 2004), and simulated annealing (Kim et al., 2004), one of the most prominent approaches is the use of genetic algorithms. The general principles will be briefly discussed next.

3.1.1. Genetic algorithms (GAs)

Genetic algorithms are optimisation algorithms inspired by the evolutionary processes found in nature. They have been identified as a particularly effective method for solving combinatorial optimisation problems (such as those discussed here) and they are capable of navigating large, complex search spaces (Coit and Smith, 1996a).

The general process is as follows:

- A *population* (set) of individual candidate solutions is randomly generated. Each individual is represented by a different *encoding*, analogous to human genes, which encapsulates the way they can vary from each other. The encoding can be thought of as the 'DNA' of each individual.
- Genetic operators—crossover and mutation—are applied to the population to obtain a new generation of child solutions, which are added to the population (or, in some genetic algorithms, become the sole new population). Crossover involves mixing the genes of two random parent candidates to produce a child; mutation involves randomly changing the genes of a candidate to produce a new solution (or alter an existing candidate).
- This process of reproduction continues until the population grows too large, at which point those with the least successful genomes are discarded. This is established by evaluating the solutions in the population according to the objective criteria (the evaluation functions are known as *fitness functions*) and retaining those with the best results—in other words, survival of the fittest.
- After a set number of generations, the optimisation ceases, and the current population should contain the best (i.e., optimal) solutions discovered so far.

The nature of the problem domain and context has a large influence on the nature of the genetic algorithm, and consequently there are many versions of genetic algorithms, each of which can be customised further by means of different parameters. Generally, however, they all share the same major features: a population of individuals represented by some form of encoding, crossover and mutation operators to produce new individuals, and some kind of fitness function to evaluate the individuals.

Encodings, which identify the characteristics of the individual solutions, can take many different forms. One of the simplest is to use a string of integers: each position of the string represents a different characteristic, and the value at that position determines the nature of the characteristic for that individual. For example, a simple system of three components connected in series could be represented by a string of three integers, and the value of each integer indicates the version or manufacturer of each of the three components. While simple, this type of encoding is also quite limited, so other forms of encoding are possible, such as hierarchical (i.e., tree-based) encodings which mimic the hierarchical structure of the system architecture.

GAs also differ in how they handle multiple objectives (assuming they do at all). One common approach is to combine the different evaluations into a single value by applying a weighting to each objective—in effect, converting a multiple-objective problem into a single-objective problem. So for example, if the objectives were safety, performance, and cost, each would be normalised and weighted—e.g. 0.4× safety, 0.3× performance, 0.3× (inverse of cost)—and the result of the formula used as the fitness for that individual. However, this is not always desirable (since it can hide useful tradeoffs) and the fitness weightings can be very difficult to derive.

Another approach is the penalty-based GA approach. In this approach, one objective is treated as the main objective and minimised (or maximised)—e.g. cost—and other objectives are treated as scaled, weighted constraints and applied as a penalty to the main objective. For example, a maximum unavailability limit might be set, and the more the design candidate exceeds the limit, the more harshly the fitness is penalised. Thus even if two solutions both achieve the same cost, the one that does not violate any constraints—or which violates them by the smallest degree—is preferred by the algorithm. However, because successful exploration of the design space often requires infeasible solutions (i.e., those that break the constraints) to be explored too, a dynamic penalty function can be employed that varies the penalty according to the length of the optimisation run so far (e.g. applying light penalties early in the process, to encourage more exploration, and gradually making them heavier as time goes on). This form of dynamic penalty-based GA has been found to have superior performance compared to either static versions or versions that only allow feasible solutions to be part of the population (Coit and Smith, 1996b).

There are also true multi-objective approaches that evaluate against multiple objectives at once and maintain a Pareto front of optimal, non-dominated solutions. This allows a broader set of solutions and generally results in a better coverage of the search space (Salazar et al., 2006), albeit at the cost of higher complexity. These types of algorithms are generally better suited to multi-objective problems because they allow the various tradeoffs between the objectives to be better represented, rather than being biased towards one objective and using the others as constraints or weighted penalties etc. There are many different multi-objective GAs, including:

- PESA-II (Pareto Envelope-based Selection Algorithm 2) (Corne et al., 2001), which seeks to maximise evenly distributed spread in the Pareto set by including crowding in the selection criteria; solutions that are found in less crowded regions of the search space are preferred for selection to encourage the algorithm to look at under-explored areas of the search space.
- SPEA-II (Strength Pareto Evolutionary Algorithm 2) (Zitzler et al., 2001). Unlike PESA-II, which adopts a "pure elitist" strategy that allows no dominated solutions to be retained, SPEA-II does keep some dominated solutions. The selection criteria are altered to include a dominance strength (how many solutions it dominates) and a density value (how close an individual is to other solutions). Thus although dominated, crowded solutions are possible, non-dominated, non-crowded solutions are preferred.
- NSGA-II (Non-Dominated Sorting Genetic Algorithm 2) (Deb et al., 2002). In NSGA-II, both non-dominated and dominated solutions are kept in the same population, ranked according to dominance (based on how many other solutions they dominate). More dominating solutions are preferred during selection, and crowding is used as a tie-breaker if two solutions have equal dominance.

3.1.2. Chosen algorithm

After comparing the three algorithms above, the algorithm chosen for use in the ATESST2 and MAENAD projects for the automatic optimisation of EAST-ADL models is a variant of the NSGA-II genetic algorithm, which was deemed to have a good balance of characteristics. NSGA-II is also used as the basis of the PerOpteryx optimisation engine (Koziolek et al., 2011) and is also used in AQOSA (Etemaadi and Chaudron, 2012). NSGA-II offers reasonably good performance, wide coverage of the design space, and (most importantly) excellent support for multiple objectives (Parker, 2010). The algorithm is not guaranteed to find all Pareto optimal solutions (though nor are any of the other algorithms, genetic or otherwise), but it should still produce a set of superior solutions much faster and more efficiently than could be performed manually.

The encoding used is a form of hierarchical, tree-based encoding, where each tree node represents a different variability point for one or more functions or components in the system architecture. If an optional function is present and also has a choice of subfunctions (or subcomponents), then the node will have corresponding child nodes beneath it which can also be varied.

The different values of the encoding represent all possible variations of the system architecture—in effect, the encoding encapsulates the entire design space. It is defined in EAST-ADL by means of the variability management mechanism, allowing more complex configurations of features to be represented, including dependencies between features (possibly hierarchical) and duplication of features. Variability is therefore used to represent the possible options for architectural variation of the design by defining the variation points of the design space; this notion of optimisation space corresponds exactly to what is called a configuration space in feature modelling terminology. Thus for example a critical component can be defined with several implementations or perhaps an alternative replicated subsystem specified (which provides additional redundancy, but increases cost); the optimisation algorithm can then choose one of these options and evaluate the effects of that option on the overall attributes of the system.

Optimisation variants can be defined in one of three main ways:

1. *Substitution*: A component/function (or subsystem) is substituted for another that has different objective attributes (e.g. safety, cost, performance). A substitute must be functionally equivalent, i.e., it must perform the same task and have a compatible interface, but need not achieve the task in the same way or make use of all connections (e.g. an electronic braking subsystem may be replaced by a hydraulic version). Different substitutes are often termed as different *implementations* of a function or component.
2. *Replication*: To improve reliability, a critical component or model element may be duplicated to achieve redundancy. Replicants are usually connected in parallel to ensure that a failure of one element does not lead to a failure of the whole subsystem. Note that the replicants have to be defined in advance as optional elements; the optimisation does not generate them itself.

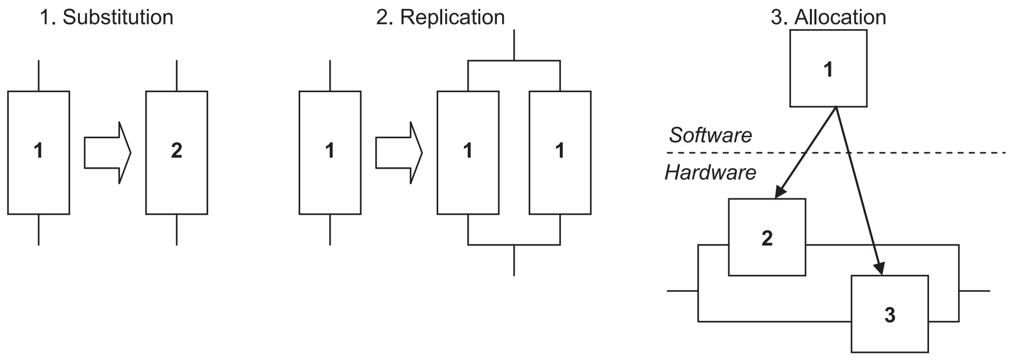


Fig. 6. Different variation strategies.

3. *Allocation*: In systems with both hardware and software elements, there may be a choice of allocation strategy, e.g. in terms of which software function is allocated to be executed on which hardware platform. This allows timing performance as well as reliability to be balanced over the available processing resources of the system.

These different variations are illustrated in Fig. 6. Furthermore, more complex optimisation is possible via a combination of two or more of the above, e.g. replication that uses more than one implementation.

During the optimisation process, different variants can then be chosen from amongst the predefined choices, and the variability mechanisms ensure that the connections between the chosen components are kept consistent. This enables relatively seamless connections between different choices of functions/components in the model and ensures that e.g. failures are propagated correctly through the system model.

This variability-based approach is very flexible and allows complex compositional variants to be modelled; for example, a single component may be substituted for another component from a different manufacturer, or replaced by a replicated subsystem with heterogeneous subcomponents. This allows more sophisticated design patterns (such as voters and monitors etc.) to be evaluated as part of the optimisation; furthermore, such configurations can often be stored in a library and reused in other models.

3.2. EAST-ADL optimisation tool architecture

This section describes the EAST-ADL optimisation tool architecture as currently being developed in the MAENAD project. The tool architecture is designed to meld all of the various analysis and optimisation tools required, using the EAST-ADL system model as the glue. A prototype optimisation environment has been developed based on the EPM tool (based on the Eclipse framework), which currently implements a core subset of EAST-ADL (Abele et al., 2012).

There are three main goals to be fulfilled by the tool architecture:

- *Design space definition:* In order for optimisation to take place, it must be possible to define the design space by means of specifying variability points in the model that the optimisation can later choose and evaluate. This is primarily achieved through the use of EAST-ADL's variability management mechanism, which allows alternative implementations and allocations to be specified. In the tool architecture, this role is fulfilled by the EPM tool.
- *Rapid objective evaluation:*For optimisation to be guided rather than random, it must be possible to evaluate new solutions to determine whether they are better or worse than previous solutions. In a multi-objective problem, this means analysing the solutions (in the form of design candidates) for each of the objectives being evaluated. In this case, HiP-HOPS is the primary tool to evaluate safety and reliability characteristics, cost (currently only a simplistic summation) is handled by a dedicated plugin, and timing analysis capabilities are provided by the MAST tool. Other plugins can be added as needed to evaluate other objectives.
- *Efficient design space exploration*: Finally, optimisation requires that the design space be explored by means of an algorithm. In this case, a variant on the NSGA-II genetic algorithm is used, based on prototype optimisation technology in HiP-HOPS but implemented as an EPM plugin instead. In combination with the design space definition and analysis capabilities, it allows automatic optimisation of an EAST-ADL architecture.

An additional crucial step is *variability resolution*. In order for a design variant to be analysed and evaluated, the variability first has to be 'resolved', i.e., to produce a model in which all of the variability points have been selected or deselected to produce a particular system configuration with a concrete set of objective attributes.

There are still certain issues and constraints to be dealt with. For example, EAST-ADL does not require a one-to-one mapping between its nominal and error model architectures, i.e., it does not ordinarily require that each function has a corresponding error model type; however, for the error model to be evaluated correctly when the nominal architecture changes, the optimisation requires a one-to-one mapping to be applied. This is resolved by explicitly defining the ErrorModel for each nominal model variant. Furthermore, it is very important that the optimisation variability ensures *substitutability*, i.e., that if the optimisation algorithm selects a different option, the resulting system design remains valid and sensible. This imposes two requirements on the variants: that they share a compatible interface (i.e., they share at least a common subset of ports and connections) and that they perform equivalent functions. Otherwise, the optimisation algorithm may select a different option that leads to an invalid or nonsensical system design. This is not necessarily a simple problem, especially in an ADL such as EAST-ADL, where the error model, software, and hardware may all be represented in separate model views and a change in one may need to be reflected in the others. At the same time, this is a criterion for validity of the model regardless of context: it is not only optimisation which requires that a valid timing model (or error model or any other model) must emerge together with the core architecture as a result of changes to the variability; this is a general issue in any variability resolution process.

The tool architecture itself is shown in Fig. 7. This diagram shows the flow of operations in the optimisation process. The initial EAST-ADL model (with variability) is passed to the Optimisation Space Definition Module (or OSDM), which generates a Master Encoding Hierarchy. This defines the search space for the Central Optimisation Engine, which generates different model encodings that are

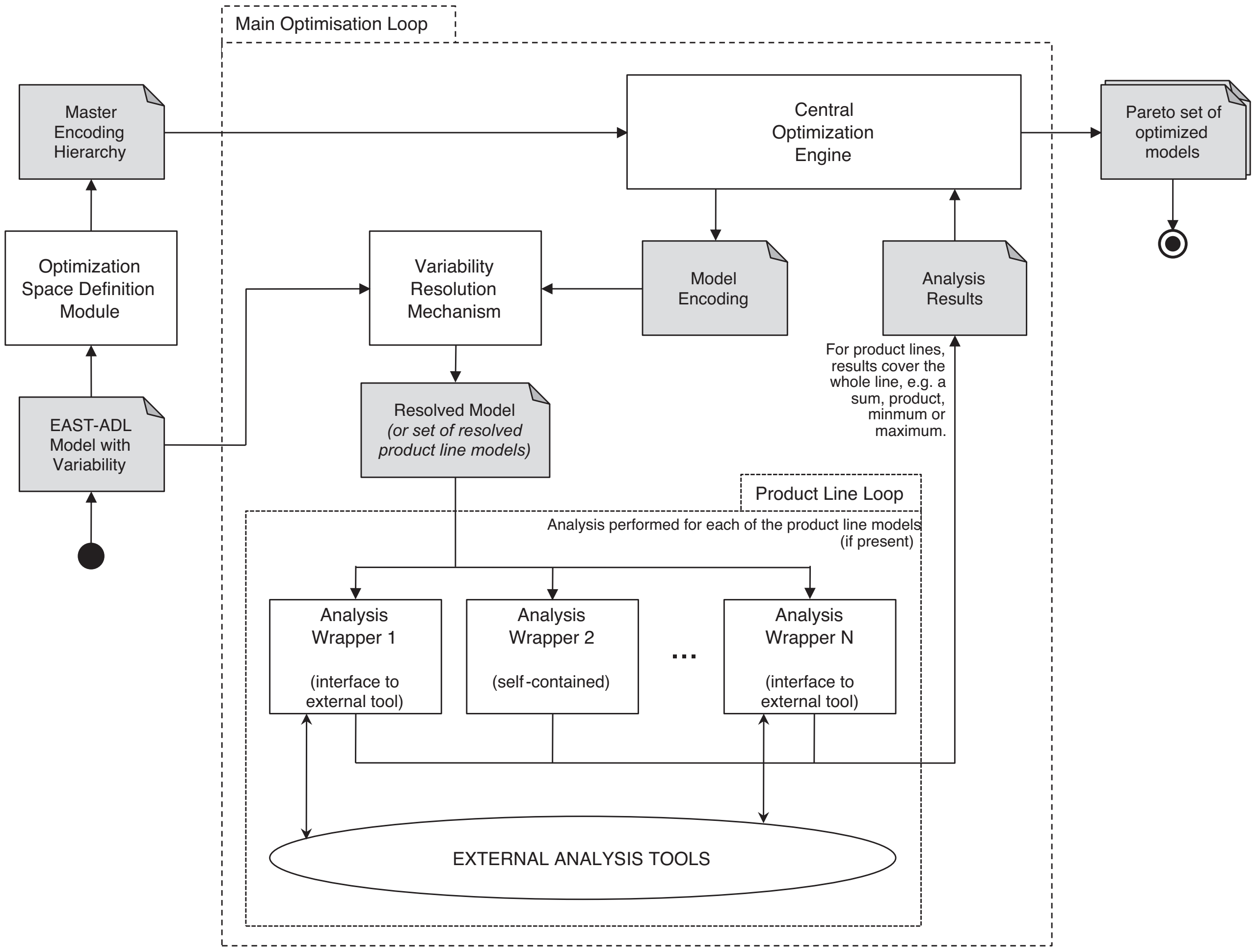

Main Optimisation Loop
Master Encoding Hierarchy
Central Optimization Engine
Pareto set of optimized models
Optimization Space Definition Module
Variability Resolution Mechanism
Model Encoding
Analysis Results
EAST-ADL Model with Variability
Resolved Model (or set of resolved product line models)
For product lines, results cover the whole line, e.g. a sum, product, minmum or maximum.
Product Line Loop
Analysis performed for each of the product line models (if present)
Analysis Wrapper 1 (interface to external tool)
Analysis Wrapper 2 (self-contained)
...
Analysis Wrapper N (interface to external tool)
EXTERNAL ANALYSIS TOOLS

Fig. 7. EAST-ADL optimisation tool architecture.

passed to the Variability Resolution Mechanism to be resolved into analysable models. The analysis wrappers then evaluate these models according to different objectives and return the results. Eventually the optimisation settles on a set of Pareto optimal solutions, which constitute the final results of the process. There is also a further planned process involving product line optimisation (see Section 3.2.5), although this is not included in the current prototype.

The major elements of the architecture will be described next.

3.2.1. *Optimisation Space Definition Module (OSDM)*

The OSDM provides the input for the overall optimisation process by taking the original "variant-rich" EAST-ADL model, which contains variability elements to define the variation points of the design, and derives from this input the set of possible encodings of the optimisation search space—the *Master Encoding Hierarchy*. This is a structure that can be manipulated to obtain the set of all valid design candidates. The Master Encoding Hierarchy allows the Central Optimisation Engine to explore the design space automatically by using an abstract structure without the need for semantic knowledge of the EAST-ADL design model itself. This is "abstract" in the sense that it does not include any information on the structure, behaviour or other properties of the individual candidates; it just provides a means to unambiguously identify each candidate. It takes the form of a tree-based hierarchical structure (based on a normal variability feature model) in which each node defines an individual variability and the nature of the node describes the type of variability. In EAST-ADL, the core technical feature model on vehicle level provides such an abstract view of the complete system's variability, as detailed in Section 2.2, and can therefore be used as is for this purpose.

The OSDM is therefore necessary to ensure that the optimisation engine itself does not need to know anything about the semantics of the models it deals with other than what variability is present in the original model and how it can be encoded. It is then the job of the VRM to convert a particular encoding—i.e., a particular configuration of the variability in the model—into a new model in which all the variability is resolved. This can then be analysed by the external tools, and the optimisation engine then determines whether to keep or discard that particular design candidate on the basis of those analysis results.

Unlike the other elements, which are used in each optimisation iteration, the OSDM is only required once, at the beginning of the optimisation, not iteratively in each optimisation cycle as with the other elements.

3.2.2. *Central Optimisation Engine (COE)*

The Central Optimisation Engine is the driver of the optimisation process, responsible for exploring the design space (by choosing which encodings to evaluate) and for collecting the best solutions so far. The COE will use genetic algorithms based on established HiP-HOPS technology (Papadopoulos et al., 2011), but is implemented as a separate plugin to facilitate easier communication with the other elements of the tool architecture.

When optimisation is initiated, the optimisation engine receives three things: a set of optimisation parameters (such as the number of generations to run for, the size of the solution population to maintain, and a specification of which objectives are to be evaluated), the original variant-rich EAST-ADL model, and the Master Encoding Hierarchy from the OSDM—a tree containing the different variation points in the model. This is essentially the base DNA of the model to be optimised. To perform optimisation, the optimisation engine chooses a particular encoding by selecting which variation points to use and which not to use according to its internal heuristics, e.g. via crossover and mutation operators (called "model encoding" in Fig. 7). This encoding is then passed on to the VRM. The VRM returns a version of the original model with the variability resolved, which is then passed to the plugin interfaces to the various analysis engines (timing, safety, cost etc.), according to the optimisation objectives. Once complete, the analysis results are returned so the optimisation algorithm can evaluate them against its objectives and then repeat the cycle.

Since EAST-ADL's core technical feature model on vehicle level is used as the master encoding hierarchy in the prototype, an ordinary feature configuration of this feature model, i.e., a selection and deselection of its features, can be used to represent a model encoding.

As mentioned above, the optimisation engine also requires the optimisation parameters, which specify the goals of the process and the objectives to evaluate. So for example, an optimisation may be initiated to minimise cost, maximise reliability, maximise processing slack, and minimise response times; alternatively, we may only be interested in minimising cost and maximising safety. Constraints can also be provided (e.g. minimum safety requirements, maximum cost etc.), although the optimisation engine will not necessarily treat these as hard limits during the optimisation process itself.

3.2.3. *Variability Resolution Mechanism (VRM)*

The Variability Resolution Mechanism (VRM) is responsible for taking the original, variant-rich EAST-ADL model and producing a new model, with all variability resolved, according to the encoding received from the optimisation engine. The basis for this step is an EAST-ADL model with variability defined. For this purpose, standard EAST-ADL variability modelling techniques, as described in Section 2.2, can be employed.

The encoding is used by the VRM to *configure* the model; essentially it produces a copy of the original model in which any unused optional structural elements (e.g. optional subfunctions, connectors, ports) are removed and selected optional elements are retained (i.e. they are no longer marked as optional). The result is a *fully resolved model* in which all variation points have been replaced by the correct variant according to the configuration defined by the encoding. This model can then be subjected to analysis for the purposes of evaluation.

As the master encoding hierarchy is represented by EAST-ADL's vehicle-level technical feature model and the model encodings are represented by ordinary feature configurations thereof (as explained above), we were able to employ standard EAST-ADL variability resolution mechanisms when implementing the VRM in the prototype. This way, the ordinary EAST-ADL variability management techniques can be used very effectively for optimisation purposes.

The consistency of the resolved model lies within the responsibility of the standard EAST-ADL variability support. With respect to structural variability, i.e. optional structural elements, EAST-ADL requires the modeller to either define the optional elements such that all permutations of the optional elements are valid or to define constraints on their selection and deselection (as part of the EAST-ADL model). In addition, several implicit dependency rules are provided in EAST-ADL to avoid the need for user-defined constraints in most standard cases. For example, if a subfunction is defined to be optional, all connectors to/from this subfunction within the containing function as well as related ports of the containing function are automatically marked as implicit optional.

3.2.4. *Analysis wrappers*

The analysis tools are responsible for analysing the model according to the objectives given to the optimisation engine. In principle there is a separate analysis for each objective, although in practical terms these analyses could be carried out by the same tool or as part of the same process (for example, HiP-HOPS will analyse safety, unavailability, and optionally a simple cost metric all as part of the same process). The results of the analysis/analyses are then fed back to the optimisation engine, which uses them to decide whether or not to retain that candidate model.

Access to each analysis tool is provided by a particular analysis wrapper, currently implemented as Eclipse plugins in our current optimisation environment. This helps ensure that there is a common interface for each objective/analysis so that the optimisation engine does not need to know the specific call procedure for each tool. In some cases, the plugin itself may carry out the analysis (e.g. for custom cost analyses); in other cases, it typically performs a model transformation on the fully resolved EAST-ADL model and passes it to an external tool (e.g. HiP-HOPS or MAST) for analysis. When external tools are involved, the plugin also needs to retrieve the results and return them to the optimisation engine. The result of the analysis is provided as a fitness value decided by the plugin (one value per objective); the rest of the optimisation engine does not need to know what this means, only whether it should be attempting to minimise or maximise the value. The abstraction layer provided by the analysis plugins also makes it possible to quickly extend the tool architecture with new objectives, e.g. analyses for energy consumption, weight, temperature, cable length, or any other analysable attribute for which a compatible tool exists. However, our prototype architecture is not yet at a stage where we have a fully defined API available to third party developers.

3.2.5. *Product line optimisation*

Variability in EAST-ADL was originally included to help cater for different product lines that may have different sets of features but share an overall commonality of architecture. It is hoped that the optimisation process can be extended to work with not just a single member of a product line, but all members of a product line simultaneously. Because product lines are already represented by the same sort of variability techniques used to define the design space for optimisation, it should be possible to include both optimisation and product line variability in the same model (with the two forms of variability distinguished by the binding time for each variation), and exhaustively analyse the variants of the product line as part of the overall optimisation loop (shown as "Product Line Loop" in Fig. 7).

The process would involve resolution of optimisation variability as normal by the VRM to create a product line according to the encoding provided by the optimisation engine. The VRM would then enumerate each of the variants of the product line in a second step, resolving them fully so that they could then be evaluated by the analysis plugins. The major complication is that analysing an entire product line is different to analysing a single product; each analysis tool would likely only analyse one product at a time, meaning that the analysis plugins would have to iterate over the product line and combine the results in some form. Whilst in some cases this may only involve a simple summation of the individual results, in other cases the result involves a more complex calculation that would have to be carried out by the wrapper itself for the whole product line. For example, cost may be determined by the sum of each piece cost multiplied by the piece count for that piece (where the number of pieces is potentially different for each product as well as each design candidate), whereas for something like safety, the wrapper may simply use a maximum unavailability value as its result instead. This would have to be determined on a case-by-case basis, as each analysis would have different semantics and would require different calculations. The proposed architecture, with the dedicated analysis wrappers for each kind of analysis, should allow different aggregation strategies depending on the specific needs of the analysis (e.g. whether the result is a sum, a product, a maximum, or a minimum etc.).

Product line optimisation is still being investigated in the MAENAD project and is intended to be undertaken after work on the standard architectural optimisation process is complete. However, the tool architecture is being developed in such a way that it can support the future extension to product line optimisation with minimum impact.

3.2.6. Optimisation results

When the optimisation process finishes (e.g. because it meets its time limit, or because no progress has been made for *X* number of iterations, or because the user manually stops the process etc.), it will output the Pareto set of all optimal encodings it has found so far, together with the evaluation results for each objective. Ultimately this would probably be presented to the user in the form of a selection dialogue; when the user selects a particular encoding (and set of evaluation results), the VRM can take that encoding and provide the user with the resolved model in question. This avoids the need to store all of the optimal models themselves. Currently the Pareto set is simply stored as a set of XML-based model encodings, together with their analysis results. The configured model can still be obtained by manually providing the VRM with the chosen encoding.

3.2.7. Realisation of interfaces

There are three types of technical realisations for the interfaces between the above components in our current prototype: pure Java interfaces, Eclipse extension points and file-based data exchange with external tools. The OSDM, COE and VRM are all implemented in Java, so currently we use plain Java interfaces between these. While some analyses are implemented as external, non-Java tools, all analyses must be represented in Java at least with a thin wrapper in the form of a Java class. The actual communication with the analysis is the responsibility of this wrapper and its implementation depends on the implementation of the analysis: file-based in the case of external tools, e.g. HiP-HOPS, or plain Java in case of a Java-implemented analysis, e.g. cost analysis.

Even though all analysis wrappers are thus implemented in Java, the interface to these analysis wrappers is not plain Java. Instead, the Eclipse plug-in mechanism based on Eclipse extension points is employed here, which makes this interface extremely flexible. In fact, this will allow analysis wrappers to be provided and installed by third parties without having to recompile any other part of the optimisation architecture. Analysis wrappers and thus additional analyses can be installed by an end user as ordinary Eclipse plug-ins using the standard Eclipse plug-in installation procedure. This plug-in interface for analysis wrappers is already present in the current prototype, but we are also considering using such a plug-in interface between OSDM, COE and VRM in the future, which would allow us to replace even these components with custom implementations.

4. Case study

4.1. Example model

In this section, a prototype version of the tool architecture described in Section 3.2 has been applied to a simple example to illustrate the benefits of the method. A more complex case study is currently being performed on a braking system, but a simpler example was considered more appropriate for showing the mechanics of the analysis and optimisation. The example demonstrates the ability to perform a multi-objective optimisation, involving multiple analysis functions, on an EAST-ADL model. In particular, it was deemed important to have a model that:

- Featured multiple perspectives, e.g. separate functional and hardware architectures in the EAST-ADL model. This allows allocation relationships between the different perspectives to be modelled as part of the optimisation and enables a meaningful timing analysis to take place, as well as testing the ability of the dependability analysis to model propagation of failures from one perspective to another.
- Featured variability to help define the size and scope of the design space. Variability in this regard came in three forms: the ability to have different implementations of a component/function with different characteristics, the ability to have different hierarchical architectures (i.e., different subsystems/subcomponents), and as mentioned above, variable allocation relationships. These variability constructs would then define and control the scope of the optimisation.
- Featured timing information to allow a timing analysis to take place and enable optimisation with schedulability as one objective. Because of difficulties in linking to the timing analysis tool (MAST in this case) from the optimisation engine, we analysed the timing characteristics for each option and enabled the optimisation engine to use these previously calculated values directly as its timing evaluation function.

The result was a generic multi-perspective model involving sensors, processors/processing functions, and actuators that could be an example of a braking system, cruise control system, or many other automotive systems. It would take input from sensors, perform some calculations based on that input, and deliver some output instructions to an actuator as a result.

The overall system is visualised in an abstract fashion in Fig. 8.

The functional layer contains six functions targeted for software implementation and two hardware functions. Input comes from the “sensor” transfer function and is fed to a pair of local device manager functions which manage the sensor by converting the measured electrical input values into the corresponding physical measure, i.e. the inverse of the sensor’s transfer function. Output is delivered by an abstract “actuator” function (which could for example be a vehicle brake actuator); like the sensor, it has two parallel local device managers (LDMs) which convert the intended physical output to a corresponding electrical signal appropriate for the actuator. For both sensor and actuator, we have omitted functions reflecting electrical components and low level software to keep down complexity.

The actuator function is the overall output of the system—a problem with the output of this function results in a system error of some kind. In between are two control functions connected in series; these are control functions that interpret the input sensor data and process it to provide correct output commands to the actuator. These functions perform the main ‘work’ of the system and are both more complex software functions than the others.

All eight functions need to be allocated to one of the four hardware components, shown at the bottom in the hardware layer. The abstract sensor and actuator functions are assumed to represent the function of the physical sensor and actuator components, but the other functions (the LDMs, Function1, and Function2) are allocated to one of two parallel ECUs (Electronic Control Units). These ECUs are connected together as well, allowing communication from one ECU to another. The dual LDMs are

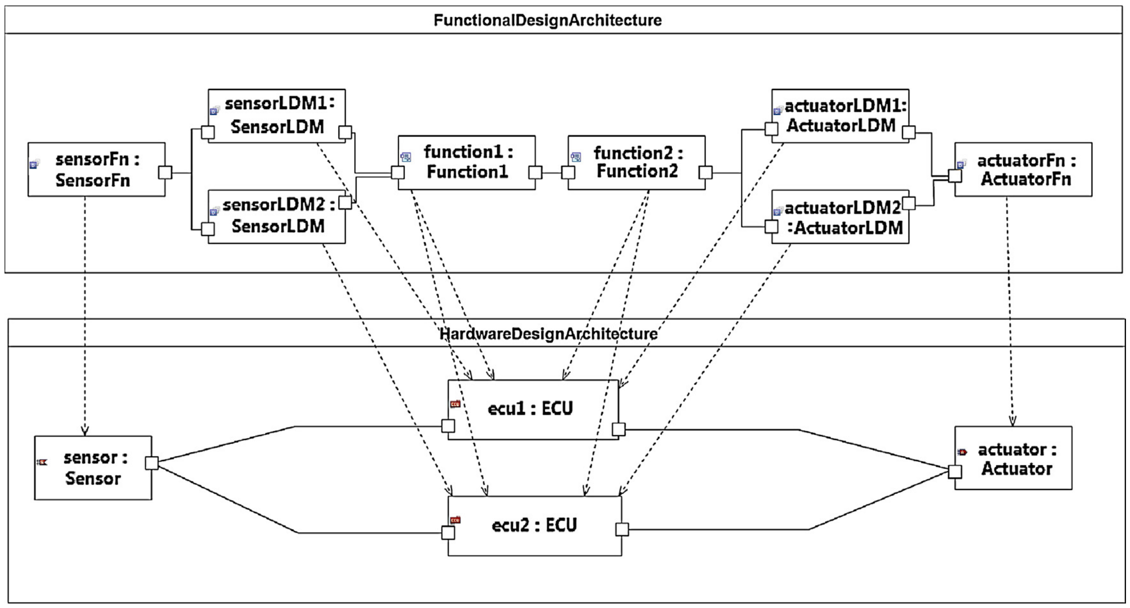


Fig. 8. The example optimisation system.

required to be allocated to different ECUs (to ensure redundancy), but the two control functions can be allocated freely.

Furthermore, each function can have one of two different implementations or variants, represented as different subsystems. The first implementation is a single function that offers no redundancy and performs all the work itself; the second implementation uses two functions, a standby and a primary, so that if the primary fails, the standby is able to take over operation, adding some redundancy to the function. It is possible for the primary and standby sub–functions to be allocated to separate ECUs.

4.1.1. *Error modelling*

There are two main channels for fault propagation in the model, as modelled by the EAST-ADL error model for the system. The first is through the hardware (in the HDA), so a failure of the sensor will propagate to ECU1 and ECU2, and any failure of an ECU (whether internal or a failure propagated from the sensor) will in turn propagate to the Actuator. The ECUs may also propagate failures to each other as well. Furthermore, a failure of any of these components will lead to the failure (omission) of any software functions allocated to them; in the case of the Actuator, this means that any propagated hardware failure will in turn propagate to the ActuatorFunction and thus cause system failure.

The failure logic of the hardware architecture (in HiP-HOPS failure expression format, relating component output failure on the left to combinations of internal failure modes and errors at the input) is shown below:

```
Error-Sensor.Out               = SensorFailure
Error-Sensor.Allocated         = SensorFailure
Omission-ECU1.Out              = Error-ECU1.In OR ECU1Failure
Omission-ECU1.Allocated        = Error-ECU1.In OR ECU1Failure
Omission-ECU2.Out              = Error-ECU2.In OR ECU2Failure
Omission-ECU2.Allocated        = Error-ECU2.In OR ECU2Failure
Omission-Actuator.Allocated    = Omission-Actuator.In OR
                                 ActuatorFailure
```

The first part of the each name (before the dash) is the failure class; in this example, only omissions are being modelled. On the right is the name of the port where the failure occurs, in *component.port* format. When the port name is “Allocated”, it represents a failure propagated to anything allocated to the current component. Names without dashes (e.g. SensorFailure) represent abstract internal failure modes of the component. These internal failure modes are defined separately with appropriate failure rates.

Thus for example an error in the output from the sensor is caused by an internal failure of the sensor (“SensorFailure”), and similarly a fault propagated along the allocation link to the hosted software is also caused by a SensorFailure. For the ECUs and Actuator, the logic is more complex, as either type of output failure (propagation via hardware and via allocation) can be caused by either an error received at the input or by an internal failure mode.

In the FDA, we can generally either assume that no internal failures occur or alternatively that internal failures are possible, but with fairly abstract failure rates (e.g. discrete failure rate classifications determined solely by ASIL). In this case, abstract failure rates were applied. As with the hardware, failures propagate from SensorFunction through the SensorLDMs, on to Function1 and Function2, and then to the ActuatorLDMs and eventually ActuatorFunction (the system output). Because each pair of LDMs are redundant, both LDMs must fail to cause a later omission. In addition, failures propagated from hardware will also enter this channel of propagation, e.g. a failure of the hardware Sensor will cause an omission of output from the SensorFunction.

The failure logic of the functional architecture is as follows:

```
Omission-SensorFunction.Out     = FromAllocation(Error) OR
                                  SensorFunctionFailure
Omission-SensorLDM.Out          = FromAllocation(Omission) OR
                                  SensorLDMFailure OR
                                  Omission-SensorLDM.In
Omission-ActuatorLDM.Out        = FromAllocation(Omission) OR
                                  ActuatorLDMFailure OR
                                  Omission-ActuatorLDM.In
Omission-ActuatorFunction.Out   = FromAllocation(Omission) OR
                                  ActuatorFunctionFailure OR
                                (Omission-ActuatorFunction.In1
                                  AND
                                  Omission-ActuatorFunction.In2)
```

The “FromAllocation(<failure class>)” construct indicates faults propagated via the allocation links from the hardware components. Thus an omission failure of the Actuator Function’s output is caused by either an internal failure mode, an omission of input to the Actuator Function, or a failure propagated from hardware.

The failure logic for the single Functions implementation are simply:

```
Omission-Function1.Out          = FromAllocation(Omission) OR
                                  SingleFailure OR
                                  Omission-Function1.In
Omission-Function2.Out          = FromAllocation(Omission) OR
                                  SingleFailure OR
                                  Omission-Function2.In
```

In other words, the single software Function can fail because of internal fault, propagated hardware failure, or omission of input in the functional layer.

For the primary/standby, it is:

```
Omission-Function1.Out          = Omission-Primary.Out AND
                                  Omission-Standby.Out
Omission-Function2.Out          = Omission-Primary.Out AND
                                  Omission-Standby.Out
Omission-Primary1.Out           = (Omission-Primary.In1 AND
                                  Omission-Primary.In2) OR
                                  FromAllocation(Omission) OR
                                  PrimaryFailure
Omission-Primary2.Out           = Omission-Primary.In OR
                                  FromAllocation(Omission) OR
                                  PrimaryFailure
Omission-Standby1.Out           = Omission-Standby.Monitor AND
                                  ((Omission-Standby.In1 AND
                                  Omission-Standby.In2) OR
                                  StandbyFailure OR
                                  FromAllocation(Omission))
Omission-Standby2.Out           = Omission-Standby.Monitor AND
                                  (Omission-Standby.In OR
                                  StandbyFailure OR
                                  FromAllocation(Omission))
```

Thus an omission of the whole Function is caused by an omission of both primary and standby subfunctions, which in turn can be caused only by an omission of input(s) to the function or combinations of failures propagated from hardware or software or internal failure modes.

After transforming the error model of the fully resolved system model provided by the optimisation engine, HiP-HOPS can take this information and analyse the propagation of failures through the system via a fault tree analysis (FTA). The primary outcomes of the FTA are the minimal cut sets (showing the set of root causes of the system failures—in this case, omission of actuator output) and an estimate of the unavailability of the system (i.e., the probability that the system will be unavailable at any given time).

4.1.2. Cost

For cost analysis, we have so far developed a simple cost analysis wrapper that uses a basic cost metric: each element of the system is assigned an abstract cost value, and the cost analysis wrapper is able to sum the constituent elements of each design candidate to produce an overall cost value. More sophisticated cost analysis schemes could also be implemented, particularly to take into account the effect of product lines (e.g. a mass produced element will provide efficiencies of scale and thus cost less than an element which is produced in smaller quantities).

In this case study, all software functions were given a cost of 10; therefore, primary/standby implementations of Functions 1 and 2 had a total cost of 20, while everything else had a cost of 10. Hardware components similarly had a cost of 10, except for the faster implementations of ECU1 and ECU2, which have cost 15.

4.1.3. Timing modelling

To provide some scope for timing analysis, two different implementations of each ECU are available: a cheaper, less reliable, slower ECU, and a faster, more reliable, but more expensive ECU. The latter, ECU-B, has 4× the performance of ECU-A (the large difference was chosen to help show larger changes in the results).

At EAST-ADL level we have a nominal execution chain from sensor to actuator. It is fed with a triggering event, in our case a PeriodicEvent of period 200 ms. Each function has an execution time, in the form of a (min value, max value, unit). Function1 is (10 and 20 ms), Function2 (20 and 30 ms), and their replicas (e.g. Primary, Standby) have the same properties. Whenever a primary/standby scheme is used, an additional execution chain has to be considered to include in the analysis possible delays due to the communication between the primary function and its replica and the execution of the replica itself. Period for this execution chain is set to 100 ms.

The controlled refinement towards a MARTE task model analysable by MAST is then applied. We obtain a MARTE model in which each execution chain is represented by an Activity diagram (see Fig. 9), called end-to-end flow. Here each task is represented by a MARTE step. As depicted in Section 2.4, the controlled refinement we apply on the EAST-ADL model allows obtaining one execution step for each function. A dedicated task called "communication step", modelling usage of communication resources, is also used. Communication steps occur when two functions allocated on different ECUs need to exchange data, e.g. between Function1 and Function2 when these are on different ECUs. When Function1 and Function2 are allocated on the same ECU, the communication step function1.send() is removed. Each execution step inherits execution times from the function mapped into it. For communication steps, communication time is set to (5 and 10 ms). As for task periods, all the steps in the same end-to-end flow have the same period set to the period of the execution chain (200 ms for the nominal end-to-end flow and 100 for reconfigurations).

Priorities for steps are all set to the same value.

The schedulability analysis methods suitable for the example are limited because of the distributed nature of the communication. The objective evaluation functions are therefore focused on the offset-based optimised analysis (Palencia and Gonzaìlez Harbour, 1998) provided by MAST. This takes into account the presence of multiple tasks involved in different end-to-end flows (called *transactions* in MAST) on the same ECU by adding offsets (extra delay time) to the worst case execution time (WCET). The optimised version also takes into account the order of the tasks, i.e., the WCET of the steps are increased by the WCET of all concurrent steps except for those which are known to occur beforehand.

The timing analysis results are given as slacks and response times. Response times include both best case execution time (BCET) and worst case execution time (WCET) in milliseconds (ms) for all end-to-end flows involved in the configuration. Slacks are percentages which show how much more the system can be loaded compared to the current timing requirements; for instance, 100% would mean you can double the timing requirement and the system should still be schedulable. In this example, the slacks provided by MAST can be very high, as we chose large periods (200 ms for the nominal activity and 100 ms for each of the reconfigurations) so that all configurations would be schedulable. The worst case performance occurs when everything is allocated on the same ECU and the "slow" ECU version, ECU-A, is chosen.

It should be noted that thanks to the proposed controlled refinement towards a MARTE task model, schedulability results given by MAST represent a sufficient condition for the schedulability of the EAST-ADL architecture.

4.1.4. Optimisation context

Having created a variability-rich EAST-ADL model containing an error model to describe the failure propagation and a timing model to represent the end-to-end flows, the optimisation parameters need to be defined. The objective functions and their goals in the optimisation are as follows:

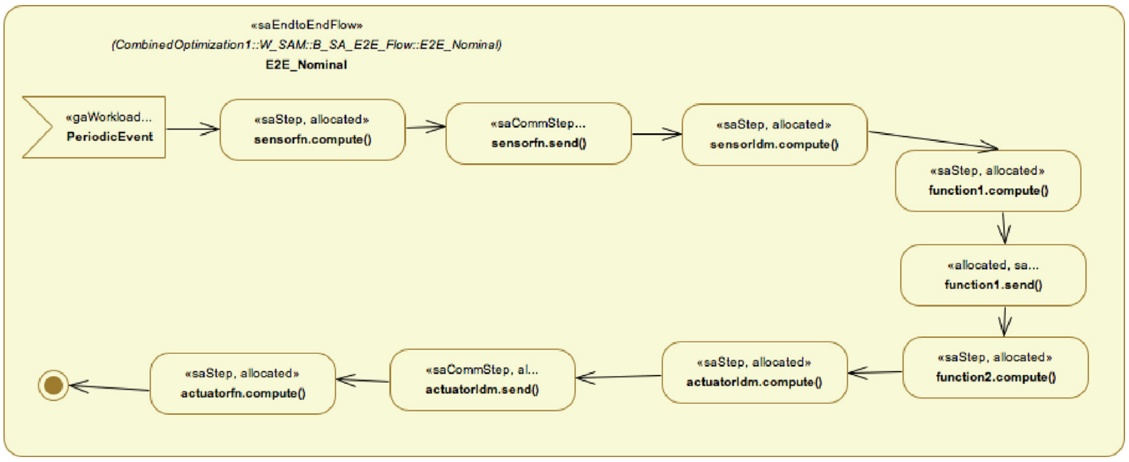

«saEndtoEndFlow»
(CombinedOptimization1::W_SAM::B_SA_E2E_Flow::E2E_Nominal)
E2E_Nominal
«gaWorkload... PeriodicEvent
«saStep, allocated» sensorfn.compute()
«saCommStep... sensorfn.send()
«saStep, allocated» sensorldm.compute()
«saStep, allocated» function1.compute()
«allocated, sa... function1.send()
«saStep, allocated» function2.compute()
«saStep, allocated» actuatorldm.compute()
«saCommStep, al... actuatorldm.send()
«saStep, allocated» actuatorfn.compute()

Fig. 9. Nominal end-to-end flow depicting tasks chain.

- [Minimise unavailability]—Unavailability is measured by the HiP-HOPS tool via a transformation of the error model. The optimisation process should seek to minimise it, i.e., reduce the probability of failure. Since in this case there is only one output failure, it can be considered proportional to safety as well.
- [Maximise slack]—As measured by the MAST timing analysis results. The goal should be to maximise the cumulative slack, i.e., maximise overall system performance. In general this also results in lower execution times and increased extensibility.
- [Minimise cost]—The cost is measured using a simple summation by the HiP-HOPS tool and should be minimised.

All three of these goals conflict to some extent: for example, using the more reliable primary/standby versions of the functions increases ECU load (because two functions are running instead), which also increases cost; similarly, using the faster ECUs also increases cost.

In addition, the parameters of the optimisation process itself are defined as follows:

- Each generation should produce 10 new individuals.
- The algorithm should be pure elitist, i.e., all dominated solutions are removed from the population in each generation.
- The mutation rate (i.e. chance for an individual to have an attribute randomly changed) is set at 5% for each generation.
- The optimisation should run for 100 generations, or stop early if no new Pareto optimal solutions have been identified for the past 10 generations.

The overall goal is to obtain the set of Pareto solutions, i.e., the set of system configurations that have superior attributes and which are not dominated by any other solutions.

4.2. Results

After 10 runs, where each run was an average of 62 generations and took only a matter of seconds (due to the small size of the model), the outcome of the optimisation is a set of 18 Pareto optimal solutions. In practice, there are several configurations that have the same fitness values for all objectives, e.g. there is no difference between allocating everything to ECU1 compared to allocating everything to ECU2, assuming all else remains equal. These are still included in the results because there may be other characteristics that differentiate the solutions while not being part of the optimisation. Consequently, there are in fact only 9 unique combinations of attributes, due to the symmetry in the model.

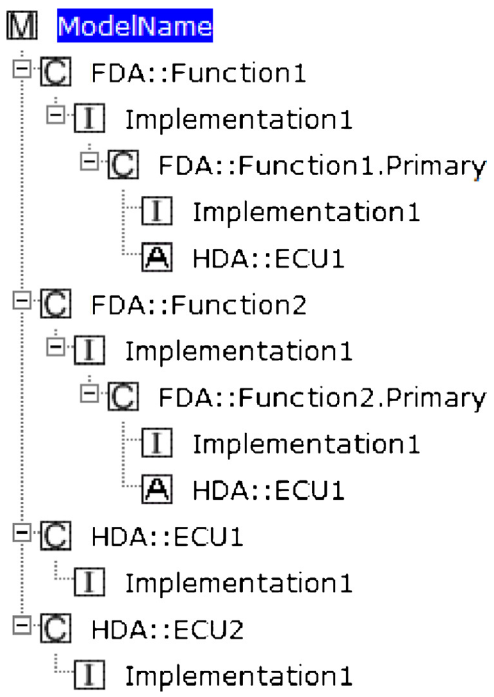
The 18 solutions are listed in Table 1.

Each of these solutions has an associated encoding that uniquely identifies it. The encoding for each of the solutions in the results can be viewed graphically, e.g. the encoding for solution #1 is shown below. This encoding can also be used to 'configure' the original, variant-rich model by resolving the variability, allowing the designer to see the actual model of that particular solution. In the case of solution #1, the result is a relatively slow system with very little slack and average unavailability, but at the lowest possible cost. As seen in Fig. 10, this is due to the selection of the slower, cheaper ECU (ECU-A, i.e., Implementation 1) for both ECUs, and the use of the single function implementation rather than the primary/standby implementation for both control functions.

By contrast, solution 17 represents both one of the most expensive and one of the most reliable solutions. In this case, the most expensive (and highest performance) ECUs were chosen, and primary/standby versions of both functions were used. To minimise the unavailability, not only are the two functions distributed across both ECUs, but so are their subfunctions, such that the two primary subfunctions are on ECU1 and the two standby subfunctions are on ECU2. This ensures that a failure of either ECU will still allow the system to function, whereas allocating e.g. both subfunctions of Function 1 to ECU1 and both subfunctions of Function 2 to ECU2 would not (this combination would result in the same cost and similar slack, but considerably higher unavailability). Finally, the highest system slack is provided by solutions 4 and 6; in these

Table 1
Pareto optimisation results.

Solution ID	System slack (%)	Unavailability	Cost
1	183.59	0.002999	120
2	1036.7	0.002999	125
3	396.88	0.003996	120
4	1889.8	0.003996	130
5	396.88	0.003996	120
6	1889.8	0.003996	130
7	183.59	0.002999	120
8	1036.7	0.002999	125
9	695.31	0.002998	150
10	183.59	0.002001	145
11	396.88	0.002001	150
12	121.09	0.002001	140
13	467.97	0.002998	145
14	467.97	0.002998	145
15	121.09	0.002001	140
16	183.59	0.002001	145
17	396.88	0.002001	150
18	695.31	0.002998	150



```
M ModelName
  C FDA::Function1
    I Implementation1
      C FDA::Function1.Primary
        I Implementation1
        A HDA::ECU1
  C FDA::Function2
    I Implementation1
      C FDA::Function2.Primary
        I Implementation1
        A HDA::ECU1
  C HDA::ECU1
    I Implementation1
  C HDA::ECU2
    I Implementation1
```

Fig. 10. Encoding for solution 1.

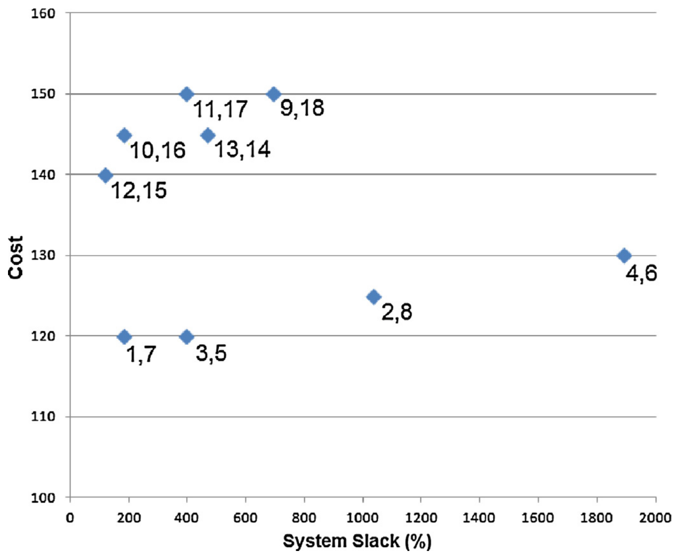


Fig. 12. System slack vs cost.

solutions, the high performance ECUs are chosen, but only the single versions (no primary/standby) of Functions 1 and 2 are used; each function is allocated to a different ECU, which means that the processing load is both minimal and balanced across the two ECUs.

The results can be viewed in pairs as one objective against another on a 2D chart (as seen in Figs. 11–13, with solutions labelled). Due to the number of objectives, it is also possible to visualise the Pareto solutions in 3 dimensions, as in Fig. 14.

Other timing metrics (e.g. worst-case execution time) can also be used as additional optimisation objectives, and in fact in one experiment we had 10 timing metrics in addition to cost and unavailability; this resulted in 107 Pareto solutions, since with so many objectives, it is less likely for any given solution to be dominated in every objective.

It is clear that the optimisation process results in a set of optimised candidate solutions, each representing a different optimal or near optimal trade-off among the parameters of the optimisation. Choosing a single solution out of this set for further development and implementation is essential and would require analysts to examine the results closely in order to assess which solution best meets requirements. Sorting the solutions in terms of their performance in individual objectives, and prioritising the most important objectives, can help with this process. In addition, constraints can be set, which would eliminate any solutions that do not meet those constraints. By sorting results in terms of unavailability and setting a maximum cost, for instance, it would be possible to quickly see the most reliable solutions for a given cost. Iterating this selection process for other objectives will make it possible to further reduce the set of candidate solutions and ultimately locate a single solution (or at least a smaller set of solutions) for further investigation and possible further development.

In general, the above process is structured and very systematic and helps to arrive at sound decisions about the architecture which are informed by a set of essential analyses that can be considered in a single framework of evaluation and optimisation. The example we used to illustrate the method was kept simple to facilitate the presentation of the method, but other applications of HiP-HOPS

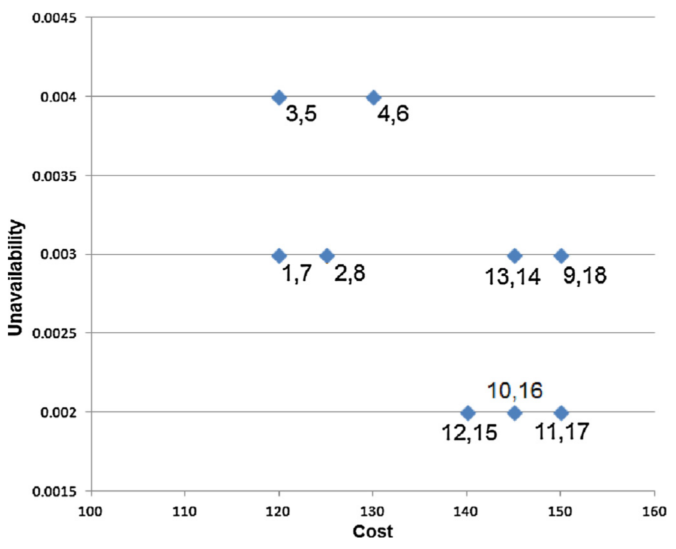


Fig. 11. Cost vs unavailability.

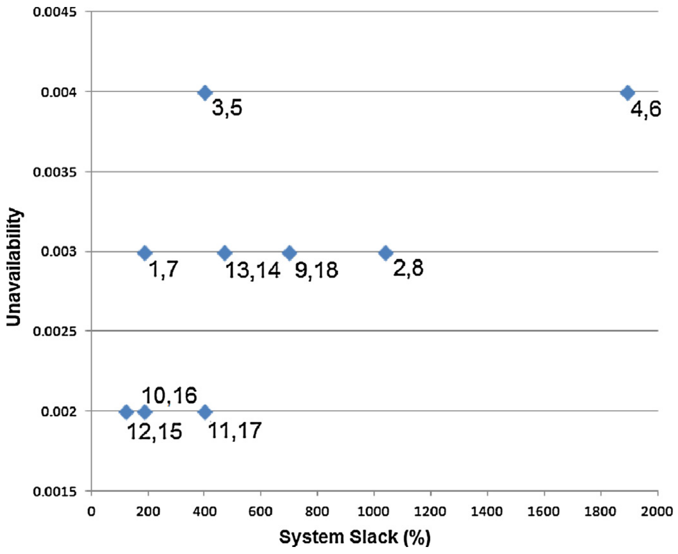


Fig. 13. System slack vs unavailability.

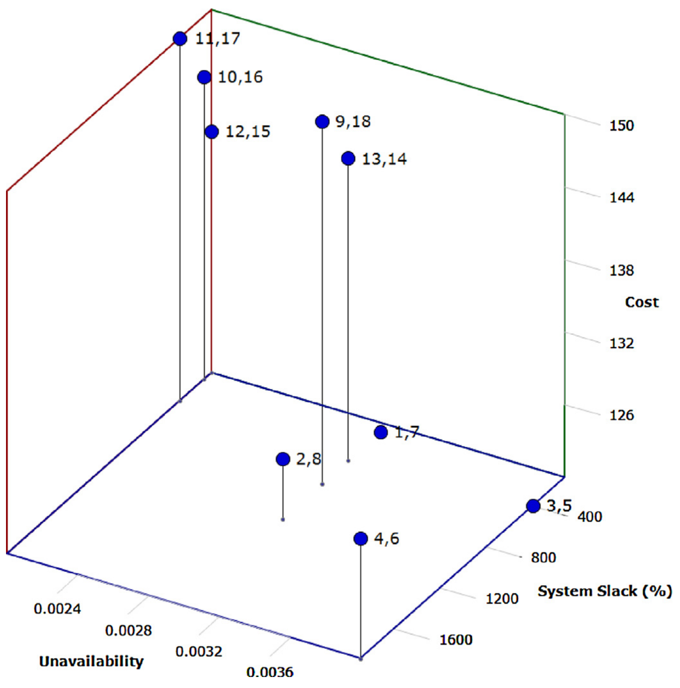


Fig. 14. Visualisation of the Pareto optimal solutions in 3D.

on optimisation (e.g. Adachi et al., 2011)—although not specific to EAST-ADL—show that the underpinning optimisation algorithms can be effective in larger scale.

5. Conclusion

In this paper, we have described how the model-based design and analysis capabilities of the EAST-ADL language can be combined with metaheuristic algorithms such as genetic algorithms to allow automatic architectural optimisation to take place. In particular, this is made possible through EAST-ADL's variability semantics and its support for varied analysis techniques of different quality attributes. Variability is used in EAST-ADL to express optionality and consequent dependencies in the model, and has been adapted here to express the architectural degrees of freedom that make up the overall optimisation design space. Three example attributes—dependability, schedulability, and cost—have so far been established as possible objectives (with associated analysis tool support) for evaluation of the candidate system designs identified by the optimisation process, allowing a multi-objective optimisation to take place.

The result is an optimisation concept and associated tool architecture design to allow for the automatic optimisation of EAST-ADL system architectures according to multiple, potentially conflicting objectives, which can be evaluated by means of heterogeneous external tools. While there are other architectural optimisation approaches in development (see Aleti et al. (2012) for a full review), we believe that basing our optimisation technique on EAST-ADL, a comprehensive ADL designed for use in the automotive domain that offers an integrated modelling and analysis solution across multiple stages of the design process, offers significant advantages in terms of consolidation and reuse of existing domain-specific information. This can offer major benefits during the design of complex system architectures, as the huge potential design space—which could never effectively be fully investigated manually—can be easily derived from existing system models augmented with established variability semantics, and then rapidly explored by the optimisation algorithm. The benefits are amplified when there are multiple, often conflicting objectives for the design (e.g. maximum safety and performance for minimum cost), which are supported by EAST-ADL's analysis support. On a more general level, such optimisation capabilities are particularly useful where there is little prior experience to inform the evolution of the design, as automatic optimisation allows many different possible design decisions to be rapidly experimented with and evaluated without the need for expensive delays and redevelopments.

The optimisation concept itself is based on the use of genetic algorithms (GAs), which mimic the behaviour of natural evolution. The algorithms themselves have already been successfully applied in the context of HiP-HOPS models of automotive systems (Adachi et al., 2011), and similar genetic algorithms have also been successfully applied in other architectural optimisation approaches, e.g. AQOSA (Etemaadi and Chaudron, 2012) and Per-Opteryx (Koziolek et al., 2011). In this paper we have repurposed the HiP-HOPS optimisation concept for application to EAST-ADL models and attributes, expanding beyond dependability to include cost and timing analysis as well. The ultimate aim would be to allow optimisation with respect to any of the qualities in EAST-ADL that can be analysed.

The design space is defined in the EAST-ADL model by means of variability mechanisms, which express variability points in the architecture—places where different implementations, replication strategies, or allocations from software to hardware can be applied. Each particular configuration of the system is represented by an encoding, which indicates which options have been chosen (if any) for each variability point by the genetic algorithm. Genetic algorithms offer two main methods for this—crossover, which perpetuates successful traits from previous generations, and mutation, which introduces new traits and prevents stagnation of the population. Multi-objective evaluation is provided by an analysis interface based on tool plugins, which provide either local analysis capabilities or link to external tools like HiP-HOPS or MAST via a model transformation process. The results of the analyses are fed back into the optimisation engine to evaluate each potential candidate, enabling successful candidates to survive and unsuccessful ones to be discarded.

The result of the process is a set of Pareto optimal solutions—those which represent the best tradeoffs between the multiple objectives found so far. Each of these is stored as an encoding and associated analysis results, so that the designers can choose a promising set of attribute characteristics (e.g. high safety, good performance, low cost) and use the encoding to configure the original model to obtain that particular solution.

The 'glue' that makes all of this possible is the use of an ADL—specifically, EAST-ADL—to model the architecture and store all of the associated functional and non-functional attributes and requirements in a consistent, organised fashion. By acting as a centralised repository of information about the system and providing a means to define architectural choices as part of the model via its variability mechanisms, EAST-ADL makes it possible to define the design space and, by providing access to the specific information needed by each objective evaluation function, explore the design space successfully by analysing potential new candidates and evolving them according to its heuristics.

Thus one of the contributions of this paper in regard to automated architecture optimisation is related to the formalisation of system parameters and quality/objective functions. EAST-ADL represents a domain-specific approach to a formalised system description. The language provides an enabling technology for capturing various quality concerns as well as the design parameters in a consolidated architecture model and thereby for allowing automated design space exploration, quality assessment, and variability management.

We applied the concept to a small example to demonstrate how the principles can be applied in practice. The model featured a mix of challenging characteristics, including a mixture of modelling perspectives (separate hardware and software layers, separate nominal and error models), the use of different variability constructs to represent different forms of architectural choices (including substitution, replication, and allocation), and the inclusion of information in the same model to allow both timing and dependability analyses.

The results demonstrate, even on a small scale, how useful this concept can be. It produced a set of different design candidates, each representing a different tradeoff between the three main objectives considered (cost, unavailability, and processing slack) and presenting them to the designer in an accessible way. If this was used as part of an overall iterative design approach, the system designers could apply the optimisation to investigate which options are available to them, choose the solution that suits their requirements most closely, and use that as the basis for the next iteration of the system design. This could be done much more rapidly and efficiently than a manual approach, as many more possibilities can be examined by the optimisation algorithm. Even in this small example, 18 different optimal solutions were identified and many more non-optimal ones discarded.

Although our results so far have been encouraging, we plan to carry out considerable further work. As part of the MAENAD project, we are currently working on a larger optimisation case study based on a brake-by-wire system, which will demonstrate the full capabilities of the optimisation tool architecture. Furthermore, we also hope to extend the optimisation to be able to work with entire product lines at once. This would mean optimising not just a single model, but a range of models. For example, while a given architectural choice may make sense when dealing with just one model, that same choice may have different implications for other models in the same product line, perhaps making them uneconomical or resulting in them failing to satisfy their requirements. This involves developing a much more sophisticated cost analysis algorithm but would result in a powerful new capability that could be of great benefit to e.g. the automotive industry.

Another promising area worthy of investigation is the inclusion of ASIL (Automotive Safety Integrity Levels) allocation and decomposition as part of the optimisation process. This would mean the optimisation algorithm would not only experiment with different architectural choices to meet a given set of requirements and objectives, but it could even experiment with how those requirements are applied. Early work into the automatic allocation of ASILs (Papadopoulos et al., 2010) and initial experiments in using optimisation algorithms to perform this allocation have yielded promising results so far, although integrating this into the wider architectural optimisation process may still pose significant challenges.

Judging from the growing body of work on architectural optimisation, as well as from our investigations, it is clear that automatic optimisation of system architectures, particularly when modelled using an ADL to centralise all of the information about those systems, could offer significant benefits for the design of complex modern systems. We believe the potential advantages presented by the conjunction of these technologies have only just begun to be explored.

Acknowledgements

The authors acknowledge the financial support provided by the European Commission (call FP7-2010-ICT-GC, Grant Agreement Number 260057).

References

Abele, A., Lönn, H., Reiser, M.-O., Weber, M., Glathe, H.,2012. EPM: a prototype tool for variability management in component hierarchies. In: Proceedings of the 16th International Software Product Line Conference – Volume 2 (SPLC'12), 2012. ACM, New York, USA, pp. 246–249, http://dx.doi.org/10.1145/2364412.2364455, ISBN: 978-1-4503-1095-6.

Adachi, M., Papadopoulos, Y., Sharvia, S., Parker, D., Tohdo, T., 2011. An approach to optimization of fault tolerant architectures using HiP-HOPS. Software Practice and Experience, http://dx.doi.org/10.1002/spe.104436.

Aleti, A., Bjoernander, S., Grunske, L., Meedeniya, I., 2009a. ArcheOpterix: an extendable tool for architecture optimization of AADL models. In: Model-based Methodologies for Pervasive and Embedded Software (MOMPES), Workshop at ICSE 2009 ACM and IEEE Digital Libraries, pp. 61–71.

Aleti, A., Grunske, L., Meedeniya, I., Moser, I., 2009b. Let the ants deploy your software – an ACO based deployment optimisation strategy. In: Automated Software Engineering (ASE 2009). IEEE Computer Society Press, pp. 505–509 (short paper).

Aleti, A., Buhnova, B., Grunske, L., Koziolek, A., Meedeniya, I., 2012. Software architecture optimization methods: a systematic literature review. IEEE Transactions on Software Engineering (September (99)), http://dx.doi.org/10.1109/TSE.2012.64, ISSN: 0098-5589.

Anssi, S., Tucci-Pergiovanni, S., Mraidha, C., Albinet, A., Terrier, F., Gérard, S., 2010. Completing EAST-ADL with MARTE for enabling scheduling analysis for automotive applications. In: Embedded Real-Time Software and Systems (ERTS2 2010), Toulouse, France, May 19–21st, 2010.

Bieber, P., Bougnol, C., Castel, C., Heckmann, J.-P., Kehren, C., Metge, S., Seguin, C., 2004. Safety assessment with Altarica. In: IFIP Congress Topical Sessions, pp. 505–510.

Bozzano, M., Villafiorita, A., 2007. The FSAP/NuSMV-SA safety analysis platform. International Journal on Software Tools for Technology Transfer 9 (1), 5–24.

Chen, D., Johansson, R., Lönn, H., Papadopoulos, Y., et al., 2008. Modelling support for design of safety–critical automotive embedded systems. In: SAFECOMP 2008, LNCS 5219. Springer, pp. 72–85.

Coit, D.W., Smith, A.E., 1996a. Penalty guided genetic search for reliability design optimisation. Computers and Industrial Engineering 30 (4), 895–904.

Coit, D.W., Smith, A.E., 1996b. Reliability optimisation of series-parallel systems using a genetic algorithm. IEEE Transactions on Reliability 45 (2), 254–260.

Corne, D.W., Jerram, N.R., Knowles, J.D., Oates, M.J.,2001. PESA-II: region-based selection in evolutionary multiobjective optimisation. In: Proceedings of the Genetic and Evolutionary Computation Conference GECCO. Morgan Kaufmann Publishers, pp. 283–290.

Deb, K., Pratap, A., Agarwal, S., Meyarivan, T., 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation 6 (2), 182–197.

Etemaadi, R., Chaudron, M.R.V., 2012. A model-based tool for automated quality-driven design of system architectures. In: Proceedings of the 8th European Conference on Modelling Foundations and Applications (ECMFA'12), 2–5 July 2012, Lyngby, Denmark.

Feiler, P.H., Rugina, A., 2007. Dependability Modelling with the Architecture Analysis and Design Language (AADL). Technical report, CMU/SEI-2007-TN-04.

Grunske, L., Kaiser, B., Papadopoulos, Y., 2005. Model-driven safety evaluation with state-event-based component failure annotations. In: CBSE'05, pp. 33–48.

Grunske, L., Lindsay, P., Bondarev, E., Papadopoulos, Y., Parker, D., 2007. An outline of an architecture-based method for optimizing dependability attributes of software-intensive systems. In: de Lemos, R., et al. (Eds.), Architecting Dependable Systems IV. LNCS, vol. 4615, pp. 188–209.

Joshi, A., Vestal, S., Binns, P., 2007. Automatic generation of static fault trees from AADL models. In: DSN'07 Workshop on Architecting Dependable Systems, Edinburgh.

Kim, H., Bae, C., Park, S.,2004. Simulated annealing algorithm for redundancy optimization with multiple component choices. In: Advanced Reliability Modelling, Proceedings of the 2004 Asian Internationals Workshop. World Scientific, pp. 237–244.

Konak, A., Coit, D.W., Smith, A.E., 2006. Multi-objective optimization using genetic algorithms. Reliability Engineering & System Safety 91 (9), 992–1007.

Koziolek, A., Reussner, R., 2011. Towards a generic quality optimisation framework for component-based system models. In: Proceedings of the 14th International ACM SIGSOFT Symposium on Component Based Software Engineering (CBSE'11), 21–23 June 2011, Boulder, CO, USA.

Koziolek, A., Koziolek, H., Reussner, R., 2011. PerOpteryx: automated application of tactics in multi-objective software architecture optimisation. In: Proceedings of the joint ACM SIGSOFT conference – QoSA and ACM SIGSOFT symposium – ISARCS on Quality of Software Architectures – QoSA and architecting critical systems – ISARCS, New York, USA, 2011, pp. 33–42, http://dx.doi.org/10.1145/2000259.2000267, ISBN: 978-1-4503-0724-6.

Kulturel-Konak, S., Smith, A.E., Coit, D.W., 2003. Efficiently solving the redundancy allocation problem using tabu search. IIE Transactions 35, 515–526.

Liang, Y.C., Smith, A.E., 2004. An ant colony optimisation algorithm for the redundancy allocation problem (RAP). IEEE Transactions on Reliability 53 (3), 417–423.

Martens, A., Adagna, D., Koziolek, H., Mirandola, R., Reussner, R., 2010. A hybrid approach for multi-attribute QoS optimisation in component-based systems. In: Proceedings of the 6th International Conference on the Quality of Software Architectures (QoSA'10), Springer, Berlin Heidelberg, Lecture Notes in Computer Science, vol. 6309, pp. 84–101, http://dx.doi.org/10.1007/978-3-642-13821-8_8, ISBN: 978-3-642-13821-8.

Meedeniya, I., Grunske, L., 2010. An efficient method for architecture-based reliability evaluation for evolving systems with changing parameters. In: International Symposium on Software Reliability Engineering (ISSRE 2010), San Jose, CA, USA, November 1–4, 2010, pp. 229–238.

Meedeniya, I., Buhnova, B., Aleti, A., Grunske, L., 2010. Architecture-driven reliability and energy optimization for complex embedded systems. In: Sixth International

Conference on the Quality of Software Architectures (QoSA 2010), 2010, Prague, LNCS 6093. Springer, pp. 52–68.

Meedeniya, I., Moser, I., Aleti, A., Grunske, L., 2011. Architecture-based reliability evaluation under uncertainty. In: Quality of Software Architectures (QoSA 2011), Boulder, CO, USA, June 2011, pp. 85–94.

Palencia, J.C., Gonzaìlez Harbour, M., 1998. Schedulability analysis for tasks with static and dynamic offsets. In: Proceedings of the IEEE Real-Time Systems Symposium, December 02–04, 1998. RTSS. IEEE Computer Society, Washington, DC, p. 26.

Papadopoulos, Y., Grante, C., 2005. Evolving car designs using model-based automated safety analysis and optimisation techniques. Journal of Systems and Software 76 (1), 77–89.

Papadopoulos, Y., McDermid, J.A., Sasse, R., Heiner, G., 2001. Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure. RESS 71 (3), 229–247.

Papadopoulos, Y., Walker, M., Reiser, M.-O., Weber, M., Servat, D., Abele, A., Johansson, R., Lonn, H., Torngren, M., Sandberg, A., 2010. Automatic allocation of safety integrity levels. In: 8th European Dependable Computing Conference – CARS Workshop, Valencia, Spain. ACM Press, Spain, pp. 7–11, ISBN: 978-1-60558-915-2.

Papadopoulos, Y., Walker, M., Parker, D., Rüde, E., Hamann, R., Uhlig, A., Grätz, U., Lien, R., 2011. Engineering failure analysis & design optimisation with HiP-HOPS. Journal of Engineering Failure Analysis, http://dx.doi.org/10.1016/j.engfailanal.2010.09.025, ISSN: 1350 6307.

Parker, D., 2010. Multi-objective optimisation of safety–critical hierarchical systems. PhD thesis, University of Hull, UK.

Salazar, D., Rocco, C.M., Galvan, B.J., 2006. Optimisation of constrained multiple-objective reliability problems using evolutionary algorithms. Reliability Engineering and System Safety 91, 1057–1070.

Selic, B., 2000. A generic framework for modelling resources with UML. IEEE Computer 33 (6), 64–69.

Wallace, M., 2005. Modular architectural representation and analysis of fault propagation and transformation. Electronic Notes Theoretical Computer Science 141 (3), 53–71.

Walker, M., Papadopoulos, Y., 2009. Qualitative temporal analysis: towards a full implementation of the Fault Tree Handbook. Control Engineering Practice 17 (10), 1115–1125, ISSN 0967 0661.

Zitzler, E., Laumanns, M., Thiele, L., 2001. SPEA2: improving the strength Pareto evolutionary algorithm. In: Proceedings EUROGEN 2001 Evolutionary methods for Design, Optimisation and Control with Applications to Industrial Problems, Athens, Greece.

Project/tool websites

ATESST Project website: www.atesst.org.

AUTOSAR Development Partnership, 2013. AUTOSAR web site: http://www.autosar.org/.

EAST-ADL Association Members, 2013. EAST-ADL Association web site: http://www.east-adl.info/.

EDONA Project website: www.edona.fr.

MARTE website: www.omgmarte.org.

MAST (Modeling and Analysis Suite for Real-Time Applications) toolset website: http://mast.unican.es/.

TIMMO Project website: www.timmo.org.

Dr Martin Walker is a Computer Science lecturer at the University of Hull, working in the Dependable Systems group. He has a PhD and MSc in Computer Science and his research is focused on the development of model-based safety analysis techniques and tools, particularly for dynamic systems. He has participated in EU projects on safety including SAFEDOR, ATESST2, and MAENAD.

Dr Mark-Oliver Reiser holds a degree in computer science from Technische Universität Berlin, Germany. From 2004 to 2008 he has been doing a PhD at Daimler AG. Among his main scientific interests lie software product line concepts, esp. variability management, requirements engineering and design modelling with a focus on applying and adapting these concepts to the automotive domain as well as other embedded system domains. He is working as a researcher at Technische Universität Berlin.

Dr Sara Tucci-Piergiovanni is a research engineer at the CEA LIST laboratory of model-driven engineering. She received her MS and PhD degrees in computer engineering from the Sapienza University of Rome in 2002 and 2006. Her master's thesis won the Confederation of Italian Industry prize for the best Italian thesis in ICT. At Sapienza, she was a lecturer in distributed systems until 2008. She publishes regularly in the areas of distributed, real-time, and mission-critical systems and software, serves as a reviewer in international journals (TPDS, TCS, JSS, SoSym) and conferences (ETFA, EDCC, ACM SAC), and co-chaired the 2008 International Conference in Distributed Event Systems.

Prof. Yiannis Papadopoulos has a PhD in Computer Science, an MSc in Advanced Manufacturing Technology and a degree in Electrical Engineering. He is a Professor of Computer Science at the University of Hull and leader of the Dependable Systems research group, where he leads the development of Hierarchically Performed Hazard Origin and Propagation studies, a novel method for model-based safety analysis and optimisation of computerised systems. Prof Papadopoulos is also active in two Technical Committees of IFAC (TC 1.3 and TC 5.1) and has been instrumental in the development of a new strand of work on "dependable control" within IFAC.

Dr. Henrik Lönn has a PhD in Computer Engineering from Chalmers University of Technology, Sweden, with a research focus on communication issues in safety–critical real-time systems. At Volvo, he has worked with prototypes, architecture modelling and communication aspects on vehicle electronic systems. He is also participating in national and international research collaborations on embedded systems development. Previous project involvement includes X-by-Wire, FIT and EAST-EEA, ATESST and TIMMO.

Dr. Chokri Mraidha is a researcher at the Laboratory of Model-Driven Engineering for Embedded Systems of the CEA LIST institute in France. He got a master degree in distributed computing in 2001 and a PhD in Computer Science in 2005. His research interests include real-time and embedded systems, model-driven development, quality-guided real-time systems design and optimized model compiling. He is involved in UML-based OMG standards for design of real-time systems like SysML and MARTE and the AUTOSAR standard and is working in European and French research projects developing model-based approaches for real-time systems design and verification for the transport industries.

Dr. David Parker is a lecturer in the Department of Computer Science at the University of Hull, working in the Dependable Systems research group. His main research focus is the application of optimisation algorithms to safety critical system designs (the subject of his PhD). He continues to play a key role in the development of the HiP-HOPS tool and has participated in several European safety related projects including SAFEDOR, ATESST2, and MAENAD.

Dr. DeJiu Chen received his PhD degree in Mechanical Engineering with a research on embedded computer control systems from KTH in 2004. His research interests are on systems and software architecture, model-based engineering, dependability and self-adaptive embedded systems. Since 2004, he has been actively involved in several research projects, including NFCCPP, ATESST, DySCAS, ATESST2, and MAENAD. From 2007 to 2009, Dr. DeJiu Chen also worked for Enea Data AB, Sweden, as a senior technical instructor. He is currently an assistant professor at KTH.

Dr. David Servat has a PhD in Computer Science at Paris 6 University and an engineering diploma from Institut Telecom Paris Tech. David Servat leads research projects in the field of component-based approach for the design of real-time distributed systems within the LISE Lab of CEA LIST. It includes research on execution platform, middleware for embedded systems, and model-driven engineering techniques in several national and European projects. In particular since 2006, he led the CEA contribution to the series of FP7 projects on automotive architecture design and optimisation: ATESST1&2 and MAENAD.