A Multi-Viewpoint Architecture Exploration Methodology for Embedded Systems

Magnus Persson, Stéphane Bonnet, Andreas Mitschke, Martin Törngren, Odile Laurent, DeJiu Chen and Marcus Shawky


Abstract

Architecture exploration is increasingly important as a design approach for embedded systems. In this paper, we present an architecture exploration methodology especially suited for multi-viewpoint systems, based on 8 different key tasks: preparation, baseline design, design variability modeling, selection of search space, architecture analysis, trade-off analysis & design, and documentation. The methodology is exemplified and validated through a concrete aerospace case study.

B.1 Introduction

Architecture design concerns the definition of the fundamental organization of a system. The word “fundamental” derives from the fact that this system organization determines key system properties, such as performance and cost. Architectural design is supported by design patterns, norms, templates and other guidance (e.g., reference architectures).

Architecture exploration is increasingly important as a design approach for embedded systems. In this paper, we present a cross-domain architecture exploration methodology, which has the goal to cover a broad span of exploration scenarios in order to provide a basis for systematic studies. We specifically address needs and challenges regarding multi-view architecture exploration and industrial adoption.

Architecture exploration involves the development and evaluation of several different architectural design alternatives as an approach to architecture design, in order to perform full or partial design optimization according to some selected design criteria. Through the usage of models, it allows a designer to systematically examine alternative design options, and thereby complements the usage of heuristics and domain experiences.

Whenever alternative behaviors, structures or mapping solutions are possible, it is the task of architecture design to describe design options and provide a selection of which option(s) is preferred. While many design alternatives often can be identified and resolved based on stakeholder consensus, heuristics, or design patterns, explicit support for architecture exploration and evaluation with models would allow a more systematic evaluation of alternatives.

Architecture exploration could be applied on architecture designs at different levels of abstraction, as long as there are sufficient amounts of information available to describe, relate, and evaluate alternative design options. As such, this means that the methodology could both be applied at early stages, using “back-of-an-envelope” rough analysis methods, as well as at later design stages, with very detailed analysis models.

The main assumption for architectural exploration to be performed is the availability of a model, in some sense of the word, of the system under design to be available for analysis. The model may be expressed using a formal or semi-formal language or simply natural language.

There are many usage scenarios for applying architecture exploration including for example refinement of an existing architecture due to addition of new functionalities, definition and evaluation of concepts for a new design, and analysis to uncover relationships between different design qualities (metrics). While the purpose of applying architecture exploration in these different scenarios differ, they all have a common theme: the designer is systematically evaluating alternative designs in order to find a design choice that is acceptable in the design context and to get a better understanding of properties of different
design alternatives. Architecture exploration does not as a concept in itself imply an automated environment, and can potentially be done completely manually by a team of engineers. However, to be able to assess different alternatives against each other effectively, a significant amount of analysis and evaluation of the architecture alternatives are necessary, and automation becomes indispensable as soon as more than a small number of variants are to be compared.

B.1.1 A Multi-View Approach to Architecture Exploration

Architecture exploration is closely related to a multi-view approach to architecture modeling, since the architecture exploration typically considers several properties of the proposed architecture. According to system engineering standard ISO/IEC 42010[25], view and viewpoint are central concepts in architectural design, with the following definitions:

“View: A representation of a whole system from the perspective of a related set of concerns.”

“Viewpoint: A specification of the conventions for constructing and using a view. A pattern or template from which to develop individual views by establishing the purposes and audience for a view and the techniques for its creation and analysis.”

However, this standard is mainly used as vocabulary. It is further important to distinguishing between information which is design information and analysis outputs, even though both may be part of a view.[36] Finally, tool support is required to efficiently develop models in practice. In the setting of architecture exploration, we hence suggest a viewpoint to include the following information, extending and clarifying what is defined by IEEE42010:

• modeling formalisms and languages used to represent the view[10].
• metrics and concerns it handles (a single viewpoint can be used for several closely related metrics).
• analysis methods to be used to compute the metrics and / or analyze the view according to the exploration objectives.
• tools that are to be used to perform the aforementioned analyses.

B.1.2 The CESAR Project

The work presented in this paper has been performed within the CESAR project[2], which has aimed to build a Reference Technology Platform (RTP)[9, 26] for tool chain integration. Further results from the CESAR project are the formalization of requirements through the requirements specification language (RSL) and requirements meta-model (RMM)[33], the definition of an integrated component model, the CESAR common meta-model (CMM)[8]. These are together with the overall RTP integration platform a possible basis for building an architecture exploration tool chain. The main results from the project will be published in form of a forthcoming book[37].
B.2 Related Work

There is plenty of architecture exploration studies for specific systems. Just one such recently published in this journal is [39], which is a study of different architectural choices for memory hierarchies on FPGAs. To focus on specific systems is however not the intention of this article, rather it is to work out methodology in order to reach a state where such studies can be performed in a systematic manner. In order to reach that state, both methodology and tool support is needed. The focus of this section is to present other work on methodology in the area, either for a monolithic system description, for a specific predefined set of view or generic approaches applicable for arbitrary choices of viewpoints.

Architecture exploration can be touched upon from many different points of view. In this brief survey of related work we touch upon the following perspectives:

- embedded systems design methodologies
- domain specific approaches, such as SoC
- process oriented approaches
- generic approaches for multi-criteria evaluation
- technology centered approaches, e.g. use of optimization techniques or specific languages

In the field of system-on-chip (SoC) hardware development, especially ASICs and FPGAs, the topic of co-design is a common challenge, often, several different implementation alternatives are possible – ranging from more or less parallel hardware solutions to pure software solutions.[20, 41] Due to this large number of realization alternatives, architecture exploration has become a popular approach to architecture design in this area.

Methods developed for the SoC context usually have a narrower mindset than the proposed methodology in this article, as they mainly work with architectures that are based on two main views: a logical/application view and a hardware/platform/implementation architecture, subject to changes. Correspondingly, they often have a limited set of quality aspects that drive the exploration, dominated by performance and cost concerns. The software to be implemented on the platform is usually taken as given (or at most with alternative implementations) while the hardware and the allocation mapping between the two is seen as the main design variability available for the exploration process. By varying the choice of hardware subsystems, their connection topology, and finally the allocation mapping, overall system architectures targeted are defined in a pre-decided manner. In other words, the range of the design variability is limited as an explicit part of the methodology.

The Y-chart approach[27] follows the pattern of separation between (hardware) architecture and applications. It further takes as an explicit notion of going through an abstraction hierarchy top-down during the exploration chain, using ever-more accurate system models during the process. Finally, a big subgroup of formalisms employing this set of viewpoints is the approach of platform-based design[12, 16], abbreviated PBD. The essential idea of PBD is to start off with software and hardware views and refine and abstract them respectively, i.e. a meet-in-the-middle approach. The hardware view is further based on using a predefined library of ready components.

There are also more generic approaches, not assuming any specific approach to system development. The SPEEDS project worked on the underlying methodology used for multi-criteria architecture evaluation. The approach recognizes the use of five views (operational,
functional, logical, technical and geometrical) but does not handle the dependencies between them. Contracts are used as a method to decompose the system into alternative subsystems. The Requirements Specification Language (RSL) is used to formalize different design constraints on the system. [13]

Some approaches do not try to design a generic architecture exploration process, but rather aim at better describing the actual real-life processes used, which often have a quite tool-centered view of the exploration process, trying to describe or work . One such example is the FTG+PM approach[34], consisting of two parts: A formalism transformation graph and a process model. They have validated their approach on an industrial case study: a power window implemented in Autosar. In order to scalably handle the transition to a large-scale system, the exploration consists of a multitude of formalisms for different system aspects and at different abstraction levels. Pruning is successively done in order not to unnecessarily evaluate instances at a too low abstraction level.

An obvious way to perform architecture exploration is to use an off-the-shelf general-purpose optimization tool. For instance, usage of meta-heuristic optimization algorithms is an approach that is often applied (see e.g. [21]). Unfortunately, most of these tools have their own special-purpose languages for problem definition, resulting in gaps to the architecture community. Further optimization tools often consider a homogeneous system model, not taking advantage of the separability of different concerns into separate viewpoints. There hence is a need for integration of design tools and optimization tools if the latter are to be useful in architecture exploration. An example of a rather optimization-focused such tool is PerOpteryx[28, 32]. As a further example, Künzli[29] works on performance evaluation of architectures and tool integration support for multi-criteria optimization tools. The Multicube project[40] targeted at providing design space exploration for SoC systems with a focus on power and performance viewpoints. They couple this with tool integration and a framework for multi-criteria optimization. A final example is AQOSA[18], which is closer to traditional architecture design.

The usage of systematic architecture exploration has so far been more limited in design of larger-scale embedded systems, e.g. of networked safety-critical systems in aerospace, automotive and automation[23, 31]. The approaches developed for the kind of systems in coverage of this paper generally have a more narrow focus, such as an explicit focus on a certain viewpoint, e.g. dependability[22, 35]. In the development of systems that are of relevance for this paper, several new aspects appear in comparison to SoC systems, causing the need for a multi-view approach:

• the number of views, and the definition of their viewpoints, is not decided a-priori as an inherent part of the methodology, but is a part of the development process of the system itself

• since the number of stakeholders and therefore number of design aspects applicable to these systems is larger than for SoCs, the number of potential non-functional design constraints is larger and less predetermined

• the views are potentially non-orthogonal and there is a need to ensure consistency between views throughout the exploration process

• the tools that need to be used for the development is an open set, and will depend on the developers’ choice of viewpoints.
A Multi-Viewpoint Architecture Exploration Methodology for Embedded Systems

B.3 Proposed Methodology

The proposed methodology presented in this paper is illustrated in Figure B.1. The methodology identifies the key work tasks and artifacts that are of particular concern in regard to architecture exploration in embedded systems:

- **Preparation**, preparing the work and producing a choice of viewpoints, including tool chain and metrics, design constraints and design goals to be used, starting from informal product requirements, use cases, scenarios and similar. See Section B.3.1.
- **Baseline architecture design**, producing a baseline architecture based on design patterns, norms, templates, existing components and known operational and environmental constraints. See Section B.3.2.
- **Design Variability Modeling**, producing a design variability model describing the design space under consideration. See Section B.3.3.
- **Selection of Search Space**, which selects a subset of the design space for actual evaluation (since exhaustive search normally is not feasible). See Section B.3.4.
- **Architecture Analysis**, providing concrete analysis results for each architecture instantiations that are evaluated. See Section B.3.5.
- **Trade-Off Analysis and Decision**, where final design decisions are taken based on trade-offs between different design goals. See Section B.3.6.

Figure B.1: An overview of the iterative architecture exploration process as described in Section B.3. The blocks represent activities, while the arrows between them represent the data flow between the activities.
• Documentation, where the outcome of the architecture exploration is documented for future use. See Section B.3.7.

These key tasks and artifacts can be instantiated and configured in different ways, depending on the exact context in which the architecture exploration takes place. For example, different companies and industrial domains are likely to use different tool environments. The methodology emphasizes necessary dependencies of work tasks and artifacts. For example, the implied workflow is not strictly sequential; the artifacts are not all mandatory, and the process is likely to be applied in an iterative manner where the same work task is revisited multiple times during the development, either with an increasing level of design detail, or due to backtracking of previous design decisions. For example, design decisions will normally be taken more than once and then fed back to another round of analysis on a more detailed model. This central feedback loop forms the part of our methodology which makes it specific to architecture exploration, evaluation and optimization. As there is no dependency on a specific tool environment, the methodology as well as actual choice of tools require adaptation before adoption.

B.3.1 Preparation

order: requirements, metrics, values of metrics and criteria.

In order to start the architecture exploration process, it needs to be prepared. Preparation can be seen as part of the requirements engineering efforts in which product requirements are analyzed, refined and formalized to make them usable for supporting architecture exploration.

As a key part of the system development process, requirements are formulated for the system under design. Some of these requirements will be quantitative requirements on extra-functional aspects of the system, for example on the needed performance, real-time properties or reliability. Each such characteristic may have one or more different concrete metrics, i.e., a concrete way of measuring the property.

The viewpoints to be used for the specific system under design need to be chosen, including choice of relevant metrics and design criteria. By choosing viewpoints, the type of phenomena (functions, types of properties) of interest and intended analysis capabilities are chosen. Given these, specific tools (languages and computer support) can be chosen. In line with the proposed multi-view approach to architecture exploration, choice of viewpoints is one of the cornerstones of the architecture exploration process, as that choice dictates the implementation of most of the subsequent steps.

It is a challenge in architecture design to refine qualitative objectives with concrete scenarios or other quantitative metrics. The proper choice of metrics is a vital prerequisite for a successful overall architecture exploration process. The choice of metrics should be directly related to the system requirements and stakeholder concerns. However, system qualities may be difficult to measure directly. Therefore, often indirect measures are used (one example is for the number of physical connectors in automotive cable harnesses as indicators of reduced reliability, especially those used in a harsh environment).

The value of the metrics for a certain system may be computed in different ways. The most common include:

• Based on analytical computation, e.g. cost calculations, safety analysis methods such as FMEA, model checking.
• Statistics, e.g., from simulation or partial test runs.
Design criteria may be formed over the metrics, in order to evaluate the architecture in question. Such assessment criteria may be more or less formal. Assessment criteria differ in nature depending on their usage. First and foremost, some metrics are better if minimized, others if as close to a certain value as possible, yet other should be maximized. Further, for some metrics, hard limits are deployed, making them into absolute design constraints (e.g., the reliability level of a safety-critical system), whereas other rather are used as design goals and only make sense relative to the results from other architecture options (e.g. overall component cost) [14]. Hence, and to adjust for scaling effects, it is common to convert metrics into some type of common scale.

B.3.2 Baseline Architecture Design

As a starting point, there will always be some design decisions which are already given. An example of this is where an existing system is to be extended with new functions, and where an architecture is thus given as a basis for elaborating how the new functions are to be fitted into an existing product. A baseline architecture can also be based on domain knowledge, design patterns, standards or informal discussion, and possibly coupled with outcomes from previous iterations of the architecture exploration process. In practice, there will also often be many constraints due to already-made design decisions. For example, a fixed network structure will set limitations on the feasible allocation of software onto a network due to bandwidth and delay constraints. Some functions may already be associated with suppliers that deliver integrated components, implying complete absence of allocation freedom. We call the result from such decisions the baseline architecture design.

The baseline architecture design then corresponds to documenting and (more or less) formalizing what this baseline is. As a special case, the baseline would be empty – corresponding to a new design built with the architecture exploration methodology from scratch. In this case, the baseline architecture design would consist of constructing basic structural alternatives, component choice alternatives, etc. This corresponds closely to the approach of platform-based design, where a set of possible platforms are created.

The decisions within the baseline architecture are not expected to change during the continued design work (unless the design concept turns out to be infeasible and rework becomes necessary). Variability may be put into the baseline architecture, either to describe unresolved design issues or to describe resulting variability.

The granularity used for the baseline architecture models may differ significantly between different organizations and development projects; a small high-level model may be used or a very detailed one.

B.3.3 Design Variability Modeling

A prerequisite for architecture exploration is a defined system design space definition [30], i.e., the set of alternative architectures given in terms of design parameters and their possible variability. These need to be described in order to make it possible to explore the architecture design space. The degree of formality of the description depends on the involved stakeholders, related work tasks and degree of automation involved in the design process. Formalization is however a fundamental necessity for ensuring that correctness of a system design model can be objectively assessed, possibly automatically by letting
a computer do the evaluation. By capturing the design variants the design space for a
certain system architecture is spanned. Together with the previous choice of viewpoints
and metrics they also describe the impact on system properties.

There are different approaches to variability modeling. One approach to create a design
variability model is to use explicit variability points, i.e., explicit model elements that
describe a finite number of alternative solutions. Using such a variability description, the
design space simply becomes all possible combinations of ways to resolve each individual
variability point. CVM[3] and PLUM[43] are such approaches. In other approaches,
variability points may be more implicit – such as is the case with a model fragment
replacement approach, as in CVL[42], which mainly uses a match-and-replace principle,
similar to many model transformation languages and graph grammars.

The baseline architecture, together with the design variability definition and the design
choices for a specific design instance, form the necessary information for the instantiation
of one of the design variants. They can be evaluated with the viewpoints and metrics from
the preparation task.

### B.3.4 Selection of Search Space

In some cases, the architect will only need to evaluate a small number of combinations, but
generally, the total size of the design space could be very large (or even infinite). Several
design variation points may combine with each other combinatorially, creating a design
space too large for exhaustive search. In such cases, for practical reasons, it may become
necessary to only evaluate a subset of the available architecture alternatives. The selection
of search space may be made by using one of the following principles [29]:

- Exploration by hand, i.e., the variants evaluated are selected manually by the
  architect
- Exhaustive search, i.e., all possible variants are checked
- Reduction to a single objective, i.e., the multi-objective design goals are reduced to
  a simplified trade-off function, e.g. a weighted arithmetic mean of similarly scaled
  metrics
- Black-box randomized search – random sampling of the design space
- Problem-dependent approaches – knowledge about the problem structure is used
to reduce the search workload. Examples include design space pruning based on
parameter independence, restriction to promising regions of design space. This
  group of approaches includes e.g. heuristic algorithms in general, genetic algorithms
  and different deductive approaches, e.g. by application of heuristics and “coupling
tightness” between different metrics.

In some cases, the selection is very simple in other cases, complex search algorithms will
be used. The issue is further complicated by the fact that the order of resolving different
variation points might greatly influence the overall amount of work to be performed, such
that there may be an optimal order of making design choices. In embedded systems design
such an early choice is the choice of hardware platform (processor type, amount of memory,
bus architecture, etc).
B.3.5 Architecture Analysis

Architecture analysis provides support for analyzing architecture properties, and thereby means for characterizing and comparing different solutions. For the analysis, it is necessary to provide analyzable models that capture the relevant properties of the design according to the adopted analysis methods and tools (e.g. component behaviors, end-to-end timing, system safety or reliability).

Each architecture analysis tool typically provides support for evaluation of one architecture instance from a single viewpoint at a time. For example: timing analysis requires models that capture the triggering, periodicities, execution time of software and scheduling algorithms; whereas one type of safety analysis requires models that capture the failure modes, errors and error propagation. There are however exceptions; either tools that support concurrent analysis of several architecture instances by including variability in the analysis model, or tools that support concurrent analysis of several different viewpoints of the same architecture instance.

Depending on the choice of representation of the baseline architecture, architecture analysis can be applied either directly on the baseline architecture or to architecture instances where design space variability has been resolved. The choice of approach will vary depending on the type of baseline design used and the general approach to architecture exploration. The former approach requires the baseline architecture to constitute a correct (i.e., a complete, analyzable) architecture model, whereas the latter allows the baseline architecture to be incomplete, and to give the remaining parts of the design as part of the variability binding for each specific architecture alternative.

B.3.6 Trade-Off Analysis and Decision

Architecture design, due to its multi-criteria nature, by necessity includes trade-offs. End system qualities, such as reliability and performance, depend on shared design parameters, e.g., the number of computers, the allocation of functions etc. Decisions about these design parameters will affect multiple qualities. It is the normal case that no single proposal can be judged to be the best one. Rather, different solutions tend to have different advantages and disadvantages in comparison to each other, depending on which stakeholder’s point of view is taken. A common case is to have a trade-off between different design properties, e.g., cost vs. performance, such that there may be a fast but expensive system that is suitable in some contexts, and a cheap but slow that is suitable in others. A solution which is the best for at least one metric, is said to be a dominating one, also referred to as Pareto-optimal. The Pareto-optimal solutions together form the Pareto border. Further, as already mentioned, it is usually not necessary to find an actually optimal solution; rather often a solution close to that level is sufficient.

Trade-off analysis comes in several flavors; it may be more or less formal; ranging from simple round-the-table discussions, over illustrative graphing techniques, to full-blown mathematical optimization.

As an analytical method for decision making, quantitative trade-off analysis [38] allows the comparison of alternative design solutions in regards to multiple metrics. Multi-objective optimization is complicated and design problems are relatively open-ended, making optimization hard. A common approach is to introduce mathematical trade-off functions (i.e. reducing the optimization problem to a single optimization goal) as a means to combine different partial metrics into an overall one. It can be noted that
such an approach is equivalent to introducing additional design goals acting as “tie-breakers” among several solutions where there is no single clearly better solution. Most such mathematical based “tie-breakers” are based on the idea of calculating one joint scoring function based on all relevant metrics. There are several ways to do so. The references [7, 14] provide a short but incomplete list: linear combination, product combination, exponential combination, sum minus product, compromise combination, certainty factors. Of these, the linear combination is most commonly chosen, of which special cases are averages and weighted sums. Which combination function is chosen has a significant impact on which system architecture will ultimately be chosen, since the choice of combination function will hide some solutions on the Pareto border.

Of course, other heuristics identifying sensitivity points and tradeoff points are also possible. Sensitivity points are points where significant impact on a metric can be had by changing a design parameter only slightly, and tradeoff points are where multiple metrics are affected. However, it depends on a mathematical formalization of the design space and an ability to generate and verify architecture instances automatically. It is thus constraining the problem to purely quantitative and objective criteria.

With the potential support of tradeoff analysis, a design decision (i.e. the implicated architectural changes the design decision entail) can now be made, binding part of the design variability. These design decisions are used in two main ways: they form part of the subsequent version of the baseline architecture (or the final architecture to be used) if iterative architecture exploration is performed, and secondly they are made part of the system documentation.

B.3.7 Documentation

One key challenge of architecture exploration is to understand and document the reasons a design was chosen over other. In engineering, design rationale captures the reasoning underlying the creation and use of artifacts [11]; it subsumes the design decision, alternatives and the design decision rationale. Such a decision may be captured at several different levels of detail:

- **R0** – artifact level: on the lowest level, there is no rationale, just the artifact that the rationale is attached to. This means that disregarded design alternatives are not captured. One example of this approach may be using iterative versions of the architecture, as is done in traditional version control systems such as Subversion.

- **R1** – decision level: provides the different alternatives that were part of the design decision, and the taken design decision. This means that the design decisions are included to the level that discarded options can be seen.

- **R2** – decision rationale level: provides the different alternatives in each design decision and the reason for them. The reason may be quantitative (e.g. analysis results) or qualitative (e.g. textual motivation).

- **R3** – rationale rationale level: additionally provides justification for the chosen quality analysis, quality models, metrics and the trade-off analysis that were selected as criteria for the decision. Remark: this would correspond to a fully documented choice of viewpoints during the preparation step of the methodology.
B.4 Multi-View Architecture Exploration: Challenges and Implications

One of the main challenges with any multi-view-based system engineering process is to ensure that all the views are consistent at least each time the architecture description has to be considered as a whole, single entity [17]. Moreover, in the case of architecture exploration, when change decisions are evaluated, it is also important to be able to capture their impact on the views of interest and the associated models. That means that some provisions have to be made so that:

- The initial baseline architecture description is consistent (or rather that all the views it is made of are consistent)

- Each evaluated architecture instance is consistent when a trade-off analysis is performed. That means that any committed design decisions will have to be propagated across the whole architecture description before the next analysis.

While consistency and impact analysis can be seen as two different concepts, they are in fact closely related, as they both rely on capturing what data is shared between parallel viewpoints and views. Hence, the same approaches may be used to handle both. The point is to be able to compare and make decisions based on an overall view of the attributes of a complete baseline architecture instance.

However, there can be phases when ensuring consistency is not necessary, and could even be adversary to the overall process. This is the case if views are handled in parallel by different engineers, to explore possible solutions for a single criterion for instance. An example is control engineers working in parallel with software engineers, using design contracts[45] as a means to avoid conflicts.

There are several ways to handle the consistency problem. How the views are chosen will influence the coupling between the views. Using a more cleverly chosen view decomposition may assist in reducing the consistency problem. Other approaches include building consistency into the system design as such, linking corresponding model elements in different views, and relying on a single source model, e.g. through sharing all model data in one single central repository only.

The approach taken within CESAR was to define a system engineering data model that captures the information shared between the different views so as to be able to import data from the views to the data model (i.e. updating the data model from the view models) and to export data from the data model to the views (i.e. updating the view models from the data model). This implies some way exist to automate those transformations. Usually, import from a master model to views of the data model is easier to automatically handle than export, since the data model itself is generally much simpler than the modeling formalisms used for the viewpoints.

Two approaches can be taken as for the definition of such a data model. Either an effort is made to try to make the data model general and suitable to every possible architecture description or it is designed to be application-specific. While the former is more appealing from a theoretical standpoint, it would involve building an all-encompassing ontology for system engineering, which may not be achievable. Hence, the second, more feasible approach is preferred as the design team can precisely decide of what goes in or not. This does not preclude some genericness however, as domain-specific data model templates can still be defined to be reusable.
The architecture exploration process may be more or less parallelized, depending on the size of the project, the number of stakeholders and development teams involved, and the nature of the project. If the project is large enough to assign specific viewpoints to different engineers, a ‘divide and conquer approach’ is a possibility. Contracts may be one way to decouple views that are not inherently separate. In other cases, a more holistic approach may be preferable. Each organization and setting would have to find their preferred alternative or combination of them, e.g. depending on product complexity or the phase within the development process. A possibility is to use the different approaches in successive design iterations. In the first case (Figure B.2a), each engineer can work separately on the viewpoint he/she is responsible for, making local design changes. The trade-off analysis and the choice of architecture modifications is then taken by examining the change possibilities for each viewpoint, and propagating the validated changes to all the impacted views once to give a new complete architecture instance. The main issue of this approach is that since different engineers work in parallel, they can make conflicting decisions. In that case, a prioritization of decisions could be made during the trade-off analysis step.

Another possibility is to only allow work on orthogonal viewpoints or local modifications on system features that do not impact other features worked on. In the second case (Figure B.2b), small incremental changes can be made to a specific viewpoint and the impact on other views immediately propagated, to have at all times a full overview of the architecture available for analysis. That approach is more in line with the usual software/hardware/allocation-based exploration process found in the literature, but can become rapidly overwhelming as the number of criteria, and thus viewpoints of interest, increases.

B.5 Applying the methodology: A Case Study

To develop and validate the methodology described in this article, an exemplary case study has been set-up with the help of Airbus and EADS Innovation Works. This case study represents a typical problem of multi-viewpoint architecture exploration. The case study
is a simplification of a real aircraft system, using realistic but not real values for the system component characteristics.

The purpose of this activity is amongst others to:

• provide a proof of concept to show the methodology is actually workable
• identify practical issues not necessarily foreseen during its theoretical definition
• provide feedback for further refinement of the approach
• assess its usefulness, benefits and drawbacks

The case study is not described in all details, due to space limitations for this article and the fact that less emphasis has been put on design variability modeling and documentation.

B.5.1 Simplified Doors and Slides Control System: System Description

The case study “Simplified Doors and Slides Control System” (SDCS) addresses the design of a system that has the following main objectives:

1. Inhibit pressurization of the aircraft cabin on ground if at least one door is not fully closed, latched and locked
2. Supervise flight locks and latches on passenger doors in order to prevent opening of doors during flight
3. Monitor the status of the indicators of the aircraft doors and associated evacuation slides

Since the SDCS system is quite complex, only the design relevant to a specific subfunction is considered in this article. The focus is on the Pressurization Prevention System (PPS) that shall ensure the first objective mentioned above. Pressurizing the aircraft while all doors are not in the proper state, followed by takeoff, could lead to a door opening in flight due to the pressure differential, and thus an in-flight decompression. Since this is a catastrophic event, the PPS is safety-critical. Consequently, a safety requirement can be formulated as follows: “The probability for incorrect activation of pressurization when any door is not fully closed, latched and locked shall be less than $10^{-9}$ per flight hour.” In addition to the safety requirement, other criteria have to be taken into account in order to achieve an optimal design solution, as described in section B.5.2.

In this example we assume an initial physical architecture, illustrated in Figure B.3, used to define the baseline for architecture exploration in this section. In particular:

• Communication between physical components are based on Avionics Full Duplex (AFDX) and ARINC 429 (A429) real-time buses
• Signal interfacing (Sensor acquisition, bus bridging) is devoted to Remote Data Controllers (RDC)
• Computations are handled by Core Processing Input/Output Modules CPIOM
• The pressurization subsystem is fixed, and features two dual-channel Outflow valves Control Units (OCU) fitted with two A429 inputs each, controlling the outflow valves (OVF) proper.
• Each door (noted 1L for the left door and 1R for the right door) is fitted with a closed sensor (Closed Sens 1L/1R) and two redundant Locked and Latched sensors (L&L1 Sens and L&L2 Sens)
Figure B.3: Physical architecture of the PPS
B.5.2 Preparation

The purpose of the preparation steps is to define and identify the exploration goals (including constraints to be met) and choose the relevant viewpoints.

B.5.2.1 Choosing Viewpoints, Metrics and Criteria

In our example, three main viewpoints were chosen for analysis: the latency viewpoint, the weight viewpoint, and the safety viewpoint. Moreover, since those share data with the functional and physical viewpoints, those two are also included as ‘reference viewpoints’, meaning they will not be directly analyzed during the process.

Latency viewpoint. The main concern addressed is the end-to-end latency between the detection of the aircraft doors closures and the pressurization authorization signals, accounting for acquisition, processing and communication delays. The viewpoint itself is very simple, as it depends on individual timing-related parameters to build a computed list of end-to-end latency metrics for the signals of interest. The analysis models needed are not so simple. Indeed, to analyze the end-to-end latency of a signal, information about the physical structure of the system, the allocation of functional components to physical components and the corresponding data flows have to be known. That means that the latency viewpoint shares information with both the functional and physical viewpoints. In addition to that information, timing-specific properties are needed, such as:

- Sensor firing periods and task / process execution periods
- Worst-case execution time of processes, depending on their physical allocation and network latencies
- Communication model between different components (i.e., event-driven, periodic, aperiodic...) and data flow paths between components

The modeling formalism chosen for the analysis model is AADL, considering the pre-existing model and tool availability. The analysis method itself is described in [19], and is implemented within the OSATE [5] tool provided by the CMU SEI, which outputs worst-case latencies for the requested data paths.

Weight viewpoint. The weight viewpoint is the simplest one in this experiment. The main concern addressed is the total weight of the physical system, excluding cables and power supplies as a simplification. In a real-world setting these should also be included. The viewpoint consists of a single overall net weight figure. However, as above, the analysis models are more complex since they have to relate a weight attribute to each component of the physical structure. Hence, the weight model actually refines the physical view of the system with the weight of each physical component. The chosen formalism for the analysis model is also AADL. The analysis tool is again integrated into the OSATE framework and is able to output partial subsystem weights and overall weight of the whole system.

Safety viewpoint. This viewpoint is the most complex of this study. The main concern addressed is the occurrence probability per flight hour of the failure condition ‘Pressurization authorized while all doors are not fully closed, locked and latched’. Indeed, while simple physical component substitution can reuse analysis models
### Table B.1: Criteria, metrics and objectives for the pressurization prevention system

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Objective</th>
<th>Type</th>
<th>Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response time between any door closed, latched and locked and pressurization authorization</td>
<td>&lt; 250 ms</td>
<td>Constraint</td>
<td>Maximum end-to-end signal latencies between door sensors and the overflow control units</td>
</tr>
<tr>
<td>Weight of the PPS</td>
<td>&lt; 60 kg</td>
<td>Constraint</td>
<td>Total weight of the components</td>
</tr>
<tr>
<td>Occurrence of Failure Condition ‘Pressurization authorized when doors not fully closed, latched and locked’</td>
<td>&lt; $10^{-8}$ occurrence per flight hour</td>
<td>Constraint</td>
<td>Direct evaluation of the failure rate</td>
</tr>
</tbody>
</table>

without much change, even the slightest change in the structure of the system or the allocation of functional components to the physical structure mean that at least a partial safety analysis has to be redone on the system. Such concerns thus make an architecture exploration process difficult to fully automate. Still, the following models are related to the safety viewpoint:

- Functional models, including the definition of failure modes of part functions and how they can propagate through the system.
- Physical models, including attributes such as failure probabilities
- Fault trees, minimal cut-sets, that may be automatically generated from the dysfunctional models

The main model formalism for the computing of actual safety metrics retained here is AltaRica [6], used to capture all these aspects, as an input to the safety tools BPA-SD9 and BPA-FT9 from Dassault Systèmes [15].

These viewpoints refer to both viewpoint-specific and shared data. Once views are instantiated, the same data will thus find its way into multiple views and some provision should be made to ensure the consistency of the overall architecture description.

For the PPS, the criteria have been chosen to represent some realistic concerns that stakeholders might express in a real settings. Table B.1 details those criteria, their associated metrics, and their nature.

#### B.5.2.2 Defining the Architecture Exploration Data Model

In the case study we choose to adopt a common system engineering data model (refer to Section B.4) in order to facilitate trade-off analysis and to ensure cross-view consis-
tency. Designing the data model requires sorting between data that is shared amongst multiple viewpoints and data that is strictly private to a viewpoint. In the case of the PPS, this analysis has been completed through round-the-table discussions between the different stakeholders (system architect, system designers and safety engineers) and led to identifying data shared between the Latency, Weight and Safety viewpoints, as shown in Figure B.4, where a common core data model represents the shared data while specific view-related data are described as simple properties. The common core data model is used for view consistency checking and impact analysis, using model comparison as described later. Viewpoint-specific data is not included in the common data model.

Once the data model has been defined, either from scratch or by reusing existing definitions from a library, it has to be implemented. The case study used Ecore and the Eclipse Modeling Framework [4] to handle data model operations. Using this framework, it is possible to use model transformations to both check for inconsistencies and perform impact analysis. In particular, the ATL [1] transformation tool was used.

Two operations have to be implemented: data import to the central shared model from each respective viewpoint and data export in the opposite direction. Using these two operations, consistency checking, change propagation and impact analysis can be implemented. For instance, checking consistency between an analysis view model and the overall system description can be performed using the following method, as illustrated by figure B.5:

1. An ATL transformation is used to transform the view model (here in AADL) into
Figure B.5: Using a common engineering data model for view consistency checking

data-model language (which is a kind of domain-specific language, noted DSL in the figure) used to describe the data model. Only the shared data ends in the new model.

2. Another ATL transformation is used to extract the shared data relevant to the corresponding viewpoint from the common data model, yielding a partial data-model instance

3. Both partial data-model instances are compared, using a comparison framework (here the Eclipse Model Compare framework) to identify inconsistencies. If no differences are found, it can be assured that the shared data present in the view model is consistent with the current state of the architecture design.

Other operations can be implemented in the same spirit, using the same tools. However, for the purpose of this case study, only consistency checking was implemented.

### B.5.2.3 Putting it all Together: Defining an Exploration Tool Chain

Once criteria, relevant viewpoints and shared data have been identified and defined, a specific tool chain can be built to enable design engineers to concretize an exploration process tailored for the problem at hand. The tool chain should build around tools that have been chosen during the definition of the viewpoints.

Such a tool chain should provide at least the following:

- Evaluation and verification services, i.e. actual evaluation tools as defined in the viewpoints, including interoperability adapters if needed / available. The main
issues here are to define how analysis models will be fed to the tools and how results will be gathered.

• Data management services, allowing the storage, versioning and access control of design artifacts during exploration.

• Analysis services, i.e. tools able to gather evaluation results, assist in the decision process, including impact and trade-off analysis, and document it. They can range from simple visualization tools for manual analysis to full edged trade-off analysis tools able to automatically check constraints and maybe also suggest design changes.

Following those considerations, a partial exploration tool chain was prototypically developed in the frame of the CESAR Project to support the PPS function analysis, see Figure B.6. The motivation for this tool chain is to automate as much as possible the tasks of data management, tool invocation, consistency checking and view updating without trying to fully automate the whole process. This tool chain however remains a “proof of concept”: much more refinement is needed to be able to propose an industrial-strength tool chain.

The main components of the tool chain are:

• Requirements Management tool: In our case we have chosen to use DOORS 9.2 for defining and storing textual requirements and objectives for the PPS function.

• Baseline Modeling tool: We are using Rhapsody 7.6 for defining an initial baseline Architecture of the SDSCS. The initial model contains information about the operational capabilities of the SDSCS (Operational View), the Functional Needs to be supported by the SDSCS (Functional View), different alternative System Architectures (Implementation View), allocations of Functions to Components (Allocation View), and the key Failure Cases (Failure Case View).
• Additional Modeling tools for specific viewpoints: One of the conclusions that we took from the CESAR project is the fact that it is not feasible to integrate all possible viewpoints into a single modeling tool. We need to add additional modeling tools to the exploration tool-chain. For our exploration tool-chain we have decided to use OSATE (AADL) for detailed weight and timing models of the SDCSC, and BPA-SD9 for a detailed Safety Model. For a real Aircraft system exploration, many more tools would have to be connected to the exploration tool-chain.

• Analysis tools: OSATE and BPA-SD9 have already built in analysis capabilities. In addition to this, we decided to build an own analysis tool that is able to calculate failure probabilities on the Rhapsody model, and hand the results over to the Isograph FT+ tool.

• A ModelBus repository with a dedicated graphical interface (= Exploration tool) which holds the following information for each architecture instance: a complete data model (as an XMI file generated from merging the view models through specific model transformations), subsets of the baseline and additional models (Rhapsody, AADL and BPA-SD9 models), evaluation results (analysis results from respective views), and documentation of design rationale in form of text files.

B.5.3 Baseline Design

In the case of the PPS function, the baseline architecture is structured as a complete but not fully resolved architecture. For instance, the type and characteristics of the physical components are not defined a priori and timing and weight parameters are missing. In order to obtain an analyzable architecture according to the chosen exploration viewpoints, more views resolving the baseline variability points have to be created, leading to an initial analyzable architecture description instance. Following the viewpoint definitions given earlier, these lead to Rhapsody models capturing both the functional and physical structure of the baseline, refined by AADL and AltaRica models including specific safety, timing and weight parameters.

These view models are the artifacts that will get changed during the exploration process to yield new and possibly more efficient architecture instances. They do not themselves contain any information about possible variations. Expressing and formalizing those variations is done at a higher level, and forms the design and search space description. Figure B.7 shows a partial Rhapsody model derived from the SDSCS initial baseline (implementation view).

Figure B.8 shows an AADL model that is used for both latency and weight analyses, as well as a BPA-SD9 model that is used for Safety Analysis using the AltaRica language.

B.5.4 Design Variability Modeling

As stated in Section B.3.4, the design space description captures all the possible designs for a given system according to a finite or infinite set of possible variations. According to the full methodology described in Section B.3, selection of the search space should be preceded by formally modeling the design variability. At a minimum, the choices designers can make during design space exploration have to be described informally. In this case study, the following variation possibilities have been selected:
• Structural modifications to the physical architecture (removal / addition of an AFDX switch)

• Changes to the allocation of software components to the hardware architecture

• Substitution of physical components, two different types each (type A and type B)

While limited, as the search space is the combinatorial composition of all those possibilities, those variations can already lead to thousands of possible architecture instances, accounting for substitutions alone. The variations were only informally described (as above), hence requiring manual model changes for evaluation of the different alternatives.
B.5.5 Selection of the Search Space

The case study used a simple manual and iterative selection of search space. That means that after each alternative has been evaluated, a new set of architectural changes were devised, whereafter repeated architecture evaluation was performed. Due to the manual character of the changes in the case study, all in all only a few architectures were evaluated.

- Initial baseline architecture
- Reduced latency of “flow 2” between left door sensors and second OCU
- Reduced overall weight by lighter components
- Change network structure (3 switches instead of two).

B.5.6 Architecture Analysis

Evaluation of a given architecture implies evaluating every view that makes its description according to the exploration criteria, in order to get an overall picture of its desirability and drive the choice of the final architecture.

In the simplest case, as performed for the PPS case study, the tools are simply executed on the view models (OSATE for the AADL models, BPA-SD9 for the AltaRica models) and yield computed metrics related to the viewpoint concerns. For example, the weight view is represented as a dialog box showing the system’s total weight. The results are both displayed on screen and added to CSV-formatted files for further analysis. While in a complete tool-chain, results for every analysis would be automatically gathered in a single place, in this case study no such data management system was available and gathering the results is a manual operation.

B.5.7 Trade-off Analysis and Decision

Figure B.9 shows some analysis results for four successive architecture instance possibilities (x-axis). The initial baseline architecture is changed to reduce the “flow 2” latency, between the left door sensors and the second OCU. The second architecture is then changed to reduce the overall weight by choosing lighter components. This has a drastic effect on flow latencies though (third architecture). The decision is made to backtrack to the second instance and change the network structure, by switching from the initial, 4-switch based network to a 3-switch triangular structure yielding the fourth architecture instance. That one improves the weight of the system and also does not affect the flow 2 latency fixed in architecture number 2. Since it also improves the first flow latency, it is accepted. Another backtracking would have had to be performed otherwise.

B.5.8 Documentation

Once a design change decision has been taken and approved, the baseline architecture instance has to be changed / updated to reflect that decision. Since according to the decision process each view is handled independently, the change will be first made into the view models whose analysis led to the decision (i.e. the models in the view containing the sensitivity point). To generate a new complete, analyzable instance of the architecture, other views may have to be updated too. A view that is not impacted at all by the decision will remain the same and may be excluded from further architecture analysis.
A Multi-Viewpoint Architecture Exploration Methodology for Embedded Systems

Figure B.9: Some metrics after evaluation of four successive baselines during iterative exploration. The horizontal axis depicts the four different architecture variants evaluated.

B.5.9 Outcomes of the Case Study

The case study has emphasized several different practical aspects of architecture exploration, which makes it a challenge in practice:

- The preparation phase is central in order to focus the process on the right aspects. It is essential to get the right stakeholders involved, finding the vital concerns and identifying suitable metrics. Without these, the rest may be done perfectly, but will be exploring and optimizing under the wrong assumptions.

- Setting up the supporting infrastructure, including models and tools, is a significant investment in time. Once the infrastructure is in place however, iterations can commence and the work can be reused. This includes information management which is particularly important in an iterative exploration process, where a history of the design has to be gathered to allow for traceability of design decisions. A tight integration with evaluation tools is often desirable, to make architecture analysis as seamless as possible. It is also vital to ensure consistency of the architecture description between different views. At a minimum, the tools should be able to warn the designers of possible inconsistencies when importing and / or exporting models.

- In order to fully automate architecture exploration, it is necessary to have formal models of architectures as well as architectural changes (design variability) and the different criteria applicable to the design. Although such formalisms are emerging, they are not yet established.

B.6 Discussion and Conclusions

In this article, we have described a methodology for multi-viewpoint architecture exploration of networked embedded systems. A prototypical tool chain has been developed to apply the methodology on a simplified case study from the aerospace domain. In this section, we will discuss further challenges that need to be resolved in order to allow a full-scale industrial application of the methodology.
B.6.1 Industrial Needs for Architecture Exploration

The CESAR project made it apparent that the process maturity varies among the industrial domains. While the aeronautics domain already has established use of validation methodologies and tool chains that are suitable for incorporation into an architecture exploration process, other domains such as the automotive are not as advanced, making domain-specific demands difficult to assess and consolidate. Part of the reason for this is a perception that exploring a design space to find optimal or at least better architectural solutions can be detrimental in domains where development cost and time-to-market are important metrics of market success.

Even in the aeronautics domain, architecture exploration as a structured approach to system design is still a rather new approach for networked systems. A key challenge that needs to be resolved to apply the methodology in industry is related to scalability, especially with regard to the number of persons involved in the architecture exploration process. In the simplified example presented in the previous chapter, the architecture exploration has been performed by a team of 3 to 4 persons. In a real aeronautic network system, dozens if not hundreds of specialists have to be involved in the architecture exploration process. Another significant challenge with all architectural development is the issue of incremental and concurrent development. Cross-dependencies throughout the entire architecture may sometimes be triggered by only minor changes. Also, many more viewpoints would have to be considered, such as maintenance, electromagnetic compatibility, thermal, structural or aerodynamic impacts. This leads to a need of integration of all kinds of detailed modeling and analysis tools.

As a result of these considerations, improvements of industrial practices have to be realized mainly with regard to the following aspects:

- Realization of a generic tool and data integration approach to enable an efficient integration of specific modeling and analysis tools for additional viewpoints
- Improvement in the usability and stability of tools and tool chains used for architecture exploration
- Improvements in the ability of tools used for architecture exploration to handle large data models
- Connection of the architecture exploration tool chain with other key services such as user management, traceability, and configuration and change Management

In Section B.5 we have presented a promising approach for data integration based on the use of a common data model. For industrial use however, improvements are needed mainly on the tool support for this approach. Tools are needed that allow to easily set-up and implement data models and tool connectors.

B.6.2 Automating Architecture Exploration

The level of automation needed within a concrete tool chain is an important preparation step before choosing which tools to use. Different levels of automation are possible:

1. Manual Design Space Exploration with some tool support
2. Semi-automatic design space exploration
3. Fully automated design space exploration
In the first case of tool supported manual design space exploration, engineers will manually define an initial baseline architecture using one or several modeling tools. Analysis tools may however be invoked automatically, and analysis results could be automatically visualized in a single dedicated exploration tool. In case that the initial requirements have been defined in a formal way, it may also be possible that the exploration tool performs an automatic check of analysis results against the initial requirements.

In the case of a fully automated design space exploration, the initial baseline architecture would be defined automatically, typically by some kind of solver. This requires however that the metrics, constraints, optimization goals and functional needs have been defined in a formal way and that all possible variations in the design have been defined as well. One approach of defining design variations is through explicit enumeration of variants using some variability description language. Semi-automatic design space exploration is between both extremes. It means that part of the design space are manually defined, other parts are left open for automatic optimization.

Results from the CESAR project seem to indicate that a fully automated design space exploration may be most adequate in the early concept phases. In these phases, the design space is huge, while the accuracy of the available data is rather limited. The main purpose in this phase is to reduce the design space from several thousands of possible architecture alternatives to a few promising ones. For this step, many metrics can be approximated, so that the use of solvers for architecture generation becomes feasible.

In later phases, the design space exploration activity can be characterized rather as an adjustment or fine-tuning of a more or less selected baseline architecture. The focus is on increasing the confidence in the baseline architecture, and ensuring that it meets all relevant requirements. Therefore, very detailed analysis results need to be acquired with a high level of confidence against many different viewpoints. For this, tool supported manual design space exploration seem to be more relevant.

B.6.3 Conclusion and Future Work

Implementing an architecture exploration process can help design teams make better decisions about the architecture as a whole, enhance traceability of those decisions and formalize their rationale by expressing hard factual arguments, in the form of metrics, for each decision. Industrial deployment of architecture exploration could target early stages, with little detailed information and formalized models, or later stages where more detailed architecture exploration and verification is possible.

The methodology presented in this paper would benefit from further case studies, for example to illustrate its use in different domains and for further validation among the multitude of various settings of architectural exploration. This should also provide opportunities for refinement of the methodology. During the work with the methodology, the following topics have also emerged as candidates for further work:

- Investigate modeling techniques for architecture exploration that are scalable, support multiple viewpoints and are able to express both design and product line variability.

- Related to modeling, there is also a need to adapt existing constraint and requirement languages such that they can be more easily integrated with architecture design and the design process as a whole.
• Evaluating architectures from multiple viewpoints leads to challenging multi-goal optimization and multi-criteria decision problems for which multiple algorithms can be employed. Here there is a need to evaluate and, if required, adapt these techniques for embedded systems architecting.

• There is a need for research involving architects in order to develop a better understanding of suitable functionalities and abstractions of architecting tools at different design stages.

• Finally, to enable tool development and information reuse, there is a need for a systematic approach for information management and tool integration. With its many concerns, architecture exploration is a typical example of a tool functionality that is highly likely to benefit from integration, reusing information such as requirements, using other analysis tools and providing information to other tools, e.g. as in the iFest approach[44]. Further research should investigate recent advances in tool integration and its applicability to architecture design.

Acknowledgment

This work was partially supported by the European Commission within the project CESAR which is funded by the ARTEMIS Joint Undertaking under the Grant Agreement No. 100016 and by national programs/funding authorities.

References


[26] Graham Jolliffe, *Cost-efficient methods and processes for safety relevant embedded systems (CESAR) - an objective overview*, in *Proceedings of Safety-critical Systems Symposium 2010* (Chris Dale and


A Multi-Viewpoint Architecture Exploration Methodology for Embedded Systems


