Quality-oriented Decision Support for maintaining Architectures of fault-tolerant Space Systems

Lukas Märtin
TU Braunschweig
Braunschweig, Germany
l.maertin@tu-braunschweig.de

Anne Koziolek
Karlsruhe Institute of Technology
Karlsruhe, Germany
koziolek@kit.edu

Ralf Reussner
Karlsruhe Institute of Technology
Karlsruhe, Germany
reussner@kit.edu

ABSTRACT

Due to hostile environments, space systems are equipped with hardware redundancies to guarantee proper operation. For reconfigurations beyond redundancies, manual decision making is needed, which results in down times, communication efforts and man hours in maintenance phases.

We investigate automated reconfiguration decision support that determines Pareto-optimal architectures w.r.t. variable hardware availability and quality properties. Reconfiguration options for control software according to available sensing and actuation hardware are derived and prioritised w.r.t. predicted qualitative impacts. The knowledge about relations of the system's variations is persisted in a decision model at design time on the level of software architectures. Upon a resources fault, the model is traversed for an alternative architecture. This promotes a transparent analysis of available deployments as well as an acceleration of the reconfiguration process during maintenance. We provide tool support for analysis and a concept for reconfigurations during operation.

For evaluation, we inspect a reengineered extension of the attitude control system of the TET-1 micro satellite.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Computer-aided software engineering, Decision tables*; D.2.4 [**Software Engineering**]: Software/Program Verification—*Reliability*; D.2.5 [**Software Engineering**]: Testing and Debugging—*Error handling and recovery*; D.2.9 [**Software Engineering**]: Management—*Software configuration management*; D.2.11 [**Software Engineering**]: Software Architectures; C.4 [**Performance of Systems**]: Fault tolerance

Keywords

software architecture, maintainability, reconfiguration, decision support, Pareto optimisation

ECSAW '15, September 07 - 11, 2015, Dubrovnik/Cavtat, Croatia

DOI: http://dx.doi.org/10.1145/2797433.2797484

1. INTRODUCTION

While developing control software for space systems, extensive mechanisms for handling hardware faults are implemented. Due to harsh environmental circumstances, such fault-tolerant systems are equipped with redundancies for stressed hardware resources to guarantee continuously operation, even when some resources fail. Today's implementations partially support the maintenance by autonomous reconfigurations of homogeneous combinations of redundant resources. To perform wide-ranging reconfigurations, beyond switching of redundant resources, personnel for manual decision making is needed.

For unmanned space systems the maintenance process basically relies on the (1) transmission of error logs to ground station, the (2) development and (3) transmission of a patch, a (4) system reboot and (5) remote-monitoring activities. Thus, huge efforts in human resources and communication result, significantly rising w.r.t. long life cycles and limited possibilities for resource replacements. In addition, each fault potentially leads to down times and reduced mission time. Apart of functional properties, the fulfilment of quality properties is crucial for appropriate operation. Thus, each reconfiguration decision has to be made in consideration of contradicting quality properties.

Flexible techniques to support the maintainability are necessary, as shown by the extraordinary reconfiguration done in course of *NASA's Kepler* mission. Following to faults in the gyroscope-like actuator system for attitude control, the spacecraft was no longer able to perform its mission. After weeks of down time, the researchers finally found a solution, called *K2: Kepler's Second Light*. By taking advantage of the pressure, expressed by photons in sun light, the system was able to continue operation with the remaining actuators.

Thus, flexibility and reliability in operation are essential for modern space crafts designs. For that, we propose a generic approach to support reconfigurations. Our intention is to reduce maintenance efforts by automated decision making, optimised towards variable hardware resources and quality properties. We inspect possible design options of the control software at the level of software architectures w.r.t. available resources and prioritise these options. The analysis knowledge of variations is persisted in a decision model at design time. Upon a resources fault during operation, this model is traversed to autonomously identify an alternate architecture. This promotes both a transparent analysis as well as an accelerated reconfiguration process at run time.

We apply our approach to a subsystem of TET-1[1], a multi-

[1]https://directory.eoportal.org/web/eoportal/satellite-

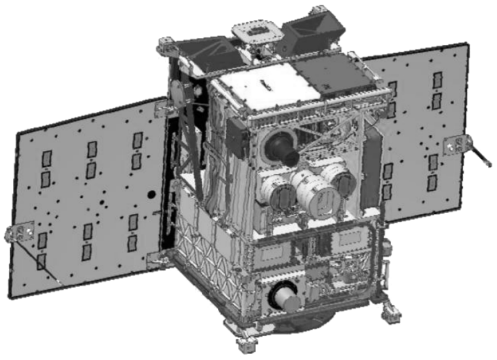

Figure 1: TET-1 Micro Satellite Bus [Image Source: Kayser-Threde GmbH, Germany]

purpose micro satellite for a wide range of missions [5], launched in 2012 to verify scientific instruments in space, shown in Fig. 1. For orienting and positioning, an Attitude Control System (ACS) [12] is used. It consists of several redundant hardware resources: five sensing systems, e.g., *GPS* and *Star Camera System* for position determination, and two actuation systems, i.e., gyroscope-like *Reaction Wheel System* and a *Magnetic Coil System* for attitude changes. The ACS provides three-axis orientation and stabilisation with high accuracy of alignment and precise pointing knowledge. In particular, a high degree of reliability is realised by redundancies and extensive capabilities for detecting faults.

In original design the variations of the ACS differ only in redundancy switches. From an architectural viewpoint, variants are quite similar to each other. For validation, we reengineered and extended the ACS by (1) cross-wise connection of sensors/actuators with compatible controllers, (2) additional combinations of actuators, (3) differing post-processors of sensors. This allows us, e.g., to (1) connect each GPS antenna to a various GPS receiver, (2) to use the magnetic coil system without reaction wheels, and (3) to use a low or a high resolution star camera. For design space exploration, we modelled and analysed the extended ACS with Feature Modelling techniques, resulting in approx. 20,000 variants.

The remainder of this paper is organised as follows. Sect. 2 discusses guidelines and techniques related to our work. The formalisation of our approach and a conceptual validation is presented in Sect. 3. Tool support for analysis at design time is shown in Sect. 4. We conclude and promote future work in Sect. 5.

2. STATE OF THE ART

In previous work, we investigated software engineering for space systems [16] and discussed possible enhancements in fault-tolerant system design [15]. The maintenance process of space systems is supported by various models, languages, and tools for *Fault-Detection, Fault-Isolation and Recovery Techniques (FDIR).NASA* investigates challenges and techniques in fault-tolerant software engineering for quality assurance in all phases of the mission life cycle, cf. [17]. In particular, n-version software techniques [13] are used for handling faults [3]. Multiple implementations, slightly differing in functionality, but related to a common base, are deployed as design options for human decisions. In implementing operational control software, the majority of space systems is still developed in plain code. To assure functionality and quality of these implementations, classified code from successful missions is reused. We focus on a more abstract level where control software is represented by software architectures and hardware by data sources (sensors) and sinks (actuator).Hence, our optimisation approach is settled in area of component-based software engineering for embedded systems [8].

Most similarities to our approach can be found here in area of cloud-based systems. Frey et al. [6] inspect reconfigurations as deployment options for cloud-based systems derived by genetic algorithms. The authors define rules at design time to systematically change the deployment of a system at run time, e.g., in case of overloads. Related to that, Jung et al. [9] determine policies to adapt a running system. These policies are derived at design time by a decision-tree learner trained with possible system configurations, generated from queuing models. Both approaches neither explicitly represent the reconfiguration space nor the qualitative trade-offs between design alternatives. Our approach preserves such knowledge from the candidate generation process to related and prioritise near-optimal candidates. Similar to changing platforms in cloud infrastructures, we consider unstable hardware resources, that might partially fail in operation.

A more hardware-oriented approach is proposed by Malek at al. [14] in context of self-managed systems [11]. The authors provide a trade-off model to identify a suitable deployment architecture for a software/hardware systems. Even if that approach can be applied as foundation for our work to distinguish between configurations, the authors do not integrated a prioritisation of all feasible design alternatives for run time decision support.

Our notion of variability in architectures relies on the concept of *Degrees of Freedom (DoF)* [10]. In software architecture optimisation, *DoF*'s describe the possible changes an optimisation algorithm can explore, both at design time (i.e., design decisions such as component selection) or at run time (e.g., configuration options such as deployments) [1].

Our previous work provides optimisation of software architecture models for performance, reliability, and costs [10]. Starting from a base architecture model, the *DoF*'s span the search space, which is explored using genetic algorithms. Multiple variations (*candidates*) of the base architecture are generated, analysed for fitness to the quality needs, and evolved using genetic operators such as mutation and crossover. The result of an optimisation run is a set of *Pareto-optimal* candidates.

In our approach, we use knowledge from the generation process to relate each candidate to alternate candidates with slightly different changes in hardware requirements and qualities. Similar to that, Barnes et al. define relations between architectures as candidate evolution paths [2]. These paths specify a search-based reconfiguration process from a source to a target architecture via a sequence of transient architectures.

Even though existing techniques assist personnel during maintenance, the near-optimal target candidate yet has to be figured out manually. In our approach, a major goal is to make such decisions semi-automatically. In addition, the decisions are done at run time in a compact design space of

missions/t/tet-1

options. For identifying the actual candidate for redeployment, our approach builds up on the work of Florentz et al. [4]. Trade-offs between contradicting quality properties are ordered hierarchically. We deal with predicted values from design time, refined by weightings at run time. Here, we follow the observations of Grunske [7], who proposes a sophisticated prediction framework for quality properties w.r.t. differing quality domains from early design stage until simulations at run time.

3. APPROACH: DECISION SUPPORT FOR RECONFIGURATIONS

In practical application human resources and communication are major limiting factors in quick and continuous maintenance. Thus, our approach is designed for autonomous reconfigurations on-board of the space system. To support transparency at the same time, we choose an approach at the structural level of software. Our approach is based on a state-based graph with software architectures as states, related by transitions. Here, we present an excerpt of the formalisation of our notions for architectures, usage profiles and quality models. Details are described in our previous work [15].

Definition 1. Let $\mathcal{R}$ be a finite set of hardware resources. An *Architecture Relation Graph (ARG)* is a triple $\langle \mathcal{A}, \triangleright, Req \rangle$ where

- $\mathcal{A}$ is a finite set of architectures, i.e., Pareto-optimal candidates,
- $\triangleright \subseteq \mathcal{A} \times \mathcal{A}$ is a distance relation between architectures, and
- $Req : \mathcal{A} \rightarrow \mathcal{P}(\mathcal{R})$ binds architectures to a subset of required hw resources.

The set of architectures corresponds to the candidates derived from a base architecture by system specific *DoF's*. Each architecture requires a minimal subset of hardware resources in *Req* for operation. Information about commonalities and differences of candidates are represented as distances $\triangleright$, resulting in transitions in an *ARG*.

In the following an *ARG* is constructed for an excerpt of our application scenario. The excerpt consists of control software modules and corresponding hardware resources for actuation by the *Reaction Wheel System (RWS)* and the *Magnetic Coil System (MCS)*. Two instances of the *DoF*'s for *Component Selection* defines design options for candidate generation: {*RwCtrl-0, RwCtrl-3, RwCtrl-4*} and {*McsCtrl-s, McsCtrl-a*}. The options for the RWS differ in necessary hardware resources (0, 3, or 4 wheels). The MCS is available in a standard implementation (*McsCtrl-s*) to solely eliminate disturbances of the RWS, and in an advanced one (*McsCtrl-a*) for stand-alone operation. As prerequisite for the candidate generating, we assume exemplary rating values in Tab. 1 for the quality properties *energy awareness* [0;3], *agility* [0;5], and *CPU load economy* [0;2] for the selectable design options. These values are calculated as part of the analysis of the candidate generation process at design time. The value scales (in square brackets) are derived from the minima and maxima. For trade-off analysis the values are normalised during the process. For the sake of illustration, these values abstract from exact float values from hardware data sheets and *RwCtrl-0* is ignored (zero values).

Table 1: Rating Values for Quality Properties q_i

Prop	RwCtrl-3	RwCtrl-4	McsCtrl-s	McsCtrl-a
En. A.	1	0	3	2
Agility	3	6	1	2
CPU E.	5	4	3	0

For comparison in trade-off analysis, we normalise these values. In our example, the generation of Pareto-optimal candidates leads us to three architectures with normalised rating values, with a lower bound of 50%, as described in Tab. 2. These candidates build up the set of architectures $\mathcal{A}$. Starting the construction of the *ARG* with all elements of $\mathcal{A}$, transitions between architectures are added. For that, *Req* holds for each $a_i \in \mathcal{A}$ a finite subset of required hardware resources from $\mathcal{R}$. Transitions are added for each subtractive change in these required hardware resources, i.e., if an alternative architecture not requires a resource, that is required by the current architecture, a transition is added. We assume, that hardware resources remain unavailable upon a fault occurrence. Thus, all transitions are unidirectional, i.e., for $(a_0, a_1) \in A \times A$ it is $Req(a_1) \subset Req(a_0)$. In addition, weightings α_i represent the current mission profile for the quality properties q_i, defined at run time for multi-purpose systems. Hence, the trade-offs between quality properties are set during operation. The transition labels are described by Equation 1.

$$p := \sum_{i=1}^{N} \alpha_i * q_i \mid \sum_{i=1}^{N} \alpha_i = 1, q_i \in [0,1] \qquad (1)$$

The transition with the highest priority p is selected to leave the current state. If there are equal priorities, the selection is performed randomly. The construction of the *ARG* holds if all elements of $\mathcal{A}$ are covered once. The *ARG* for our example is constructed as depicted in Fig. 2. It consists of three nodes, representing the architectures with required hardware resources, and the yet unweighed quality sums as transition priorities. Architecture a_0 is applied for initial system's operation.

Let us now assume, that one of required actuators of *RWS*, used in a_0, fails. Following to the autonomous fault detection in the original ACS design for fault tolerance by 3-of-4-redundancy, the reaction wheel would be deactivated and the system continue operation with the remaining three wheels. Thus, architecture a_1 would be applied.

However, also in our approach, a state change from a_0 is enforced by a violation of $Req(a_0)$, but now two outgoing transitions to alternate architectures a_1 and a_2 are enabled. The decision which state is selected depends on the current mission profile and the quality rating of the alternate architectures. Under the assumption, that the current mission needs good support for energy awareness and agility, but CPU load is less important, the weightings are set to $\alpha_{1,2} := 0.45$ and $\alpha_3 := 0.1$. In combination with the rating values from Tab. 2 an near-optimal reconfiguration decision is possible now. The transition to a_1 has a lower priority and the Pareto-optimal architecture a_2 is selected for further operation. Aligned to the current mission profile, the reconfigured ACS saves energy (q_1) while still considering agility (q_2) and CPU load economy (q_3).

Table 2: Pareto-optimal architectures with Qualities

a_i	DoF Choice	En. Aw. q_1	Agility q_2	CPU Ec. q_3
a_0	{RwCtrl-4,McsCtrl-s,4Wheels,3Coils}	0.50	**1.00**	0.86
a_1	{RwCtrl-3,McsCtrl-s,3Wheels,3Coils}	0.67	0.70	**1.00**
a_2	{RwCtrl-0,McsCtrl-s,0Wheels,3Coils}	**1.00**	0.50	0.50

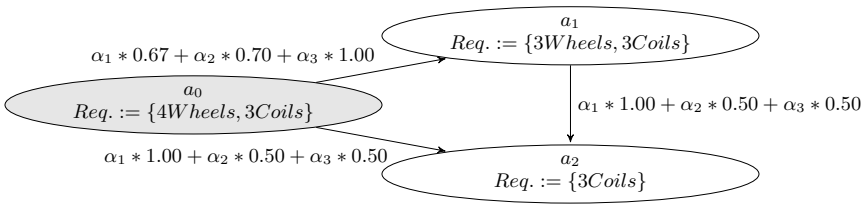

Figure 2: ARG for reduced ACS

4. TOOL SUPPORT

For analyses at design time and simulation of reconfigurations at run time, we integrated several modelling and analysis tools in a prototypical tool chain as depicted in Fig. 3. In the design phase, we apply PerOpteryx[2] to generate candidates out of a set of *DoF*'s and a base architecture, modelled in Palladio Bench[3]. For each generated candidate, energy awareness, agility, and CPU load economy ratings are determined by an numeric approximation. Using genetic selection and mutation, the set of Pareto-optimal candidates is evolved and persisted as related architectures in the *ARG* data structure. The reconfiguration process is supported by our tool AReva2[4]. Based on the current mission profile, the predicted quality properties from design time are ordered hierarchically with weightings This information is used to set the values for α_i for the transition labels in the *ARG* and calculate priorities. The selection algorithm, described in Sect. 3, is still under development as an extension of AReva2 for extensively evaluating the *ARG* concept at simulated run time.

5. CONCLUSIONS AND DISCUSSION

As technique for maintaining fault-tolerant space systems, our approach supports decision making for reconfigurations triggered by hardware faults. By additional efforts for precalculations during design time, reconfiguration decisions at run time are automated to significantly reduce human resources and communication efforts. Additionally, our state-based decision model, the *ARG*, serves as graphical representation of feasible design options to promote transparency of the process. For extensive evaluation in application, our concept needs enhancements in efficiency and tool support.

At present, the state changes are controlled solely by weightings of quality properties and hardware availability. It is necessary to encode business decisions as additional constraints in the *ARG* to prioritise preferred architecture candidates by domain experts. E.g., this could be realised by manually removing transitions. The resulting reduced degree of connectivity would raise efficiency of queries on the *ARG*, but might also interfere automated construction of the graph.

In space domain usage profiles are characterised by mission goals. Thus, the weightings of the quality properties could also been set at design time. Thus, we would be able to pre-define a tailored *ARG* for each mission profile. This avoids parametrisation of the weightings at run time, resulting in a increased performance at run time. However, the flexibility for unexpected or long-term mission goals is lost.

Our approach observes required hardware resources currently in active use. Of course, also non-active resources might fail, resulting in invalidity of each alternate architecture, that require such resources. If a reconfiguration is triggered by a faulty active resource, our approach potentially prefer one of these invalid architectures. Possible solutions for that might be (1) a dynamically *ARG*, purged for invalid architectures upon every fault occurrence, (2) satisfaction checks before enabling transitions, or (3) to allow transient yet invalid architectures and multiple state changes during a reconfiguration. At the moment (3) appears promising to us, but also might lead to termination problems of the query in the *ARG*.

Apart of complete tool-support, we plan simulations and empirical studies with experts from our industrial partner in future work. For that, we extend our tool with a mechanism for fault injection and multiple mission profiles to inspect impacts of changes in the available hardware platform and the usage of the system. As basis for further validation, we improve our process to cope with large input sizes, which would obstruct a complete iteration of all possible configurations by means of search-based techniques. Due to the varying hardware resources, multiple sets of Pareto optimal architecture candidates result from the mass generation. This enables the exploration of multiple Pareto frontiers and their relations during the analysis process. By inspecting distances between these frontiers, we try to identify critical hardware resources leading to large gaps between frontiers. In the long term, we plan to reduce interconnections of the ARG by removing transitions between architecture candidates of far-off frontiers to prevent massive losses of qualities.

6. ACKNOWLEDGEMENTS

This work was partially supported by the DFG (German Research Foundation) under the Priority Programme SPP1593: Design For Future — Managed Software Evolution. The authors would like to thank Sven von Höveling for programming support.

[2]https://sdqweb.ipd.kit.edu/wiki/PerOpteryx
[3]http://www.palladio-simulator.com
[4]Alpha release: https://github.com/lmaertin/areva2

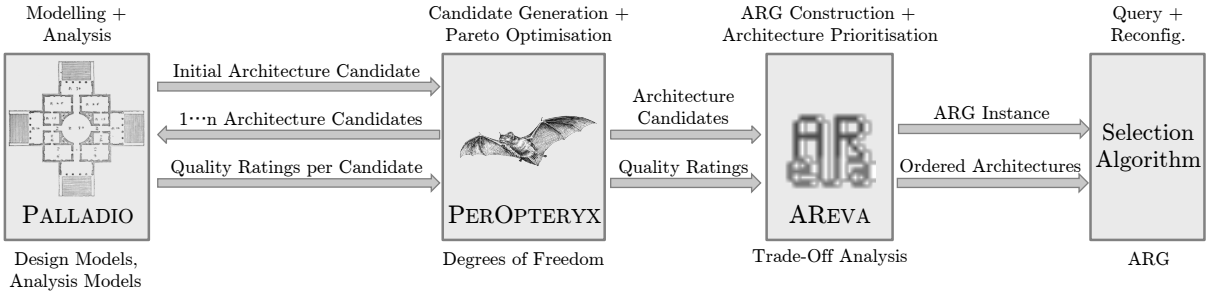

Figure 3: Tool Chain

7. REFERENCES

[1] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya. Software architecture optimization methods: A systematic literature review. *IEEE Trans. on Software Engineering*, 39(5):658–683, 2013.

[2] J. M. Barnes, A. Pandey, and D. Garlan. Automated planning for software architecture evolution. In *28th Int. Conf. on Automated Software Eng.*, pages 213–223, 2013.

[3] G. Cieslewski, A. Jacobs, C. Conger, and A. George. Advanced space computing with system-level fault tolerance (invited talk). In *1st WS on Fault-Tolerant Spaceborne Computing – Employing New Technologies*, 2008.

[4] B. Florentz and M. Huhn. Embedded systems architecture: Evaluation and analysis. In C. Hofmeister, I. Crnkovic, and R. Reussner, editors, *Quality of Software Architectures*, volume 4214 of *LNCS*, pages 145–162. 2006.

[5] S. Föckersperger, K. Lattner, C. Kaiser, S. Eckert, S. Ritzmann, R. Axmann, and M. Turk. The on-orbit verification mission tet-1: Project status of the small satellite mission and outlook for a one year mission operation phase. In *2010 Int. Conf. on Space Optics*, pages 4–8, 2010.

[6] S. Frey, F. Fittkau, and W. Hasselbring. Search-based genetic optimization for deployment and reconfiguration of software in the cloud. In *35th Int. Conf. on Software Engineering*, pages 512–521, 2013.

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

[8] L. Grunske, P. Lindsay, E. Bondarev, Y. Papadopoulos, and D. Parker. An outline of an architecture-based method for optimizing dependability attributes of software-intensive systems. In R. de Lemos, C. Gacek, and A. Romanovsky, editors, *Architecting Dependable Systems IV*, volume 4615 of *LNCS*, pages 188–209. 2007.

[9] G. Jung, K. Joshi, M. Hiltunen, R. Schlichting, and C. Pu. Generating adaptation policies for multi-tier applications in consolidated server environments. In *5th Int. Conf. on Autonomic Computing*, pages 23–32, 2008.

[10] A. Koziolek and R. Reussner. Towards a generic quality optimisation framework for component-based system models. In *14th Int. ACM Sigsoft Symposium on Component based Software Engineering*, pages 103–108, 2011.

[11] J. Kramer and J. Magee. Self-managed systems: An architectural challenge. In *Future of Software Engineering*, pages 259–268, 2007.

[12] S. Löw, J. Herman, D. Schulze, and C. Raschke. Modes and more – finding the right attitude for TET-1. In *12th Int. Conf. on Space Operations*, 2012.

[13] M. R. Lyu. *Software Fault Tolerance*. 1995.

[14] S. Malek, N. Medvidovic, and M. Mikic-Rakic. An extensible framework for improving a distributed software system's deployment architecture. *IEEE Trans. on Software Engineering*, 38(1):73–100, 2012.

[15] L. Märtin and A. Nicolai. Towards self-reconfiguration of space systems on architectural level based on qualitative ratings. In *35th Int. Aerospace Conf.*, 2014.

[16] L. Märtin, M. Schatalov, M. Hagner, O. Maibaum, and U. Goltz. A methodology for model-based development and automated verification of software for aerospace systems. In *34th Int. Aerospace Conf.*, 2013.

[17] S. Savarino, R. Fitz, L. Fesq, and G. Whitman. Fault management architectures and the challenges of providing software assurance. In *31st Space Symposium, Technical Track*, 2015.