An Approach for Dependability Conformance Analysis from Code to Architecture Description

Rovedy Aparecida Busquim e Silva
Instituto de Aeronáutica e Espaço (IAE), São José dos Campos, SP, Brazil, 12228-904
Instituto Tecnológico de Aeronáutica (ITA), São José dos Campos, SP, Brazil, 12228-900
rovedyrabs@iae.cta.br

Abstract

In the context of an increasing importance of software in aerospace field, the work presented in this paper aims to verify dependability attributes in the software architecture and implementation levels for conformance analysis. To fulfill this goal, the work focuses on the definition of an approach to be incorporated into the software verification process that helps software engineers to identify faults. The steps of the proposed approach include the modeling of software architecture using AADL models and formal verification program using specified tools. The AADL models are built by using features of the AADL Error Model Annex and AADL properties. A case study to show the application of the proposed approach is described.

1. Introduction

Critical Real-Time Systems are systems whose failure can result in injuries, environmental damage, or even loss of lives. Due to the criticality of this type of system, dependability requirements should be incorporated in its development. However, dependable software development remains a challenge.

According to ESA [7], the dependability assurance shall be implemented by means of a systematic process for specifying requirements for dependability and to demonstrate that these requirements are achieved. A software verification process evaluates whether dependability requirements are achieved through software fault handling methods and techniques (fault prevention, fault tolerance, and fault removal). The use of a software verification and validation process can help find faults between the phases of software development life cycle, aiming to determine whether the software requirements are implemented correctly and completely and are traceable to system requirements.

One of software verification activities is the conformance analysis between the software architecture and source code. The requirements, specification, and various levels of the design are all descriptions of the same system, and thus should be functionally equivalent [14]. If each one of these descriptions is prepared in a suitable form, it is possible to prove this equivalence, thereby greatly increasing the reliability in the development process. When a system is defined informally, usually the process of verification relies heavily on the skill of the development team and involves activities such as documentation review, source code inspection, and checklists. The perception of equivalence of functionalities between the phases may not be so obvious. The use of formal methods can be used in ways that allow functional comparison.

In a software verification process, individually and formally verified software architecture and source code must comply with each other. However, studies frequently show that there is a degradation in the knowledge related to software architecture during the development process [9]. Shaw and Clements explain that the lack of conformance dooms architecture to be irrelevant as the code sets out on its own independent trajectory [13]. A review about the same topic, explains that the best architecture is worthless if the code does not follow it [3]. This is a risk during initial development; in many cases the risk becomes a certainty in post-deployment maintenance. Tools to analyze code for architecture conformance are still inadequate and rely on humans making suggestions. In this context, the proposed approach is to carry out a formal conformance analysis in the software architecture and source code considering the dependability requirements. The results of analysis must identify inconsistencies that can indicate a fault and thus, contribute to the fault removal technique. The fault removal is beyond the scope of this work. This idea has been considered in the future works.

The remainder of the paper is organized as follows. Section 2 presents the background related to dependability, description architectural language, and formal methods. Section 3 is an overview of our stepwise approach for depen-

dability conformance analysis. Section 4 presents the case study and Section 5 concludes the paper.

2. Background

2.1. Dependability

Dependability of a computing system is the ability to deliver service that can justifiably be trusted [12]. This property integrates the following basic attributes: reliability, availability, safety, confidentiality, integrity, and maintainability. The development of a dependable computing system calls for the combined utilization of a set of techniques related to faults (prevention, tolerance, removal, forecasting).

Within the techniques, our work aims to contribute with software fault removal methods and techniques. Removing software faults directly improves dependability because they are no longer potential causes of failure. This technique is related to the software verification process. Software fault removal methods and techniques are those that are used to verify that the undesired behavior of the software product does not have a severe effect on the system's overall behavior, and these should be assessed for every software development activity and then removed or controlled to reduce their effect. According to [5], the software fault removal is composed of the following steps: fault identification, fault detection, fault isolation, and fault removal. The faults considered in this work are related with reliability issues such as timeliness.

2.2. AADL

The Architecture Description Languages (ADLs) were created to help model software architecture and hardware, each one representing specified features. The use of one architecture description language supplies an important foundation to the verification, since that describes how the system must behavior in the high level viewpoint, including software and hardware.

One of ADLs which we have focused our study on is Architecture Analysis & Design Language (AADL), whose first standard was available in 2004 [8]. The AADL standard provides modeling concepts for description and analysis of system architecture in terms of software components, hardware components, components of computing platform, and their interactions. AADL is an evolution of META-H language and for this reason, it was specially created for specifying and analyzing embedded real-time systems.

AADL has two important features that can be inserted in the AADL specification: set of properties and annex libraries. Both allow the architecture designer to extend the language to customize an AADL specification to meet specific requirements of domain. For example, the error model annex can be used to define reliability models and properties that facilitate the tree fault analysis of architecture. The features are used to incorporate new analysis in the architectural design.

2.3. Formal methods

Accidents caused by errors in software products are a reality. Despite of the importance of software testing, it is not the solution to achieve the forecasting in real-time systems. According to Butazzo, the reason is that in real-time applications, the flow of programs depends on input date and environmental conditions that during the test phase can not be completely replicated [2]. The test phase can provide a partial verification related to a subset of input date.

Formal methods are complementary to traditional methods and increasing their application is recommended. The recommendation of the use of formal methods can be found in important standards such as ECSS-E-ST-40C [6], specifically used in the aerospace area. Formal methods include a set of elements: formal specification languages, model checkers, and theorem provers. A formal specification language depicts the features of a system through a accurate vocabulary, a syntax and a set of semantic. Model checking is an automated formal technique based on the exploitation of states that can be used to obtain evidence that system security is not busted [1]. Theorem proof is a tool that allows finding proof of a system from a set of axioms [4].

3. Proposed Approach

The dependability conformance analysis, a verification activity, is the main focus of this work. The approach verifies whether one dependability requirement implemented in the source code satisfies or is correct with the same dependability requirement represented in the software architecture. A source code SRC satisfies an architecture ARQ if and only if specified dependability requirements of code R_{SRC} satisfies dependability requirements of architecture R_{ARQ}:

$$SRC \models ARQ \text{ iff } R_{SRC} \models R_{ARQ}$$

The dependability requirements are verified in both phases and the goal is fault identification. The main contribution of this proposal is to help the dependability fault removal technique. Figure 1 shows the idea of the proposed work.

The scope of this work considers a source code already concluded and an architecture that has not been described with an architecture design language. The application domain is embedded real-time systems. The solution uses an architecture description language, model checking, and theorem prover for running conformance analysis. The work

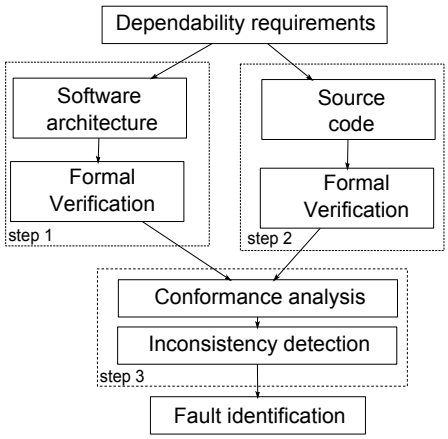

Figura 1. Proposed approach.

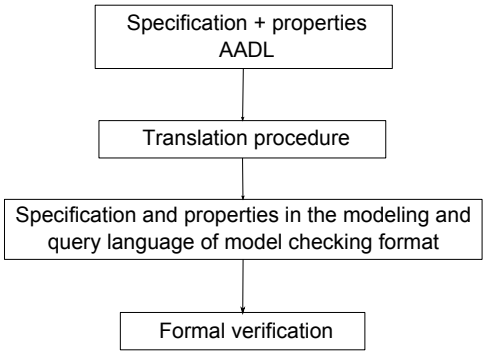

Figura 2. Formal verification of architecture.

starts with activities to identify potential hazards and safety-critical functions based on software documentation available. The next step is an analysis of the safety-critical functions identified and the definition of AADL models which will be created and verified. After that, the same safety-critical functions identified are then traced into source code functions and later formal verification. Finally, the results of verification should be analyzed and classified in terms of conformance criteria. This work can be divided in three main steps.

The first step consists of elaborating an architectural description and verification. The methodology for architectural description follows the rules of standard IEEE 1471-2000 [11]. The standard describes recognized tendencies to build description architectures and recommends that description architecture be organized in architecture viewpoints. The formal architectural verification will be carried out by model checking. The model checking selected represents the behavior of the software by timed automata. In this task, it is required to define how to expand the AADL models in the way that will be possible to extract timed automata models from AADL models automatically. The basic idea is to develop a procedure that allows an architecture specification to translate to model checking language, as pointed out in Figure 2. However, some works are being studied in order to verify the feasibility of their use in this step. Also the possibility of using theorem prover will be studied.

The second step is defining one procedure to execute a formal verification implementation. Formal methods can be applied directly to source code, which is called in the literature: formal verification of software or program. The formal verification of programs helps to verify the correct behavior of the programs. The technique selected for this step is the source code annotation process. It is based on an annotation specification language to produce first-order logic proof obligation called Verification Conditions that can be manipulated by a proof tool. The elementary idea is that if all verification conditions can be proved valid then the program is assured to be correct. Figure 3 shows one diagram with the annotations source. The comments written in natural language in the source code can be formalized as annotations and the requirements also are source for insertion of annotations in the code. Checking dependability requirements in the source code is a challenging job.

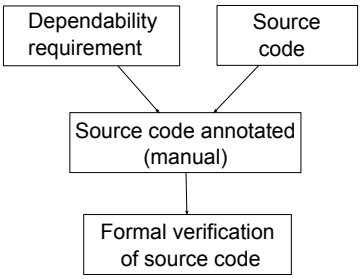

Figura 3. Annotation sources to formal verification.

Finally, the last step consists in defining a procedure for dependability conformance analysis. The formal dependability conformance analysis will be defined according to the architecture verification and source code used. The basic idea is establishing an equivalence relation between dependability properties proved in the architecture and dependability properties proved in source code. Additionally, the analysis considers three types of conformance: convergences, divergences, and absences. Convergence is a relation between two components that are allowed or were implemented as intended. Divergence is a relation between two components that are not allowed or were not implemented as intended. Absence is a relation between two components

that were intended but not implemented. The result of this final step supplies evidence of possible fault in the source code, since the software architecture was verified and used as reference model.

4. Case study

The scenario is based on a spacecraft capable of launching satellites weighing maximum 350 kg at altitudes up to 1000 km. The embedded software is a critical in real-time, whose main assignment is to perform the functions of initiation, verification, and control of the vehicle, from minutes before its launch up to putting the satellite into orbit. Its function is to navigate and guide the vehicle, control its actuators, manage the sequence of events, address and send the telemetry data, and carrying out the pre-launcher. The case study is based on the Brazilian Satellite Launcher Vehicle [10]. In the software VLS project, the dependability is implemented by fault prevention, fault tolerance, and fault removal. The fault prevention is provided through the assurance product plan, which focuses in the following attributes: functionality, reliability, maintainability, and efficiency. The fault removal is obtained through simulation and tests. The fault tolerance has an information redundancy used in communication with the ground system. Currently, the verification process of the software is a typical process without formality.

The application of methodology in the case study starts with the study of Software System Specification and Software Requirements Specification documentation in order to analyzing the safety-critical functions and to identify dependability requirements. According to step 1, in the section 3, the AADL models will be elaborated based on selected requirements. The actual architecture is depicted in task diagrams in a static view without any associated verification and AADL models will allow the verification and analysis. Next, the step 2 is applied in the source code implemented in programming language C. The annotations will be inserted in the code in a manual way. Then, the step 3 is applied and the conformance analysis will report results that might reveal inconsistencies. From those results is possible to identify faults in the VLS software.

5. Conclusions

This paper presented an approach for dependability conformance analysis in the software architectural and implementation levels. The aim of the approach is to contribute with the software removal fault technique through the fault identification, thereby showing if each of the dependability requirements is met individually and, if possible, indicating that they are met collectively. The main characteristic of this approach is the use of formal methods. The application of formal methods in the conformance analysis provides greater accuracy than others techniques that depend on the skills of the development team.

The dependability plays an important role in the safety-critical software development. The fulfillment of dependability software requirements should increase the quality of the final product and decrease the probability of lost mission in the aerospace field. Further, the resulting approach can be adapted to a process of correct-by-construction software development into the aerospace field.

Acknowledgments
I would like to thank my research advisors, Jose M. Parente de Oliveira and Jorge Sousa Pinto, for their support and assistance.

Referências

[1] G. Behrmann, A. David, and K. G. Larsen. A tutorial on Uppaal. In *School on Formal Methods for the Design of Computer, Communication, and Software Systems*, 2004.

[2] G. C. Buttazzo. *Hard Real-Time Computing Systems - Predictable Scheduling Algorithms and Applications*. Springer, Italy, 2005.

[3] P. Clements and M. Shaw. The golden age of software architecture revisited. *IEEE SOFTWARE*, 26(4):70–72, July 2009.

[4] CoqTeam. *The Coq Proof Assistant Reference Manual*, 2010.

[5] ECSS. ECSS-Q-80-03 Space product assurance - methods and techniques to support the assessment of software dependability and safety, March 2006.

[6] ECSS. ECSS-E-ST-40C Space Engineering - Software, March 2009.

[7] ECSS. ECSS-Q-ST-30C Space Product Assurance - Dependability, March 2009.

[8] P. Feiler, D. Gluch, and J. Hudak. The architecture analysis & design language (AADL): An introduction. Technical Report CMU/SEI-2006-TN-011, Carnegie Mellon University, Software Engineering Institute, February 2006.

[9] M. Feilkas, D. Ratiu, and E. Jurgens. The loss architectural knowledge during system evolution: An industrial case study. In *IEEE 17th International Conference on Program Comprehension*, pages 188–197, Vancouver, BC, May 2009.

[10] IAE. Instituto de Aeronautica e Espaco - Projeto VLS, 2011.

[11] IEEE. IEEE Recommended Practice for Architectural Description of Software-Intensive Systems - IEEE Std 1471, 2000.

[12] J. Laprie. *Dependability: basic concepts and terminology*, volume 5 of *Dependable Computing and Fault-tolerant Systems*. Springer-Verlag, Vienna, 1992.

[13] M. Shaw and P. Clements. The golden age of software architecture. *IEEE SOFTWARE*, 23(2):31–39, March 2006.

[14] N. Storey. *Safety-Critical Computer Systems*. Addison-Wesley, 1996.