Performance Modelling for Avionics Systems

Visar Januzaj[1], Ralf Mauersberger[2], and Florian Biechele[3]

[1] Technische Universität Darmstadt, Fachbereich Informatik,
FG Formal Methods in Systems Engineering - FORSYTE
Hochschulstr. 10, 64289 Darmstadt, Germany
januzaj@forsyte.de
[2] EADS Innovation Works, 85521 Ottobrunn, Germany
ralf.mauersberger@eads.net
[3] EADS Defence & Security, 85077 Manching, Germany
florian.biechele@eads.com

Abstract. The new paradigm of Integrated Modular Avionics (IMA) [1] necessitates the analysis and validation of non-functional requirements for IMA systems. This includes the analysis of their performability. In this paper we present an initial approach of a performance modelling framework, based on the SAE standardised modelling and analysis language AADL [2, 3], to integrate performance analysis in the toolchain of this practical context. The proposed framework is a hybrid of static and dynamic systems analysis and includes aspects of performance evaluation.

1 Introduction

Due to the increasing demands and complexity of avionics systems emerges the need of a methodology to cope with system development issues, such as spatial demands, power resources onboard the aircraft and high maintenance costs. To overcome these issues the *Integrated Modular Avionics* (IMA) [1] defines avionics system as an integrated system with multiple functions hosted on a cabinet of processors. In consequence IMA systems are comprised as distributed embedded systems where software components interact with each other and the hosting physical architecture by a set of standardised interfaces, similar in function to operating system calls in general-purpose computer systems. This abstraction enables dynamically configurable systems where software components are deployed on host processors according to criteria such as criticality, system load and run-time faults. The deployment/binding between hardware and software components is defined in so called system configuration tables or *blueprints*. System blueprints including an initial system configuration as well as reconfigurations for identified system failure types are loaded at runtime and conditions are monitored by designated system components. Despite its advantages, IMA demands sophisticated analysis of avionics systems. Thus a blueprint generation requires foregoing system analysis, e.g. schedulability analysis, to assure a correct operation of avionics software so that deadlines are met and memory resources are sufficient. To achieve a successful application of the IMA concept in the

R. Moreno-Díaz et al. (Eds.): EUROCAST 2009, LNCS 5717, pp. 833–840, 2009.
© Springer-Verlag Berlin Heidelberg 2009

avionics domain, supporting techniques are essential for blueprints generation and their integration into the system development process.

We present a performance modelling framework which facilitates the automatic generation of blueprints for IMA systems. Our framework not only proposes a method for execution time prediction in modern avionics systems, more importantly, it shows how those predictions can be integrated and used for the determination of stable and feasible system configurations (blueprints). In order to support the IMA technology we use the SAE standardised *Architecture & Analysis Description Language* (AADL) [2,3] which offers an abstract but precise description of the components of a system architecture and facilitates the application of performance analyses [4].

2 The Performance Modelling Framework

Figure 1 describes our modelling and analysis framework. Our approach is divided into two major stages. Within the first stage, *component analysis*, we separately analyse the hardware and software components. For each analysis a corresponding profile is generated. The analysis is performed separately for the following reasons:

- once the hardware is analysed, we do not need to repeat its analysis for possible software changes, or if new software is available
- we do not need to run tests for each software on each available hardware

This approach can also be applied for execution time prediction of software being still under development. In the second stage, *modelling and system analysis*, we model, according to the collected profile data, and analyse the generated system. The purpose of the latter analysis is to determine possible configurations in order to finally generate feasible and stable system blueprints.

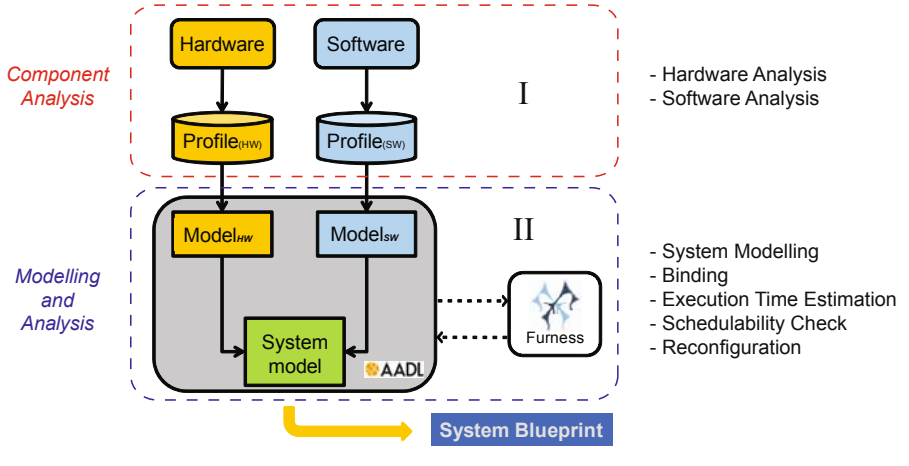

Fig. 1. The performance modelling framework

2.1 Component Analysis

This analysis concerns the execution time prediction and aims at generating corresponding *profiles* for each component type. These profiles hold measurement information about the estimated execution time for each instruction/operation (on a hardware) and their occurrence during an execution run (of a software), respectively.

Hardware Analysis. For the sake of simplicity the hardware is analysed on a test-based manner, i. e. carrying out a number of benchmarks. For each benchmark we collect the execution time and perform an operation counting (cf. Software Analysis), i. e. we count during the execution of a benchmark the occurrence of each operation type, e. g. `ADD`, `MUL` and `ALLOC`. The benchmarks are C programs that try to reflect as good as possible the behaviour and nature of the software used in avionics systems.

We represent each benchmark as a *linear inequation* built by the number of the operations occurred and the observed execution time, e. g. for a benchmark b we get : $200 \cdot \texttt{ADD} + 300 \cdot \texttt{MUL} + 50 \cdot \texttt{DIV} + 9 \cdot \texttt{ALLOC} + \ \ldots \ \leq \ 33,9 \ ms$.

All resulting linear inequations are put together creating a linear inequation system. Mathematically we represent such a linear inequation system as follows : $O \ \underline{v} \ \leq \ \underline{t}$, where O is a $n \times m$ matrix built out of the number of the occurrences for each operation (called in the following *variable*) in each benchmark. The number of rows n corresponds to the number of benchmarks and the number of columns m to the number of variables. The $1 \times m$ vector v represents the variable vector. t represents the $1 \times n$ vector of execution times for each benchmark.

In order to get more accurate execution time estimations (or even to get a solution at all) we add in both sides of our linear inequations a variation vector δ, representing potential differences on execution times that a benchmark might have (during different runs). Thus we get the following representation : $\underline{t} - \underline{\delta} \leq O \ \underline{v} \leq \underline{t} + \underline{\delta}$.

To avoid arbitrary solutions we need to add further constraints to our linear inequation system. These constraints are based on processor data sheets and represent a weighting for each variable. The varying weights evolve from the fact that, e. g. a floating-point division takes much longer than a simple address load. The constraints play an important role since they not only reflect the processor's architecture but also lead to more qualitative and better solutions.

The estimated execution time of the variables is calculated by minimising over the sum of all variations δ (subject to the linear inequation system and the corresponding constraints): $min \ \sum_{i=1}^{n} \ \delta_i$, where n is the number of the benchmarks. The minimisation problem is solved using linear programming solvers. The solutions represent the estimated execution time for each variable and are stored in the *hardware profile*.

Software Analysis. In this step the source code structure is extracted and a rather symbolic execution time estimation is calculated. To achieve this we perform *code instrumentation* using the LLVM compiler framework [5], i. e. the source code is translated into a LLVM bytecode intermediate representation

```
property set New_Properties is
     Variables : list of aadlstring applies to (thread, processor);
     ExecTime  : list of aadlreal applies to (processor);
     Occurrence: list of aadlinteger applies to (thread);
end New_Properties;
```

Fig. 2. New properties

```
processor CPU
   properties
      New_Properties::Variables => ("MUL", "ADD", "LOAD");
      New_Properties::ExecTime  => (0.0035, 0.0013, 0.0007);
end CPU;

thread T
   properties
      New_Properties::Variables  => ("MUL", "ADD", "LOAD");
      New_Properties::Occurrence => (309, 53, 137);
end T;
```

Fig. 3. Applying new properties

(IR). The language- and platform-independent IR is represented by a reduced set of RISC-like instructions. Furthermore, by providing a modified `gcc` compiler the LLVM framework facilitates the analysis of optimised code, thus increasing the software analysis accuracy. To keep track of each instruction type execution we add to the IR corresponding *counters*. These counters are incremented each time an instruction is executed. A program run is symbolically represented as an equation of the following form:

$$3579 \cdot \texttt{ADD} + 759 \cdot \texttt{MUL} + 53 \cdot \texttt{LOAD} + \ \ldots \ = \ exT, \tag{1}$$

where exT is the execution time which at this point is unknown and will be estimated later on. The code structure (at the function level) and the symbolic execution time estimation form together the *software profile*.

2.2 Modelling and System Analysis

We model the component profiles and the resulting system using AADL. A detailed overview of AADL can be found in [2, 3]. The AADL components are divided into three categories: (i) *Hardware components* : `processor`, `memory`, `bus`, `device`, (ii) *Software components* : `process`, `thread`, `subprogram`, `data`, `thread group` and (iii) *System components* : `system` (representing the composition of all components).

The hardware profile is mapped to hardware components and the software profile is mapped to software components, respectively. However, most of the profiles data cannot be mapped to AADL components. Using own property definitions one can add supplementary attributes to each AADL component.

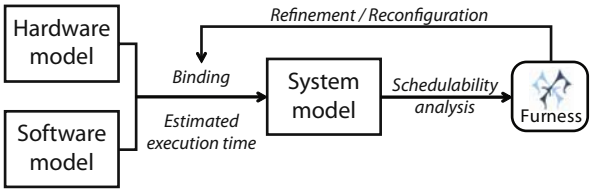

Fig. 4. System analysis

The newly defined properties (cf. Fig. 2) for both hardware (`processor`) and software components (`thread`) represent the variables, their occurrence and their estimated execution time. The application of the new properties is illustrated in Fig. 3. Using the data stored in the property entries we determine the estimated execution time for each thread depending on a particular binding.

System Analysis. In this section we introduce the final step of our approach, which involves the following actions:

- calculate a set of possible bindings for hardware and software components
- determine the execution time estimation for each chosen binding
- find feasible system configurations by performing schedulability analysis

These actions are repeated until a feasible and sufficient schedule is found. Hence, the automated system deployment process is completed. Figure 4 depicts the three steps introduced above.

In order to include possible hardware failures we perform *reconfigurations*, i. e. for each found feasible system configuration we exclude a hardware component and repeat the actions above. We keep excluding hardware components until no binding can be found. All computed reconfigurations are stored and integrated into the AADL system model as *failure modes*. We will however not discuss in this work the reconfiguration process.

Binding. This step is one of the important ones, since the quality of determined bindings influences plainly the quality of the system deployment and the generated blueprint. Possible bindings are determined by cross combining the available hardware and software in consideration of various important criteria, e. g. taking into account the speed of the processor in order to find the more appropriate hardware to handle the specified software and by clustering processes that have dense intercommunication to achieve an efficient deployment.

Execution Time Estimation. The estimated execution time of a software bound to a particular hardware is calculated by simply building equations of the form shown in equation (1) and evaluating the generated equation. Each chosen binding is specified by the `Allowed_Processor_Binding` property of the `system` component. Such an entry could be for instance: `Allowed_Processor_Binding => reference c1 applies to p1.t1;`, indicating that thread `T` of the process

P is bound to the processor CPU, i. e. T will run on CPU. Note that t1, p1 and c1 are references to the implementations of T, P and CPU, respectively.

To calculate the estimated execution time of T running on CPU (cf. Fig. 3), as mentioned above, we create out of the properties (Variables and Occurrence) of thread T the following equation:

$$309 \cdot \texttt{MUL} + 53 \cdot \texttt{ADD} + 137 \cdot \texttt{LOAD} = \ exT$$

In order to determine exT we replace each variable (ADD, MUL and LOAD) with the values found in the execution time properties of processor CPU, namely in ExecTime. Thus we get the following:

$$309 \cdot 0.0035 + 53 \cdot 0.0013 + 137 \cdot 0.0007 = \ exT$$

The value of exT (= 1.2463) is then added in the specified timing property of thread T as follows: `Compute_Execution_Time => 1.2463 Ms .. 1.2463 Ms;` , indicating that T needs 1.2463 milliseconds for its execution.

Schedulability Analysis. Having defined the binding and calculated the corresponding estimated execution time we perform the schedulability analysis. The schedulability analysis checks the compliance of thread scheduling constraints using the Furness toolset [6]. Furness applies the ACSR [7] process algebra to determine schedulability. If a model is not schedulable Furness displays a failing trace (a timed system trace). Otherwise, it shows the analysis of best-case and worst-case time responses [6], thus facilitating the validation of various system configurations and the establishment of a thoroughly feasible system deployment.

Depending on these schedulability analysis results one can decide if it is required to repeat the previous actions, i. e. to choose a different binding, calculate the corresponding estimated execution time and run the schedulability analysis again. The received results are evaluated by the system developer. Nonetheless, one can automatically run the schedulability analysis for a set of chosen bindings and identify a feasible binding.

3 Related Work

Execution time predictions support in particular the development of embedded systems, enabling various system analysis regarding specified performance, such as schedulability analysis to examine if task deadlines are met [8]. A detailed overview of techniques and tools that deal with the execution time analysis is given in [9] and a comparison of different timing analysis methods and approaches can be found in [10, 11]. A modular architecture for a timing analysis approach combining different analysis methods is introduced in [12], making possible the exchange of results and the comparison of such methods. In [13] an approach is presented for computing tight bounds for WCET using *static* analysis. Wang et al. [14] present an approach based on *source code instrumentation* concerning binary translation and microarchitecture-related issues. A similar approach is presented in [15]. The authors investigate different methods and propose a

statistical analysis-based approach which improves estimates of execution time. Both latter approaches have similar aspects to our component analysis approach. However, the main distinction is that we keep the analysis of hardware and software strictly separated. As a result of research on timing analysis a number of tools evolved, such as aiT [16], SWEET [17], Chronos [18] and OTAWA [19]. The importance, challenges and the evolution of the embedded system design is discussed in [20]. Different methodologies and development environments, such as SCADE [21] and Matlab/Simulink [22], have been applied on the design and development of embedded systems. We use in our framework the SAE standardised language AADL [2,3]. Powerful tools such as Furness [6] and Cheddar [23] allow the schedulability analysis of systems modelled in AADL.

4 Conclusion

We introduced in this work an approach applicable to the performance modelling of modern avionics systems. Following the IMA ideology, a separate analysis of hardware and software ensures a decoupling of any possible analysis dependencies/impact in case of an update, be it a hardware or a software update. Furthermore, the application of AADL facilitates a direct support of the IMA concept. The execution time estimation method which we present in this paper is well suited for observing differences between various hardware. However, the quality of these observations is very closely bound to the quality of the benchmarks used and the constraints built. Due to the clearly defined interface between component analysis and the system modelling and analysis, one can smoothly integrate any other method for execution time prediction. We have tested our method with a set of benchmarks and the results look promising with regard to future activities, achieving an accuracy up to 90%. Since the work is still in progress, the current results have rather an experimental character. Nevertheless this activity is extensively fed by an industrial context.

The distinctiveness of our approach is based on the fact that it not only tackles important non-functional properties in modern avionics system design and development, such as execution time estimation, resource awareness composition and reusability of function units, but it also shows how those particular observations can be integrated and used for the deployment of such systems.

Acknowledgements

We would like to thank Andreas Holzer and Stefan Kugele for their fruitful comments in finalising this paper.

References

1. Garside, R., Joe Pighetti Jr., F.: Integrating modular avionics: A new role emerges. In: IEEE/AIAA 26th Conference on Digital Avionics Systems, DASC 2007 (2007)
2. Society of Automotive Engineers: SAE Standards: Architecture Analysis & Design Language (AADL) - AS5506 (November 2004) and AS5506/1 (June 2006)

3. Feiler, P.H., Gluch, D.P., Hudak, J.J.: The architecture analysis & design language (aadl): An introduction. Technical report, SEI, Carnegie Mellon (2006)
4. Feiler, P.H.: Modeling of system families. Technical report, Software Engineering Institute, Carnegie Mellon (2007)
5. The LLVM Framework, http://www.llvm.org
6. Furness Toolset v.1.6., User Guide, http://www.furnesstoolset.com/files/Furness%20Toolset%20User%20Guide.pdf
7. Brémond-Grégoire, P., Lee, I., Gerber, R.: ACSR: An Algebra of Communicating Shared Resources with Dense Time and Priorities. In: Best, E. (ed.) CONCUR 1993. LNCS, vol. 715, pp. 417–431. Springer, Heidelberg (1993)
8. Engblom, J., Ermedahl, A., Sjödin, M., Gustafsson, J., Hansson, H.: Worst-case execution-time analysis for embedded real-time sys. STTT 4(4), 437–455 (2003)
9. Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., Muller, F., Puaut, I., Puschner, P., Staschulat, J., Stenström, P.: The worst-case execution-time proble overview of methods and survey of tools. ACM Transactions on Embedded Computing Systems (TECS) 7(3), 1–53 (2008)
10. Engblom, J., Ermedahl, A., Stappert, F.: Comparing Different Worst-Case Execution Time Analysis Methods. In: The Work-in-Progress session of the 21st IEEE Real-Time Systems Symposium (RTSS 2000) (November 2000)
11. Kirner, R., Knoop, J., Prantl, A., Schordan, M., Wenzel, I.: Wcet analysis: The annotation language challenge. In: WCET (2007)
12. Ermedahl, A.: A Modular Tool Architecture for Worst-Case Execution Time Analysis. PhD thesis, Uppsala University: Acta Universitatis Upsaliensis (June 2003)
13. Ferdinand, C., Heckmann, R., Langenbach, M., Martin, F., Schmidt, M., Theiling, H., Thesing, S., Wilhelm, R.: Reliable and precise WCET determination for a real-life processor. In: Henzinger, T.A., Kirsch, C.M. (eds.) EMSOFT 2001. LNCS, vol. 2211, pp. 469–485. Springer, Heidelberg (2001)
14. Wang, Z., Sanchez, A., Herkersdorf, A., Stechele, W.: Fast and accurate software performance estimation during high-level embedded system design. In: edaworkshop, Hannover, Germany (May 2008)
15. Giusto, P., Martin, G., Harcourt, E.: Reliable estimation of execution time of embedded software. In: DATE 2001: Proceedings of the conference on Design, automation and test in Europe, pp. 580–589. IEEE Press, Los Alamitos (2001)
16. AbsInt, http://www.absint.com/ait/
17. SWEET, http://www.mrtc.mdh.se/projects/wcet/sweet.html
18. Chronos, http://www.comp.nus.edu.sg/~rpembed/chronos/
19. Otawa, http://www.otawa.fr/
20. Henzinger, T.A., Sifakis, J.: The discipline of embedded systems design. IEEE Computer 40(10), 32–40 (2007)
21. SCADE, http://www.esterel-technologies.com/products/scade-suite/
22. The MathWorks Inc.: Using Simulink (2000)
23. Singhoff, F., Legrand, J., Nana, L., Marcé, L.: Cheddar: A flexible real time scheduling framework. In: Proceedings of the 2004 Annual ACM SIGAda International Conference on Ada, Atlanta, GA, USA (2004)