A Unified Benchmarking Process for Components in Automotive Embedded Systems Software

Wolfgang Forster[1] Christof Kutschera[2] Dietmar Schreiner[1,2] Karl M. Göschka[1]

[1]Vienna University of Technology
Institute of Information Systems, Distributed Systems Group
Argentinierstrasse 8/184-1, A-1040 Vienna, Austria
{w.forster,d.schreiner,k.goeschka}@infosys.tuwien.ac.at

[2] University of Applied Sciences Technikum Vienna
Department of Embedded Systems
Höchstädtplatz 5, A-1200 Vienna, Austria
{kutschera}@technikum-wien.at

Abstract

During the last years, component based software development has become a well accepted software engineering paradigm within the automotive industry. This fact is not only reflected by upcoming development tools but also by newly arising automotive software standards. In component based software engineering, applications are built by assembling small reusable building blocks, the components. Typically more than one component implementation meets the application developer's requirements, so proper selection of the assembled components becomes a key element of the whole procurement and engineering process. This paper's contribution is twofold: First, a basic set of performance and dependability metrics and measures for automotive components is identified. Second, a unified benchmarking process is proposed, that allows an unambiguous comparison of distinct component implementations of a given component class.

1 Introduction

The importance of embedded systems within the automotive industry has rapidly grown over the last years. State of the art vehicles contain up to 70 electronic control units (ECUs), typically connected by various bus systems [8]. Automotive software has to be developed for those heterogeneous, distributed systems with respect to safety, quality and costs. These requirements are well met by the component based software engineering (CBSE) paradigm.

One of the main achievements of component based software engineering is the reusability and substitutability of equivalent components: A component may be replaced by another one, if both components implement the same interfaces and provide the same functionality. To compare them, a unified process of comparison has to be defined.

This paper's[1] contribution therefore is twofold. First we identify the most common measures and metrics within component based software engineering for automotive system software. As the terms *component*, *interface* and *component class* are often used in literature with different semantics, we use the definitions provided in [15]. Second, we propose a unified benchmarking process. We define a general benchmarking framework that specifies which measures have to be acquired for a general component, and how the acquisition has to take place. Based on the general framework a specialized benchmark can be derived for each component class. Application developers will be able to choose between different vendors of required components. This fact is of great importance for establishing a working component market.

2 Metrics and Measures for Automotive Components

Many software metrics have already been proposed for object oriented [7, 16] and component based systems [4, 2,

[1]This work has been partially funded by the FIT-IT [embedded systems initiative of the Austrian Federal Ministry of Transport, Innovation, and Technology] and managed by Eutema and the Austrian Research Agency FFG within project COMPASS [5] under contract 809444.

14]. Most of them are based on well known source code. This paper introduces relevant measures and metrics for component based system design in the safety-critical automotive domain. In [9] the term measure is defined as directly observable data or countable entities. Further the term metric is defined as result from combining measures. Our approach covers platform dependent performance and dependability attributes and is based on the restriction of a *black box benchmark evaluation*.

2.1 Performance

Performance in the domain of embedded systems basically addresses resource usage and execution duration of a dedicated piece of code on the available platform.

2.1.1 Execution Time

Due to the fact that this paper proposes black box evaluation a *worst case execution time* (WCET) can not be identified without extensive analysis of the component's object code [17]. Nevertheless with a dedicated workload simulating typical use cases of a component it is possible to determine a *worst observed execution time* (WOET) and consequently also an *average observed execution time* (AOET) for each operation of a component. To support a component system integrator in resource estimations an *execution time indicator for average observed execution time* (ETI_{ao}) is introduced. It is defined as the sum of weighted average observed execution times of each operation of the component

$$ETI_{ao} = \sum_{i=1}^{m} \sum_{k=1}^{n} w_{ao_{i,k}} \cdot AOET_k \qquad (1)$$

where *n* and *m* define the number of operations per interface and the number of interfaces of the component. The weights $w_{i,k}$ for operation k of interface i depend on a dedicated application and have to be determined under consideration of (i) the call-frequency of an operation and (ii) the usage of an operation at a critical point in time.

In safety-critical automotive systems the *component initialization duration* (CID) is of major importance. It is defined as the worst observed execution time of the initialization operation.

2.1.2 Memory Consumption

Memory consumption on embedded systems can be classified into (i) *program memory consumption* (PMC) and (ii) *run-time memory consumption* (RMC). The program memory consumption is defined as the number of bytes a component requires to store the corresponding object code in the read only memory (ROM). The run-time memory consumption is defined as the number of bytes a component requires during execution in the working memory (RAM) of an embedded system and can be subclassified into *Static Run-time Memory Consumption* (Static RMC) and *Dynamic Run-time Memory Consumption* (Dynamic RMC). Static RMC consists of (i) memory that is shared between all instances of one specific component (*Common Static Memory Consumption*) and (ii) memory that is accesible by each instance exclusively (*Instance Static Memory Consumption*). The only significant type of dynamic memory allocation in safety-critical automotive systems is the *stack memory usage* (SMU). It is a significant metric for the run-time behavior of a component and is defined as the maximum stack memory usage of all component operations.

2.2 Dependability

Dependability is *the ability of a system to avoid service failures that are more frequent or more severe than acceptable* [1]. It can be described by five characteristics:

1. ***Availability*** is determined by *mean time to failure* (MTTF) and *mean time to repair* (MTTR) [7, 16].
2. ***Reliability***: Possible scenarios are the (i) *detection* and notification of illegal parameters , (ii) *hang-up* in case of no response and (iii) *propagation* if an illegal call is falsely assumed as valid response.
3. ***Safety***.
4. ***Integrity*** is actually not state-of-the-art in the domain of component based automotive embedded systems and therefore subject of future work.
5. ***Maintainability***.

The metrics for maintainability are derived from [2], where maintainability is divided into two characteristics: *changeability* and *testability*. Both characteristics are in turn represented by (i) *customizability ratio*, (ii) *change control capability*, (iii) *start-up self-test* and (iv) *test suite provided*. Additional this paper proposes *structural complexity* as part of maintainability to assess integration of components into final systems. Structural complexity indicates the integration complexity of a component [2]. In accordance to [4] we propose the *component interface complexity* (CIC) as the sum of *method complexities* (MC) of all used interfaces of the component, whereas method complexity is defined by simple and complex arguments of component operations.

Figure 1 gives a compact overview about the proposed metrics and measures. It illustrates a refinement of general terms on the left side of the figure to the underlying measures and their units on the right side. For example the general term performance can be split up into execution time and memory consumption. Furthermore one characteristic of execution time is the *execution time indicator for average execution time* of a component (ETI_{ao}). This indicator

METRICS				MEASURES	UNITS
PERFORMANCE	Execution Time	Component WOET		WOET[1]	Seconds
		ETI_{ao}		AOET[1]	Seconds
		CID		$WOET_{init}$	Seconds
	Memory Consumption	PMC		ROM Usage	Bytes
		RMC	Static	Common RAM Usage	Bytes
				Instance RAM Usage	Bytes
			Dynamic	Stack Memory Usage[1]	Bytes
DEPENDABILITY	Availability	MTTF			Seconds
		MTTR			Seconds
	Reliability	Error Detection Rate		Detected Illegal Calls	{0,1 .. n}
				Illegal Calls	{0,1 .. n}
		Hang-up Rate		Hang-ups	{0,1 .. n}
				Illegal Calls	{0,1 .. n}
		Error Propagation Rate		Propagated Illegal Calls	{0,1 .. n}
				Illegal Calls	{0,1 .. n}
	Safety	Safety Properties[2]			{True,False}
	Maintainabiliy	Changeability	Customizability Rate	Comp. Parameters	{0,1 .. n}
				Comp. Interfaces	{0,1 .. n}
			Change Control Cap.		{True,False}
		Testability		Start-up Self-test	{True,False}
				Test Suite Provided	{True,False}
		CIC	MC*	Simple Arguments	{0,1 .. n}
				Complex Arguments	{0,1 .. n}

[1] .. must be determined for each operation of the component

[2] .. must be defined depending on the component's context

Figure 1. Component Metrics and Measures Overview

depends on the *average observed execution time* (*AOET*) of each component's operation.

3 Benchmarking

When assembling a component based application, the used components have to be carefully selected. To support the process of selection a standardized tool for component comparison has to be defined. This automotive benchmark framework intends to provide necessary definitions to enable the realization of external [12] benchmarks. Based on [12, 11, 13] we developed the following definition for *benchmark*:

A benchmark is a specified set of rules (agreed by the community) for assessing specific features and measures of a system. The specification of a benchmark must include definitions for (i) the considered system in its operating environment, (ii) the benchmark context[2], (iii) the benchmark measures and (iv) everything needed for experimentation.

To evaluate different implementations of a component class, properties of the examined components have to be acquired and compared. In Section 2 we identified a set of measures and metrics that are relevant for automotive components and shall therefore be benchmarked regardless to the particular component functionality on a dedicated platform.

[2]The "benchmark context" consists of specifications for the following issues [10]: (i) Life Cycle Phase, (ii) Benchmark User, (iii) Benchmark Purpose, (iv) Result Scope and (v) Benchmark Performer

3.1 The Approach

The general approach described in the following sections is based (i) on a benchmarking framework developed to enable harmonized development of benchmarks for automotive components and (ii) on benchmark definitions for specific component classes and platforms. To ensure reproducible results the particular benchmark definition must include an accurate and detailed description of the assessment methods and benchmark setups.

3.1.1 Framework and Benchmarks

The proposed benchmark framework standardizes the definition of component benchmarks. Platform independent information on the process of benchmarking is included in the framework definition, whereas platform specific information has to be defined in the benchmark definitions. Component benchmark definitions have to be derived from the benchmark framework, so each benchmark in turn is a specialization of the framework.

Based on [10, 12, 18, 19] we developed an automotive benchmark framework which extends the focus of benchmark definitions beyond dependability benchmarks to comprehensive benchmarks. The benchmark framework addresses merely the assessment of automotive components, providing a standardized way for defining benchmarks for them.

The benchmark concept is based on three major categories significant for benchmark definitions:

Categorization dimensions: Categorization dimensions specify general issues for automotive benchmarks. They provide information on the operating environment, the application area and context information [12]. The benchmark framework introduces specifications for the categorization dimensions as they are considered to be common for benchmarking components in the automotive application domain.

Quantification dimensions: Quantification dimensions deal with metrics and measures of general properties of automotive components. These properties shall be acquired regardless of the component class of the assessed components. Several common measures and metrics – divided into the major categories performance and dependability – have been identified in the course of the benchmark framework definition. Furthermore, component class specific measures and metrics may be specified in the component benchmark definition for each component class, e.g. data throughput, for components that are part of a communication

sub-system. The quantification dimensions are also referred to as *"Measure Dimensions"* [10].

Experimentation dimensions: Experimental dimensions deal with rules and methods for acquiring specific measures. As the acquisition of common measures is platform dependent, it is not feasible to generally specify how to acquire them. Nevertheless the framework supplies rules and methods, e.g. feasible benchmark architectures, for acquiring common measures to support the development of component benchmark definitions. This enables a general procedure for acquiring certain measures throughout different component benchmarks. Therefore reproducibility and the development of benchmark definitions will improve. The actual benchmark setup and measurement approach has to be specified in the according benchmark definition and may differ from the guidelines provided by the benchmark framework, if necessary. Workload and faultload are platform and component dependent and are therefore not specified in the automotive benchmark framework, but have to be specified in each component benchmark definition.

3.2 Benchmark Architecture

To acquire measures of dynamic runtime properties, e.g. the stack usage, the assessed component has to be invoked under real world conditions within a benchmark setup called benchmark architecture. The benchmark architecture includes the assessed component itself – the component under benchmark (CUB) – and anything that is necessary for invoking the component.

3.2.1 "Integrated Component"-Architecture

If a component is assessed within a "real" environment, the benchmark architecture consists of all components involved in the application's execution including the operating system and the hardware platform. The exact benchmark architecture has to be defined accurately for each benchmark in order to provide reproducibility. Figure 2 shows a general example for the definition of such an architecture. The CUB is part of an automotive composition. To enable assessment, all interfaces of the CUB are utilized by wrappers. The wrappers signal each interaction of the CUB with other components to a so called evaluation component. This evaluation component collects and processes this information and transfers the results to a logging component for later analysis. All parts in Figure 2, introduced for the acquisition of data, are marked with "*". Along with the benchmark architecture an adequate workload for each measurement scenario has to be defined in the related automotive component benchmark definition.

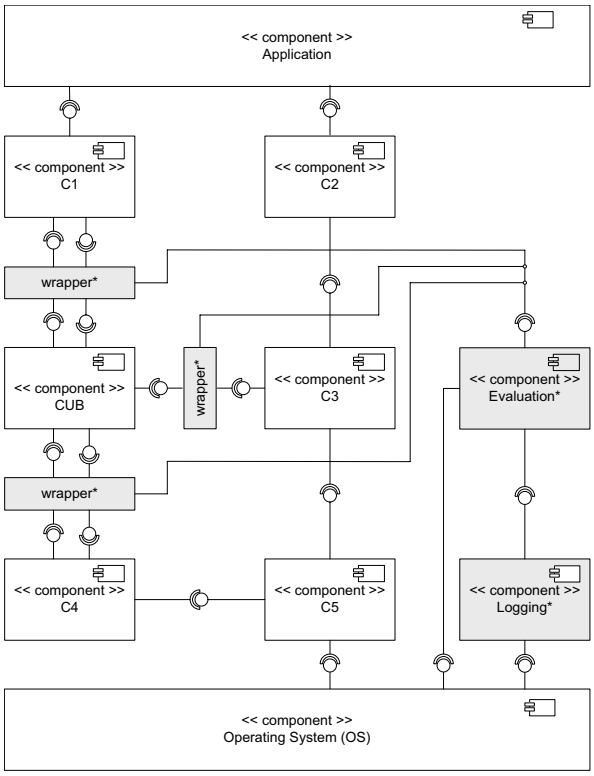

Figure 2. Benchmark architecture for integrating the assessed component in a real workload scenario

3.2.2 "Isolated Component"-Architecture

For some measures it is not feasible to use a real workload scenario as suggested. For those cases a benchmark architecture is proposed, allowing a single component to be assessed in isolation from other components. This benchmark architecture consists of the assessed component, an environment simulator, and the operating system. The environment simulator has to be configured at the begin of each benchmark run. It creates the workload and the faultload respectively. Furthermore it stores the benchmark data in a provided data storage. Since the environment simulator creates the workload, it is an artificial workload that can be either **realistic** or **synthetic** workload [3].

The two presented benchmark architectures are capable of covering a wide range of various assessment scenarios. Benchmark setups are therefore supposed to be based on one of the two benchmark architecture approaches. The actual setup including hardware, operating system, application, workload, faultload, etc. has to be accurately specified for each series of measurements by its according automotive component benchmark definition. Since many different measures are supposed to be acquired for each component,

usually more than one benchmark setup has to be specified in a benchmark definition.

4 Related Work

Measures and metrics to assess performance and dependability properties of a component are proposed in [7, 4, 2, 6, 14]. Most of them are based on the analysis and evaluation of the component's source code or object code. It is not possible to apply such measures and metrics under the restriction of a black-box component evaluation which is typical in automotive systems.

A benchmark framework approach is presented in [10, 12]. A benchmark consists of dimensions that have to be examined and specified. The proposed dimensions are separated into three classes: Categorization, measure and experimentation dimensions. The framework of dimensions was introduced in order to supply support for defining and specifying benchmarks. A complete specification based on this framework furthermore contributes to satisfy the properties reproducibility and representativeness. In our approach we propose not only to address dependability but other component attributes, like performance.

An approach for harmonizing dependability benchmarks is presented in [18]. Dimensions of dependability like availability, data integrity and disaster recovery are established. A classification of the application space into a set of types is proposed, e.g. transaction processing or process control. The authors further propose minimum requirements for each dimension and application type. Our automotive benchmark framework extends this approach by defining standardized measures and metrics for automotive components.

5 Conclusion

Component based software engineering is a well established engineering paradigm for embedded automotive software. To support system integrators and application developers in assembling a proper component architecture, available components have to be comparable regarding to relevant properties. This can be achieved by (i) identifying these relevant properties and by (ii) defining a unified benchmarking process. We identified the most common metrics and measures for automotive software, which were fewer and more simple ones than expected but have been agreed upon by all involved engineers. To make distinct implementations of components within the same component class comparable, we introduced a unified benchmarking process. This process relies on derivation of specialized benchmarks from a common framework. It allows a sound specification of component-class specific benchmarks that are capable of handling relevant properties.

References

[1] A. Avizienis, J.-C. Laprie, B. Randell, and C. E. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Sec. Comput*, 1(1):11–33, 2004.

[2] M. F. Bertoa and A. Vallecillo. Quality attributes for COTS components, Dec. 09 2002.

[3] K. Buchacker, M. D. Cin, H.-J. Höxer, R. Karch, V. Sieh, and O. Tschäche. Reproducible dependability benchmarking experiments based on unambiguous benchmark setup descriptions. In *DSN*, pages 469–478. IEEE Computer Society, 2003.

[4] E. S. Cho, M. S. Kim, and S. D. Kim. Component metrics to measure component quality. In *APSEC*, pages 419–426. IEEE Computer Society, 2001.

[5] COMPASS. *Component Based Automotive System Software*. http://www.infosys.tuwien.ac.at/compass.

[6] P. Donzelli, M. Zelkowitz, V. Basili, D. Allard, and K. N. Meyer. Evaluating COTS component dependability in context. *IEEE Software*, 22(4):46–53, July/Aug. 2005.

[7] N. Fenton and S. L. Pfleeger. *Software Metrics: A Practical and Rigorous Approach*. International Thomson Computer Press, London, UK, second edition, 1996.

[8] P. Hansen. New s-class mercedes: Pioneering electronics. *The Hansen Report on Automotive Electronics*, 18(8):1–2, October 2005.

[9] G. T. Heineman and W. T. Councill, editors. *Component-Based Software Engineering*. Addison Wesley, 2001.

[10] K. Kanoun, H. Madeira, and J. Aria. A framework for dependability benchmarking, June 18 2002.

[11] H. Madeira, J. Arlat, K. Buchacker, D. Costa, Y. Crouzet, M. D. Cin, J. Dures, P. Gil, T. Jarboui, A. Johansson, K. Kanoun, L. Lemus, R. Lindström, J.-J. Serrano, N. Suri, and M. Vieira. Dependability benchmark definition: Dbench prototypes, 2002.

[12] H. Madeira, K. Kanoun, and J. Arlat. Towards a framework for dependability benchmarking, 2002.

[13] H. Madeira and P. Koopman. Dependability benchmarking: making choices in an n-dimensional problem space, Sept. 09 2001.

[14] O. P. Rotaru and M. Dobre. Reusability metrics for software components. IEEE Computer Society, 2005.

[15] D. Schreiner and K. M. Göschka. Explicit connectors in component based software engineering for distributed embedded systems. In *SOFSEM 2007: Theory and Practice of Computer Science, Proceedings*, volume 4362 / 2007 of *LNCS*, pages 923–934. LNCS, Springer, Jan 2007.

[16] T. J. Teorey and W. T. Ng. Dependability and performance measures for the database practitioner. *IEEE Trans. Knowl. Data Eng*, 10(3):499–503, 1998.

[17] I. Wenzel, R. Kirner, B. Rieder, and P. P. Puschner. Measurement-based worst-case execution time analysis. In *SEUS*, pages 7–10. IEEE Computer Society, 2005.

[18] D. Wilson, B. Murphy, and L. Spainhower. Progress on defining standardized classes for comparing the dependability of computer systems, June 18 2002.

[19] J. Zhu, J. Mauro, and I. Pramanick. R-cubed (r3): Rate, robustness, and recovery - an availability benchmark framework, 2002.