Reusable Component Analysis for Component-Based Embedded Real-Time Systems

Johan Fredriksson[†⋆] Rikard Land[†]

[†]The PROGRESS Centre for Predictable Embedded Software Systems,
Mälardalen Real-Time research Centre, Västerås, Sweden
[⋆]CC Systems, Västerås, Sweden

Abstract *Component-Based Software Engineering (CBSE) promises an improved ability to reuse software which would potentially decrease the development time while also improving the quality of the system, since the components are (re-)used by many. However, CBSE has not been as successful in the embedded systems domain as in the desktop domain, partly because requirements on embedded systems are stricter (e.g. requirements on safety, real-time and minimizing hardware resources). Moreover these requirements differ between industrial domains. Paradoxically, components should be context-unaware to be reusable at the same time as they should be context sensitive in order to be predictable and resource efficient. This seems to be a fundamental problem to overcome before the CBSE paradigm will be successful also in the embedded systems domain. Another problem is that some of the stricter requirements for embedded systems require certain analyses to be made, which may be very complicated and time-consuming for the system developer.*

This paper describes how one particular kind of analysis, of worst-case execution time, would fit into the CBSE development processes so that the component developer performs some analyses and presents the results in a form that is easily used for component and system verification during system development. This process model is not restricted to worst-case execution time analysis, but we believe other types of analyses could be performed in a similar way.

1 Introduction

Embedded and real-time systems have requirements not found in desktop systems such as real-time requirements and resource efficiency. This is one reason to why embedded and real-time systems have more difficulties adapting to component-based software engineering (CBSE) than, e.g., desktop systems.

Expected benefits from using CBSE include more effective management of complexity, shorter time-to-market, improved quality and higher maintainability. Reuse is the main characteristics for component-based development that would bring these benefits. We believe that CBSE is easier applied in embedded and real-time systems if context dependencies can be relaxed or parameterized without sacrificing predictability and analyzability. Thus, the aim of this work is to simplify reuse by relaxing context dependencies through dividing analysis into one reusable part and one specific part.

Our research goal is to support real-time for general component-models, this paper presents one step further towards that goal. We specifically focus on supporting worst-case execution time (`wcet`) analysis. We present a development framework that decreases the gap between embedded real-time system and component-based development paradigms.

In previous work [12, 13, 14] we have developed methods for increasing analysis accuracy and resource efficiency of embedded real-time systems. In this paper we put these methods in a larger scope, describing how the different actors, component developer and system developer, may use these methods to enable higher potential for component reuse by performing the analysis in such a way that analysis is reused. The contribution of this paper is to utilize our previously proposed methods for aggregation of analysis results to create a system development process. By introducing a reusable component analysis model where part of the analysis is performed by the component developer the overall development process becomes arguably more efficient than with traditional analysis, which implies that the complete analysis is required to be performed by the system developer.

The rest of the paper is outlined as follows; in section 2 we describe background and related work. Section 3 introduces the context sensitive analysis and our proposed development method. In section 4 we discusses how our methods fit in development process for component-based development. Finally, section 5 concludes the paper and discusses future research.

2 Background and Related Work

The embedded systems industry is under competitive pressure to continually shorten its time-to-market, increase product differentiation and at the same time offer more customer value. As a result (i) embedded systems become increasingly software intensive and (ii) individual components integrate increasing functionality over different projects and reuse cycles. Integrating more functions into a single component gives rise to increasingly varying behavior, some variants only invoked in a particular deployment context, some only as part of adaptive behavior triggered by context-sensitiveness or deployment-specific configuration parameters. Extra-functional properties (EFPs) of the component such as time and reliability are variable and context-sensitive and the variance may be large. For these reasons software components in embedded systems make it difficult to predict EFPs and hence guarantee quality of the components and the services they provide through analyses [7]. At the same time, componentization has been the key to structured design processes with predictable properties in many other engineering domains. For software, in particular, a context-unaware characterization of component properties is inadequate for accurate predictions. A common obstacle to combine predictability with correctly dimensioned hardware is the inaccuracy of the system analysis. Real-time analysis is based on worst-case assumptions, and the composition of worst-cases make the system impractically oversized and under utilized [10].

In [15] a framework has been developed that considers the usage of a system; however, neither software components nor reuse is considered. In [9] a framework for probabilistic `wcet` with static analysis is presented. Recent case-studies show that it is important to consider mode- and context-dependent `wcet` estimates when analyzing real sized industrial software systems [19]. There are suggested models of the overall component-based life cycle processes [2, 8] as well as more concrete methods for e.g. component assessment [4, 18]; our work illustrates how the division into context-unaware and context-sensitive analyses would be integrated into these models.

2.1 Real-Time for Component-based systems

During the last decade advances have been made in component-based development for desktop and internet applications. A few de-facto standards have completely transformed the way such software is developed. These standards are mainly Microsoft's .NET, SUN's Enterprise Java Beans and OMG's Corba Component Model (CCM). Component models for embedded systems are usually designed with very domain specific requirements in mind [17]. For instance the well known component model by Philips, Koala [20], considers low resource usage, but does not consider, e.g., real-time properties that are important in many other embedded domains. There is a large set of different component technologies that approach different problems in different ways such as ABB's PECOS [21], Rubus [16], SaveComp [1] and many more. None of these however have yet been successful outside of their intended domain. Thus, for embedded systems it seems difficult to define de-facto standards due to highly diverging requirements on different industrial segments [7].

One of the more important component properties is unquestionable *reuse*. It is commonly accepted that reuse, if used properly, increases productivity and lowers development costs. However, support for reuse requires generality of components which often leads to low accuracy of component properties which is enough for desktop systems. But for embedded real-time systems low accuracy of component properties leads to low resource efficiency, and low resource efficiency leads to higher manufacturing costs in terms of hardware resources. On the other hand, lack of support for reuse increase development costs and increases time-to-market.

Reusable components should, by definition, be used in different applications [6], i.e., they should be context unaware. All possible deployments are not known and the extra-functional behavior of components in a new deployment is often very hard to predict. This is not a problem for desktop applications where resources are abundant and the requirements on, e.g., timing and safety are relatively low. There are very few component models that support general component properties at the same time as they are highly resource and run-time aware, and vice versa. Component models that are specifically designed for a particular group of systems are often not adaptable and general enough to be used in other systems or other domains. In order to achieve reuse, most component technologies of today intentionally do not consider the system context, e.g., inputs, hardware and run-time system. As a result performance prediction is often inaccurate.

3 Introducing the Reusable Component Analysis Model

Traditional `wcet` analysis (figure 1) is not suitable for large scale components since the analysis considers the absolute worst-case for all possible uses. Components and systems are developed separately, sometimes even by different companies and it is important to be able to get accurate predictions from the components when they are used in a specific system. The method to do this has been to use specific component

models for embedded real-time systems. When using highly specialized components it becomes virtually impossible to reuse them in any other system or context. Thus there is a trade-off between the generality required for efficient reuse, and the particularity of accurate component properties and efficient transformation to real-time system. The potential benefits of reuse are particularly high in the embedded domain where product differentiation is ever increasing and competitiveness is driven by time-to-market and costs; thus there is reason to find a solution to the trade-off.

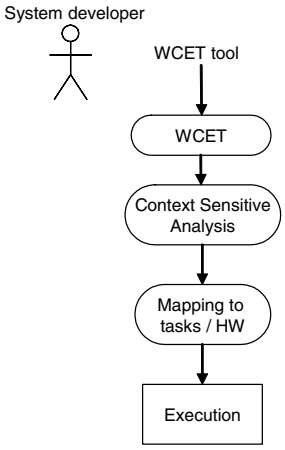

Figure 1. Traditional Analysis model

3.1 Context and Reuse

In order to be easily reused a component should be general and context-unaware [5]. As components grow and become more general and more reusable, they get a more diverse behavior, thus more varying execution-times. This is not an issue in most desktop systems but in embedded and real-time systems it is not "cost-efficient" to dimension hardware resources after the absolute worst-case, although that would be required in lack of more sophisticated methods to predict the *actual-case execution time* of the system as a whole.

To achieve reuse at the same time as accurately predicting `wcet` we propose division of the analysis into (i) context-unaware analysis, and, (ii) context and usage sensitive analysis, as depicted in figure 2. In addition monitoring and feedback to the context (usage-profile) is added. The analysis model is driven by a context-unaware `wcet` analysis (depicted as `wcet`-*tool* and *measurements* in figure 2), and context-sensitive *expert domain knowledge* about the system usage. The context-unaware `wcet` analysis (*Reusable Analysis*) is combined with the context-sensitive usage knowledge (*Usage-profile*) to assess an accurate, context-sensitive, `wcet` (*Context-sensitive analysis*). When the system analysis has been performed, and the system has been mapped to a run-time system and is executed, observations (*Monitoring and Usage feedback*) are used to fortify the usage-profile, depicted as a feed-back to the usage-profile in the figure. Reuse in a new system, or changed usage of the system potentially results in a more accurate `wcet` with higher confidence. An even greater gain is reuse without performing a new analysis with `wcet`analysis-tools.

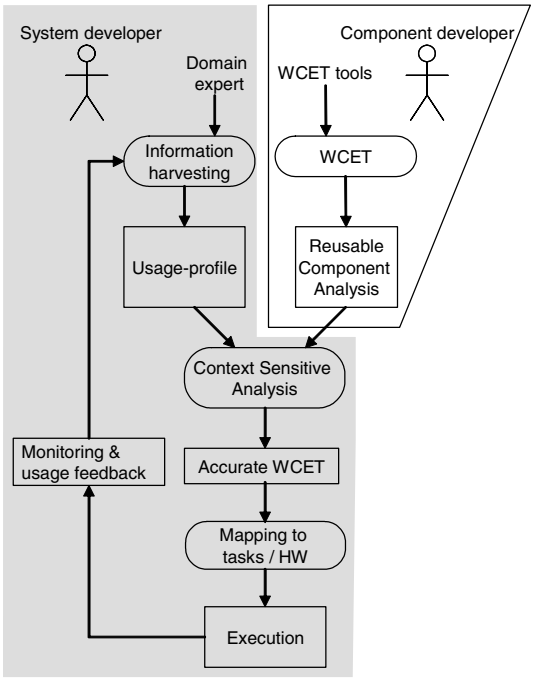

Figure 2. Reusable Component Analysis model

3.2 Reusable Timing Analysis

Analysis is performed for different reasons in the development. Of course components are required to be analyzed when the component is developed or changes to the code are made because of, e.g., maintenance, updates, etc. However, because of the generality of components there are more occasions when analysis must be renewed to get enough accuracy in the analysis. When the usage is changed, i.e., the input-profile or the composition, analysis must be updated to avoid a too pessimistic `wcet`. The need for reanalysis at every reuse is a definite obstacle for reuse in embedded real-time systems.

In [13] we have proposed a context-unaware analysis by making `wcet` analysis for a large number of input combinations. The `wcet`'s and inputs are gathered in a table-like structure. In that work we show how optimization can be used to minimize the size of the table and get as few entries as possible without

loosing information. Let us exemplify by introducing an example of the resulting analysis.

Ex. The component developer has performed `wcet` analysis of the component with respect to inputs (usage). We assume a component with two inputs, A and B. The method has resulted in a very simple table (see table 1). The system wide `wcet` (for any usage) is 2400 ms.

No	A [0,255]	B [0,255]	wcet
1	$0 \leq A < 30$	$0 \leq B < 100$	2400
2	$0 \leq A < 30$	$100 \leq B < 180$	1900
3	$30 \leq A < 100$	$100 \leq B < 180$	1200
4	$100 \leq A < 150$	$B < 100$	1000
5	$A < 100$	$B > 200$	400

Table 1. wcet with respect to any input $0 \leq A < 256 \wedge 0 \leq B < 256$. All predicates are true.

For sake of readability we have made the example very simple (for a real component there may be many input parameters, and based on our simulations the many combinations could easily result in a lookup table with thousands of entries; there is a correspondence of course between the number of entries and the chosen size of clusters). In previous work [13] we also consider probabilities of inputs and outputs with respect to inputs to get a composable approach. For the sake of simplicity we leave out all details and only give a general view of the concept.

3.3 Context-sensitive analysis

By assessing which input combinations are valid for a specific system and applying them to the reusable analysis we attain a seamlessly reusable analysis. In this way a component can be reused and reanalyzed without having to do any actual `wcet` analysis. The benefits of not having to perform traditional `wcet`-analysis for every usage is that only the developer of the component is required to have licenses and expertise of commercial `wcet`-tools. It is our firm conviction and experience that it is easier to do a lookup in a table compared to using commercial `wcet`-tools. Domain expert information and observed behaviors are utilized to attain specific information about each system. This information is refined during development, testing and execution by monitoring the system, as indicated in figure 2.

We propose a usage-profile with input dependencies, i.e., probabilistic relationships between inputs. Consider the following example:

Ex. After an assessment of the usage by the system developer and the domain expert, the usage-profile has been determined to be the following: $0 \leq A < 100 \wedge 130 \leq B < 200$. Thus, we see that some of the `wcets` may be ignored, leading to a more accurate prediction of the system behavior.

By applying the usage-profile to table 1, we see right away that rows 2 and 3 (marked with gray) are affected, thus resulting in a `wcet` of $max(1900, 1200) = 1900$, as shown in table 2.

No	A [0,255]	B [0,255]	wcet
1	$0 \leq A < 30$	$0 \leq B < 100$	2400
2	$0 \leq A < 30$	$100 \leq B < 140$	1900
3	$30 \leq A < 100$	$100 \leq B < 140$	1200
4	$100 \leq A < 150$	$B < 100$	1000
5	$A < 100$	$B > 200$	400

Table 2. wcet with respect to a usage-profile $0 \leq A < 100 \wedge 130 \leq B < 200$. The predicates of rows 2 and 3 are true.

3.4 Monitoring and usage feedback

Further information about the usage is gathered throughout testing and execution by monitoring the system. The information that is required to be gathered is the inputs of the components and a cross reference so that it is possible to assess the input state, i.e., the values of all inputs at every observation point. Similar work has focused on gathering information about execution times to make the worst-case prediction itself more accurate [3].

Ex. Lets revisit the component with the two inputs *A* and *B*. After a large number of input observations the probability it may be possible to further asses a more refined usage, e.g., $170 \leq B < 240$ can be identified, further refining the analysis.

By applying the revised usage-profile we can further remove some `wcet`'s from table 3, resulting in a `wcet` of 400. Note that this is an artificial example and refinements can be arbitrarily large depending on the initial knowledge of the usage.

No	A [0,255]	B [0,255]	wcet
1	$0 \leq A < 30$	$0 \leq B < 100$	2400
2	$0 \leq A < 30$	$100 \leq B < 140$	1900
3	$30 \leq A < 100$	$100 \leq B < 140$	1200
4	$100 \leq A < 150$	$B < 100$	1000
5	$A < 100$	$B > 200$	400

Table 3. wcet with respect to the usage-profile $0 \leq A < 100 \wedge 170 \leq B < 240$. The predicate of row 5 is true.

For each time the system is executed the usage-profile is improved. In turn re-analysis will be more accurate and have higher confidence.

4 Positioning the Analysis in the System Development Process

Turning to the highest level of system development, the benefits of the proposed division into reusable and context-sensitive analysis become apparent. The component-based system development process can be described in two parts: a system development process and a component development process [8] (this distinction is useful even when component development and system development takes place within the same organization). The interface between these two processes may be fairly complex. First during system development, existing components may be surveyed already during requirements phase (and influence the entire scope and direction of the system). Later components are tested to assess functionality and quality characteristics; they are used in prototyping during design, and of course finally integrated and deployed with the system. And conversely, requirements on the system may also affect the evolution of its constituent components more or less directly (depending on the business relationship).

Figure 3 shows how a general model for CB processes would be extended with the *exhaustive component analysis* as part of the *Verification* phase of the component development process, and the analysis results packaged with the component in the *Release* phase. These results may then be used in several ways during the *context-sensitive analysis*, which would be done in the *Verify* phase of the component assessment process. One may dimension the hardware by looking up the `wcet` by the known or estimated input, and/or in some circumstances it may be possible to restrict the allowed input, to gain a low (known) `wcet` and thus fit the component into certain resource-constrained hardware. During the *Verification* and *Operation* phases of the system, the actual usage of the component is monitored, which may lead to the usage profile being fine-tuned and the context-sensitive analysis to be redone.

This development process allows for `wcet` analysis to be moved from the system development to the component development. In this way, not only the component itself but also the `wcet` analysis is reused several times. The system developer escapes the effort of learning and using advanced `wcet` analysis tools, and the overall process becomes more efficient. In our experience it is a matter of several man weeks to setup, learn and effectively use many of the commercial and research `wcet` analysis tools, in comparison to a few minutes of simple table lookups, as proposed in our work. The work effort is effectively moved from the system developer to the component developer, and for every reuse there are potentially big time gains.

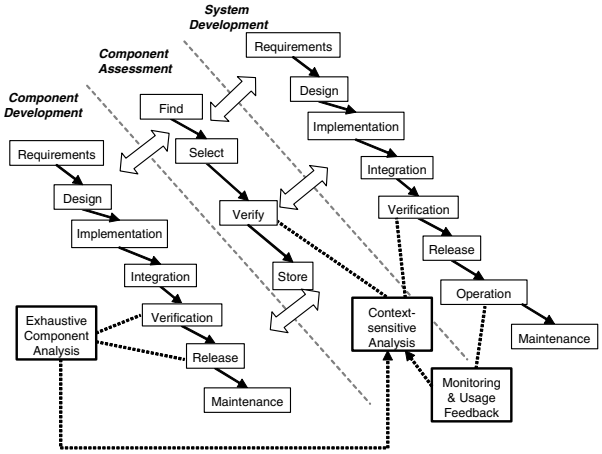

Figure 3. Development process

5 Future Work and Conclusions

We are witnessing an enormous expansion in the use of software in embedded systems. Software is no longer marginal in technical systems, but has now become a central factor in many fields. System features based on software are becoming the most important factor in competing on the market.

We have previously developed methods for reusable real-time analysis and component to task mappings for increasing predictability and resource efficiency [13, 14]. In this paper we have described a development process where these methods are central, focusing on support for reuse; in the resulting reusable component analysis part of the analysis is performed by the component developer instead of the system developer as is implied by existing analysis methods. The development process facilitates analysis of the system characteristics and thereby facilitates dimensioning of resources for the system. The model is independent of component technology, and in on-going work [11] we explore how to apply the same principles to other types of analyses. The provision of context-unaware analysis models for reusable components would be a selling argument for component vendors.

On-going and future work also includes evaluating the process and methods with simulations, experiments and industrial evaluations.

Acknowledgements

This work is partly funded by SSF, the Swedish Foundation for Strategic Research.

References

[1] M. Åkerholm, J. Carlson, J. Fredriksson, H. Hansson, J. Håkansson, A. Möller, P. Pet-

tersson, and M. Tivoli. The save approach to component-based development of vehicular systems. *The Journal of Systems and Software*, 2006.

[2] A. S. Andreou, A. C. Zographos, and G. A. Papadopoulos. A three-dimensional requirements elicitation and management decision-making scheme for the development of new software components. In *ICEIS (3)*, pages 3–13, 2003.

[3] G. Bernat, A. Colin, and S. Petters. pWCET, a Tool for Probabilistic WCET Analysis of Real-Time Systems. In *WCET*, pages 21–38, 2003.

[4] J. Bhuta and B. Boehm. A method for compatible cots component selection. In *International Conference on COTS-Based Software Systems*, pages 132–143, 2005.

[5] O. Ciupke and R. Schmidt. Components as context-independent units of software. In *WCOP'96 in conjunction with ECOOP'96*, Linz, Austria, July 1996.

[6] I. Crnkovic. Component-based software engineering – new challenges in software development. *Focus Review, Software Focus*, 2(4):127–133, 2002.

[7] I. Crnkovic. Component-based approach for embedded systems. In *WCOP'04*, Oslo, June 2004.

[8] I. Crnkovic, M. Chaudron, and S. Larsson. Component-based development process and component lifecycle. In *ICSEA'06*, Tahiti, French Polynesia, October 2006. IEEE.

[9] L. David and I. Puaut. Static determination of probabilistic execution times. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS'04)*, pages 223–230, 2004.

[10] M. Duranton. The challenges for high performance embedded systems. In *9th Euromicro Conference on Digital Systems Design, DSD*, pages 3–7. ieee, September 2006.

[11] J. Fredriksson and T. Nolte. Contract-based reusable analysis for software components with extra-functional properties. Technical report, Mälardalen Real-Time Research Centre, April 2007. (submitted to ECRTS'07).

[12] J. Fredriksson, T. Nolte, A. Ermedahl, and M. Nolin. Clustering worst-case execution times for software components. Technical report, Mälardalen Real-Time Research Centre, April 2007. (submitted to WCET'07).

[13] J. Fredriksson, T. Nolte, M. Nolin, and H. Schmidt. Contract-based reusable worst-case execution time estimate. Technical report, Mälardalen Real-Time Research Centre, April 2007. (submitted to RTCSA'07).

[14] J. Fredriksson, K. Sandström, and M. Åkerholm. Optimizing Resource Usage in Component-Based Real-Time Systems. In *Proceedings of th 8th International Symposium on Component-Based Software Engineering (CBSE8)*, May 2005.

[15] J.-I. Lee, S.-H. Park, H.-J. Bang, T.-H. Kim, and S.-D. Cha. A hybrid framework of worst-case execution time analysis for real-time embedded system software. In *Aerospace, 2005 IEEE Conference*, pages 1–10. ieee, March 2005.

[16] K. L. Lundbäck, J. Lundbäck, and M. Lindberg. Component-based development of dependable real-time applications, 2003.

[17] A. Möller, M. Åkerholm, J. Fröberg, and M. Nolin. Industrial grading of quality requirements for automotive software component technologies. In *ERTSI'05 in conjunction with RTSS'05*, 2005 Miami, USA, December 2005.

[18] C. Ncube and N. A. Maiden. *Component-Based Software Engineering: Putting the Pieces Together*, chapter Selecting the Right COTS Software: Why Requirements Are Important. Addison-Wesley, 2001. ISBN 0-201-70485-4.

[19] D. Sehlberg, A. Ermedahl, J. Gustafsson, B. Lisper, and S. Wiegratz. Static wcet analysis of real-time task-oriented code in vehicle control systems. In *2nd International Symposium on Leveraging Applications of Formal Methods (ISOLA'06)*, Paphos, Cyprus, November 2006.

[20] R. van Ommering. *Building Reliable Component-Based Software Systems*, chapter The Koala Component Model, pages 223–236. Artech House Publishers, July 2002. ISBN 1-58053-327-2.

[21] M. Winter, T. Genssler, et al. Components for Embedded Software – The PECOS Apporach. In *The Second International Workshop on Composition Languages, in conjunction with the 16th ECOOP*, June 2002.