

# On the Design of Consistent Executable Assertions for Distributed Software\*

Arshad Jhumka, Martin Hiller, Vilgot Claesson and Neeraj Suri  
Dept of Computer Engineering  
Chalmers University  
412 96, Göteborg, Sweden  
Email:{arshad,hiller,vilgotc,suri}@ce.chalmers.se

## Abstract

Software (SW) is increasingly defining the services for embedded systems. As the dependability requirements for embedded systems grow, consequently the dependability requirements extend to SW as well. In order to produce such reliable SW, it is important to know the nature of faults that can affect it and also how the resulting errors propagate through the system. Over the design of embedded SW, Executable Assertions (EAs) are seeing increasing usage in aiding detection of data errors in SW. However, designing a set of EAs that are located in different modules of the SW such that each provides complementary protection is a difficult problem, i.e., whether localized EAs add up to implement a global EA. To address this issue, we present a two-pass approach. First, we develop a semantics-based framework that allows EAs to be specified and verified for consistency. This pass is based on abstract interpretation. Second, whenever the set of EAs is found to be inconsistent (they do not provide complementary coverage), we have developed an algorithm that systematically generates a set of consistent EAs (consistent both within and across modules).

## 1 Motivation

Software (SW) is increasingly determining the provision of dependability in safety-critical embedded systems, with corresponding strict requirements on the correctness of delivered services, and the ability to handle faults - design and/or operational. In order to produce such reliable SW that can meet these stringent requirements, it is important to know the nature of faults that can affect it and also how the resulting errors propagate through the system. To mitigate the effects of such errors, these errors need to be detected and corrected as early as possible. Error Detection Mechanisms (EDMs) such as EAs [3] have been shown to efficiently detect data errors in SW [3]. EAs are essentially predicates over the program state variables, stating the properties of the program that must exist at this point. The ability to detect (and correct) data errors is of fundamental importance in preventing errors

occurring in one part of the system from propagating to other parts of the system.

However, the design of EAs that provide high error coverage has been an ad-hoc process, requiring in-depth knowledge of the system. In our previous work, we have developed techniques (a) to measure the effectiveness of EA [3], and (b) to systematically locate EAs for effective fault coverage [4]. However, as distributed SW is considered, an important problem is to determine if locally placed EAs (EAs in a given module) provide synergistic or complementary coverage on a global basis, i.e., consistency across local EAs to implement a global EA.

### 1.1 Approach

In this fast abstract, we present definitions of consistency, and also the nuances and scenarios for inter-EA consistency. We will then provide a sketch of an algorithmic approach that aids in systematic generation of consistent EAs.

**Definition 1.**  $EA_1$  and  $EA_2$  are said to be consistent with each other whenever the code linking the two EAs transforms  $EA_1$  into  $EA_2$ . They are partially consistent if the transformation of  $EA_1$  overlaps with  $EA_2$ . Otherwise, they are inconsistent.

For example, consider *Function 1*: the global EA is the EA monitoring the output to the environment (user), and in this case,  $EA_2$ .  $EA_1$  is a local EA that monitors the input. From this example,  $EA_1$  and  $EA_2$  are not consistent since, when  $0 \leq param \leq 5$ ,  $EA_2$  detects an error unlike  $EA_1$ .

```
Function 1  
Function MultiplyBy5(int param)  
int val;  
ASSERT(param > 0); EA 1  
val := param * 5;  
ASSERT(val > 25); EA 2 - Global EA  
return(val);  
End Function
```

Overall, we identify the following cases of EA consistency to cover partial, inconsistent and consistent behavior:

\*Copyright ISSRE and Chillarege Corp. 2001

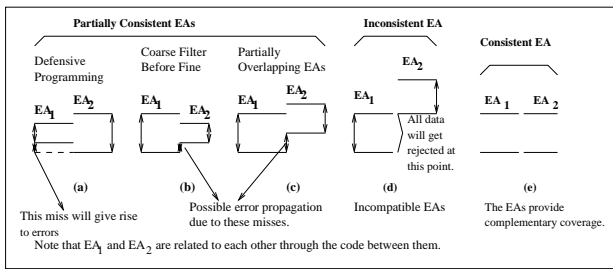


Figure 1: EAs with varied consistencies.

Fig. 1(a)–(c) show examples of partially consistent EAs, while Fig. 1(d) and (e) shows examples of inconsistent and fully consistent EAs respectively. Function 1 above is an example of partial consistency.

To facilitate provisioning of consistent EAs, we have adopted a two-pass approach. The first pass incrementally verifies the EAs (both within a module and across modules) for consistency. A semantics-based framework is developed for the specification of EAs. The consistency verification is based on abstract interpretation [2], through the use of annotations, as in [1]. If the set of EAs is not consistent, we have developed an algorithm, as depicted in Fig. 2 that systematically generates a set of consistent EAs [6].

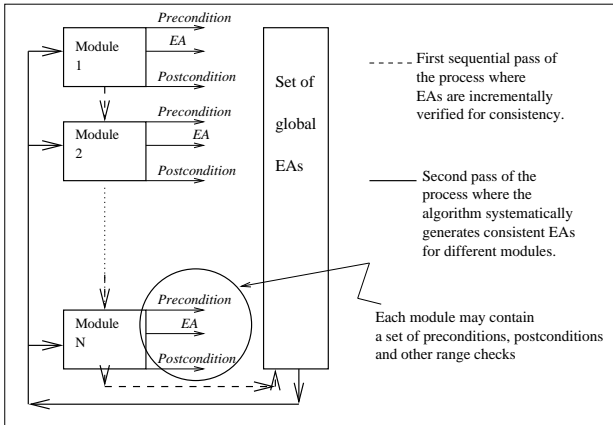


Figure 2: Two-pass algorithmic approach for consistent EA generation

## 1.2 Discussions

Having presented the approach, some salient aspects of the approach are discussed.

- As the first pass is based on abstract interpretation, generation of consistent EAs can be done at compile time. This will provide the programmer with additional information pertaining to the effectiveness of the EAs in the code with respect to

the global EA. Also, this process is fully automatic and does not require manual annotations.

- One important aspect of this approach is that it improves on the intractability of formal methods-based verification, due to the very large state space. At the code level, the intractability problem is alleviated.
- A broader implication of our work is on security aspects. Security checks that are placed in a given module and across different modules need to be ascertained to conform to an overall security property. [5] addresses some of these security issues, more specifically across modules (flow of control). Thus, our approach of inter-EA consistency (along with the formal semantics of EA specifications) has utility in developing globally compliant security filters/firewalls.

Overall, we have presented an algorithmic approach that generates consistent EAs at compile time. We envision the EAs generated by the algorithm to provide for enhanced error coverage and, enhanced error detection latency.

## References

- [1] A. Ermedahl, J. Gustafsson, “Deriving Annotations For Tight Calculation of Execution Time”, *Proc EuroPar’97, RT System Workshop*
- [2] P. Cousot, R. Cousot, “Abstract Interpretation: A Unified Model for Static Analysis of Programs by Construction or Approximations of Fix Points”, *Proc. 4th ACM Symposium on Principles of Programming Languages*, pp. 238–252
- [3] M. Hiller, “Executable Assertions for Detecting Data Errors in Embedded Control Systems”, *Proc. DSN 2000*, pp. 24–33
- [4] M. Hiller, A. Jhumka, N. Suri, “An Approach for Analyzing The Propagation of Data Errors in Software”, *Proc. DSN 2001*, pp. 161–170
- [5] T. Jensen, D. LeMetayer, T. Thorn, “Verification of Control Flow Based Security Properties”, *Proc. of Symp. on Research in Security and Privacy*, pp. 89–103, 1999.
- [6] A. Jhumka, M. Hiller, V. Claesson, N. Suri, “A Systematic Approach for Designing Consistent Executable Assertions for Embedded Software”, *Technical Report 01-8, Dept of Computer Engg, Chalmers University, Sweden*