# Essential Ingredients for a WCET Annotation Language*

Raimund Kirner, Albrecht Kadlec, Adrian Prantl, Markus Schordan, Jens Knoop
Vienna University of Technology, Austria

## Abstract

The situation for Worst Case Execution Time (WCET) analysis is very heterogeneous: Within the real-time community, it is a well known fact that manual annotations are needed to assist non-perfect analyses. Various tools exist providing different levels of sophistication. However, as the *WCET Tool Challenge* [1] has shown, few tools share the same target hardware, analysis method or annotation language.

For a fair comparison, a common annotation language is required, together with an accepted set of benchmarks in order to evaluate the various tools and methods. Still, as a direct consequence of the first WCET tool challenge, a set of accepted benchmarks has already being collected, without unified annotation support.

To enable annotations within these benchmarks, the *WCET Annotation Language Challenge* [2] has formulated the need for a common annotation language. This language has the purpose of specifying the *problem-inherent information* in a *tool- and methodology- independent* way, supporting, e.g., static analysis equally well as measurement based methods, thus allowing the comparison or the combination of their results. It also has the difficult task to enable annotations at the *source* level, which is the natural specification level, as well as supporting the annotation of binary or object code, if the source code is not available, such as for operating systems or libraries.

Within this paper we present a list of essential ingredients for a common WCET annotation language. These selected ingredients comprise a number of features available in different WCET analysis tools and add several new concepts we consider important. The annotation concepts are described in an abstract format that can be instantiation at different representation levels.

The essential ingredients of the WCET annotation language are:

- We distinguish analyzable *information* and user-supplied *annotations*.
- The novel concept of *overrules* adds a potent *what-if* querying mechanism.
- *Layers* are provided to reflect the affiliation of annotations with specific platform layers.
- *Symbolic grouping* is provided to aid in maintainance.
- The *addressable units* within a program that may need to be annotated are identified.
- The annotations are categorized into
  - *Source code related high-level annotations*
    e.g.: loop and recursion bounds, variable limits,
  - *Object- or binary- level annotations*
    e.g.: code vs. data distinction,
  - *Control-flow refinement annotations*
    e.g.: reachability and predicate evaluation information, and
  - *Hardware specific annotations*
    e.g.: memory map information, clock speeds, absolute time bounds.

We hope that comparing the different tools will be easier, using a common set of benchmarks that have the necessary annotations already included. This work may also encourage more researchers to support annotations on the *source* level.

## References

[1] Jan Gustafson. The WCET tool challenge 2006. In *Preliminary Proc. 2nd Int. IEEE Symposium on Leveraging Applications of Formal Methods, Verification and Validation*, pages 248 – 249, Paphos, Cyprus, November 2006.

[2] Raimund Kirner, Jens Knoop, Adrian Prantl, Markus Schordan, and Ingomar Wenzel. WCET analysis: The annotation language challenge. In *Proc. 7th International Workshop on Worst-Case Execution Time Analysis*, Pisa, Italy, July 2007.

*This work has been partially supported by the Austrian Science Fund (Fonds zur Förderung der wissenschaftlichen Forschung) under contract P18925-N13 and the ARTIST2 Network of Excellence, `http://www.artist-embedded.org/`.