Overview

With this project we target the development of reliable safety critical embedded applications, such as used widely in aerospace, in automotive applications, and in medical devices.

We have identified two core issues with the current state of the art:
1. Embedded systems software is normally structured in a way that does not leverage available analysis tools. The software is developed in an analysis-agnostic style that often obscures the abstractions made and hampers analyses;
2. Analysis tools do not leverage available knowledge of the embedded systems' environment. For example, embedded systems software relies critically on specific schedulers that differ significantly from those used on common desktop and mainframe platforms.

Approach

We address the above two core issues by designing and building a demonstration environment for safety-critical embedded systems software development.

1. The first issue is addressed by developing a new executable specification language that enforces a code structure that is analysis-friendly. Because scheduling disciplines can have an important influence on both execution and analysis, the language includes a sublanguage for describing schedulers.
2. The second issue is addressed by building an analysis tool that allows direct execution of specifications and also offers verification support based on model checking. The verification algorithms exploit the structure of the code and the schedulers introduced using the specification language. Constraining the analysis to traces that can occur in practice might lead to a significant performance improvement.

The architecture of our framework consists of three main components:
- an executive that keeps track of the current state and updates it by taking a step corresponding to an action from the language;
- a scheduler that, by looking at the current state, decides what actions can be taken next;
- an analysis tool (e.g., an interpreter or a model checker) that monitors the execution, picks an action from those returned by the scheduler, and instructs the executive to take the corresponding step.

Specification Language

We are developing a new specification language that is close to a systems implementation language (e.g., C), but has a formally defined semantics, provides higher-level abstractions and allows us to write formal specifications as part of the source code in the form of interface standards and assertion requirements. The language is also close to a typical specification language used in model checking (e.g., Promela), but is richer by providing functions and abstract data types.

The specification language is statically typed and provides support for concurrency and channel-based message passing. The memory model used by the language relies on the notions of ownership and ownership transfer and avoids features that can obstruct formal verification attempts (e.g., explicit pointers and unsafe casts). The specification language also provides features used by the scheduling language: defining process attributes and assigning processes to schedulers.

Scheduling Language

We are developing a constraint-based language for specifying common schedulers used in embedded systems. The language operates on predefined sets of processes, like the set of enabled processes, and can access the process attributes defined using the specification language.

References