Summary of the paper "Model-checking Middleware-based Event-driven Real-time Embedded Software" [Deng et al., 2004]

Kenneth Rohde Christiansen

Department of Mathematics and Computing Science
Rijksuniversiteit Groningen
Blauwborgje 3
NL-9747 AC Groningen

k.r.christiansen@student.rug.nl / kenneth@gnu.org

Abstract
Distributed systems are to a great extend being developed using component and middleware frameworks. With the introduction of Real-time CORBA and CORBA Component Model, these component frameworks are increasingly being used for developing mission-critical DRE (distributed real-time embedded) software systems as well. Due to safely concerns and soft and hard real-time requirements (temporal properties) model-checking proves a practical way to reason about these systems. In this summary we discuss a paper presenting a model-checker infrastructure for these software systems. The summary assumes that the reader has followed the Real-time Systems course given at the RUG or at least a similar course given at another research institution.

Keywords: Distributed Real-time Embedded Systems, CORBA Component Model, (Real-time) CORBA Event Service, Model-checking, Computer Science.

1 Introduction

Most distributed systems today are being build with sophisticated component and middleware frameworks such as .NET, Enterprise Java Beans and CORBA. Since these frameworks have proved advantageous for developing such systems, special real-time versions have seen the light and are increasingly being adopted. Real-time CORBA and CORBA Component Model (CCM) are two of these systems.

These systems usually work in the following manner: Loosely connected components communicate with each others using a middleware layer that hides the complexities that exists when moving data across the network. The middleware framework includes various *services* to support commonly required tasks in these systems, such as performing transactions and insuring persistence.

The components and the middleware framework are developed using standard object-oriented programming techniques and design patterns with all the usual advantages such as: improved software quality, reduced costs, reuse as well as easier maintenance and evolvement of the software product. The reuse of components is often important in these DRE systems.

The loosely coupling of the components is often archived by using the *CORBA Event System*, which is a asynchronous event-based communication infrastructure, or similar systems. Event services are advantageous as they provide a way to easily plug and unplug components into the systems (via subscription or publishing mechanisms) as well as provides support for creating new event types, event filtering and event correlation. Real-time versions often provide additional capabilities as QoS[1] and

[1] Quality of Service

temporal attributes and are often different in the way they support threads. Real-time event services, thus, don't allow components to include more than *one* thread. Instead, the *event service* provide a thread pool and the service itself provides and manages threads which then execute event-handlers of the subscribing components in the case an event is publishes. This gives greater control of the scheduling and timing constraints.

2 Introducing Cadena

Model-checking is widely used in real-time systems and there are therefore a need for using model-checking for these newer DRE systems as well. Unfortunately the tools on the market does not scale to these DRE systems. A lot of research has been done, mostly within two different domains:

- High-level modeling and analysis of async. systems.
- Model-checking of object-oriented software.

The systems for modeling distributed systems, unfortunately, lack important object-oriented features such as dynamically created objects and connection of components; and the tools designed for checking object-oriented software are not taking advantage of the numerous opportunities for reducing the state exploration that exist in these DRE systems, by for instance taking the threading and scheduling policies into account. This means that these existing tools, as for instance Bandera [Corbett et al., June 2000], Java Path Finder [Brat et al., July 2000] and dSpin [Demartini et al., Sept. 1999], does not scale to realistic systems due to the state expansion problem.

In order to fill this hole, the authors of the paper have developed a system called *Cadena*, which makes it possible to verify DRE systems developed using CCM. Cadena adds various extensions to the IDL (CCM Interface Definition Language) to enable light-weight specification of component behavior and dependencies. Additionally, Cadena includes a easy-to-use graphical user interface and integrates into the open-source Eclipse[2] IDE[3] being developed by IBM and others.

2.1 Background

Before the current version of Cadena, the authors simply made Cadena translate *CCM system descriptions* into the input language of the dSpin model-checker. This work is described in [Deng et al., 2003]. This approach proved well, but had some shortcomings. dSpin natively supports most object-oriented features such as dynamic creating and deletion of objects, but unfortunately it does not take the different threading methods and scheduling policies found in these DRE systems into account. This means that it does not take advantage of the state reduction methods available and, thus, does not scale to larger, realistic systems.

In the summarized paper the authors describes a new improved version of Cadena that has been developed in cooperation with Boeing and Rockwell-Collins. Due to this cooperation, the authors believe that Cadena is now more usable in real contents.

The cooperation was formed in order to make it possible to use Cadena for reasoning about the *system design and assembly phase* of the Boeing *Bold Stroke* system. Bold Stroke is a product line of object-oriented mission critical aircraft software for many of the military aircrafts producted at Boeing. Bold Stroke is developed using CORBA middleware and it a good example of the DRE systems that we described earlier.

2.2 The design

The new version of Cadena does not use dSpin as a back-end. Instead a new model-checker called *Bogor* has been developed, which supports the things missing from dSpin and additionally makes use of various state-reduction techniques. Bogor also includes a mechanism for adding new primitives to the

[2] http://www.eclipse.org
[3] Integrated Development Environment

modeling language.

The Boeing developers described their work as

- Hooking various general-purpose components together from a component library.
- Selecting distribution strategies.
- Selecting execution priorities.
- Selecting particular event communication layers.

In order to ensure the real-time deadlines, the developers used rate-monotonic scheduling theory and conventionally scheduling analysis techniques. This, however, resulted in many *costly* iteration of the design process as the developers, for instance, couldn't reason about the high-level control-flow.

In particular, what the developers requested what that they were able to reason about the following:

1. Intra-component control-flow realized by conventionally constructs as locks, conditionals etc.
2. The mode-state of the components and what effect it would have disabling/enabling a particular component action or the communication patterns.
3. Intra-component control-flow realized by event subscription patterns and the ordering of reception, broadcast, and correlation of events and methods calls.

In order to full fill these requirements three different kinds of system descriptions are used. These three descriptions will be described shortly. For a more elaborate description and examples on how to model the Bold Stroke system with these we refer to the summarized paper [Deng et al., 2004].

2.2.1 Component Models

According to the authors, the semantic description of the components only need to include simple control-flow skeletons with transitions actions in order to full fill requiremtn 1) and 2): reads/writes of state variables, the receiving and sending of events, as well as method calls to local objects. The interface of the components are described using the standard CORBA IDL[4] and the behavior by using a new Cadena specific format called CPS[5]. This new format makes it possible for developers to describe many semantic component properties such as the transition system and mode-aware dependencies. An example of this format is given in [Deng et al., 2004].

2.2.2 Middleware Infrastructure Models

The inter-component communication (to full fill requiremtn 3) is a bit more difficult to model as the level of abstraction is hard to get right. Too high a level of details and you quickly run into the state-expansion problem and too low level of detail and all property violations might not be explored. The solution has been to define several variations with varying precision and space/time requirements. This makes it possible for the developer to choose a fitting middleware variant when designing a system model. This idea makes use of the extension system of Bogor.

2.2.3 Connection actions

System initialization (creating of objects, followed by connection actions) is modeled by a new CAD[6] format. The Bold Stroke applications employs a static component allocation and configuration policy, which means that components are created and connected in the system initialization phase. While this is *standard practice* in most mission-critical systems, CAD additionally supports component to be created/deleted and connected/disconnected on the fly.

[4]Interface Description Language
[5]Component Property Specification
[6]Component Assemble Description

3 Strategies for reducing states

As described earlier, Cadena employs various strategies in order to forecome the state-expansion problem. The idea is to exploit knowledge of specific timing and scheduling strategies when analyzing these concurrent systems. This is feasible as many of the possible interleavings of concurrent actions will never take place in realistic systems. Without these kinds of reduction strategies the model-checker will only be able to model-check very small systems and practically only be of academic interest.

In the following we describe a few of these strategies:

Priority-based scheduling

This strategy makes use of the fact that exploring interleavings without considering thread priorities will create infeasible schedules that does not exist in realistic systems. This could for instance be a schedule that executes transactions from a low priority thread while a high-priority thread is activated.

Inter-rate-group timeout constraints

Like the strategy above, the idea is to remove interleavings that are unrealistic. In this strategy the actual idea is to remove schedules where for instance a 5 Hz event is ran more often than a 25 Hz event.

Lazy-time with priority scheduling

This strategy removes infeasible schedules by considering timing estimates for system transactions.

4 Results

In order to test the developed systems the authors described the Bold Stroke system developed by Boeing by the various Cadena description models. A test was made with different configurations. For instance the system was model-checked without considering the scheduling policy. Thus, it is *priority unaware* which means that *none* of the state reduction techniques can be applied and all interleavings are therefore checked. Three other tests were made with scheduling strategies applied as well as other properties.

The test showed that the state-expansion problem is in fact a real problem as only two scenarios, the smallest of the four, could be run with all four test configurations. The work performed by the authors have thus shown its application. The full description of the test cases, as well as the experiment data can be found in the paper ([Deng et al., 2004]).

5 Conclusion

The authors of the paper looked into what was required in order to model-check realistic distributed real-time embedded (DRE) systems and have developed an application Cadena that implements these requirements as well as takes advantage of various state-reduction strategies which removes many infeasible schedules found in real life DRE systems. The authors proves that these strategies made it possible to move the Cadena system out of the academic world and into real use in the industry, as the state-expansion problem was seriously reduces. This made the model-checker scale to handle the size of realistic systems found in the industry.

6 Acknowledgements

We want to thank our teacher Sietse Achterop for an interesting and worthwhile course and for introducing us to model-checking.

References

Brat, G., Havelund, K., Park, S., and Visser, W. (July 2000). Java pathfinder - a second generation of a java model-checker. *Proceeding of the Workshop on advances in Verification*.

Corbett, J. C., Dwyer, M. B., Hatcliff, J., Pasareanu, S. L. C. S., Zheng, R., and Zheng, H. (June 2000). Extracting finite-state models from java source code. *Proceeding of the 22nd International Conference on Software Engineering*.

Demartini, C., Iosif, R., and Sisto, R. (Sept. 1999). dspin : A dynamic extension of spin. *Theoritical and Applied Aspects of SPIN Model Checking*, **LNCS 1680**.

Deng, W., Dwyer, M. B., Hatcliff, J., Jung, G., and Prasad, V. (2003). Cadena: An integrated development, analysis, and verification environment for component-based systems. *Proceeding of the 25nd International Conference on Software Engineering*.

Deng, W., Dwyer, M. B., Hatcliff, J., Jung, G., Singh, R., and Singh, G. (2004). Model-checking middleware-based event-driven real-time embedded software.