

# CAR-SoC - Towards an Autonomic SoC node

Florian Kluge<sup>\*,1</sup>, Jörg Mische<sup>\*,1</sup>,  
Sascha Uhrig<sup>\*,1</sup>, Theo Ungerer<sup>\*,1</sup>

*\* Institute of Computer Science, University of Augsburg, 86159 Augsburg, Germany*

---

## ABSTRACT

The CAR-SoC project aims to develop an embedded hard-real-time system architecture, that is controlled by autonomic computing principles. Several SoCs can be connected to build an autonomic/organic network. The SoC is based on a multithreaded processor core with an interface for reconfigurable hardware. The system software fulfills autonomic computing requirements by implementing methods for self-configuration, self-healing, self-optimization and self-protection. These management functions will be running as helper threads concurrent to the hard-real-time application thread.

KEYWORDS: Autonomic computing, organic computing, multithreading, real-time scheduling, helper threads

## 1 Introduction

The paradigm of Autonomic Computing [Hor01, KC03] was introduced by IBM in 2001. This idea focuses on the design of computing systems that behave more like organic entities. Thus they should autonomously adapt to new challenges, heal themselves after injuries, and protect themselves against attacks. A similar concept was defined by the Organic Computing Initiative [OCI] as goals for the development of robust, flexible and highly adaptive embedded systems.

Our solution is to fulfill the Autonomic and Organic Computing (AC/OC) and the hard-real-time demands at hardware level by combining a multithreaded processor core with appropriate hardware scheduling within a SoC [UMU05]. These *Connective Autonomic Real-time SoC* (CAR-SoC) will be able, to dynamically form a network of several SoCs. The nodes will have Autonomic Managers running as helper threads, without influencing the real-time behavior of concurrent real-time threads.

---

<sup>1</sup>E-mail: {kluge,mische,uhrig,ungerer}@informatik.uni-augsburg.de

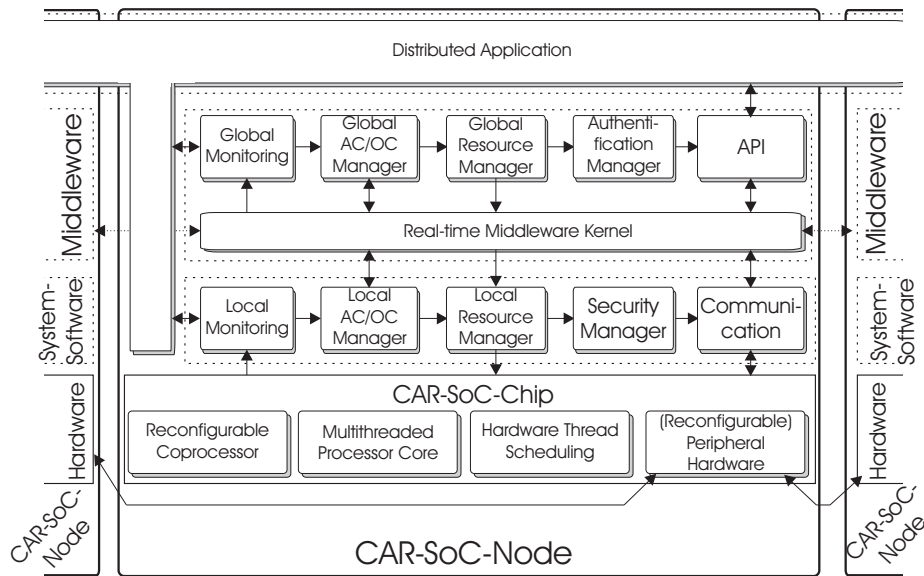


Figure 1: CAR-SoC project layers

## 2 Project Overview

Figure 1 shows an overview of the CAR-SoC project. The top layer of a CAR-SoC system consists of a distributed application which is based on a real-time middleware. Within each node, the local system software is executed on a multithreaded processor core with integrated real-time scheduling. The system software as well as the middleware support the autonomic/organic principles with the help of integrated closed-loop controls. While the system software provides a loop on the local, node-wide level, the middleware provides one on the global, system-wide level. Both loops consist of a monitoring function, an autonomic manager as control element, and a resource manager as actor.

## 3 Hardware

The processor core, called *CarCore* is based on Infineon's TriCore microcontroller, extended by simultaneous multithreading capabilities. It contains two pipelines, one for integer data and one for address calculation. The scheduler tries to fill both pipelines with instructions from the same thread, but if this is not possible, both pipelines can execute instructions from different threads. Currently only fixed priority and round robin scheduling is available, but adopting the *Guaranteed Percentage* scheduling developed in the Komodo project [KSP<sup>+</sup>00] will allow several threads to meet hard-real-time constraints.

The CarCore contains an interface to a *Reconfigurable Unit*, that allows to speed up time critical calculations. When one thread switches its execution to the reconfigurable unit, the other threads can exclusively use the CarCore pipeline, until the first thread switches back. This may lead to conflicts between the core and the reconfigurable unit, as both access memory simultaneously in a real-time capable way. To support hard-real-time execution in both parts, the reconfigurable unit cannot access memory directly, but it has to get an acknowledgement from the thread scheduler [UMKU06]. The realized technique leads to memory-accesses of the pipeline and the reconfigurable unit in an interlocked fashion.

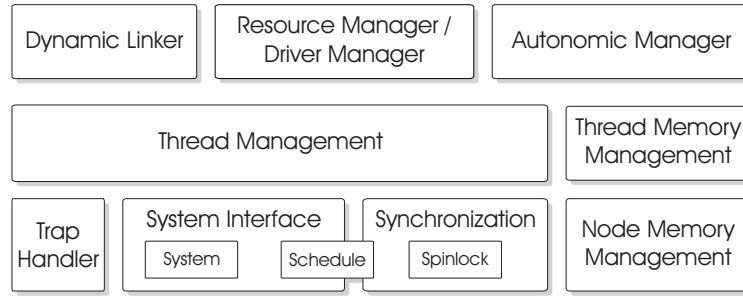


Figure 2: An overview of the CAR-SoC System Software

## 4 System Software

The system software pursues several goals. It must provide basic memory management and thread management such that hard-real-time is possible, even with several threads running. AC/OC support will be implemented, and also an extensive hardware monitoring. At last, it will provide a suitable interface to the AC/OC middleware.

We implemented the system software following the micro kernel paradigms. This micro kernel consists of three layers, as shown in figure 2. In the lowest layer, we provide fundamental accessors and hardware wrappers. The medium layer contains some more sophisticated structures for the thread management. The third layer provides support for the AC/OC principles.

The memory management is divided into two stages: A *Node Memory Management* in the first layer allocates big blocks of memory for each thread. The allocation of data memory for a running task is done in the second layer by the *Thread Memory Management* for each thread separately. Thus we achieve a non-blocking, realtime-capable memory management in the second layer, requiring that enough memory is reserved at thread creation. The first stage of memory allocation will always be blocking, as we work in a multithreaded environment. This is not that bad, as this allocation usually will only be called at thread creation, which itself is not a realtime capable operation.

The *System Interface* mostly provides functions for accessing special memory and processor registers. The displayed scheduling unit is also used by the synchronization unit for suspending and waking up threads. A *Trap Handler* catches exceptions that occur during regular program run.

The *Thread Management* is built upon the system interface and provides all functions necessary for creating, starting, suspending and resuming threads. A two-phase thread creation is utilized by the *Dynamic Linker*. This makes it possible to migrate code at runtime from one node to another. It also provides a more sophisticated mechanism for thread creation. Furthermore, there is a *Resource and Driver Manager*. This module provides a uniform interface for all available hardware resources. At boot time, it configures itself by loading the appropriate hardware drivers.

Finally, the *Autonomic Manager* is designed to monitor system parameters, and, if limits are exceeded, to initiate counteractive actions autonomously. The monitoring process is done by lots of small independent modules, each of which is to monitor only one parameter. The Autonomic Manager is running as a helper thread on the CAR-SoC. As control-mechanism we developed a *Classifier System* as proposed in [Hol71] enhanced by an adaptive parameter set. Currently we are working on several metrics suitable as decision criterions.

## 5 State of the Project and Future Work

The CarCore processor is designed and realized as SystemC model and the basic functions of the system software are implemented in C and assembler.

The main focus in the hardware layer lies on the enhancement of the scheduling. The guaranteed percentage algorithm will be extended to deal with several parallel pipelines and to control the reconfigurable unit, without violating hard-real-time constraints or blocking each other. Additionally, the scheduler will be able to handle a large amount of threads (hardware thread slots as well as software threads).

On the software part, our classifier system will be extended. Some more decision metrics must be developed and analyzed. We also want to equip the Autonomic Manager with learning functions. For the final implementation into the CAR-SoC system software, several optimizations have to be done. When these parts are finished, the real-time behavior of the complete system will be evaluated.

## References

- [Hol71] John H Holland. Processing and processors for schemata. In E. L. Jacks, editor, *Associative Information Processing*, pages 127–146. New York: American Elsevier, 1971.
- [Hor01] Paul Horn. Autonomic computing: IBM's perspective on the state of information technology. IBM Manifesto, IBM Corporation, October 2001.
- [KC03] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.
- [KSP<sup>+</sup>00] Jochen Kreuzinger, Alexander Schulz, Matthias Pfeffer, Theo Ungerer, Uwe Brinkschulte, and Christian Krakowski. Real-time Scheduling on Multithreaded Processors. In *7th International Conference on Real-Time Computing Systems and Applications (RTCSA 2000)*, Cheju Island, South Korea, pages 155–159, December 2000.
- [OCI] Organic Computing Initiative. <http://www.organic-computing.org>.
- [UMKU06] Sascha Uhrig, Stefan Maier, Georgi Kuzmanov, and Theo Ungerer. Coupling of a reconfigurable architecture and a multithreaded processor core with integrated real-time scheduling. In *13th Reconfigurable Architectures Workshop (RAW 2006)*, 2006.
- [UMU05] Sascha Uhrig, Stefan Maier, and Theo Ungerer. Toward a processor core for real-time capable autonomic systems. In *Proceedings of the Fifth IEEE International Symposium on Signal Processing and Information Technology*, 2005.