Safe Self-Evolving Embedded Software via System Virtualization

Stefan Groesbrink and Franz J. Rammig

Heinz Nixdorf Institut – Universität Paderborn, Paderborn, Germany

{s.groesbrink,franz}@upb.de

Abstract. *Self-evolving embedded software is assumed as a fundamental paradigm in designing Cyber Physical Systems (CPS). CPSs are characterized as a combination of interconnected embedded systems with the Cyber Space via Cloud computing. System virtualization is argued to be a promising approach to allow for safe self-evolving software. A system virtualization platform with an adaptive yet safe and real-time-capable resource management scheme based on a multi-mode protocol is proposed in this paper. Finally first results are presented.*

1. Introduction

Cyber Physical Systems (CPS) can be seen as an approach to combine distributed embedded systems with globally available data and services [Ra2012]. Due to the tight coupling of the involved embedded systems (ES) to their physical environment such systems have to continuously adapt to the evolving characteristics of their environment. In this sense, a CPS needs to be evolutionary by nature. The respective physical environment can be characterized by a set of parameters being divided in such ones that are under control of an ES belonging to the CPS and such ones being defined externally. Whenever changes in the externally defined parameters of the physical environment are observed that no longer can be handled by the current version of the respective ES, then this ES as a constituting part of the CPS has to be modified. A CPS with its access to globally available services can be interpreted as a special case of Service-Oriented Computing (SOC). In distinction from classical SOC these services have to be linked in as locally running components. The reason is the close connection to the controlled physical environment of an ES, which strictly requires predictable real-time behavior. Numerous open questions are subject of current research, including identifying the need for adaptation [Ca2012], look-up for matching components [MS2004], and ultra-late dynamic binding techniques [Ma2009]. In this paper we are focusing on a special aspect to support the dependability of such evolving architectures. In particular we are concentrating on a proper real-time system software environment enabling strict separation of dynamically interchangeable components from the rest of an ES. This is enforced by application of system virtualization, based on our real-time virtualization platform Proteus [BK2009]. The rest of the paper is organized as follows: In Sect. 2 the scenario we are dealing with is characterized while in Sect. 3 the resource management scheme for a virtualization platform to support safe component adaptation in a CPS is presented. First results are provided in Sect. 4 while Sect. 5 is a short summary.

2. An Adaptive CPS Scenario

A CPS can be seen as a network of agents. Each agent acts locally as a real-time system being directly connected to its local sensors and actuators and providing some locally real-time-capable services, typically closed-loop applications. In a CPS, it cannot be assumed that the parameters constituting the environment remain within a predefined domain over the entire lifetime of the system. As soon as those parameters are running out of such bounds the respective model of the environment is no longer valid. As a consequence some system adaptation is needed. We are restricting to component-based software systems where the entire software environment is composed of well-defined components with well-defined and highly standardized interfaces. Any kind of adaptation then means removing, adding, or exchanging such components. In the context of this paper we are concentrating on the challenge to ensure safe adaptation by means of system virtualization and assume that every involved embedded system is running on top of a real-time-capable virtualization platform. We reserve dedicated partitions for components that may be replaced while other partitions are reserved for static constituents of the embedded system. Due to the isolation mechanisms enforced by the virtualization platform a potentially malign component that has been dynamically loaded cannot harm components running in other partitions. Even any kind of inter-partition communication can be subject to additional checks.

3. Safe CPS Adaptation Based on System Virtualization

We propose to follow real-time system virtualization, as for example realized by our hypervisor Proteus [BK2009]. According to system virtualization, a hypervisor allows the sharing of the underlying hardware among multiple operating systems (OS) executed within isolated virtual machines (VM). Multiple existing software artifacts are combined to a system of systems. Such a virtualization architecture is well-suited for CPSs, primarily due to capabilities such as integration of legacy code, scalability, transparent use of multiple processor system-on-chips, cross-platform portability/migration, and isolation of applications, especially for open systems, in which subsystems may be added or removed at runtime. The coexistence of mixed criticality levels (e.g. safety-critical, mission-critical and subsystems of minor importance) has been identified as one of the core foundational concepts for CPSs [Ba2010] and system virtualization provides a natural way to support mixed criticality by consolidating systems of different criticality levels in separated VMs.

CPSs adjust their goals and behavior at runtime according to changes of the environment or corrections received from a higher level, resulting in varying resource usage patters, and therefore require a dynamic resource management. This work focuses on safe resource partitioning and exploits system virtualization for a dynamic reallocation of resources.In previous work, the Flexible Resource Manager (FRM) has been developed in our group [Ob2010] and was recently adapted to system virtualization [Gr2012]. The FRM approach assumes that components are available in various profiles, which are functionally more or less equivalent. However, they vary substantially concerning nonfunctional properties. Profiles represent implementation alternatives or service levels with differing resource requirements, as they exist for example in case of optimization applications (relax optimality for lower resource utilization) or control applications with variable frequency. Task profiles define

minimum and maximum resource requirements: a task can only allocate resources in this range. VM profiles unite the active profiles of the VM's tasks to specify the allocation limits of a VM. Considered resources are computation time, memory, and access to I/O devices. The FRM is in charge of switching between the profiles at runtime.

System virtualization implies resource management decisions on two levels. The hypervisor retains the ultimate control of the hardware resources and assigns resource shares to the VMs. The OSs on the second level assign the obtained resources to their tasks. A FRM component is therefore added to both the hypervisor and the OS. The OS-FRMs inform the Hypervisor-FRM about the dynamic resource requirements and current resource utilization. The Hypervisor-FRM's resource allocation among the VMs is based on this information. The Hypervisor-FRM informs the OS-FRMs about the assigned resources, which facilitates each OS-FRM to manage its resource share. This results in a partitioned approach with decisions on both levels (hierarchical FRM) and communication in both directions.

The possibility to switch between profiles on task level and on VM level enables a dynamic resource assignment across VM borders. In particular, reserved but temporarily unused resources can be assigned to other tasks, even to tasks of other VMs. If resources were reallocated from a task to another task and the resource-lending task at a later point in time needs more resources than left, a resource conflict occurs and has to be solved under real-time constraints. To achieve this, an acceptance test precedes each profile switch and a resource reallocation is accepted if and only if a feasible reconfiguration is identified. This special reconfiguration is a set of profile switches that activate a fallback configuration, which fulfills the worst-case requirements of all tasks. It is not sufficient to identify such a reconfiguration plan; the acceptance test checks whether the time required executing it does not lead to a timing requirement violation.

4. Results

The approach is currently integrated into our real-time multi-core hypervisor Proteus and our real-time operating system ORCOS for 32-bit multi-core PowerPC 405 platforms. A low memory footprint and a high configurability characterize Proteus. A configuration with the base functionality requires 11 kilobytes and a configuration with all functional features requires 15 kilobytes. The interrupt latencies and the execution times for synchronization primitives, hypercall handlers, emulation routines, and virtual machine context switch are all in the range of hundreds of processor cycles. The detailed WCET analysis of all routines makes it possible to determine the WCET of a hosted application. Executed with a clock speed of 300 MHz, a virtual machine context switch takes 1.3 µs. Virtualization increases the interrupt latency. The additional latency is 0.5 µs for a programmable timer interrupt and 0.3 µs for a system call interrupt.

Proteus supports both full virtualization and paravirtualization, without relying on special hardware support for virtualization. According to paravirtualization, modified OSs that are able to communicate with the hypervisor are hosted. The implementation of the hierarchical FRM requires paravirtualization, since the OS-FRMs have to pass information to the hypervisor. The requirement to modify the OS is outweighed by the advantages in terms of efficiency, run-time flexibility, and cooperation of hypervisor

and OS. A specific advantage of paravirtualization for real-time systems is the possibility to apply dynamic real-time scheduling algorithms. The emulation of privileged instructions is the core functionality of the hypervisor and is requested via interrupt (full virtualization) or hypercall (paravirtualization). Paravirtualization speeds up the execution in average by 39%. A library provides the paravirtualization communication primitives in order to reduce the modification effort. If legal or technical issues preclude the modification of an OS, it can be hosted fully virtualized with a fixed resource allocation.

5. Summary

System virtualization may serve as an appropriate means to provide for safe self-evolving software in the context of CPS. Based on our virtualization platform Proteus we are building such a system that offers the necessary services with little overhead and an affordable footprint. We propose a resource management scheme for system virtualization that overcomes the limitations of static resource allocation. CPSs adjust their goals and behavior at runtime according to changes of the environment and are open systems, resulting in varying resource requirements, and therefore demand an adaptive resource management. The two-level architecture offers flexibility across VM borders and adds runtime adaptability with maintained VM isolation. Main feature is a mode change protocol with real-time conflict resolution. Paravirtualization is exploited in order to reduce the performance overhead and to enable a cooperation of hypervisor and OS. To address the limited applicability of paravirtualization, Proteus is designed for the co-hosting of paravirtualized and fully virtualized OSs.

References

[MS2004] Maximilien, E. M. and Singh, M. P. (2004) “A Framework and Ontology for Dynamic Web Services Selection”. In: IEEE Internet Computing, 8(5), pp. 84-93.

[Ca2012] Calinescu, R. et al. (2012) “Self-Adaptive Software Needs Quantitative Verification at Runtime”. In: Communications of the ACM, 55(9), pp. 69-77.

[Ma2009] Mabrouk, N. B. et al. (2009) “QoS-aware Service Composition in Dynamic Service Oriented Environments”. In: International Conference on Middleware.

[Gr2012] Groesbrink, S. et al. (2012) “Towards Adaptive Resource Management for Virtualized Real-Time Systems”. In: Workshop on Adaptive and Reconfigurable Embedded Systems (CPSWeek).

[Ra2012] Rammig, F. (2012) “Self-Coordination as Fundamental Concept for Cyber Physical Systems”. In: 2nd SBC Workshop on Autonomic Distributed Systems - WoSiDA, Ouro Preto, Brazil, pp. 45-48.

[Ob2010] Oberthür, S. et al. (2010) “Flexible Resource Management for Self-X Systems: An Evaluation”. In: Workshop on Self-Organizing Real-Time Systems.

[Ba2010] Baruah, S. et al. (2010) “Towards the Design of Certifiable Mixed-Criticality Systems”. In: Real-Time Technology and Applications Symposium.

[BK2009] Baldin, D. and Kerstan, T. (2009) „Proteus, a Hybrid Virtualization Platform for Embedded Systems“. In: International Embedded Systems Symposium.