A Comparison of Partitioning Operating Systems for Integrated Systems

Bernhard Leiner[1], Martin Schlager[1],
Roman Obermaisser[2], and Bernhard Huber[2]

[1] TTTech Computertechnik AG
Schoenbrunner Strasse 7, 1040 Vienna, Austria
phone: +43 1 5853434 0, fax: +43 1 5853434 90
{bernhard.leiner,martin.schlager}@tttech.com
[2] Vienna University of Technology
Treitlstrasse 3, 1040 Vienna, Austria
{ro,huberb}@vmars.tuwien.ac.at

Abstract. In present-day electronic systems, application subsystems from different vendors and with different criticality levels are integrated within the same hardware. Hence, encapsulation of these subsystems is required in the temporal as well as in the spatial domain. Partitioning Operating Systems (OSs) are employed to allow shared access of applications to critical resources within an integrated system.

In this paper we will discuss fundamental properties of partitioning OSs and compare features of existing solutions. Thereby, we will investigate on LynxOS which is a partitioning OS according to ARINC653, on Tresos, a partitioning OS in accordance with AUTomotive Open System ARchitecture (AUTOSAR), as well as on two prototypical partitioning OS realizations that have been implemented within the Dependable Embedded COmponents and Systems (DECOS) project, an integrated project within the Sixth Framework Programme of the European Commission.

Keyworks: Embedded Systems, Dependability, Partitioning OS.

1 Introduction

Dramatic advances within the last years have paved the way for the integration of application subsystems by different vendors into a single coherent embedded system architecture. Thereby, important initiatives (e. g., AUTOSAR [1], Integrated Modular Avionics (IMA) [2], DECOS [3]) in the automotive, avionic and related domains have been concerned with a systematic, domain oriented process to bundle different application subsystems within the same hardware. These approaches target at increased interoperability, a reduction of the number of Electronic Control Units (ECUs), cables and connectors, and an increase in reliability of the overall system.

A fundamental pre-requisite for the integration of different application subsystems is given by a reliable protection mechanism that partitions a system into execution spaces that prohibit unintended interference of different application

F. Saglietti and N. Oster (Eds.): SAFECOMP 2007, LNCS 4680, pp. 342–355, 2007.

subsystems. Reliable protection in both the spatial and the temporal domains is particularly relevant for systems where the co-existence of safety-critical and non safety-critical application subsystems shall be supported. Partitioning on node level enforces fault containment and thereby enables simplified replacement/update and increased reuse of SW components. A major commercial benefit of partitioning comes with significantly reduced certification effort for mixed criticality systems.

This paper investigates on different existing OSs that are designed to support the partitioning of hardware resources in order to enable the integration of different application subsystems. Such *partitioning OSs* can be found in the automotive (e. g., Tresos according to AUTOSAR), the avionic (e. g., LynxOS according to ARINC653 for IMA systems), or in cross-industry approaches (e. g., DECOS Encapsulated Execution Environment (EEE) [4], DECOS partitioning OS-based on Real-Time Application Interface (RTAI) [5]).

In this paper we identify fundamental features of partitioning OSs and compare these features based on the existing partitioning OSs: Tresos, LynxOS, DECOS EEE, and the RTAI based DECOS partitioning OS.

The remainder of the paper is structured as follows: Subsequent to this introduction, section 2 describes the concepts of an integrated architecture as tackled by AUTOSAR, IMA, and DECOS. Section 3 outlines the fundamentals of partitioning OSs that enable spatial and temporal partitioning. Section 4 outlines properties of TRESOS, LynxOS, and the DECOS OSs, whereas section 5 compares the features of these solutions. Section 6 concludes this paper.

2 Integrated Architecture

Characteristic of an integrated architecture is the sharing of computational resources (e. g., CPU time, memory) and communication resources (i. e., network bandwidth) among multiple software components. This strategy leads to a reduction of the number of deployed node computers and avoids unnecessary resource duplication. In the following we discuss our system model of an integrated architecture. In particular, we describe the model of an integrated node computer, which exploits a partitioning operating system to establish an execution environment for multiple software components.

2.1 System Model

Today large distributed computer systems are typically constructed out of node computers (e. g., denoted as ECUs in the automotive domain). However, there is a shift towards a component-based integration at a finer level of granularity. In integrated architectures, such as DECOS or AUTOSAR, vendors supply software components instead of node computers. In an additional step these software components are then allocated to the ECUs of the target platform [6,7]. This integration of software components on an integrated node is depicted in Figure 1 and will replace the "1 Function – 1 ECU" methodology of today's federated architectures.

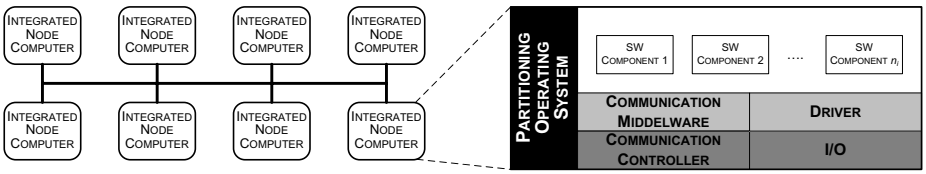

Fig. 1. Integrated Architecture

In order to provide an execution environment that allows the execution of software components without unintended interference, temporal and spatial partitioning for both computational and communication resources is required. For both communication and computational resources, one can distinguish two types of partitioning [8]:

- **Spatial Partitioning.** Spatial partitioning ensures that one software component cannot alter the code or private data of another software component. Spatial partitioning also prevents a software component from interfering with control of external devices (e. g., actuators) of other software components.
- **Temporal Partitioning.** Temporal partitioning ensures that a software component cannot affect the ability of other software components to access shared resources, such as the common network or a shared CPU. This includes the temporal behavior of the services provided by resources (latency, jitter, duration of availability during a scheduled access).

While partitioning of communication resources in an integrated architecture has been addressed in [9], this paper focuses on the partitioning of computational resources.

2.2 Model of an Integrated Node Computer

An integrated node computer provides an execution environment for multiple collocated software components of one or more application subsystems as shown in Figure 1. The model of an integrated node computer comprises:

- *Software components:* The software components implement the application functionality. A software component is part of an application subsystem and represents the unit of distribution. Each software component is the responsibility of a single organizational entity (e. g., a specific supplier). The interaction with other software components occurs through the communication services provided by the communication middleware.
- *Partitioning operating system:* The purpose of the partitioning operating system is the establishment of multiple encapsulated execution environments for combining multiple software components within a single node computer. The encapsulated execution environment provided for a software component is denoted as a *partition* and provides guaranteed computational resources

(CPU time, memory). The partitioning operating system implements mechanisms for spatial and temporal partitioning in order to protect the computational resources of the individual partitions. The scheduling of partitions needs to ensure that a timing failure of a software component, such as a worst-case execution time violation, does not affect the CPU time available to other partitions. In analogy, the spatial partitioning mechanisms of the partitioning operating system include memory protection between partitions (e. g., hardware-enforced with a Memory Management Unit (MMU)). Thereby, each partition emulates a virtual node computer that is dedicated to a single software component only.

- *Communication middleware:* The main purpose of the middleware is the management of the communication resources as previously described. The middleware provides a technology invariant interface to the software components that abstracts from any hardware-specific implementation details. For example, in AUTOSAR [7] the runtime environment (RTE) provides such a generic communication service for the applications. In DECOS the high-level virtual network services perform this task [10].
- *Communication controller:* The purpose of the communication controller is to provide access to the underlying communication system. By the use of hardware drivers and the provision of standardized Application Programming Interface (API) one typically abstracts from the used hardware and thus ensures reuse of existing code in future systems.
- *Input/Output (I/O) and drivers:* The software components hosted on a node computer exploits the input/output subsystem for interacting with the controlled object and the human operator. This interaction occurs either via a direct connection to sensors and actuators or via a fieldbus (e. g., Local Interconnect Network (LIN) [11]). The latter approach simplifies the installation – both from a logical and a physical point of view – at the expense of increased latency of sensory information and actuator control values.

3 Partitioning OS Fundamentals

Integrated node computers as those introduced in Section 2 raise the demand for an OS which is in charge of managing the available resources. Besides the usual feature set of an OS like process scheduling, memory management, inter-process communication and Input/Output, support for *partitioning* in the temporal and the spatial domains as mentioned in Section 2 is important for encapsulation as the basis for composability. A more detailed discussion is included in [12].

The goal of a partitioning OS as depicted in Figure 2 is to provide fault containment equivalent to an idealized system in which each partition is allocated an independent processor and associated peripherals, and all inter-partition communications are carried out on dedicated lines [8]. As mentioned above, partitioning can be splitted into *spatial* and *temporal* partitioning. Both of this partitioning dimensions have an influence on the implementation of the basic OS features listed below as well as on the general OS API. For instance, no system call shall corrupt other partitions or the OS itself.

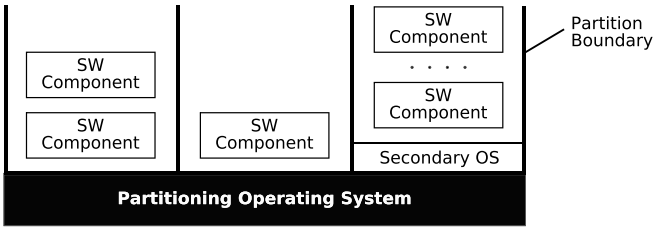

Fig. 2. Concept of a partitioning OS

In the following, we will investigate on the fundamental properties of an partitioning OS, namely scheduling, memory management, and communication.

3.1 Scheduling

Scheduling is concerned with the allocation of resources to software components including the instants of invocation, the assignment of memory regions and the right to access I/O. We can distinguish between static (i. e., offline) and dynamic (i. e., online) scheduling approaches. In [13] the following four classes of scheduling paradigms in the area of real-time systems are outlined:

Static table-driven scheduling: The resource allocation is based on a static schedulability analysis. At runtime, the invocation of tasks is triggered according to a pre-defined schedule, which is usually called table.

Static priority driven preemptive scheduling: A static schedulability analysis is performed but, in contrast to static table-driven scheduling, software components are scheduled at runtime according to a "highest priority first" strategy.

Dynamic planning-based scheduling: With dynamic planning-based scheduling, resource allocation to a software component is decided dynamically based on a feasibility check (that is also performed at runtime).

Dynamic best effort scheduling: The focus of dynamic best effort scheduling is to provide an efficient allocation of given resources to software components. No feasibility checks are performed. Hence, no guarantees with respect to the real-time behavior can be given and tasks may be aborted during their execution.

A partitioning OS typically supports a static table-driven scheduling approach that is very well suited for safety-critical, hard real-time systems since its static nature makes it possible to check the feasibility of the schedule in advance. Furthermore, the maximum time between two partition activations is known in advance.

In addition to partitioning OS services, it should be possible to host several software components within a single partition. In this case, the partitioning OS schedules the partitions and a *secondary OS* schedules the software components

within a partition (*multi-level scheduling*). This secondary OS can implement a simple table-driven scheduling or even a full featured OS or virtual machine.

3.2 Memory Management

Memory management deals with the allocation of memory to a partition (and to the software components residing in that partition). Hence, the OS shall ensure that no undesired interference between any two partitions might occur. Therefore, it must be avoided that a software component is able to write into or execute code from a different partition if not explicitly granted to do so. Spatial partitioning is only possible if the processor provides hardware support for memory protection, i. e., a dedicated protection unit that assigns memory access rights to a certain partition and that avoids errors by strictly blocking illegal requests of faulty partitions.

In existing implementations, two approaches can be found with respect to memory protection that depend on the available hardware support.

Virtual Address Space: Memory protection in modern OSs is typically organized by providing a virtual address space to each process (e. g., Windows, Linux). Virtual addresses are translated to physical addresses by an MMU. In case a virtual address cannot be translated into a physical address (due to an illegal request from a software component), a protection trap is raised.

Protection Blocks: When using protection blocks, multiple memory areas are assigned to software components with different access rights, e. g., read, write, execute. This approach is comparably simpler and thus favored for low-end embedded devices that offer a Memory Protection Unit (MPU).

3.3 Interaction

A partitioning OS must support the interaction of a software component with other software components and with its physical environment. We distinguish between: (a) communication that takes place between software components on the same physical node (i. e., *intra-node communication*), (b) communication between software components that are located on different nodes (i. e., *inter-node communication*), and (c) interaction of a software component with its physical environment (i. e., *I/O interaction*).

Intra-Node Communication. The interaction between software components that are located in different partitions on the same physical hardware node must be supported by a partitioning OS. The most simple mechanism for intra-node communication is to provide shared memory regions that can be accessed by more than one partition. More sophisticated approaches provide message channels / queues for intra-node communication.

Inter-Node Communication. Inter-node communication is typically supported by a middleware layer as discussed in Section 2 that provides a message-based interface to a software component. The inter-node communication must be

supported by a partitioning OS in the sense that the realization of a middleware layer is enabled that provides a message-based communication interface to the partitions and that accesses and instruments the communication controller of a node.

I/O Interaction. I/O interaction of a software component with its physical environment takes place either across a standardized fieldbus (e. g., LIN [11], Controller Area Network (CAN) [14]) or via direct I/O. Thus, I/O interaction is concerned with the protection of I/O hardware of a given microcontroller (e. g., a particular I/O block). In case a transducer or a fieldbus is instrumented only from a single partition, it is sufficient to grant this partition access to the transducer/fieldbus. If more than one partition needs to access I/O, typically a shared partition is implemented that offers I/O services to other partitions by inter and intra-node communication.

It should be mentioned that if a memory-mapped device is employed, controlled I/O interaction must be supported by the memory management mechanism as previously discussed. This means that a partition is granted access to the memory region to which the memory-mapped device is linked.

4 Overview of Partitioning OSs

In this section we discuss the capabilities of partitioning OSs that have been selected from the avionics (i. e., LynxOS), automotive (i. e., Tresos), and cross-industry domains (i. e., DECOS OSs). Our survey is based on data available from the respective vendors/research groups. No actual verification of the stated properties has been carried out for this survey.

4.1 Encapsulated Execution Environment – DECOS Core OS

The Encapsulated Execution Environment (EEE) was developed by TTTech within the DECOS project and consists of the DECOS Core Operating System (COS) as well as a set of graphical configuration tools to generate configuration files.

The COS has been written from scratch with strict temporal and spatial partitioning of all system resources in mind. Using a static configuration is the fundamental mechanism to implement the partitioning functionality. Resource ownership and scheduling is defined statically and can be checked for feasibility in advance. Besides partitioning mechanisms, the COS provides a system interface for intra-node communication as well as error handling and health checks.

Scheduling. The COS uses a two-level scheduling hierarchy. The top-level schedule table divides the schedule cycle into multiple time slots, each of them assigned to a single partition. A second schedule table is used to trigger events and their corresponding event handler function within those partition intervals. Both of these schedule tables are generated during system configuration. The

table responsible for scheduling the partitions is fixed whereas the secondary table can be reconfigured during runtime. The COS ensures that SW components can only reconfigure events belonging to their partition as well as that the new scheduled time for an event lies within a partition interval of the same partition.

For event handlers within partitions fixed priority-based preemptive scheduling is used. This is also necessary for hardware interrupts. An interrupt belonging to a partition is only enabled if this partition is currently scheduled. If this interrupt is triggered, it is mapped to a high-priority partition event with the ISR as event handler.

Temporal partitioning among different partitions is guaranteed since at the end of a partition time slot, the COS preempts all running event handlers, disables all interrupt sources belonging to this partition and finally sets up the environment for the next partition.

Memory Management. The memory protection mechanisms of the COS depends on an MMU that allows the concurrent assignment of permissions to at least five different memory regions (protection blocks). Each partition has execute rights for all its private code as well as for a memory region containing the system interface functions and libraries. Additionally, each partition has read and write permissions for its private data and a dedicated part of a node-wide shared memory. The last protection block is used to grant read access to the whole shared memory.

Interaction. Intra-node communication can be done by using the *message channel* service provided by the COS which can either be *sampled* or *queued* and support an arbitrary number of producers and consumers. A simpler way of intra-node communication can be achieved by using the node-wide shared memory. In the DECOS project this memory is also used for private inter-node communication lines (memory mapped I/O).

Another way to provide private access to I/O devices is to deny direct access to hardware for usual partitions[1]. A dedicated I/O partition with the necessary access rights is used which provides an API for the different hardware components.

4.2 RTAI-Based Partitioning OS

In the course of the DECOS project, a partitioning operating system has been developed, which is based on the real-time Linux variant RTAI and which exploits the Linux Real-Time (LXRT) extension of RTAI for realizing spatial and temporal partitioning. LXRT is an extension of RTAI that enables the development of hard real-time programs running in user space by utilizing the real-time scheduler provided by RTAI. The RTAI real-time scheduler executes the Linux kernel as an idle task, i. e., non real-time Linux applications are only executed when no RTAI/LXRT tasks are active.

[1] Only possible if the target platform provides support for such a restricted run level.

The central element of the partitioning OS is a time-triggered dispatcher, an RTAI kernel module that is responsible for the allocation of processor time to the individual partitions. The partitions are implemented as LXRT tasks, thus executed in user space while preserving the temporal benefits of RTAI.

Scheduling. The activation of partitions is performed by the time-triggered dispatcher that extends the built-in functionality of RTAI/LXRT operating system. The dispatching points are derived from a static dispatching table which is created during system configuration. The dispatching table consists of the activation of a particular partition along with its maximum time granted for execution.

For providing temporal partitioning, individual partitions must not be able to exceed their maximum assigned processor time as defined in the dispatching table, even in the case of a software fault within a partition. For this purpose, deadline monitoring is done by the time-triggered dispatcher: The dispatcher is periodically executed according to the dispatching table. Due to its realization as an RTAI kernel module with the system-wide highest priority, an eventually active partition would be preempted by the dispatcher. The dispatcher analyzes the processor state of the previously executed partition right after its activation and, eventually, removes the partition from the *ready-queue* of the RTAI scheduler. Due to this enforced preemption by RTAI, the partitions are not required to act cooperatively and release the processor on their own.

Memory Management. Memory protection of the RTAI-based partitioning operating system relies on the MMU functionality provided by the processor on which it is executed. Since a processor that is equipped with an MMU basically distinguishes between supervisor mode and user mode, where memory access is only protected in the latter one, spatial partitioning by exploiting the functionality of the MMU can only be provided for applications running in user mode.

Like in many real-time Linux variants, applications using the RTAI API are implemented as Linux kernel modules. Thus, they are executed in supervisor mode and could circumvent memory protection. However, due to the realization of the partitions as LXRT tasks which are executed in user mode, the memory protection mechanisms of Linux are preserved and thus spatial partitioning between individual partitions is provided.

Interaction. According to the system model of DECOS, a software component, denoted in DECOS as *job* [3], is the basic unit of work that is distributed among the nodes of a DECOS cluster. Usually, a mapping of one job per partition is established. Communication between jobs is realized via virtual network services [10]. Thus, from the point of view of the individual jobs it is transparent whether an interaction occurs via intra-node or inter-node communication.

The virtual network service is provided by the virtual network middleware, which is implemented in a dedicated partition on each DECOS node. The interaction of jobs with the virtual network middleware occurs via shared memory,

denoted as *ports*. The memory layout and access rights of these ports is determined during system configuration. The code sequences for allocating and initializing the corresponding memory areas are then statically linked to the application code in order to prevent an application developer from allocating forbidden areas. The same strategy is followed for protecting I/O regions, which are accessed by the use of *memory mapped I/O*, i. e., by mapping the physical address of the I/O in the virtual address space of a particular partition.

4.3 AUTOSAR OS – Tresos

The AUTomotive Open System ARchitecture (AUTOSAR) standard contains the operating system specification AUTOSAR OS which defines the main features of an AUTOSAR-compliant OS: real-time performance, static configuration combined with a priority-based scheduling strategy and protective functions for memory and timing during runtime. A further requirement is the capability to be hostable on low-end microcontrollers. In order to support this, the AUTOSAR OS specification introduces four scalability classes, each of them with different mandatory features. Those scalability classes also affect partitioning. E.g., memory protection is only required for class three and four, timing protection for class two and four. The example OS for this section is Tresos, developed by Elektrobit, which supports scalability classes one to four.

The AUTOSAR standard does not know the concept of partitions. Instead, multiple *OS-Applications*, which form a cohesive functional unit composed of tasks, Interrupt Service Routines (ISRs) and other resources, are used. OS-Applications can either be *trusted* or *non-trusted*. Since trusted OS-Applications are allowed to run with monitoring and protection features disabled at runtime, a non-trusted OS-Application is the best fit to the partition notion like defined in Section 3.

Scheduling. The AUTOSAR OS standard is based on OSEK/VDX which is widely used in the automotive industry. In order to be OSEK-compatible, AUTOSAR uses the same fixed priority-based preemptive scheduling strategy. The scheduling is event-triggered and a high-priority event is always able to reserve the CPU which prevents strict temporal partitioning. To support temporal partitioning to a certain extent, two mechanisms are available in AUTOSAR: (1) *Schedule tables* which allow defining a statically, periodic activation of events and (2) *time monitoring* which is used to limit the maximum execution time of tasks/ISRs, the maximum time they are allowed to hold a shared resource/disable interrupts and the arrival rate of tasks/interrupts.

Memory Management. The basic memory protection requirement that shall be fulfilled by the OS is to protect the data, code and stack section of tasks within an OS-Application from other non-trusted OS-Applications. Additionally, it should provide protection for private data and stack for tasks within the same OS-Application. This requires hardware support in form of an MMU or an MPU. Since this is highly platform specific, the AUTOSAR OS standard does not define

implementation details. Tresos uses static allocation of memory to tasks and OS-Applications in combination with an MPU to achieve spatial partitioning of memory.

Interaction. All AUTOSAR software components run on top of the AUTOSAR Runtime Environment (RTE), which acts as a communication abstraction layer. The same interface in form of ports is provided whether intra-node or inter-node information channels are used. The current version of the AUTOSAR RTE specification (Release 2.0) does *not* support memory protection mechanisms even if provided by the OS.

4.4 ARINC653-Compliant Partitioning OS – LynxOS-178

The operating system LynxOS-178 is a real-time operating system that has been developed by LynuxWorks for safety-critical avionic applications based on Integrated Modular Avionics (IMA) [2]. LynxOS-178 adheres to the ARINC standard 653 [15], which is known as APplication EXecutive (APEX) and defines the services of the avionic software environment. APEX provides services for partition management, process management, time management, memory management, interpartition communication, intrapartition communication, and diagnosis. In addition, LynxOS-178 distinguishes between a small partitioning kernel, which establishes the encapsulated partitions, and higher software layers (e.g., for POSIX support) that run within the partitions. LynxOS-178 supports certification to the highest criticality levels, namely DO-178B level A [16]. LynxOS-178 has already been deployed in safety-critical avionic military and aerospace systems.

Scheduling. For the scheduling of partitions, LynxOS-178 uses fixed cyclic scheduling. Each partition is statically assigned CPU time via a periodically recurring time slice. Thereby, interference between partitions is prevented in the temporal domain. Within a partition, on the other hand, LynxOS-178 offers a process-based execution environment with priority-based preemptive scheduling, priority inheritance, and priority ceilings according to the POSIX model.

Memory Management. In analogy to the allocation of the CPU time, LynxOS-178 statically performs the allocation of memory to the partitions. The memory allocation of a partition is fixed at design time and the configured memory size cannot be changed at runtime. An MMU is employed for isolating the partitions from each other. In contrast to the memory allocation at the partition level, dynamic memory management is supported within a partition. Therefore, LynxOS-178 offers an API with POSIX-compliant calls. The software layer for establishing this POSIX interface is not part of the LynxOS-178 partitioning kernel, but executed in the partitions.

Interaction. For the interaction between partitions, ARINC653 specifies communication channels that are accessible via two types of ports: sampling and

queuing ports. At *sampling ports*, successive messages contain identical but updated data. Received messages overwrite old information, thus requiring no message queuing. In *queueing ports*, messages are assumed to contain uniquely different data. Messages are buffered in queues, which are managed on a first-in/first-out (FIFO) basis.

Inner-partition communication services (e. g., message queues, black boards, semaphores, and events) are not part of the LynxOS-178 partitioning kernel, but can be provided by software layers within the partitions.

5 Feature Comparison

Table 1 gives a brief overview of the features of the four partitioning OSs discussed in this paper. LynxOS-178 has the highest maturity level and provides the

Table 1. Feature Comparison Overview

	DECOS COS	DECOS RTAI	Tresos	LynxOS-178
Vendor	TTTech	Vienna University of Technology	Electrobit	Lynuxworks
Maturity	prototype	prototype	commercial	certified
Standard	—	—	AUTOSAR	ARINC653
Footprint	$\leq$ 1 MB	1 – 10 MB (incl. Linux Kernel)	$\leq$ 1 MB	$\geq$ 100 MB (incl. secondary OSs)

Temporal Partitioning	
DECOS COS	Temporal partitioning ensured by static cyclic scheduling of partition time slots. Deadline monitoring to detect faulty SW components within a partition.
DECOS RTAI	Temporal partitioning ensured by static cyclic scheduling of partition time slots. Deadline monitoring to detect faulty SW components within a partition.
Tresos	No strict partitioning due to preemtive scheduling. Time monitoring to detect faulty SW components.
LynxOS-178	Temporal partitioning ensured by static cyclic scheduling of partition time slots.

Spatial Partitioning	
DECOS COS	Private memory statically assigned and protected by MPU. Private communication lines for I/O, inter-partition communication and cluster-wide communication.
DECOS RTAI	Private data protected by MMU. Private communication and I/O by mapping memory into the virtual address room of partitions.
Tresos	Private memory statically assigned and protected by MPU. Communication middleware (RTE) does not support memory protection.
LynxOS-178	Private data protected by MMU.

most features at the cost of higher hardware requirements. Tresos concentrates on the compliance with the AUTOSAR OS specification, which focuses more on backward compatibility and sophisticated communication middleware than on partitioning. Both OSs for the DECOS project have been developed with partitioning as core feature. The DECOS COS is a small OS written for low-end embedded hardware whereas DECOS RTAI profits from the many features provided by underlying Linux kernel.

6 Conclusion

The bundling of software components by different vendors and with different levels of criticality on an integrated node computer as currently undertaken in integrated architecture approaches (e. g., AUTOSAR, ARINC, DECOS) requires strict partitioning of these software components at the OS level. A number of partitioning OSs exist that aim at providing encapsulation of software components in the temporal and the spatial domains.

In this paper we presented four partitioning OSs, i. e., DECOS EEE Core OS, RTAI-based partitioning OS, Tresos, and LynxOS, and compared the features of these partitioning OSs. Thereby, we investigated on the temporal and spatial protection mechanisms as well as on the code size and the targeted area of application of these partitioning OSs. It turned out that although all presented partitioning OSs adhere to the same core principles, there are notable differences with respect to maturity, code size, and support for spatial protection of these OSs. In the future we expect significant improvements and further establishement of partitioning OSs on different markets (particularly in the automotive domain).

Acknowledgments

We would like to thank our colleague Bernhard Wenzl and the anonymous reviewers for their comments on earlier versions of this paper. This work has been supported by the European IST project DECOS under contract No. IST-511764.

References

1. GbR, A.U.T.O.S.A.R.: AUTOSAR – Technical Overview V2.0.1 (June 2006)
2. Aeronautical Radio, Inc., 2551 Riva Road, Annapolis, Maryland 21401. ARINC Specification 651: Design Guide for Integrated Modular Avionics (November 1991)
3. Obermaisser, R., Peti, P., Huber, B., Salloum, C.E.: DECOS: An integrated time-triggered architecture. e&i journal (Journal of the Austrian professional institution for electrical and information engineering) 3 (March 2006)
4. Schlager, M., Herzner, W., Wolf, A., Gründonner, O., Rosenblattl, M., Erkinger, E.: Encapsulating application subsystems using the DECOS core OS. In: Górski, J. (ed.) SAFECOMP 2006. LNCS, vol. 4166, pp. 386–397. Springer, Heidelberg (2006)

5. Huber, B., Peti, P., Obermaisser, R., El Salloum, C.: Using RTAI/LXRT for partitioning in a prototype implementation of the DECOS architecture. In: Proc. of the Third Int. Workshop on Intelligent Solutions in Embedded Systems (May 2005)
6. Heinecke, H., Schnelle, K.-P., Fennel, H., Bortolazzi, J., Lundh, L., Leflour, J., Maté, J.-L., Nishikawa, K., Scharnhorst, T.: AUTomotive Open System ARchitecture - An Industry-Wide Initiative to Manage the Complexity of Emerging Automotive E/E-Architectures. In: Proceedings of the Convergence Int. Congress & Exposition On Transportation Electronics, Detroit, MI, USA, October 2004, SAE, 2004-21-0042 (2004)
7. Scharnhorst, T., Heinecke, H., Schnelle, K.-P., Fennel, H., Bortolazzi, J., Lundh, L., Heitkämper, P., Leflour, J., Mate, J.-L., Nishikawa, K.: AUTOSAR – challenges and achievements 2005. In: VDI Berichte 1907, Verein Deutscher Ingenieure (2005)
8. Rushby, J.: Partitioning for avionics architectures: Requirements, mechanisms and assurance. NASA contractor report CR-1999-209347, NASA Langley Research Center (October 1999)
9. Obermaisser, R., Peti, P.: Realization of virtual networks in the DECOS integrated architecture. In: Proc. of the Workshop on Parallel and Distributed Real-Time Systems 2006 (WPDRTS), April 2005, IEEE Computer Society Press, Los Alamitos (2005)
10. Obermaisser, R., Peti, P., Kopetz, H.: Virtual networks in an integrated time-triggered architecture. In: Proc. of the 10th IEEE Int. Workshop on Object-oriented Real-time Dependable Systems (WORDS2005), Sedona, Arizona, February 2005, pp. 241–253. IEEE Computer Society Press, Los Alamitos (2005)
11. LIN Consortium. LIN Specification Package Revision 2.0 (September 2003)
12. Conmy, P.M.: Safety Analysis of Computer Resource Management Software. PhD thesis, University of York (2005)
13. Ramamritham, K., Stankovic, J.A.: Scheduling algorithms and operating systems support for real-time systems. Proceedings of the IEEE, 55–67 (1994)
14. Robert Bosch Gmbh. CAN Specification, Version 2.0. Stuttgart, Germany (1991)
15. Aeronautical Radio, Inc., 2551 Riva Road, Annapolis, Maryland 21401. ARINC Specification 653: Avionics Application Software Standard Interface, Part 1 - Required Services (March 2006)
16. Radio Technical Commission for Aeronautics, Inc (RTCA), Washington, DC. DO-178B: Software Considerations in Airborne Systems and Equipment Certification (December 1992)