Timing

AUTOSAR CP
AUTOSAR AP
POSIX

01 INTRODUCTION

For many years classical AUTOSAR operating systems, based on the OSEK/VDX standard, suited the automotive world quite well.

Initiated in 2016 as an extension to the AUTOSAR CP (“**C**lassic **P**latform”), the AUTOSAR AP (“**A**daptive **P**latform”) targets high-performance ECUs such as autonomous driving controllers [3](2.1).
AUTOSAR AP comes with major impacts on the way software is developed – including all timing aspects.

This poster sheds some light on how timing is addressed in the existing AUTOSAR CP world, the new AUTOSAR AP world and beyond.

02 AUTOSAR CP

2.1 Architecture
When AUTOSAR was introduced in 2003, it defined the methodology for how to develop a system top-down. See the left side of the central diagram which deals with AUTOSAR CP.

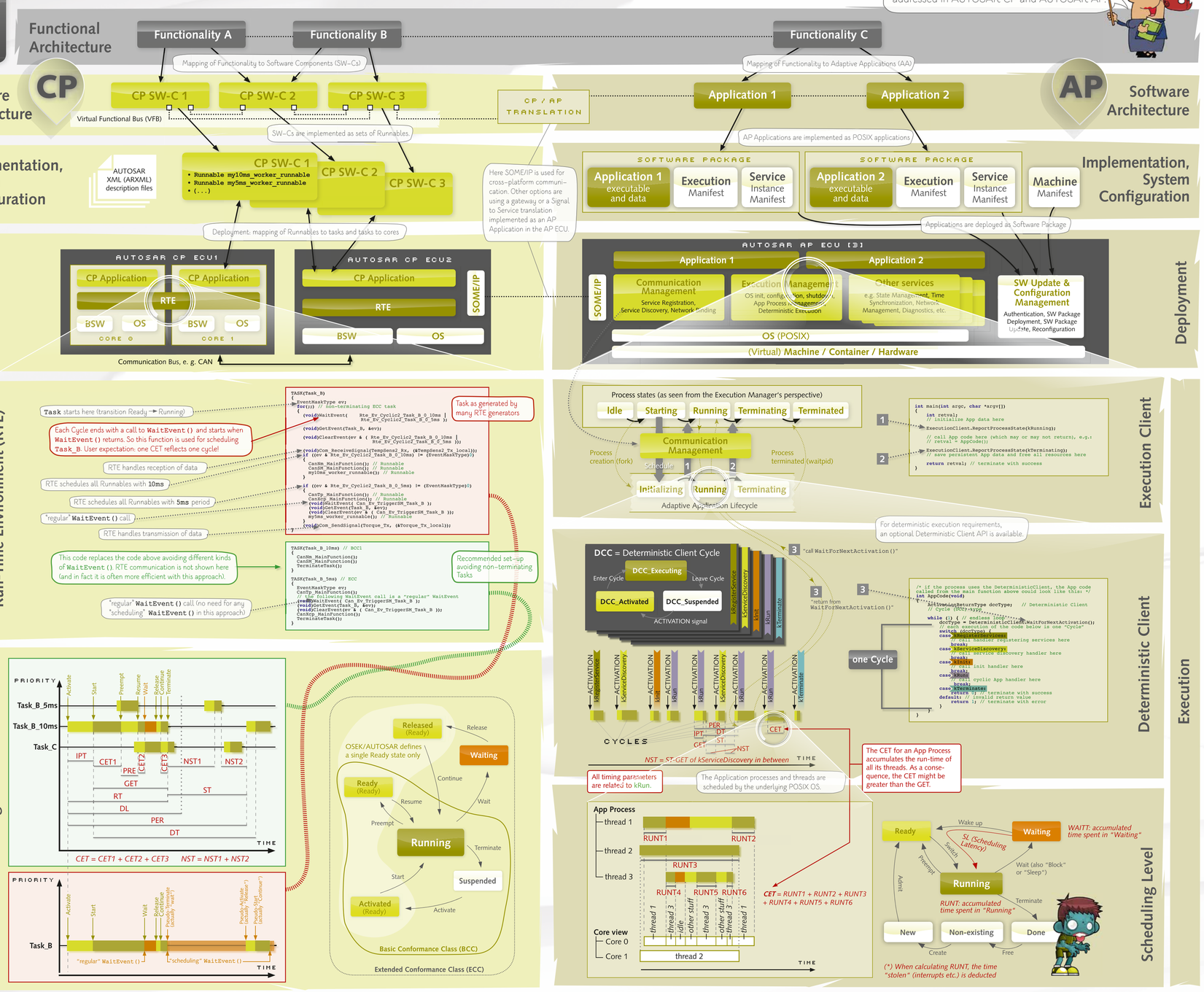
2.2 Scheduling and Execution Management
In a nutshell: OSEK defines tasks as containers for code to be executed. Tasks have fixed priorities – leaving aside temporary priority modification through the priority ceiling protocol – and their attributes are defined at compile-time. Multicore Partitioned Scheduling is also statically configured off-board.

Typically, the RTE generator creates tasks like the non-terminating Extended Conformance Class (ECC) TASK_B shown on the right. However, in many cases it would be better to create separate terminating Basic Conformance Class (BCC) tasks to increase efficiency and reduce complexity [8].

2.3 Timing Parameters
AUTOSAR CP release 4.1.3 introduced the definition of timing parameters. The figure on the right shows all timing parameters in red color.

03 GLOSSARY

AA Adaptive Application
Abstract System Description Description of features on an abstract level without considering the actual mapping to hardware, software and physical communication interfaces
AMP Asymmetric Multiprocessing implies that different cores are assigned different roles
AP Machine A (virtualized) ECU onto which software can be deployed
ARXML AUTOSAR XML, including system, software components and communication configuration
Common Software Architecture overview of all software entities and their communication relation independent of the mapping to AUTOSAR CP and AUTOSAR AP
ECU Electronic Control Unit, an embedded system that controls motor vehicle subsystems
MMU Memory Management Unit, a hardware component necessary to implement virtual memory
Multicore Partitioned Scheduling Scheduling algorithm in which tasks are statically partitioned between cores. Often chosen for AMP
Multicore Global Scheduling Scheduling algorithms in which cores are resources dynamically assigned to tasks. Often chosen for SMP
Multiprocessing (OS) OS support for multiple concurrent processes
OSEK/VDX A Standard Automotive Embedded OS reused for the AUTOSAR OS layer
PID Process Identifier - a unique identifier for the process
POSIX Portable Operating System Interface (Unix-like)
Process An address space realized with virtual memory, with one or more threads executing within that address space
RTE AUTOSAR Run-Time Environment
SMP In Symmetric Multiprocessing, all cores are considered equal
SOME/IP Scalable service-Oriented MiddlewarE over IP, an automotive middleware solution for control messages
SW-C Software Component (CP)
Task A run-time software component as viewed by the OS Task Scheduler
Thread A single flow of control within a process. Process memory is accessible to all threads in the same process
VFB Virtual Function Bus (CP)
Virtual Memory An abstraction layer which allows multiprocessing through multiple different views of memory



Timing Parameters

OSEK/AUTOSAR CP AND AUTOSAR AP DETERMINISTIC CLIENT			
ABR.	EXPLANATION	ABR.	EXPLANATION
IPT	*initial pending time*	PER	*period*
CET	*core execution time*	ST	*slack time*
GET	*gross execution time*	PRE	*preemption time (AUTOSAR CP only)*
RT	*response time*	DL	*deadline (“max. RT”)*
DT	*delta time*	NST	*net slack time*

POSIX OS SCHEDULING	
ABR.	EXPLANATION
SL	*scheduling latency*
WAITT	*accumulated time spent in “Waiting”*
RUNT	*accumulated time spent in “Running”*

CPU-utilization (**also referred to as CPU-load**): the sum U of all CETs within a defined time-frame t_O (the “observation frame”) in relation to t_O times the number of relevant cores (n_C).

Such CETs can relate to a specific code-fragment, a runnable, a task, a thread, a process, a core or the complete system.

$$U = \frac{\sum_{n=1}^{N} CET(n)}{n_C \cdot t_O}$$

04 AUTOSAR AP

4.1 Architecture
AUTOSAR AP differs from AUTOSAR CP in that it defines a SOA (Service-Oriented Architecture [14]) for the **A**daptive **A**pplications (AA).

The AUTOSAR **R**untime for **A**daptive Applications (ARA) constitutes the environment in which the applications can run and access a wide variety of high level services, plus low level interfaces to the OS [3](3.1). AAs are developed in C++ [6]. The OS of AUTOSAR AP is **POSIX** as opposed to OSEK/VDX of AUTOSAR CP.

The ARA runs on an AUTOSAR AP machine, which can represent
- a physical machine – including an MMU in order to support virtual memory for multiprocessing [3](4.4)
- a virtual machine – supported by any kind of hypervisor [10]
- a container – a standardized encapsulated software environment [12]

The most important components of the ARA in this context are:
- Execution Management [1]
- Communication Management [2]
- POSIX Operating System [9]

4.2 Organizing Execution
The execution of an AA is organized into multiple layers of abstraction on top of the OS scheduling level. Each layer can be modeled with a dedicated state machine as shown in the AP part of the picture on the left.

The Execution Manager and AA need to synchronize their view of the AA state. The Execution Manager creates the AA process (“Idle” to “Starting”). AA's execution state will start at “Initializing”. After initialization, the AA reports its “Running” execution state.

In the case of self-termination, the AA reports its execution state as “Terminating”. The AA will then clean up, commit to storage, and exit normally from the process. The Execution Manager will wait on the process exit status (“Terminating” to “Terminated”). If the execution management initiates the termination, it sends a POSIX signal to the AA, which will react by running the same procedure.

4.3 Deterministic Client
Beside other features, the Deterministic Client API [1](7.6.2) offers event-driven and time-driven cyclic execution. The picture on the left shows a code example for such a cycle, its state machine and the execution over time.

The timing parameters we propose (CET, DT, ... in red) relate to a specific cycle type, in the picture “kRun”. For each cycle type a separate set of timing parameters is calculated.

4.4 OS Scheduling
AA tasks are processes and threads, scheduled to run on the available cores according to the POSIX scheduling policies, including “FIFO”, “RR”, and “OTHER” policies [9].

In contrast to the Multicore Partitioned Scheduling used in OSEK, POSIX OS implementations generally use Multicore Global Scheduling, dynamically assigning cores to threads at run-time. A useful general model for process and thread states is shown in the scheduling state diagram [13] (3.2, Fig 3.1) at the bottom right in the picture.

When a process is created (“non-existing” to “new”), usually by the “parent”, the OS assigns it a Process Identifier (PID), an address space and the first thread of execution, also called the “main thread”, for which AP standardizes the configuration [1](7.7.3.3) (7.6.3.1).

The new thread is admitted (“new” to “ready”), and can then be picked up for execution via a context-switch (“ready” to “running”). A thread can release the core voluntarily (“running” to “waiting” / blocked / sleeping), or involuntarily (preemption: “running” to “ready”).

The OS wakes up and switches-in waiting threads and preempted threads respectively at the appropriate time (“waiting” to “ready”, “ready” to “running”).

When a thread terminates (“running” to “done” / dead / terminated), the OS will keep the exit status information. When the exit status has been collected, it will be freed and the thread will cease to exist (“done” to “non-existing”).

A “done” process whose parent process does not collect its exit status is called a “Zombie” process, whose long lasting presence should be avoided, as it uses up valuable PIDs.

4.5 Timing Parameters
GLIWA introduced the timing parameters shown in the CP section of the picture on the left several years ago into AUTOSAR CP [15]. With this poster we propose a mapping of the timing parameters to the cycles of the deterministic client as outlined in the picture.

05 POSIX

POSIX [9] is a set of standards for operating systems, which allows applications to be easily moved from OS to OS without requiring changes.

Its scope is mostly the portable interface presented to the application and leaves most of the internal OS implementation out of scope. The POSIX standards include the C language, processes, threads, signals, timers, file system, I/O, IPC, real time extensions and more. POSIX fundamentally requires virtual memory in order to support multiprocessing.

POSIX certifications for operating systems are available [7]. However, many OSes do not go through the process, for practical reasons. A prominent example of such a POSIX-like operating systems is GNU / Linux, although specific variants are certified.

06 REFERENCES

[1] AUTOSAR,
Specification of Execution Management AUTOSAR AP Release 18-10
https://www.autosar.org/standards/adaptive-platform/

[2] AUTOSAR,
Specification of Communication Management AUTOSAR AP Release 18-10
https://www.autosar.org/standards/adaptive-platform/

[3] AUTOSAR,
Specification of Adaptive Platform Design AUTOSAR AP Release 18-10
https://www.autosar.org/standards/adaptive-platform/

[4] AUTOSAR,
Methodology for Adaptive Platform AUTOSAR AP Release 18-10
https://www.autosar.org/standards/adaptive-platform/

[5] AUTOSAR,
Specification of Manifest AUTOSAR AP Release 18-10
https://www.autosar.org/standards/adaptive-platform/

[6] AUTOSAR,
Guidelines for the use of the C++14 language in critical and safety-related systems; AUTOSAR AP Release 18-10
https://www.autosar.org/standards/adaptive-platform/

[7] IEEE, POSIX Certification
http://get.posixcertified.ieee.org

[8] Peter Gliwa, GLIWA, *A systematic approach for timing requirements EMCC (Embedded Multi-Core Conference) 2018 in Munich*
https://gliwa.com/index.php?page=downloads&lang=eng/

[9] The Open GROUP, *POSIX Standard*
https://publications.opengroup.org/standards/unix

[10] WIKIPEDIA, *Hypervisor*
https://en.wikipedia.org/wiki/Hypervisor

[11] WIKIPEDIA, *POSIX*
https://en.wikipedia.org/wiki/POSIX

[12] Red Hat, *Topics, What are Linux containers?*
https://www.redhat.com/en/topics/containers

[13] Kay A. Robbins, Steven Robbins,
UNIX Systems Programming: Communication, Concurrency, and Threads, Prentice Hall, 2003

[14] SOA Manifesto Authors, *SOA Manifesto*
http://www.soa-manifesto.org

[15] AUTOSAR, *Recommended Methods and Practices for Timing Analysis and Design within the AUTOSAR Development Process*
https://www.autosar.org/standards/classic-platform/

T1 TIMING 1. CLASS
T1 – state of the art timing-analysis, stack-analysis, memory-access analysis

GLIWA embedded systems
www.gliwa.com
GLIWA GmbH embedded | Pollinger Str. 1 | D-82362 Weilheim

V. 2019