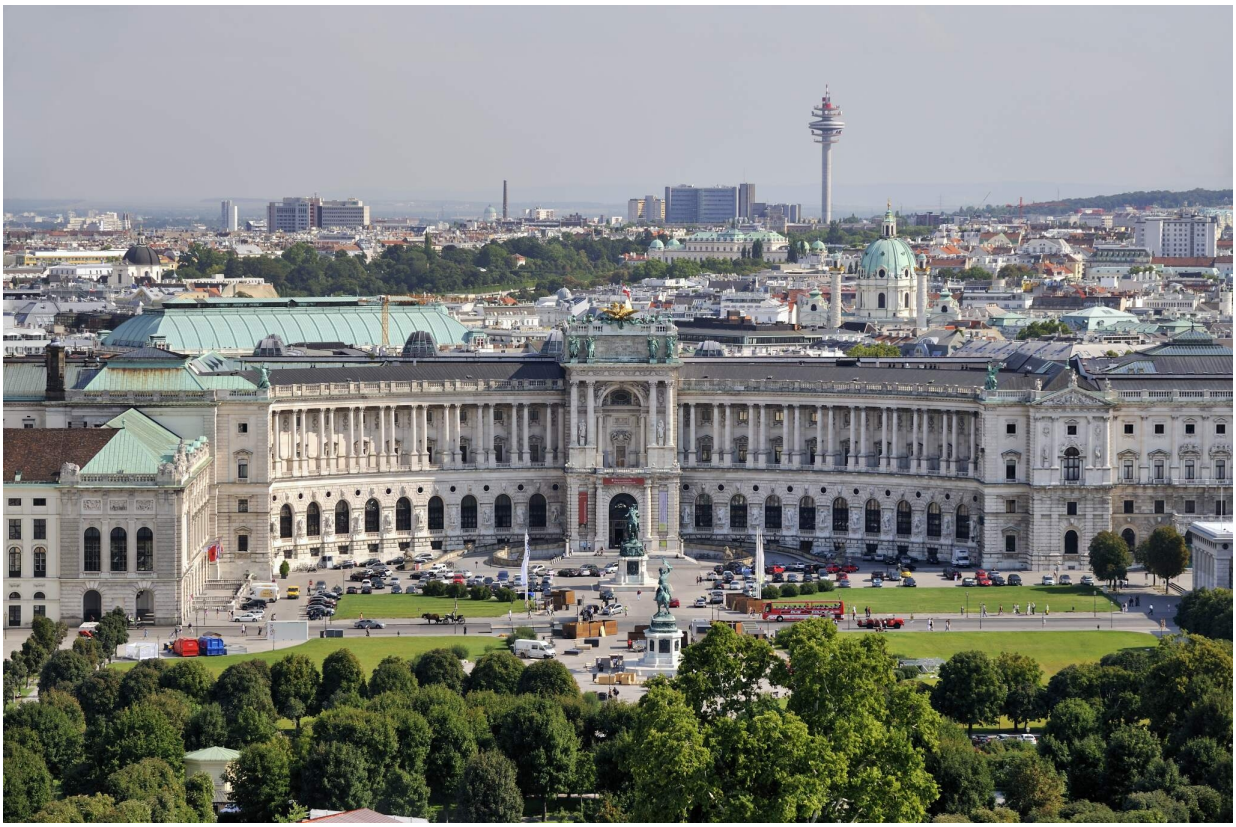
22nd International Conference on Reliable Software Technologies

Ada-Europe 2017



12-16 June 2017, Vienna, Austria

Abstracts

Industrial Presentations

http://www.ada-europe.org/conference2017

In cooperation with

acm In-Cooperation SIGAda SIGPLAN

Ada Resource Association

Astronomical Ada.
Using Ada to control Telescopes.

White Elephant GmbH – software@white-elephant.ch

A very basic introductory level astronomical telescope either has no motor or a single motor to compensate for the rotation of the earth.

However a keen astronomer will quickly want an automatic means of locating a galaxy, star or planet, typically using his or her PC to direct the telescope to the specified location. For this to be possible the telescope needs a second motor.

Freeware virtual planetariums such as the well-known Stellarium can be used to instruct the telescope to position and then track a specific object.

However, very keen astronomers will want to do more. At the time of writing, the only way to position and track "Near Earth Objects" (NEOs), is to write your own telescope control system.

So that is what White Elephant GmbH did!

SkyTrack is a PC program that is able to control the movement of a telescope to find and follow stars, planets, deep sky objects or near earth objects such as satellites and asteroids.

The program is written entirely in Ada 2012 and can be complied to execute under any environment that is supported by GNAT and Gtk3. For example: Microsoft Windows, Linux and Apple OSX.

It supports telescopes that have simple stepper motors (without decoders) as well as both Equatorial and Azimuthal mountings.

The telescope can be controlled using the program's graphical user interface (GUI), a numeric keypad or via Stellarium. In the latter case the program sends feedback to Stellarium so that it can keep its display up to date. The use of Stellarium or other third party products is not a requirement.

Other features that are often missing from contemporary products include:

- Favourites : a list of frequently observed objects.
- Horizon : For fixed telescopes and those at their "home" location, the actual horizon can be defined. The actual horizon differs from the true horizon when buildings obscure the true horizon. This feature allows the program to improve the accuracy of its filtering of visible objects.
- Only list objects that can currently be seen or, in the case of NEOs, are soon to be seen.
- Catalogues : Objects can be named according to a variety of catalogues. The same astronomical object is called different things according to the catalogue used.
- On Earth Objects : For demonstration purposes, when the weather is overcast, it is sometimes useful to be able to position the telescope onto a nearby terrestrial object (eg a road sign). In this case the rotation of the earth must be ignored.
- Park position : For fixed telescopes or those at their "home" location, it is useful to park the telescope to a known position before powering off. This saves the need to realign the telescope next time it is used.

- Normal and Expert modes : Useful to prevent shared or public telescopes from being accidently misconfigured.

Originally we attempted to control the telescope using a dedicated microprocessor. However the requirement of a graphical interface as well as the phenomenal amount of calculations that needs to be performed persuaded us that a standard PC was the best suited platform.

However PCs equipped with their standard operating systems are not real time. The control of stepping motors, without decoders to ascertain their actual position, requires hard real time. Steps must be precisely calculated and steps may not be lost.

Consequently a small microprocessor is used to actually control the stepper motors. The PC program sends a stream of future actions to the microprocessor using standard Ethernet UDP/IP. In this arrangement the PC does all the processing work whilst the microprocessor maintains the real time control of the motors.

Ada was an ideal choice of implementation language mainly because of the large amount of concurrency involved. Input commands may come from the GUI, Stellarium or the keypad. In addition communication with the microprocessor must be maintained and new actions generated. Dynamically displays must be updated as objects appear and disappear from the visible sky.

For performance reasons, a part of the large SIMBAD astronomical database, which currently contains over nine million objects, was converted into an Ada constant table and then compiled into a static library. The Universe, from our perspective, is relatively stable and therefore a large structured constant is much better suited for fast real time access than a conventional database.

Challenges:

- For portable telescopes – a fast yet easy method to align the telescope. For equatorial mounts this means bringing the telescope polar axis parallel with the Earth's axis, even without visibility of the Polar star (Polaris).
- Time. NEOs are typically moving so fast that they traverse the viewed area within a fraction of a second. A small inaccuracy in time can therefore make the difference between seeing the object or not.
- NEOs, although travelling at a constant speed, appear from the earth to be accelerating and decelerating according to their position and the angle at which there are observed. This requires that the telescope continually vary its tracking speed.
- When the tracked object is changed to another visible object, it is highly desirable that the telescope repositions using the optimum route. Large heavy telescopes move relatively slowly so it is important not to waste time making unnecessary movements.
- The motors have finite limits depending on the load and their angle. The software needs to take this into account in order to prevent losing steps when the tracked object moves too fast to be accurately tracked. In this case the telescope will be trying to catch up with the object. This rare situation occurs with Equatorial mounts when the object passes close to the Polar star or with Azimuthal mounts when the object passes close to the zenith.

The software executing on the microprocessor used to control the stepper motors is currently written in Modula-2, a poor man's Ada. A future project will be to port this to Ada.

IP Network Stack in Ada 2012 and the Ravenscar profile

Stephane Carrez

Proposed industrial presentation at Ada-Europe 2017

Abstract

Ada Embedded Network is a small network stack intended to be used by small embedded Ada applications running on ARM. It implements the standard ARP, IPv4, UDP, DNS and DHCP protocols on top of an Ethernet driver. It runs on the STM32F746 board.

The talk presents the components from the Ada implementation point of view. It will highlight the Ada features that have been used and show some of the benefits of the Ravenscar profile that have helped the project.

First, the Ethernet driver uses interrupts to receive and transmit packets. It implements the transmit side by using a dedicated protected object that controls the transmit ring. On its side the receive ring is controlled by a second protected object. This design increases the concurrency between the network transmission and reception.

On top of the Ethernet driver, the ARP module maintains the ARP table that maps the IPv4 address to the Ethernet address. The ARP module uses a protected object to maintain that map. It handles the ARP resolution with retransmission of ARP requests and the detection of stale entries.

The IPv4 layer also sits on top of the Ethernet driver but it also relies on the ARP module for the Ethernet address resolution. The IPv4 layer is only responsible for setting the IPv4 packet header. It dispatches received packets to upper protocol layers.

The UDP layer defines a simple socket-like interface by providing a `Socket` abstract tagged type that represents the UDP server port. Applications have to override a `Receive` procedure to receive packets. The simple design allows to easily send and receive UDP packets.

The DHCP module uses the socket framework to implement the DHCP client state machine, allowing applications to get their network configuration. The DHCP state is controlled by a protected type because the transmission and reception of DHCP packets are handled by different tasks.

Like the DHCP module, the DNS module uses the UDP socket as a client to make DNS requests to the DNS server whose address was received by the DHCP configuration.

Finally the presentation will also describe how EtherScope and its two tasks design uses the stack to make the real-time analysis of network packets, display the result on the 480x272 display, and interact with the user through the touch panel.

The conclusion will explain how Ada helped the project and what were the difficulties found during the design and implementation.

Hardware-Based Data Protection/Isolation at Runtime in Ada Code for Microcontrollers

This industrial presentation will describe an approach for using the hardware-based memory protection capabilities from modern low-end microcontrollers, to control access to data structures and peripherals in bare-metal embedded software, written in Ada 2002.

Software for low-end microcontrollers is typically written bare-metal or using an RTOS. In the case of Ada, the role of the RTOS is typically played by an Ada runtime library, such as one of the Ravenscar profiles. This type of software consists of one of more Ada tasks and interrupt handlers running in a single address space. Traditionally, there has been no data protection mechanism within a single address space. An errand pointer can corrupt any location in the address space, including variables in RAM and memory-mapped I/O registers. Although this is less likely in code written in Ada than in code written in C/C++, it is still possible and for safety-critical/high-integrity software, this possibility must be reduced to the minimum. Besides, Ada programs that call libraries written in C need to protect themselves from buggy or malicious C code that can corrupt the Ada data structures.

Many modern low-end microcontrollers come with a memory protection unit (MPU) as an alternative to the memory management unit (MMU) from high-end microcontrollers. A memory protection unit (MPU) is a hardware module that enables software to control access to areas of physical memory known as regions. These regions can vary in size and typically can be as small as 32 bytes, in modern MPUs. The set of memory areas (regions) that a given Ada task is supposed to access may be different at different points in the execution of the task, and it depends on the software components (Ada packages) that the task invokes. A given component's invoked subprogram can access variables in RAM or memory-mapped I/O registers of a given peripheral, or both.

This presentation will show an approach for using a microcontroller's MPU to provide component-level data protection within a single address space, for bare-metal microcontroller software, written in Ada 2012. First, an MPU-independent API to support this approach will be presented. Then, an implementation example of this API for a specific microcontroller will be described. Next, Ada code design techniques to structure an application's code components to enforce component-level data protection will be illustrated. Finally, the required changes to the task creation and context switch in the Ada runtime library (GNAT Ravenscar Small Footprint profile) will be described.

Automated Testing of SPARK Ada Contracts: Progress & Case Study Report

Industrial Presentation for Ada-Europe 2017

Background & Introduction

More than fifty percent of avionic software development cost is spent on software verification. The design, maintenance, and execution of unit tests based on software requirements, takes up a significant proportion of this. The Federal Aviation Administration (FAA) and European Aviation Safety Agency (EASA) outline that all avionic software must meet certification standards and the DO-178 certification guidance is the most widely recognised means of compliance. This guidance places a significant amount of burden on the cost of avionic software. The AUTOSAC research project, funded by the National Aerospace Technology Exploitation Programme (NATEP), has developed a tool chain that automates the generation of unit tests, whilst still maintaining their credibility as functional tests within the eyes of the DO-178 guidance. This is achieved through the use of Ada contracts (a feature of the SPARK language) and a process developed in collaboration with Rolls-Royce Control Systems.

In this presentation we will describe recent progress on the project – including results from industrial case studies. We will discuss key results including certification arguments regarding conformance to DO-178 guidance, and present one of the key innovations: a *post-condition coverage criterion* designed to generate high-value test cases similar to those that would be written by an experienced test engineer.

Case Studies & Results

The primary industrial case studies to which the prototype tools have been applied are samples of existing industrial code written by Rolls-Royce for their FADEC (Full Authority Digital Engine Controller). The first study was drawn from the Operating Software (OS) – lower level platform code. The second was drawn from the FADEC Application Software (AS) which runs on top of the OS layer. Both are examples of DO-178 DAL-A (safety critical) software that has previously been subjected to manually-written unit tests. The code was written in the SPARK ’95 subset of Ada, but had not previously been contractualised. For the purpose of the case study we added a functional specification using SPARK 2014 pre- and post-conditions. The AUTOSAC tools were then used to generate unit test cases from the SPARK specifications and executed with the post-conditions enabled to provide an oracle for the test results.

By applying the AUTOSAC tools to existing, certified code, we are able to make a comparison in three important dimensions: the quality of the automatically-generated tests vs. those that a test engineer would write; the extent of structural coverage that is achieved automatically; and a comparison of time, costs and process complexity.

Introducing static analysis to a mature project

Jacob Sparre Andersen

JSA Research & Innovation, Jægerparken 5, 2. th., 2970 Hørsholm, Danmark

Static analysis tools (eg. AdaControl, CodePeer) are accepted as a useful way of identifying potential problems in source text before they turn into system failures or make a project prohibitively expensive to maintain.

I have previously been presented with anecdotal evidence that – as good as such tools may be – if they aren't used on a project from day one, it is very hard to get any benefit from them, since they have a tendency to need the developers to stay within a limited set of patterns, which the tools can recognise as correct.

Since January 2016 I have been working with a client, maintaining an application initially developed in 1987, which includes of more than one million lines of Ada source text. This application has traditionally been developed without the use of static analysis tools (besides a small selection of enabled compiler warnings).

As the customer decided to focus on increasing quality (to reduce support costs), we experimented with using AdaControl to identify problems in the source text. The good news was that AdaControl had a single rule matching a significant part of the registered incidents. The bad news was that AdaControl reported many more (potential) issues than we had resources to do something about[1].

We are handling this in several ways:

- My client funded an extension of AdaControl, such that the rule recognises the most common, correct implementation pattern in the application.
- For a start, we decided only to run AdaControl on units which are touched anyway; either due to a change request or due to an incident.
- We track individual rule violations on a per-file basis in our continuous-integration system, to ensure that the number of violations of each rule is not increasing.

Extending the tool reduces the number of false positives on checks of our most critical problem.

Only checking units, which are touched by a developer anyway, limits the effort spent fixing sources to where we are likely to introduce new problems.

Tracking rule violations will allow us to increase quality, even with an explicit non-zero-warnings policy. If we have zero-warnings, we are probably not checking our sources for enough kinds of problems.

As an indication that the focus on increasing quality already has paid off, the frequency of incidents has been reduced significantly since the beginning of 2016[2].

The presentation will:

- Report the effect of each of the three attacks on the problem listed above.
- Contrast zero-warnings and non-zero-warnings policies.

Throughout the presentation both business needs and software engineering "needs" will be discussed.

I hope to close with a lively discussion including the experiences of the audience on the subject.

[1] Matching the above-mentioned anecdotal evidence.

[2] This change is not **only** due to the work presented here.

Challenges and Opportunities for Improvements of the Testing Process for Ada based Safety Critical Systems

Dr. Guillem Bernat, CEO

Rapita Systems Ltd. (UK)

Atlas house, Osbaldwick Link Road

YO10 3JB York, UK

Abstract

The Ada language continues to be used widely for safety critical systems. This success is underpinned by features of the language that support safety critical software development. However, some of these features are difficult to test and, as a result, additional manual effort during the testing process is required making it less attractive.

The new revision of DO-178, DO-178C, has also impacted the testing process by shifting the emphasis to integration testing and testing on target. This introduces additional challenges during testing that without proper automation and tool support increase the burden, cost and effort to satisfy the DO-178 objectives.

In this talk we outline the language features of safety critical systems written in Ada that are challenging to test to satisfy DO-178C objectives and show strategies on how to address them. These include: private types, subtypes, suprograms, generics, nested generics, local scope variables, pragma import/export and complex package dependencies. We also present strategies to address other challenges, including reusing legacy code and legacy tests, as well as merging structural code coverage evidence from multiple sources of tests.

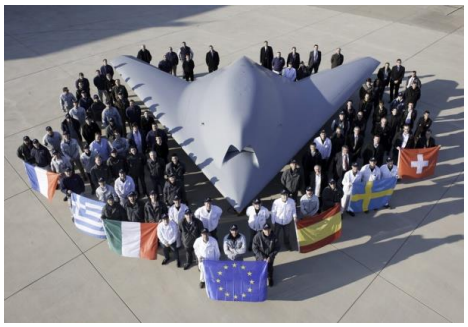
The tool RapiTest Framework, from Rapita Systems, has been designed from the ground up to provide substantial improvements to the challenges of testing safety critical Ada programs. The tool focuses on process automation and provides support for the challenges listed above. In this talk we will present the main features of the tool that address those challenges and provide evidence from a real life industrial case study of process improvements and quality metrics to satisfy DO-178B and C objectives.

Experiences with Ada in the Safety-Critical Communication and Ground Control Systems of the nEUROn UCAV

Luis Pabón, Artemio Jiménez, and José M. Martínez

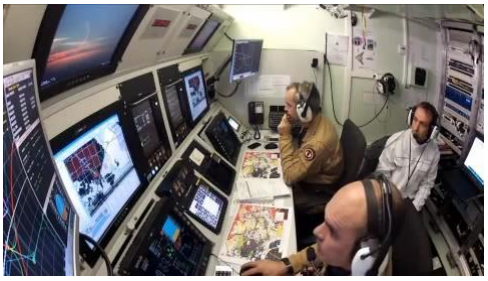
Airbus Defence & Space, paseo John Lennon s/n, 28906 Getafe, Spain
{luis.pabon,artemio.jimenez,jose.d.martinez}@airbus.com

Extended Abstract



The nEUROn is an experimental unmanned combat aerial vehicle (UCAV) developed with international cooperation, led by the French company Dassault Aviation and with the collaboration of several European aerospace industry partners (Airbus, Saab, Alenia, Thales, RUAG and HAI). Countries involved in this project include France, Greece, Italy, Spain, Sweden and Switzerland.

The main objective of the program launched in 2003 was to acquire the necessary knowledge and experience for future UAS programs by developing a Pan-European large size stealth UCAV platform to demonstrate the maturity and the effectiveness of technical solutions. The aircraft made its maiden flight in December 1st 2012, and numerous flights have been performed since then to test the capabilities of the system.



Ada is a key part of the embedded, real-time and safety-critical systems contributed by Airbus Defence & Space to the nEUROn program: the *Data Link Management System* (DLMS) and the *Global System Monitoring & Control* (GSMC), whose software has been developed to be certifiable according to DO-178B up to levels D and C respectively.

This industrial presentation focuses on the advantages and challenges found after the choice of Ada for the development of these two software systems, putting the focus on real problems and issues such as the fulfillment of safety requirements and constraints, the integration on ARINC 653 partitioned environments, the generation of graphical human-machine interfaces (HMI), the design of computationally intensive algorithms or the application of different verification and validation techniques.

The presentation summarizes the main lessons learnt during this successful industrial case study, and gives an insight on how the gathered experience has helped Airbus Defence & Space to improve other Ada-based technical solutions in subsequent projects.

Experience with Use of Model Driven Code Generation on the ASIM Project

Steen Palm
Terma A/S, Vasekær 12, 2730 Herlev, Denmark; Tel: +45 4594 9665; email: sup@terma.com

On the Ada-Europe 2012 conference, I presented the paper "Use of Model Driven Code Generation on the ASIM Project". The paper described the approach used for the development of the Ada software to control the two ASIM instruments, MMIA (studying the high-altitude electrical discharges in the stratosphere and mesosphere above severe thunderstorms, the so-called red sprites, blue jets, and elves) and MXGS (observing terrestrial gamma flashes associated with severe thunderstorms), that will be placed on the International Space Station. The development approach is based on the principles of SAVOIR and ASSERT. For each of the two instruments, a UML component design is constructed where interfaces are decorated with stereotypes like ‹‹cyclic›› and ‹‹sporadic›› defining their concurrency behaviour. From the design of an instrument application (the Interface View) and Ada packages implementing the sequential behaviour of the interfaces (the Functional View), a final Ravenscar compliant implementation of the instrument software (the Concurrency View) is automatically generated.

At this point, all functional, electrical, and mechanical tests of the ASIM payload are completed. Currently, the flight model is undergoing an environmental test and later this year ASIM will be launched with a Falcon-9 rocket and placed on the Columbus module of the International Space Station (ISS).

The presentation on this year conference will elaborate on the experiences gained during the development of the software (boot and application software) for MMIA and MXGS.

The presentation will have two parts. The first part will summarize the development approach, which is based on SAVOIR/ASSERT principles. To support these principles, Terma has developed a modelling tool chain, which will secure a Ravenscar compliant implementation:

- The design will be expressed as a component model in UML. Components may be composite or simple (leaf components). Enterprise Architect from Sparx Systems has been selected as UML tool.
- The Interface View is supported by specific UML stereotypes (like ‹‹protected›› and ‹‹cyclic››) that can be used to decorate interfaces provided by model components. The stereotypes are complemented with real-time attributes like worst-case execution time and period, which are supplied as UML tagged values associated with the provided interfaces.
- The Functional View is implemented as passive Ada packages. For each leaf component in the design, there will be a corresponding Ada package that defines the sequential behaviour of the operations provided by the component.
- The Deployment View is not supported. As the instrument software is running on a single node, a Deployment View was considered superfluous.
- The Concurrency View (obtained by vertical transformation) is automatically generated from the UML design (more precisely, from an exported XMI file). The Concurrency View consists of Ada tasks and protected objects, which will call the passive Ada subprograms defined in the Functional View. The Concurrency View constitutes the final implementation.

The second part of the presentation will describe the experience gained.

Overall, the use of the model driven approach based on the ASSERT/SAVOIR principles was very successful. The concurrency aspects were in a very natural way defined as part of the design of the behaviour and interaction between components in the UML design (the Interface View). Also, all issues related to concurrency could easily be discussed and resolved at design level without any concern about the source code.

Of course, minor updates to the tool chain have been done in order to accommodate various needs. The most important updates are related to schedulability analysis and the approach used to obtain worst-case execution times for tasks and protected operations.

The software can be built such that it will automatically measure the genuine execution time (as opposed to elapsed time) of tasks and protected operations each time they are executed and keep track of the largest execution time for each. By defining performance tests running worst-case scenarios, it was possible to easily get the worst-case execution times for the tasks and protected operations.

The presentation will go into much more depth of the approach and the gained experience than described here.

A Time-Triggered Middleware for Safety-Critical Automotive Applications

Ayhan Mehmed, Wilfried Steiner, and Maximilian Rosenblattl

TTTech Computertechnik AG, Vienna, Austria
{ayhan.mehmed,wilfried.steiner,maximilian.rosenblattl}@tttech.com

Advanced driver assistance systems (ADAS) are one of the fastest growing sectors in the automotive industry. Initially developed as add-on comfort features, ADAS now target highly intelligent, fully autonomous, vehicle control systems. Towards this goal various technical challenges have to be addressed to guarantee dependable and deterministic vehicle behavior.

A clear challenge is the growing number of ECUs, which drives complexity, weight, power, space consumption and ultimately the cost. For this a higher degree of integration of function per ECU is needed. At the same time, the integration of functions from different domains and with differing requirements, will require interference-free coexistence of mixed-criticality functions. The demand for further ECU consolidation, requires virtualization like techniques, which allow the coexistence of multiple operating systems on the same ECU. Last but not least, the vehicle systems of today have to execute their functions with respect to real-time, meet the highest safety requirements (up to ASIL-D) and must accelerate their development cycles.

To address these challenges we outline a novel time-triggered middleware for vehicle cyber-physical systems (CPS), namely TTIntegration. Complementary to the TTIntegration middleware, the AUTOSAR software architecture and concepts have been followed as a reference. Starting from top, on application level the software components communicate according to the AUTOSAR standardised application interfaces. One level down, the TTIntegration middleware is placed for a clear separation between the integrated applications and the basic software and hardware. Apart from serving as abstraction layer the middleware enables the execution of tasks according to a time-triggered paradigm - a set of concepts and principles, which provide (i) a predictable timing behavior of each application, by ensuring sufficient CPU-time and memory, and (ii) a guaranteed freedom of interference between the applications. Furthermore, for high degree of function integration, the middleware enables multi-core system-on-chips (SoCs) to run in parallel with different operating systems (e.g. AUTOSAR, VxWorks) and collaborate with each other. By extending the AUTOSAR environment for all possible operating systems, all applications can then be moved between different SoCs with ease. Finally, to facilitate the process of software integration, the middleware provides parallel, multi-vendor development and integration paths for individual software components.