$$\frac{(Spacecraft\ Bus\ Controller) + (Automotive\ ECU)}{2} = Ultimate\ Controller$$

Sergio Montenegro, Frank Dannemann, Lutz Dittrich, Benjamin Vogel
DLR Institute for Space Systems
Dpmt. Central Avionics
Robert-Hooke-Str. 7
D-28359 Bremen
Sergio.Montenegro@DLR.de, Frank.Dannemann@DLR.de, Lutz Dittrich@DLR.de
Benjamin.Vogel@DLR.de

Ulf Noyer, Jan Gacnik, Marco Hannibal, Andreas Richter, Frank Köster
DLR Institute of Transportation Systems
Dpmt. Automotive
Lilienthalplatz 7
D-38108 Braunschweig
Ulf.Noyer@DLR.de, Jan.Gacnik@DLR.de, Marco.Hannibal@DLR.de,
Andreas.Richter@DLR.de, Frank.Koester@DLR.de

Abstract: In many cases, similar challenging problems arise in different domains and are often solved with an only limited focus. Synergy effects between different research fields can lead to an enormous technical improvement. This is illustrated by combining experiences and competencies of the institute of Space Systems and the institute of Transportation Systems of the German Aerospace Center (DLR). Their cooperation/collaboration aims at more reliable and highly available platforms for embedded (software/hardware) systems. Both institutes have developed software frameworks to be used in embedded systems of their field of work. In this paper we present an approach of integrating an embedded real time OS to be used in satellites into the DOMINION implementation-platform, which is used for the specification of embedded in-vehicle software. Finally a safety critical application with hard real time requirements will be chosen as a use-case to demonstrate the combination of aerospace and automotive technologies.

1. Introduction

As pointed out above, similar challenges in the context of embedded systems related research are currently to be focused in the space as well as in the automotive domain, e.g. dependability, reliability, availability, and safety of the built systems has to be improved. The fast growing complexity of the systems leads to the famous "design-gap" which has to be handled with a more maintainable and structurable design. Re-usability of requirement analysis, designs, and implementations (e.g. as an intellectual property (IP) core) lead to faster time to market. If the system can heal itself by being adaptable to changing circumstances (self-x-properties), it can be used in unknown or at least unpredictable environments, i.e. the space or a car engine.

Besides sharing similar problems, similar trends can be determined. The growing trend of software driven innovations in the system design emerges as using a large share in the development costs (up to 70%). While the automotive industry is already fighting the complexity to be handled and is trying to reduce it, space industry only recently realizes the growing complexity. With the planned technology transfer, the complexity collapse could be avoided.

Both above mentioned domains have different approaches and views of common problems and have reached different maturity in different fields. The following table shows some of these differences and shows which branch may profit from the achievements reached by the other one (see the transfer arrow).

Automotive	**Transfer**	**Space**
Real industrial production (thousands of units per day)	→	Artisan level (hand-craft): each unit hand made, and unique (thousands of days per unit)
Europe has a leading role	→	Europe is not in the cutting edge
Good reuse practice	→	Hardly reuse
Scarcely fault tolerance	←	Widely spread fault tolerance: No single point of failure is allowed
Safety/dependability relies mainly on mechanic fall back mechanisms and on the driver	←	All safety/dependability measures are based on autonomous reactions of the avionics (no mechanic/operator fall back mechanisms)
Traditionally based on mechanics	←	Electronic and software based from its origin
One important optimisation factor is the cost.	→	Cost was not important. Now changing, but lack of experience.
Power, mass and volume were not important. Now changing but lack of experience.	←	Design optimized for lowest resources: power, mass, volume
Interoperability between different suppliers and between main contractor (e.g. AUTOSAR)	→	No interoperability between suppliers.

Complexity management	← →	Complexity Management
Active steps to reduce the complexity (e.g. reduce number of ECUs: AUTOSAR)	→	No active steps to reduce the complexity
Standardized communication, e.g. CAN	→	No global accepted standard. Each device has its own protocol. Now one standard is emerging: SpaceWire, but it is too complex to be accepted widely.
Complexity is already out of control		Complexity is becoming out of control
Experience on safety critical hard real-time systems	→	Time violations are not considered to be mission critical
Experience on handling sensors failures and errors due to environment conditions	→	Systematic sensors errors/failures are not a central theme of fault tolerance
Transient computation and storage errors are not a central theme for fault tolerance	←	Experience on handling transient computations and storage (CPU + MEMORY) errors, due to radiation effects.
Experience using diversity to avoid systematic errors (especially for sensoric)	→	Diversity is not a central theme for dependability
Large experience on quality management (but different than space)	← →	Large experience on quality management (but different than automotive)
Large experience on system tests (but different than space)	← →	Large experience on system tests (but different than automotive)

Both domains can learn from each other about quality management, systematic tests and complexity management because both have totally different solutions for the same kind of problems. In the other fields one has more experience than the other one, and in one filed (getting rid of complexity) both are having the same problem. In any case both can profit from sharing experiences and competencies.

2. Partners

2.1 The DLR Institute of Space Technology

The DLR Institute of Space Technology (www.dlr.de/irs) has the aim to investigate and evaluate complex astronautic systems in the context of space research given consideration of technological, economic as well as socio-political and security aspects. Furthermore, the dynamic team of employees at the institute develops concepts for innovative space missions on national and international levels. The focal points of the working groups are: system analysis for space transport systems and space segments, system technology (rover wheels, satellite data management, and attitude control, cryogenic rocket propulsion systems) and operation of test and simulation facilities.

The Department Central Avionics develops software and hardware for the core avionics of a spacecraft (avionics, excluding devices). The central element in our architecture is a network to interconnect devices, computers, storage units, and radio communication units. Usually devices are attached directly to the central computer, in our architecture the computer has only link(s) to the network and all devices are attached to the spacecraft network (SAN).

2.2 The Institute of Transportation Systems

With basic research activities and numerous projects with industrial as well as scientific partners the institute of Transportation Systems (www.dlr.de/ts) aims for improving safety and efficiency of road and rail transportation. Therefore, scientists from different disciplines like engineering, psychology, and computer sciences strive for innovative solutions for automotive and railway systems as well as for future traffic management.

Beside activities in the institute's departments of Railway Systems and of Traffic Management, the human-centered design of advanced driver assistance and automation systems is the major topic of the institute's automotive department. This and other research activities are supported by elaborated research facilities: With the DLR ViewCar® driving in real traffic is observed with regard to understand driver behavior and the dynamics of surrounding traffic. Based on such results assistance and automation systems are developed to support the driver in an adequate manner and to prevent errors as well as accidents. First prototypes are usually implemented in a virtual reality laboratory and other simulation facilities for fast and efficient testing and evaluation. If necessary, a special dynamic driving simulator can be used for further evaluation steps. Experimental vehicles (FASCar-I/II) are available to demonstrate proposed concepts of assistance and automation in reality.

For different purposes technologies from the aerospace sector are adapted. That includes e.g. image processing, location services, and control- as well as planning algorithms. For an effective integration of functions flexible and modular system architectures are being developed.

3 RODOS - Building Blocks Execution Framework

In the first step, the Real Time Onboard Dependable Operating System (RODOS) [3] is provided by the DLR Institute of Space Technology to the joint venture between the space and automotive industry. In the near future, network technologies, and hardware implementations of middleware's may be included to the joint venture.

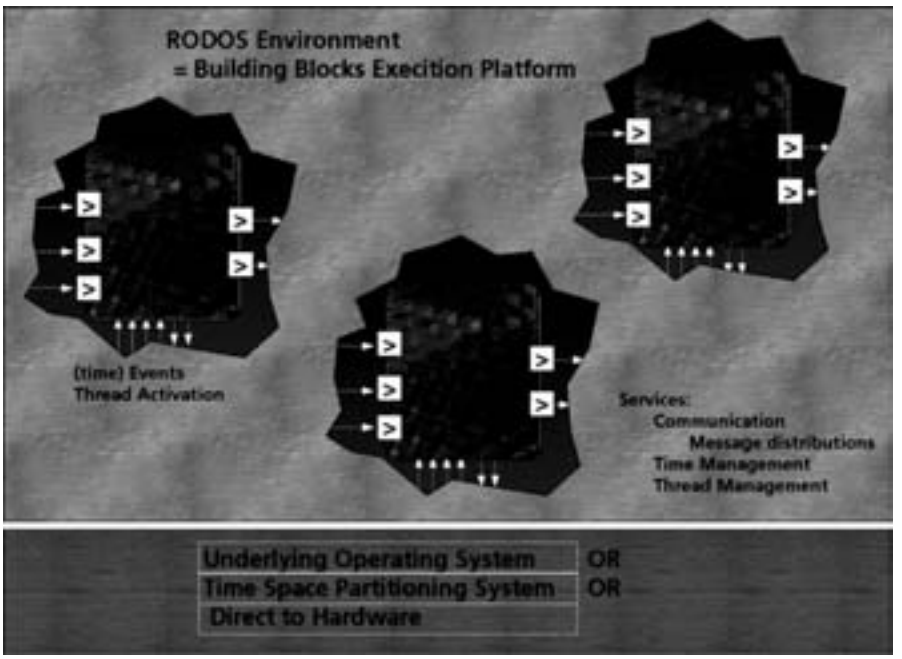

Figure 1: The RODOS Environment executes S/W building blocks.

RODOS is a building block execution platform (see figure 1) designed for space applications and for applications demanding high dependability. Simplicity is our main strategy for achieving dependability, as complexity is the cause of most development faults. The system was developed in C++, using an object-oriented framework simple enough to be understood and applied in several application domains. Although targeting minimal complexity, no fundamental functionality is missing, as its microkernel provides support for resource management, thread synchronization and communication, input/output, and interrupts management. The system is fully preemptive and uses priority-based scheduling and round robin for threads sharing the same priority level.

The execution platform provides a (software) interconnection network between applications / building blocks. A building block requires some services (incoming messages) in order to be able to provide other services (outgoing messages). The execution platform RODOS distributes such services (messages) from producer to consumers.

RODOS may be executed on top of other operating systems or TSP (Time Space Partitioning) systems, or directly on the hardware in case no other operating system is running on the target hardware. In all cases, the interfaces to the building blocks (or applications) remain the same, and a network of applications may be executed on different platforms and operating systems without modifications.

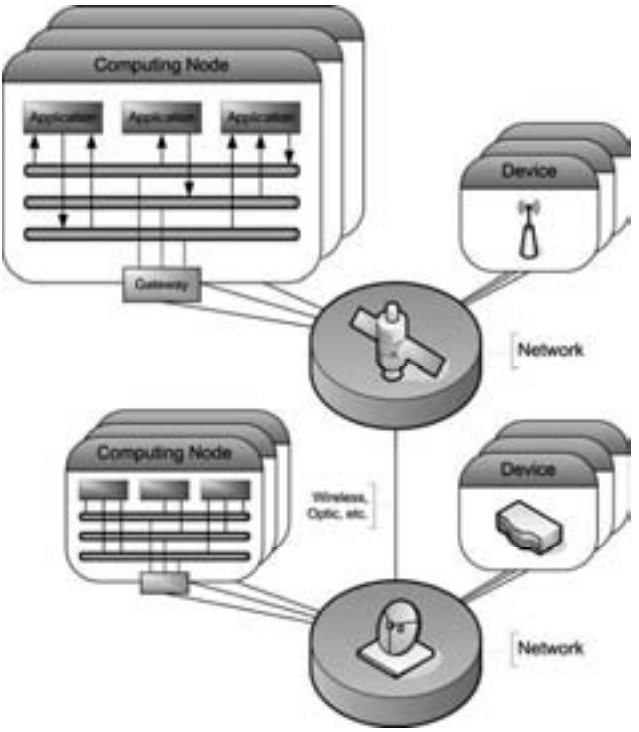

Figure 2: Example of the network centric network

RODOS provides a middleware which carries out transparent communications between applications and computing nodes. The messages exchange is asynchronous, using the publisher-subscriber protocol. Using this approach, no fixed communication paths are established and the system can be reconfigured easily at run-time. For instance, several replicas of the same software can run in different nodes and publish the result using the same topic, without having to know each other. A voter may subscribe to that topic and vote on the correct result. The middleware core distributes messages only locally, but using the integrated gateways to the "NetworkCentric" network [2] [3], messages can reach any node and application in the network (see figure 2). The communication in the whole system includes software applications, computing nodes and IO devices.

Our interconnection links in the network define a very simple message distribution protocol. Each message in the network receives a topic tag which identifies its content and purpose. Examples of topics could be: position, temperature, and attitude. Services will be published as topics regardless of whether they are produced by software tasks or by hardware devices. Any application or device may subscribe to the topics which it needs to help to produce its services.

This method is used to interconnect service providers and consumers including hardware and software. All our devices and software components provide this interface. In order to be able to attach COTS (Commercial Off The Shelf) devices (with their own protocols) to the network, the network provides the required interfaces and protocol converters. The COTS devices receive and send their own messages, and then the protocol converters translate them into our internal "universal language". The network will perform all required transformations in order to make the message transport transparent.

4. Implementation-Platform DOMINION

DLR Institute of Transportation Systems has defined an architecture for development of in-vehicle services called DOMINION [1]. The step especially seems reasonable with the current standardization of Intelligent Transport Systems (ITS) technologies, including wireless communications (C2X – car to car, and car to infrastructure communication) and related services (e.g. Internet enhanced navigation, hazard warnings etc.).

Part of DOMINION uses modified versions of business standards, like VSDL (in-Vehicle Service Description Language) and VPEL (in-Vehicle Process Execution Language) . This concept integrates model-driven aspects like code-generation for real-time targets as well as the deployment of very flexible SOA services. The focus of the development of in-vehicle services is in the context of assistance and automation systems. A lightweight, platform independent (currently Windows, Mac OS X, iPhone OS, and RT-patched Linux are supported) DOMINION runtime environment has been developed, providing a flexible test environment of time-critical services. By using service descriptions, the description of communication and data structures etc. are independent of the programming language. A built-in code generator generates source code for heterogeneous platforms and programming languages. This allows deploying DOMINION based software on various environments and makes DOMINION a valuable approach for testing in different research facilities like simulators or even entire vehicles. Furthermore, to integrate the DOMINION runtime environment and SOA technology, DOMINION services might be deployed into both worlds by configuring the bindings of ports and operations of a service. Moreover, DOMINION offers general development services, especially when dealing with recording and storing data from testing assistance and automation systems.

5. Merging Automotive and Space Technology

The first step in our joint efforts is a software rendezvous: Generating RODOS applications out of DOMINION's semi-formal service descriptions.

Model driven development including high-level formal specification of services is more and more used within the automotive domain, which abstracts from low-level target implementations. Using DOMINION, services are specified in a simple, reproducible way, independent from implementation details.

In the aerospace domain, the complementary part to this approach is located. RODOS is an execution platform which handles real hardware of embedded systems taking into account all possible limitations and restrictions of real resources like memory, CPU performance, power consumption etc. RODOS targets dependability even if running on non-dependable components, while the fault tolerance mechanisms are transparent to the applications. These fault tolerance mechanisms can be added to the DOMINION applications without having to modify them. The available redundant hardware resources will be managed by RODOS to obtain the maximum of possible redundancy at any moment without intervention of the DOMINION applications.

A demonstrator which will be controlled by the RODOS-DOMINION pair is developed by the two DLR institutes (see Chapter 5.1 for details). During the development of the demonstrator, the shared problems can be handled by the know-how of both institutes. In the joint-venture, a fault tolerant embedded computer running RODOS is combined with the automotive applications specified in DOMINION. The code generator of DOMINION will be adapted to generate C++ code for RODOS. The DOMINION specification only contains a complete description of the service interface and its timing aspects, whether functional aspects nor low-level dependencies and restrictions are included. Fault tolerance aspects will be added by RODOS in a transparent way. This allows the application programmer to concentrate all his efforts on functional development. Both systems, RODOS and DOMINION, support the distribution of services, but in a different way. RODOS uses a publishers/subscriber message passing system while DOMINION uses a shared memory paradigm. As the distribution mechanism is transparent to the applications, i.e. an application doesn't need to know how data is transported, the DOMINION-RODOS system is easily ported to the message passing system.

5.1. Demonstrator (Car Case Study)

A safety critical application with hard real time requirements, which is a standard practice in automotive engineering, should be chosen to demonstrate the combination of aerospace and automotive technologies as described in the previous chapters. An emergency brake assistance system represents a suitable application. Such an assistance system is proposed to avoid rear-end collisions in case of critical situations, or at least attenuate possible impacts.

Reliability of hard- and software components as well as plausibility of sensor information illustrate typical problems with such an assistance system. To deal with that problem the system is equipped with two independent front radar sensors that are connected to a set of three redundant Electronic Control Units (ECUs).

Handling transient computations and storage errors is one of the main problems in aerospace and is mainly caused by cosmic radiation. A possible way of dealing with this problem is the usage of triple modular redundancy (TMR). In a TMR-based system, three ECUs perform a calculation, while the result is checked by a voting instance. Only if two of the three ECUs deliver the same result, the result is treated as correct. In automotive engineering, different problems arise in the reliabilty of sensor information and defective hardware components such as ECUs. As an example, different ECUs process sensor inputs with different algorithms providing a wide variety of different sensor information combined with different processing algorithms.

DOMINION's automatic code-generation for both DOMINION and RODOS run-time environments allows parallel development and testing of the assistance system in DLR's driving simulators using the DOMINION run-time environment as well as in test vehicles using the RODOS run-time environment.

On the one hand, this allows an economic and fast functional processing already in very early stages of development (software-in-the-loop implementation and evaluation). On the other hand, hardware components can also be tested early and without the need of expensive integration into test vehicles. In addition, it is possible to test functionalities or system failures, especially in simulated critical situations without any "real-life" risks and with the benefit of exact reproducibility of the simulated situations.

Since DOMINION provides a run-time environment supporting all research facilities in the transportation system branch, it is easy to migrate from one facility to another during different development steps or changing requirements. Necessary steps for migration are supported by DOMINION. During the development process of advanced driver assistance systems, the actual site of operation is completely transparent from the developers or users point of view. For this reason, it is not necessary to gain expert knowledge regarding the different research facilities such as driving simulators or test vehicles. Thereby, the developer can completely concentrate on the implementation of their functions.

The components of the emergency brake assistance system that will be developed in this car case study, shall be, as shown, developed, parametrized, and evaluated in the simulators to be finally tested under real-life conditions in the test vehicle.

Detailed Description

The figure 3 shows the usage of the combined DOMINION and RODOS environment in practical usage. Inputs for the DOMINION-code-generation are the architecture definition as well as the behavior description.

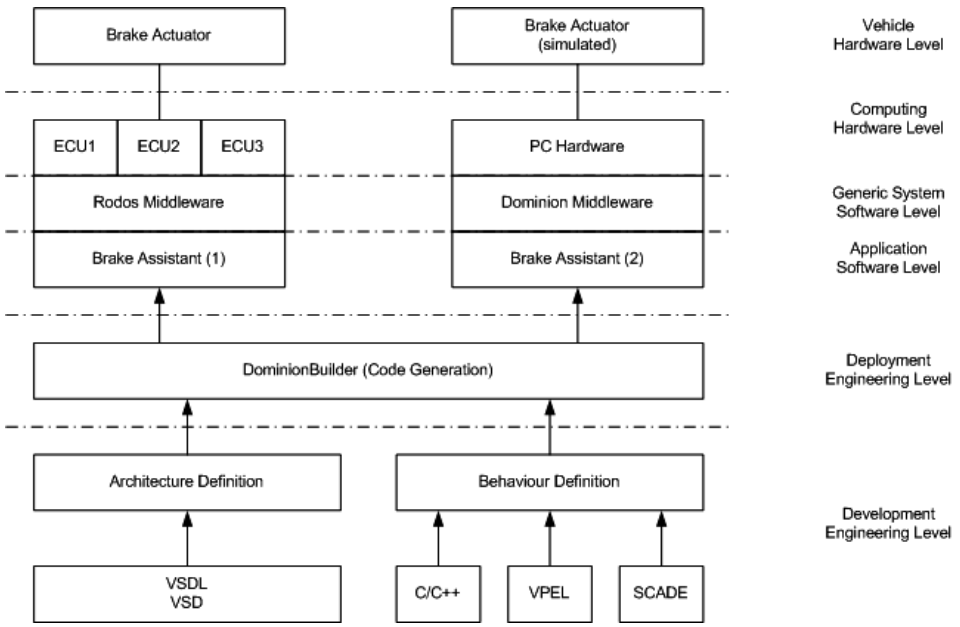

Figure 3: Combination of RODOS and DOMINION

The architecture definition describes modules and their input and output parameters, specified in VSDL. The code-generation can be switched to produce code for different target platforms. The application logic itself can then be described in programming languages like C/C++, VPEL, or SCADE. In case of critical functional development model-driven tools like SCADE support the development process by the application of certified code generators [1].

In the shown case, RODOS and DOMINION are based on C/C++, so code generation transforms all input specifications into valid C/C++ code. Practically, for the conversion of the code to another target platform only the specially generated base classes have to be exchanged.

The right part of the figure shows the classic DOMINION work-flow. The application is based on the DOMINION foundations and is deployed on a standard PC. To test the application, a simulation environment providing vehicle dynamics, street traffic, and a visualization is used. In this simulation environment, an easy and comparatively cheap evaluation of the newly added functions is possibly.

After successful testing, the same application is generated for the RODOS platform (see left part of the figure). Code generation provides the necessary base integration classes, whereby the behavior description (i.e. the actual logic) of the considered application itself remains unchanged. No manual programming is necessary to adapt the application for the RODOS platform. To control the brake, three completely separated instances of the application are built to be run on three EDUs. The redundant setup then controls the brake. To minimize the influence of erroneous sensor signals, every application compares sensor inputs itself and performs plausibility checks. With that scenario, the most likely state of the environment is considered (sensor fusion). As pointed out, triple modular redundancy is used before the hardware brake is triggered. In this way the application can be tested in the FASCar-I as a real testing vehicle. The FASCar-I is equipped with several diferent sensors (e.g.. radar, laser scanner, (D)GPS). Furthermore, steering and acceleration can be completely controlled by application logic on the on-board computers [2].

Because of the previous testing in the simulator the function is already in a mature stage and should be even work in critical situations as designed. However, since reality is more complex than any simulation, still further more testing and development is always necessary. Furthermore, development cycles can be performed, in which new advances are tested in the simulator again, before deploying them in the testing vehicle.

For the presented scenario the application logic for the brake controller application is diverse implemented three times by different developers. It is important to mention that only the application logic is implemented three times. As the application logic is only concerned with basic physical formulas for the braking conditions, it is short and reasonably well to understand. Therefore, errors in the application logic should be detectable by code reviews or similar precautions. Furthermore, the application logic is also checked with static software analysis for error recognition, which is the standard procedure for software in the space industry.

6. Summary

In this paper the merging of two software frameworks with a different background is discussed. The one is named RODOS and is mainly concerned with safety issues like redundancy and real-time communication with space industry background. The other platform, DOMINION, is developed within the context of automotive systems and allows to formally specifying services for later code generation and deployment on different target platforms. With that approach, we are basically able to combine the strength of both products. That allows us to use service descriptions and code generation for safety critical applications, dramatically accelerating software development in such scenarios. Until now, basic functionalities of RODOS and DOMINION have been successfully combined to show the potential of this approach. For further development, a complex scenario for a redundant, safety critical brake is planned, which proofs the value of the approach under realistic conditions.

Space sector and automotive employees have recognized that they can learn many things from each other and can even combine their work to reach more than just adding two parts. Based on the results of the planned scenario, further developments would be easily possible, i.e. a redundant steer-by-wire wheel controller. The authors believe that the described combination of the products offers great chances for future software development activities in the space and automotive sector.

References

[1] Schröder, Mark und Hannibal, Marco und Gacnik, Jan und Köster, Frank und Harms, Christian und Knostmann, Tobias (2010) *Ein Labor zur modellbasierten Gestaltung interaktiver Assistenz und Automation im Automotive-Umfeld.* AAET 2010 , 10.-11. Feb. 2010 , Braunschweig

[2] Noyer, Ulf und Schomerus, Jan und Mosebach, Henning und Gacnik, Jan und Löper, Christian und Lemmer, Karsten (2008) *Generating High Precision Maps for Advanced Guidance Support.* In: IEEE Intelligent Vehicles Symposium 2008 , Seite 67. IEEE Intelligent Vehicles Symposium 2008 , 2008-06-04 - 2008-06-06 , Eindhoven (Niederlande).

[3] Dr. Sergio Montenegro, Frank Dannemann *RODOS: Real Time Kernel Design for Dependability* , In: DAta Systems In Aerospace (DASIA) May 26 - 29, 2009, Istanbul, Turkey

[4] Dr. Sergio Montenegro, John Richardson: *RODOS for Network Centric Core Avionics* In: Conference on Advances in Satellite and Space Communications July 20-25, 2009 - Colmar, France

[5] Dr. Sergio Montenegro, Gunter Schoof, Ebrahim Haririan, In: *Network Centric Core Avionics,* DASIA 2009 DAta Systems In Aerospace, 26 to 29 May 2009, Istanbul