The AIDA System

A 3rd Generation IMA Platform

DIANA Project White Paper
1 The DIANA Project and its Goals

DIANA, Distributed, equipment Independent environment for Advanced avionics Applications, is an aeronautical research and development project funded through the European Commissions 6th Framework Programme and led by Skysoft, Portugal. It aims at the definition of an advanced avionics platform, called AIDA (Architecture for Independent Distributed Avionics), supporting execution of object-oriented applications over virtual machines and secure distribution services. The development process for the new platform will be based on the OMG standard called Model Driven Engineering (MDE).

The execution environment for object-oriented applications aims at neutrality. A neutral execution platform does not specify how it has to be implemented on certain hardware architectures or operating systems. In the contrary, the execution platform is expected to behave identically on different hardware and different operating systems. This will reduce dependencies between avionics applications and given hardware or operating systems strengthening a multi-supplier approach.

New technologies already used in other IT domains for many years, such as the object-oriented programming paradigm, will be integrated. Object-oriented programming is considered to support source code reuse, to ease software maintenance and to increase productivity during development. An object-oriented language designed with platform independence in mind and hence the ideal candidate for the project goals is the Java programming language. Main technologies analysed and used for the neutral execution environment are therefore the evolving Safety Critical Java Profile and the Real Time Profile, it is based on, and Aonix implementation of the Safety Critical Profile, the PERC Pico Java Virtual Machine.

Important design objectives of the secure interoperability architecture are to enhance interoperability means, increase software reuse and secure communication. Therefore, it will be based on technologies such as ARINC 653 Inter-partition Communication, CORBA middleware and data-centric communication technologies as well as security concepts like Multiple Independent Levels of Security (MILS).

To enhance interoperability means, AIDA proposes the use of a publish and subscribe data centric mechanism with content filtered topic supported by standard ARINC 653 ports and channels for data communication support. The use of content filter mechanism will reduce bandwidth by only communicating requested data to requesting applications.

Another important aspect of interoperability enhancement is location transparency. Instead of addressing application communication ports directly in application code and configuration, an abstraction layer based on CORBA middleware is introduced.
Traditional software reuse concepts are not enough for the avionics domain: the main focus in the avionic domain should not be limited to source code reuse but should include certification credits. An architectural strategy to increase this type of reuse is reducing the impact of the change at partition level. In other words, an application/partition shall be considered a sort of software component.

In civil avionics, safety is addressed as a fundamental requirement which must be satisfied by on-board software. Scenarios from the military context show that also security issues must be solved to ensure that safety constraints hold. Airborne applications like navigation or collision avoidance systems rely on information from remote sources. This information must be protected against attacks from the outside world. Additionally, a secure application should also protect its data from manipulation and disclosure by other applications on board.

The introduction of the DIANA concepts is expected to bring a significant development cost and time reduction. DIANA integrates promising technologies, such as CORBA, Java and data-centric communication mechanisms with field-proven IMA (Integrated Modular Avionics) approaches. The usage of such technologies in real time environment and the update of standards, like ARINC 653, will provide new opportunities to create the future IMA architectures for the next generation of aircraft.

DIANA cooperates closely with well-known technology providers like Windriver, SysGo and Objective Interface Systems (OIS). DIANA demonstration platforms will be built on high-quality products provided by these vendors, like the Real-Time Operating Systems by Windriver (VxWorks-653) and SysGo (PikeOS) and the Real-Time CORBA middleware by OIS (ORBexpress).

2 Reference Architecture

The AIDA Platform Architecture is inspired by several aeronautical and IT standards. First of all, AIDA envisions backward compatibility for ARINC 653 compliant applications and consequently it is deeply funded on the ARINC 653 concepts.

2.1 The existing Baseline Model for AIDA

ARINC 653 concept model foresees support to IMA systems partitioning: one purpose of a core module in an IMA system is to support one or more avionics applications and to allow independent execution of these applications. This can be correctly achieved if the system provides partitioning, i.e., a functional separation of the avionics applications, usually for fault containment, such that a failure in one partitioned function cannot cause a failure in another partitioned function; in consequence the partitioning approach eases verification, validation and certification. The unit of partitioning is called a partition. A partition is basically the same as a program in a single application environment: it comprises data, code and its own context configuration attributes.
Partitioning separates applications in two dimensions: space and time. Spatial separation means that the memory of a partition is protected. No application can access memory out of the scope of its own partition. Temporal separation means that only one application at a time has access to system resources, including the processor; therefore only one application is executing at one point in time – there is no competition for system resources between partitioned applications.

ARINC 653 defines a static configuration where each partition is assigned a set of execution windows. The program in the partition associated with the current execution window gains access to the processor. When the execution window terminates, the program is pre-empted; when the next execution window starts, the partition associated to this one will gain access to the processor.
Processes within the scope of a partition are scheduled by a priority-based preemptive scheduler with FIFO order for processes with the same priority. This second level scheduler is invoked whenever an execution window assigned to its partition starts and the partition gains access to the processor. It is pre-empted by the first level partition scheduler when the execution window terminates.

ARINC 653 defines an API providing applications with a set of services. These services are partition-internal inter-process communication means, like events, message buffers, black boards and semaphores; time services; an interface to the health monitor; and interfaces to the partition associated with the application and the processes it consists of.

Figure 3: ARINC 653 API

The communication among partitions occurs through ports defined at partition level. A port is either incoming or outgoing; two behaviours are defined: a port may be a queue holding zero or more messages or a so called sample port containing the current instance of a periodically updated message.

3 Neutral Execution Platform

An Execution Environment is a software and/or hardware framework which enables software applications to run. Typical execution environments include hardware boards, operating systems, programming languages and runtime libraries. The specification of an execution platform covers its general properties and the API. An execution platform that is considered neutral does not specify how it has to be implemented on certain hardware architectures or operating systems. In the contrary, the execution platform is expected to behave identically on different hardware and different operating systems.
3.1 The Runtime

An execution environment is tightly coupled with the programming languages it supports. It aims at abstracting lower software and hardware layers as shown in the figure below.

![Execution Environment Layers](image)

A Run-Time System (RTS) provides support for the dynamic features of a given programming language, like memory allocation and concurrency functionality. The concurrency model can be mapped onto the RTOS model in two different ways. With the first way, called all-in-one, all threads are mapped on a single process of the underlying operating system whilst with the second, called one-to-one each thread is given a dedicated RTOS process. Whatever the adopted approach is, it must comply with the ARINC 653 concurrency model for avionics systems.

A Virtual Machine (VM) is more complex than a traditional RTS because it provides a full abstraction of the underlying operating system and hardware. The VM functions may include a code interpreter, dynamic loading of code etc.

The Middleware layer includes distribution and communication components like CORBA or RMI as well as data bases components. Middleware components are used by the applications via well-defined Application Programming Interfaces (API).

RTOS: Any Real-Time Operating Systems complying with the avionics ARINC 653 specification. RTOS provides memory, threads/processes, input/output, as well as file and communications management.

The Board Support Package layer (BSP) provides the low-level devices management.
### 3.2 Java Neutral Execution Platform

**Safety Critical Java Technology Profile** The JSR 302, Safety Critical Java Technology is an ongoing specification work that will define a profile for the use of Java in safety and mission critical applications.

In order to limit the amount of code to certify, SCJT will support a reduced version of the standard Java API. The exact list of classes and methods supported by the profile is not defined yet, but we can have a rough idea of what could be the standard class that are supported by looking at another minimalist profile which is the Java Micro Edition CLDC profile.

The JSR 302 is derived from the RTSJ specification for its real-time extensions. SCJT will reuse some classes from RTSJ but restricts their API (as for classes from the Java Standard Edition), and proposes new classes for new concepts like, for instance, the Mission class describing the Mission phase of program.

**Aonix PERC Pico Implementation** Aonix is an active member of the JSR 302, and proposes a virtual machine called PERC Pico that targets critical applications. As this virtual machine is developed concurrently with the JSR 302 Safety Critical Java Technology specification, PERC Pico as it is today does not necessarily match exactly what will be the final SCJT specification. Therefore, Aonix plans to adjust its implementation once the final JSR 302 specification is released to ensure full compliance.

**Compilation Model** The Java language has been designed to support byte-code interpretation and dynamic code loading, but nothing imposes it if the application doesn’t need it. Byte-code interpretation and dynamic code loading raise a lot of issues for certification. So in order to target safety critical application, PERC Pico adopts a traditional static compilation model, in which the Java application is compiled into binary code and statically linked on the host to obtain a traditional executable for the target system. Currently, PERC Pico generates C code from Java byte-code and then uses a standard C compilation model to obtain the binary machine code.

**Threading Model** PERC Pico supports the scheduling model proposed by the SCJT profile, which is fixed priority, FIFO within priority pre-emptive scheduling, with the priority ceiling emulation protocol as priority inversion control mechanism.

The implementation of this scheduling on top of APEX can be problematic. The main problem with APEX (and with many other operating systems), is the absence of the priority ceiling emulation protocol for locks. To make up for this limitation, and implement the Java scheduling model on top of APEX correctly, PERC Pico doesn’t use a one-to-one mapping for scheduling, so a Java thread is not equivalent to an APEX PROCESS. PERC Pico handles the scheduling of Java threads itself.
For communication with the operating system, PERC Pico supports a special kind of threads, called native threads. Native threads are scheduled by the operating system. They are not authorised to synchronise with normal Java threads, unless they transfigure themselves to be scheduled by PERC Pico.

In order to allow a coherent cooperation of the scheduler of PERC Pico with the scheduler of the operating system, PERC Pico creates one operating system thread (i.e. one APEX PROCESS) per Java priority level. The mapping of Java priority on APEX priority is configurable when building the application. This way it is possible to have native threads scheduled by APEX with a priority within Java priority range.

Memory Allocation  
Automatic Garbage collection technologies are not considered certifiable for the moment because the memory heap, modified concurrently by different threads and the garbage collector, are too complex for static analysis. The RTSJ specification has defined an alternative allocation mechanism for Java programs that is based on scoped memory areas.

With RTSJ Scoped Memory areas it is possible to design deterministic allocation and deallocation of objects. On the other hand, the analysis has to prove that the usage of scope and the assignment of references to scoped objects will not raise any runtime exception. This analysis can be very complex. The RTSJ specification only determines that the area will not be allocated from the current memory area, leaving the programmer in doubt about the success of this operation, especially in case of memory fragmentation.

PERC Pico has adopted a more restrictive approach, in which every scope is allocated on the Java stack in a strict LIFO (Last-In First-Out) order. This way the memory fragmentation is avoided, and the computation of the maximum memory usage of a program is reduced to the computation of the maximum stack usage which is a tractable problem.
PERC Pico introduces a series of annotations that allow the programmer to specify in which context, an object will be used. Instead of requiring the programmer to allocate every scope and to handle the scope change using the RTSJ API, PERC Pico automatically creates a local scope for every method, and uses the programmer’s annotations to determine where to safely allocate objects when a new operation is invoked.

Although the manipulation of scope is automated, the scope model of PERC Pico conforms to the RTSJ specification. The single parent rules of RTSJ’s scope are ensured by constructions with PERC Pico due to the strict LIFO stack model used for scope. The assignment rules of RTSJ are imposed to prevent dangling reference: an object can only reference objects allocated in the same or in an outer scope – with immortal memory as the outermost scope. By default, PERC Pico enforces these rules to be statically verified at compilation time.

PERC Pico verifier is conservative in the sense that it will by default try to allocate new object locally in the scope of the current method; but when this object is stored in a field, or passed to a method as an argument that has not been annotated, the verifier will consider the worst case scenario (this object might be referenced till the end of the program) and forces the object to be allocated in immortal memory. Immortal memory allocation is of course forbidden by default so that it is highly probable that a program that has not been annotated will be rejected by the Pico verifier.

The annotations imposed by PERC Pico for classes, fields, methods and arguments, allow the safe composition of software components that have been developed independently. The analysis is performed locally for every method, and only depends on the signature, extended with annotations, of referenced classes, fields and methods.
4 Interoperability Architecture

AIDA supports all available ARINC 653 services, part 1 (required services) as well as part 2 (extended services) and provides a seamless integration of new AIDA services with existing ARINC 653 services. The following picture provides an overview about some of the fundamental Interoperability concepts for AIDA high level architecture.

AIDA remote services design shall allow multi-supplier integration compliance with DO-297 recommendations such as ARINC 661 Cockpit Display System standard service architecture.

Modern avionic architectures request a decoupled and fault tolerant communication model in a system that is considered reliable and static. As a consequence, the AIDA architecture includes an asynchronous communication model (command / event based) that allows a location transparency of remote service implementation and the object-oriented support adapted for the AIDA needs.

Another characteristic found in avionics systems, and not usually found in other applications domains, is the extensive presence of health monitors. The AIDA proposed communication model takes advantage of that, not associating by default a status to a command.

One important peculiarity of the proposed architecture with respect to existing approaches, such as ARINC 661, is that AIDA does not impose the use of commands to update data. AIDA proposes the use of a publish / subscribe data centric mechanism with content filtered topic supported by standard ARINC 653 ports and channels for data communication support. The use of content filter mechanism will largely reduce AIDA inter partition communication required bandwidth (see the two figures below).
From an implementation point of view, inter-partition data communication is supported by middleware level data-centric services (publisher and subscriber services) accessing partition memory and distributing data over predefined ARINC653 standard channels.

In summary, the above vision does not affect the approach of keeping the high level architecture compliant with ARINC 653: inter-partition communication remains port-based even if a mechanism that enhances location-transparency and reduces system integrators' effort is used. Intra-partition communication remains based on ARINC653 API services supported by the capabilities of compliant RTOS.

### 4.1 AIDA Change Containment Concepts

In AIDA vision, partitions are considered the change containment units: based on the premise that avionics system integration is mainly based on data exchange, AIDA proposes an improvement to the current ARINC 653 inter-partition communication through the introduction of some CORBA and data-centric communication concepts.

AIDA vision considers ARINC 653 partitions and ports as the de facto software interface among applications in the aeronautic domain. Based on this premise, AIDA proposes the introduction of additional features to conventional ARINC 653 partition philosophy instead of investing in a completely new software component model.

Partitions provide a strong containment in two aspects:
- they allow applications with different software levels to share the same platform without requiring all (software) parts to be qualified at the highest level. Here, partitioning defines fault containment areas: an error in one partition cannot jeopardise the service provided by another partition;
- they allow qualification activities to be performed separately for each application, whatever their criticality levels are. The responsibilities of each function supplier are then clearly identified and bounded.

The proposed AIDA vision considers that traditional software reuse concepts are not enough for the avionics domain: the main focus in the avionic domain should not be limited to source code reuse but should include the certification credits and/or FTSO reuse. An architectural strategy to increase this type of reuse is reducing the impact of the change at partition level. In other words, an application/partition shall be considered a sort of software component. Reuse of certification credits implies a specification of software component usage.
domain, i.e., a set of constraints on the way it shall be used and constraints on the platform properties on which it is executed, and means to demonstrate that an implementation complies with this usage domain.

The AIDA approach to the above mentioned enhancement of certification credits reuse is represented by the adoption of configurable services and middleware that deal with all adaptations and filtering aspects needed when dealing with two existing applications. A service is activated by a command or event model, is supported by middleware capabilities and uses existing ARINC 653 inter-partition messages to communicate with remote locations. This approach will also promote the compatibility with already existing ARINC 653 applications. See the next figure for more details.

Figure 8: Interaction Concept

The separation of concerns between functional (e.g. an FMS) and IMA architecture related aspects (e.g. Redundancy) for a hosted application is the key to enabling a truly reconfigurable logic architecture. AIDA proposes the adoption of reconfigurable (multi-static) logic architecture. A configuration is a set of active software components and communication links including multiple A653-like XML configuration records.
The objective of such multi-static configuration is to

- enhance system availability when a HW fault is detected. Fault Management shall set the System Configuration to the minimum degraded configuration selected from authorised configurations;
- allow system reconfiguration to keep the minimum required redundancy without adding additional hardware to enhance dispatchability;
- enable the system to have a set of alternative logic configurations previously verified and validated;
- avoid the non certifiable aspects, associated with not predictable architectures.

4.2 Security Concepts

Security concerns related to the avionics domain were first addressed by the military sector. Classified military information on aircrafts or satellites must be protected from unauthorised access from the outside world. It is necessary to authenticate connections made from ground control and to map users with authorisations to access certain parts of the system. Authorisation has to deal with different roles, controllers may be assigned in system maintenance and control. An engineer or system administrator who is entitled to change system configuration or to execute system control tasks is not necessarily privileged to see the content belonging to payload applications. This means not only protection of the entire system against outside attackers is mandatory, but also the separation of the system parts from each other.

In civil avionics safety is addressed as a fundamental requirement which must be satisfied by on-board software. Scenarios from the military context show that also security issues must be solved to ensure that safety constraints hold. Airborne applications like navigation or collision avoidance systems rely on information from remote sources. Not only communication between ground and aircraft must be secured but also the information flow within the on-board avionics system. An avionics application with a high safety-level should not only protect itself from faults caused by other, less critical, systems but should also protect its data from manipulation and disclosure by other applications on board.

MILS These problems led to the development of a security design that resembles safety architectures like ARINC 653: Multiple Independent Levels of Security (MILS). MILS uses partitioning for security enforcement as ARINC uses partitioning for safety. A partition in MILS is understood as basic building block of informational segregation. The MILS component responsible for partitioning is called Separation Kernel (SK) which can easily be integrated with an ARINC kernel. The first step to security segregation is already made by ARINC: no application may access the memory of another partition. The MILS SK is stronger than the ARINC partitioning in one respect: ARINC 653 does not strictly forbid that a partition reads memory of another partition. An
A653 implementation that also follows the MILS architecture must follow the stronger MILS requirement and prevent read access to other partitions.

A more complex task is the protection of data leaving their partition walls. For this purpose the MILS Message Routing (MMR) component is designed. The MMR contains a privilege table defining which applications may communicate with each other and which information an application may gather from another. In ARINC 653 the messaging component may be seen as the complement to this MILS facility. The A653 messaging relies on a configuration which is defined before start-up and describes all possible communication paths within a system. This way, it is impossible that a partition sends or receives a message without authorisation. The picture changes with applications that force users to authenticate. Now, the actor, the sender or receiver of a message, is not necessarily a neutral application, but an actor with a role in the systems interaction. The respective role may have authorisation to access certain information or may not. In such a situation, the existence of a static routing description addressing partitions, ports and messages is no longer sufficient. A real access control is needed which identifies users and grants privileges to them.

The general access control performed by the MMR can already contain detailed information like client/server-configurations, message types and access levels. But it is not intended to enter protocol specifics of single applications. The MMR itself, being an OS component or a central system service, would become too complex if it integrated details of all possible types of applications and their communication protocols.

To extend the capabilities of the information flow control, made available by the MMR, MILS provides the concept of guardian applications. These guards are applications of their own, running in separate partitions, but dedicated to the protection of an application or a set of applications. Each application or set of applications may have a guard to validate the exit and entry of messages in the given context sender/receiver, contents, purpose etc. This way, security related tasks are removed from the applications and concentrated in specialised protection software.

The main goal of this architecture is a design of a system that is not obliged to trust the applications hosted on it. The ARINC partitioning treats applications as not to be safe that way it enables the co-existence of applications certified at different criticality levels without questioning the safety of the entire system or the applications with high safety levels. MILS treats applications as not to be trusted, making the co-existence of high and low classified information possible. Important is, that all data leaving or entering partitions is treated in this manner, including error messages or data garbage.
5 Development Means

The AIDA system development means have been requested to support a system Life Cycle, of a kind commonly used in the IT world and able to support various phases in order to allow: risk identification and resolution, evaluation of design alternatives, iterative development and verification, definition of a development plan.

The main key points of the defined AIDA Development tool chain are:

- adoption of the Model Driven Engineering (MDE) approach
- implementation of a tool chain which is well integrated with one central extensible development framework
- having a tool chain for the entire System development process covering:
  - Requirements Analysis and Top-Level Design
  - Architectural Design
  - Behavioural Design
  - Non-Functional Design of security, dependability and performance aspects
  - Code Development and Automatic Code Generation
  - Test Design and Automatic Test Generation
  - Debugging and Simulation Means.

AIDA concept proposes to couple an UML/SysML based Model-Driven Development (MDD) environment with a Model-Based Design (MBD) environment for dynamic behaviour and mode logic design. An example for MBD tool is Simulink, but in the philosophy of AIDA tool chain it can be replaced with some other tools in the future. We suggest this approach since embedded systems developers are faced with the challenge of developing systems that must not only meet functional requirements, but also achieve real-time performance, size, safety, testability, and specialised hardware control goals. Furthermore, certification needs must be considered. Typically, the way this problem has been managed is by selecting specific tool solutions to meet the unique needs of each stakeholder. Although effective, the single tool philosophy may not be enough to fully meet the demands placed on it. Combining environments can be a powerful strategy to solve the problem. Ideally, this approach allows users to exploit the best aspects of each tool to create a better solution by combining the power of the tools to create an environment that is collectively better than they would be individually. By combining these solutions, users benefit from an environment that covers the complete process, from requirements to specification, design, development, implementation and test.

See in the following figure a representation of the coupling among both MDD and MBD tools with the objective to implement an integrated tool chain.
Figure 9: MDD and MBD Coupling

Using a combination of MDD and MBD tools, engineers can ensure that all facets of the process are covered.

The combination of powerful MDD and MBD environments form a hybrid-modelling environment capable of capturing requirements, designing systems and software architectures, analysing dependability, schedulability and performance, and developing logical and mathematical algorithms while supporting multiple workflows.

6 DIANA CONSORTIUM

Supported by major avionic manufacturers as well as by recognise air framers the DIANA consortium has been built to represent the aeronautical community. This includes a relevant set of end-users, key players in the European and global markets, namely, Dassault Aviation, Alenia Aeronautica and Embraer (air framers/system integrators), and Thales Avionics, major provider and system integrator of avionics applications.

The technological solutions will be based on the software scientific know-how from Universities - Budapest University of Technology and Economics and University of Karlsruhe - which have proven scientific skills in the areas of MDE, Safety Critical Java and OO Oriented Development, while the Dutch Aeronautics Research Institute, NLR, will provide its well known expertise in the areas of aeronautics, certification, simulation and system integration.

The definition and design of the solutions will be supported by the software provider partner companies, namely Skysoft Portugal, Alenia SIA and Aonix.
Alenia Aeronautica is the Italian leader in aeronautics and a major European player in the aerospace industry, with full system integration capabilities.

It designs and builds high-performance combat aircraft, military and commercial transport aircraft and advanced structures for commercial aircraft, and offers global customer service.

AleniaSIA is the company resulting from the merger, completed at the end of 2005, between Società Italiana Avionica S.p.A. and Teleavio s.r.l.

The major market areas are: system analysis, design and development of environment to support integration, simulation and training, embedded software development, integrated logistic support, flight control systems, flight management systems.

AleniaSIA is certified UNI EN ISO 9001:2000 and employs more than 330 engineers (more than 50% of them with University degree).

With two decades of experience in delivering tools for the most demanding and rigorous applications around the globe, Aonix is at the leading edge of real-time and embedded software solutions. Java developers and Ada developers use our tools to build reliable and scalable applications in avionics, aerospace, telecom, network infrastructure, telematics, industrial automation, military, transportation, office automation, and automotive industries.
The Fault Tolerant Systems Research Group at Budapest University of Technology and Economics Department of Measurement and Information Systems (BUTE DMIS) has been founded in 1994; currently it consists of 30 members, including 5 PhDs.

The main research field of the group is the model-driven development and analysis of dependable computer systems: systems modelling and formal analysis, dependability assessment, verification and validation of IT systems, analysis and synthesis of IT infrastructure, system optimization and dependability consolidation, service oriented computing and integration, model transformation.

Dassault Aviation is one of the major players in the global aviation industry, with a presence in more than 70 countries across 5 continents, whose varied activities cover several areas of high technology. With 8500 people, the Dassault Aviation's primary vocation is the design, development, production, sale and maintenance of aircraft.

Embraer is a customer oriented company with nearly 38 years of experience in designing, manufacturing, selling and supporting aircraft for the global markets of Commercial Aviation, Executive Aviation and Defense and Government, with headquarters in So Jos dos Campos, Brazil, and offices, subsidiaries and customer service bases in China, France, Portugal, Singapore and the United States. As of March 31, 2007, Embraer had a total workforce of 21,005 employees, and its firm order backlog totaled US$ 15.0 billion.

The Dutch National Aerospace Laboratory (NLR) carries out applied research on behalf of the aviation and space sectors.

NLR is an independent technological institute. NLR performs research to develop new technologies for aviation and space travel, not only from a scientific perspective, but also for the application of this research in industrial and governmental sectors.
Skysoft is a systems and software house, located in Lisbon, with over a decade of experience in the aeronautics, space, and telematics markets. It currently counts with a staff of around 80 persons with experience in international aeronautics and space programmes.

Thales is a leading international electronics and systems group, serving defence, aerospace and security markets worldwide, supported by a comprehensive services offering. The groups civil and military businesses develop in parallel to serve a single objective: the security of people, property and nations. Leveraging a global network of high-level researchers, Thales offers a capability unmatched in Europe to develop and deploy critical information systems. Thales employs 60,000 people in 50 countries and generated revenues of 10.3 billion in 2005, with a record order book of over 20 billion.

As the oldest technical university in Germany, founded in 1825, the profile of Universität Karlsruhe is determined primarily by the technical and natural sciences as well by engineering.

The Department of Computer Science (Fakultät für Informatik) was founded in 1972 as the first department of its kind in Germany. With its presently 1500 students, 26 professors and 180 scientific staff it has since then grown into one of the largest departments of the university, and one of the largest informatics department in Germany. Its characteristic is the unparalleled broad scope in teaching and research, ranging all the way from computer technology and architecture to theoretical computer science, telematics, software techniques and systems, parallel and distributed systems, to applications mainly in the engineering sciences.