From ARTEMIS Requirements to a Cross-Domain Embedded System Architecture

R. Obermaisser, H. Kopetz

Vienna University of Technology, Real-Time Systems Group

http://ti.tuwien.ac.at/rts/people/obermaisser

Abstract: This paper gives an overview of the cross-domain component-based architecture GENESYS for embedded systems. The development of this architecture has been driven by key industrial challenges identified within the ARTEMIS Strategic Research Agenda (SRA) such as composability, robustness and integrated resource management. GENESYS is a platform architecture that provides a minimal set of core services and a plurality of optional services that are predominantly implemented as self-contained system components. Choosing a suitable set of these system components that implement optional services, augmented by application specific components, can generate domain-specific instantiations of the architecture (e.g., for automotive, avionic, industrial control, mobile, and consumer electronics applications). Such a cross-domain approach is needed to support the coming Internet of Things, to take full advantage of the economies of scale of the semiconductor industry and to improve productivity.

Keywords: System Architecture, ARTEMIS, Composability, Robustness, Diagnosis, Cross-Domain Architecture

1. Introduction

Embedded systems enable the real-time computer control of physical devices and systems, ranging from mobile phones, to television sets, to automotive engines and to industrial robots (to take a few examples) in order to achieve an unprecedented level of performance and utility. In all of these domains the demands on the functional capabilities and the dependability of and consequently the opportunities for embedded systems grow rapidly as society moves further into the information age.

In this context, the ARTEMIS Joint Technology Initiative (JTI) was launched by the European Commission. Within ARTEMIS major European embedded system suppliers, OEMs and research organizations created a Strategic Research Agenda (SRA) with key challenges for more advanced embedded systems. Although the participants in ARTEMIS belonged to a broad range of application domains, many common challenges emerged. Composability and robustness are examples of cross-domain challenges that exist in embedded systems ranging from consumer electronic and mobile systems to safety-critical applications in the transportation sector.

Despite these cross-domain challenges, today's landscape of embedded system architectures is highly fragmented. Different system architectures have evolved in different application domains such as Integrated Modular Avionics (IMA) [1] in the avionic domain, the AUTomotive Open System ARchitecture (AUTOSAR) [2] in the automotive domain or the Network on Terminal Architecture (NoTA) [3] for consumer applications.

By advancing from the different domain-specific architectures towards a cross-domain embedded system architecture, this fragmentation can be avoided. Thereby, a significantly larger market for embedded system technologies (e.g., semiconductor devices, embedded components, software and tools) emerges. A large market and the resulting ability of exploiting the economies of scale is of particular importance for the semiconductor industry where the engineering costs associated with the design and tooling of chips grows rapidly. According to the International Technology Roadmap for Semiconductors (ITRS) the production of masks and exposure systems is becoming a bottleneck for the development of chips with finer geometries [4]. For example, consider the design investments in the Cell Broadband Engine (CellBE) processor, which amount to 400 million USD [5]. The cross-domain usability helps in amortizing such non-recurring costs.

In addition, a consolidation of embedded system architectures is needed to facilitate the interoperability of systems from different domains. Consider for example the electronic systems in a modern car, which encompass control subsystems, multimedia subsystems and gateways to mobile systems [6]. In particular, a consolidation is required to support the integration of systems into the Internet-of-Things.

Based on these insights, the European research project GENESYS (GENeric Embedded SYStem Platform) was launched, which took the ARTEMIS SRA challenges as a starting point and developed a candidate of a domain-independent reference architecture for embedded systems. By

promoting a strict *component-based* design style and identifying services that can be deployed in different application domains, the design and production costs of new applications can be significantly reduced by reusing existing components and taking advantage of the enormous economies of scale of the semiconductor industry.

GENESYS is a platform architecture that provides architectural services as a baseline for the development of applications. The architectural services consist of a minimal set of core services, which are domain-independent and deployed in every instantiation of the architecture. On top of the core services, optional services can be implemented as system components on top of the core services. These optional services, which can be domain-specific, are used to customize the GENESYS architecture for different applications.

The remainder of the paper is structured as follows. Section 2 gives an overview of the ARTEMIS requirements. The GENESYS cross-domain architecture, which facilitates solutions to these challenges, is the focus of Section 3. Section 4 describes the core architectural services, which are mandatory in every instantiation of the GENESYS architecture. Examples of domain-independent optional services are discussed in Section 5. Section 6 provides examples of domain-specific optional services. The paper concludes with a discussion in Section 7.

2. ARTEMIS Requirements

The European Technology Platform on Embedded Systems, ARTEMIS, has proposed a Strategic Research Agenda (SRA) that outlines the most important areas requiring research in the context of embedded systems. Central to the ARTEMIS SRA is the concept of pan-application domain relevance of embedded systems research: the avoidance of further fragmentation in tools and techniques and multiple re-inventions of the wheel. One major axis of the SRA is “generic reference designs and architectures”, i.e., reference architectures that can be used across multiple application domains.

The following key challenges have been identified for such a reference architecture [7]:

1. *Composability.* Architectures must support the constructive composition of large systems out of components and subsystems without uncontrolled side effects.
2. *Networking and Security.* Subsystems have to communicate with each other and with the external environment at multiple levels (e.g., network-on-chip, cluster-level, local-area networks), observing given timing and dependability constraints. Also, novel approaches are required to deal with malicious attacks from intruders, to maintain the confidentiality of sensitive data and to protect intellectual property.
3. *Robustness.* This challenge comprises the provision of an acceptable level of service despite the occurrence of transient and permanent hardware faults, design faults, imprecise specifications, and accidental operational faults.
4. *Diagnosis and Maintenance.* Architectures should support the reliable identification of failed subsystems in order to autonomously recover from transient failures and guide untrained users in the replacement of defective subsystems with permanent failures.
5. *Integrated Resource Management.* Architectures need to support an integral management of resources (e.g., power, time, communication bandwidth, memory).
6. *Evolvability.* Architectures shall support the evolution of systems, driven by novel hitherto unknown user requirements, and the need to integrate new technological capabilities.

3. GENESYS Cross-Domain Architecture

This section gives an overview of the GENESYS cross-domain architecture. First the strict component-orientation of GENESYS is discussed. In addition, we provide information on the structuring of architectural services, which serve as a baseline for the development and integration of autonomous component services.

3.1 Component-Orientation

From the hardware and software points of view, the GENESYS architecture follows a strict component orientation. In GENESYS, a component is a self-contained hardware-software subsystem that is used as a building block in the design of a larger system. It encapsulates the implementation and hides the implementation details. The provided services are exposed via a set of message-based interfaces. Since a component hides the (complex) internal structure from its user, it offers an appropriate unit of abstraction for system design. Therefore, it helps to tackle the challenge of design complexity in large embedded systems.

The GENESYS architecture distinguishes between components in the logical view and the physical view. In the logical view a system consists of *jobs* that capture the behavior along with non functional properties while abstracting from the physical implementation platform (e.g., an FPGA, a general purpose CPU, the physical allocation). In the physical view of a system, three integration levels are distinguished: chip-level (level L1), device-level (level L2) and system-level (level L3). At the *chip-level*, the components of the physical view are *IP cores*. At the *device level*, the components are *chips*. The components at the *system-level* are *devices*.

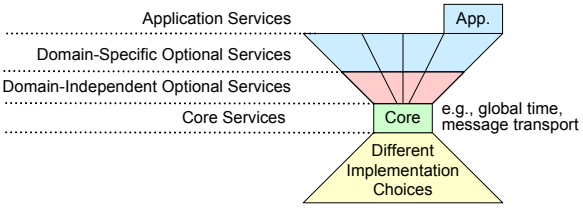

Figure 1: Service Hierarchy in GENESYS

3.2 Architectural Services

The capabilities of the platform are offered to the components, which realize the application services, by a set of architectural services. These architectural services are hierarchically structured as shown in Figure 1. At the bottom of this service hierarchy are the *core services* that define the platform. The core services must be present in every instantiation of the GENESYS architecture. Above these core services are the *domain-independent optional services* that provide enhanced capabilities to the users of the platform in order that functionalities that are needed in many, but not all application domains, are provided in a ready-to-use form. It is up to the user to decide, which one of these optional services are to be included in a concrete instantiation of the architecture. The implementation of these optional services uses the core services. At the next level of the hierarchy are the *domain-specific services*, which are highly specialized optional services of particular domains, followed by the *application services*, as shown in Figure 1.

The *optional architectural services* refine the core services for individual applications and application domains. Based on the core services, a broad range of higher-level *optional services* can be realized which are tailored to the requirements of specific application domains and refine or extend the core architectural services. The GENESYS architecture is intended to be used across multiple heterogeneous application domains with highly specific and potentially mutually contradicting requirements with respect to the architectural services. Examples of different requirements are the desired properties of communication protocols: Safety-critical applications (e.g., X-by-wire systems) have to deliver their application services in all specified load and fault scenarios. Therefore, safety-critical applications typically use communication services that are highly deterministic in the temporal domain (e.g., time-triggered protocols). In contrast, many non safety-critical systems have to be optimized for average load in order to reduce cost (e.g., in a multimedia system) and thus require flexible architectural services.

Additionally, optional architectural services have to be provided to support the reuse and the integration of legacy applications by emulating the corresponding legacy platforms. Examples are emulations of standard execution environments (e.g., a CAN [8] communication stack for automotive applications, a KHRONOS OpenMAX [9] middleware for multimedia applications).

Consequently, the set of optional architectural services is *extendable* and *configurable*. Extendability is an important factor since it is not possible to anticipate how applications and their requirements will evolve in the future.

Configurability is required to facilitate the construction of resource-efficient systems. Therefore, the architecture is adaptable in such a way, that any particular instantiation contains only those services that are required in the actual system.

4. Core Services

The core services are mandatory in every instantiation of the architecture, since they form the foundation for all higher-level services. The core services can be grouped into four categories (cf. Figure 2): the basic time services, the basic communication services, the basic configuration services and the basic execution control services.

4.1 Basic Time Services

The basic time services establish a *global time base*, which allows the temporal coordination of distributed activities (e.g., synchronized messages), as well as to interrelate timestamps assigned at different components.

The basic time services provide to each component a counter value that is globally synchronized within the given integration level. If the counter value is read by a component at a particular point in time, it is guaranteed that this value can only differ by one tick from the value that is read at any other correct component at the same point in time. This property is also known as the reasonableness of the global time [10], which states that the precision of the local clocks at the components is lower than the granularity of the global time base. Optionally the global time base can also be synchronized with an external time source (e.g., GPS or the global time base of another integration level). In this case, the accuracy denotes the maximum deviation of any local clock of the integration level to the external reference time.

4.2 Basic Communication Services

The basic communication services enable the components to send unidirectional multicast messages on communication channels. The basic communication services support three communication modes: time-triggered, event-triggered and streaming communication.

Time-triggered communication provides a predictable time-guaranteed messaging service, where the instants of message transmissions are specified by

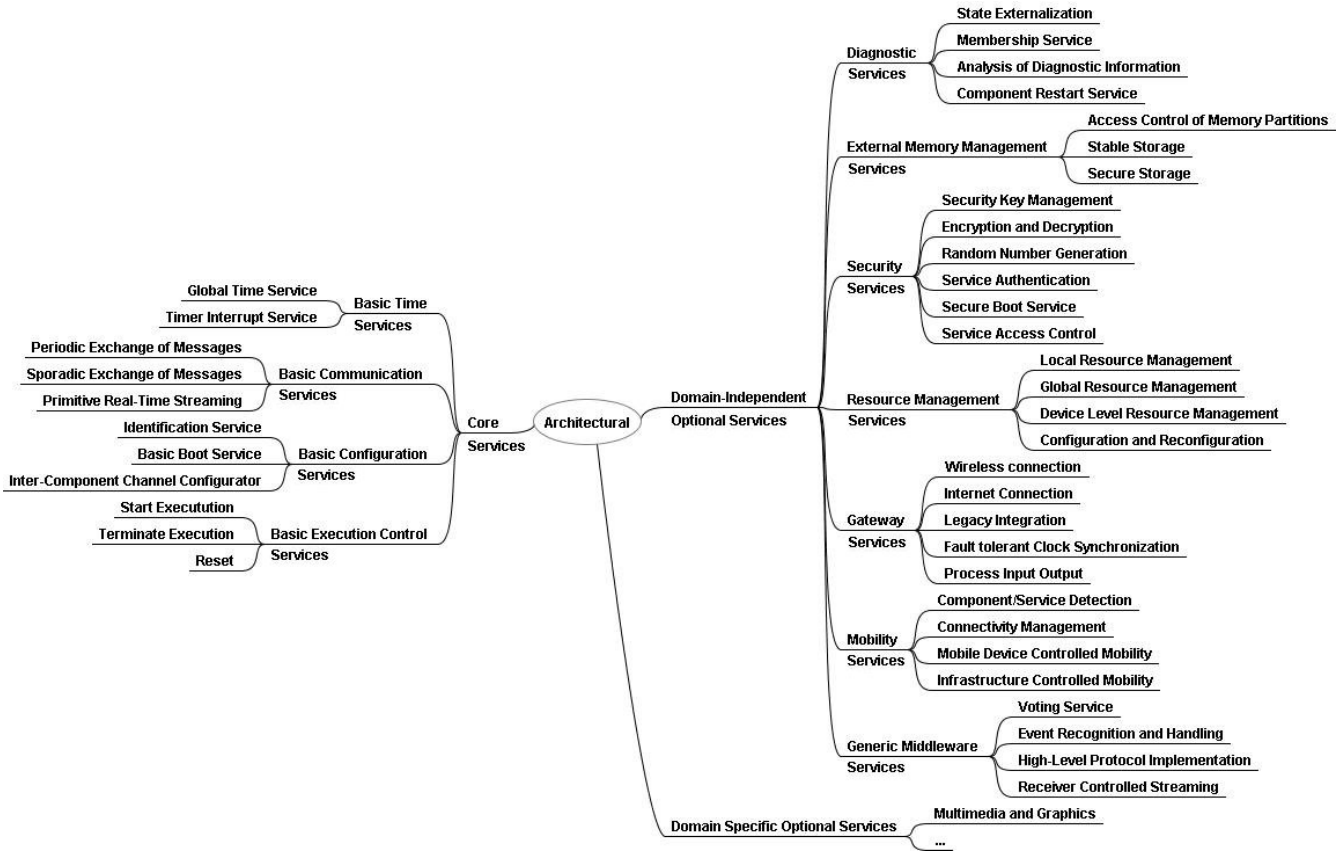

Figure 2: Architectural Services

an a priori planned conflict-free communication schedule. For time-triggered messages, the communication infrastructure is deterministic and guarantees temporal properties such as latency, latency jitter, bandwidth, and message order.

The second communication mode is *event-triggered communication*. Event-triggered messages arrive sporadically and must be queued at the sender and at the receiver. They are characterized by bandwidth, sender queue length, receiver queue length, and minimum inter-arrival time and provide a flexible communication service. The transmission of event-triggered messages can be triggered by any significant event, i.e., not necessarily by time events. Due to the temporal unpredictability of the events that trigger event-triggered messages, this communication mode offers weaker temporal guarantees, e.g., best-effort communication or probabilistic statements about the bandwidth or latency.

Finally, the *primitive streaming communication* enables the transmission and on-the-fly processing of a sequence of variable-size data elements (e.g., frames in a video stream) with corresponding temporal properties (e.g., average data rate with bursts). Similar to the sporadic message exchange service, this service uses also queues at the sender and receiver side. Queues compensate for irregular data rates that are typical for streaming applications (e.g., MPEG4). The primitive streaming service does not exercise any flow-control from the receiver to the sender.

4.3 Basic Configuration Services

The basic configuration services are needed to introduce the components to the platform and to connect the ports of the components for the *establishment of the communication channels* among the components.

The basic configuration services establish naming and connect/disconnect the ports and communication channels of the components according to a schedule that is provided to the basic configuration services by a scheduler. The basic configuration services determine the validity of the supplied schedule by checking whether collisions occur on the underlying communication medium (e.g., a network-on-a-chip at L1) and by evaluating a set of safety assertions. An example of a safety assertion would be a check whether the communication channels of the safety-related components are unmodified.

If the schedule is found to be valid, the basic configuration service sets the protected communication parameters of the involved components. These pro-

tected communication parameters designate the instants w.r.t. the global time base for the time-triggered message transmissions, the minimum inter-arrival times for event-triggered transmissions, the maximum duration of a transmission, the type of message and the address of the receivers/sender.

In addition, the basic configuration service provides a *unique identifier* of the hardware under consideration (e.g., chip at L1, device at L2). This identification is tamper resistant as needed to inform the boot service about the type of available hardware and the attributes of the unit under consideration.

4.4 Basic Execution Control Services

The basic execution control services are used to control the execution of a component. Execution control is realized by sending an appropriate message to a local resource manager of the respective component. It is assumed that in every component there is such a local resource manager that accepts and executes the messages from the basic execution control service.

The basic execution control consists of the following three commands:

(1) *Execute Requests* is a message requesting a component to start its execution at the next restart instant with the restart state that is contained in this message. In case this restart state is empty, the component will restart with the static restart data that are contained in its core image.

(2) *Terminate Request* is a message requesting a component to terminate its execution. The execution can be restarted with an Execute Request.

(3) *Reset Request* is a message that is interpreted directly by the component hardware without any control by the system or application software. It resets the component hardware and starts the execution until the point where an ExecuteRequest is awaited.

5. Domain-Independent Optional Services

These services build upon the core services and are generic in the sense that they can be used in multiple application domains. These services are optional in the sense that they are not required in every instantiation of the architecture. If needed, developers can pick them out of component libraries with existing, validated architectural services for the different levels of integration. The set of domain-independent optional services is open, i.e., new optional services can be added if the need arises.

At present, the domain-independent optional services are grouped into the following categories (cf. Figure 2): diagnostic services, external memory management services, security services, resource management services, gateway services, mobility services, and generic middleware services.

In the following, the domain-independent services are exemplified using two of these categories, *diagnosis* and *security*, which represent two important cross-domain challenges.

5.1 Diagnostic Services

Many application domains – ranging from industrial systems to ambient intelligence applications – require the ability for monitoring the behavior of components in the value and time domains [7]. The reliable identification of failed components can be used for the autonomous recovery of the system service in case of a transient failure, and provide guidance for the physical replacement of defective components in case the failure is permanent.

The GENESYS architecture provides a number of diagnostic services, which are implemented in a self-contained diagnostic component. The message multicasting as described in Section 4.2 supports the non-intrusive observation of the component behavior by the diagnostic component.

State externalization is a service in this category, which periodically externalizes the internal component state in a ground state message at predefined cyclic recovery instants. State externalization serves for state validity checking, taking global state snapshots, state exchange, logging and power gating. In particular, state externalization is important for the fast recovery of components affected by transient faults. The externalized information can be used to 1) allow error detection and/or 2) enable checkpointing and retry mechanisms.

Based on the externalized information, a *membership service* provides a vector with a coherent global view of the operational status of the components [11]. The membership service simplifies the provision of many application algorithms, since distributed components can rely on the consistency of diagnostic information and react in a coordinated manner to detected failures.

Another member of the diagnostic services is responsible for the analysis of diagnostic information. The *diagnostic analysis* service checks the incoming ground state messages to find anomalies and records these anomalies in a local database. Additionally, error messages that are received from the components are examined. The application component implementations must ensure that all errors, even if they are masked by redundancy, are faithfully reported to the diagnostic component. An application specific diagnostic model analyzes all this data base information to assist the maintenance engineer in finding faulty components. Particular emphasis is placed to find out whether a transient, an intermittent or a permanent failure is present in a component, since the maintenance strategy is different for each one of these faults.

The *component restart service* resets and restarts a

failed component with an internal state that is expected to be acceptable at the next future restart instant. The component restart service is also used to restart and perform state restoration in case of power gating.

5.2 Security Services

Security becomes a key issue when the connectivity of systems is increased beyond their traditional system boundaries (e.g., connecting parts of the in-vehicle electronic system of a car via gateways to the Internet in order to enable in-system updates). In addition, security becomes more and more important as consumer devices are used increasingly for identification, authorization, payment as well as for storage of sensitive personal information.

The GENESYS architecture provides by design mechanisms for the security related isolation of a job. The physical separation of the IP cores together with the one-to-one mapping between IP cores and jobs ensure data separation of different jobs. The restriction of the inter-component communication exclusively to message passing ensures by design that no hidden channels among components of different confidentiality can exist that bypass the message passing mechanisms.

A basic identification service – the provision of a tamper-resistant unique identification of any GENESYS MPSoC is part of the core services. Using this core service, a dedicated optional security component can be provided that makes available enhanced security services, such as secure key management, encryption and decryption of all messages that leave or enter the MPSoC and a secure boot service.

The objective of the *secure key management* is to establish a chain of trust. Keys are required for many cryptographic functions. In many cases the security of the system depends on the secrecy of the employed keys. Therefore, the keys should not be stored in plain text in the memory or on a disc, but have to be sealed in a cryptographic envelope. To operate on such an envelope a non-encrypted key is required, which is usually called the root key. The root key serves as the starting point for the chain of trust.

The secure key management service embeds the root key in a tamper resistant memory which guarantees that no other service or job can access it in plain text. Whenever a job or any optional service wants to access a key contained in an envelope (protected by the root key) it has to issue a decryption request to the secure key management services for that specific key. If the request is granted, the security key management service will decrypt the requested key in a protected region that can only be accessed by the security key management service itself. In the next step it delivers the decrypted key securely to the requester. The granting of decryption requests depends on the employed security policies (e.g., one possibility would be that only jobs authenticated by the secure boot service may have access).

The *encryption and decryption service* provides the basic algorithms for the encryption and decryption of messages given that the trusted keys are available.

The *random number generation service* generates true random numbers from a physical process as required in both symmetric and asymmetric cryptography for generating keys.

The *secure boot service* ensures that the binary image of the job itself that is downloaded from a trusted server has not been modified by an intruder. It performs an authentication and integrity check on the job and it admits the job's execution only if the checks succeed.

The GENESYS architecture also provides application developers with basic mechanisms to access the services provided by the architecture as well as the application services built on top of the architectural services. *Service authentication* is performed when there is a request to register a service. Components that intend to use a registered service need to invoke an *access control* service in order to acquire the necessary access rights.

6. Domain-Specific Optional Services

The domain-specific optional services are services that are specialized for a considered application domain. The do-main-specific optional services build on the core services and a well-defined subset of the domain-independent optional services. In the following, two examples of do-main-specific optional services are explained.

6.1 CAN Communication and Gateway Service: A Domain-Specific Optional Service for Automotive Systems

The Controller Area Network (CAN) protocol [12] is widely deployed in the automotive domain. For example, present day cars contain multiple CAN buses deployed in different subsystems such as comfort or powertrain [13, 14].

The *CAN communication and gateway service* is a domain-specific optional service for the automotive domain. This service enables components to interact using the CAN protocol. The mechanisms of the CAN protocol, such as message prioritization based on identifiers, acknowledgments, bidirectional request/reply patterns, are implemented on top of the communicated modes of the core architectural services. For this purpose, existing solutions for the establishment of CAN based on a time-triggered communication service [15] can be exploited in the GENESYS architecture.

In addition, this domain-specific service supports the interconnection between a CAN bus and a communication network supporting the three communicated modes introduced in Section 3.2. For example, messages from a network-on-a-chip at L1 can be redirected to a CAN bus at L3. This gateway capability involves the provision of a physical interface (mechanical and electrical) that is compatible to CAN [12], the address mapping from the internal addresses of the chip to the CAN identifiers and the protocol translation between the network-on-a-chip and the CAN bus.

6.2 Multimedia and Graphics Services: Domain-Specific Services for Consumer Applications

Multimedia and graphics is a mature domain, where solutions, standards and Application Programming Interfaces (APIs) exist that can be established in the GENESYS architecture. However, the core services of the GENESYS platform have to be available such that the domain-specific optional services for multimedia and graphics application services can be easily implemented.

Examples of domain-specific multimedia services supported in GENESYS are the services defined by the Khronos Group, and in particular the OpenMAX standards. The OpenMAX includes APIs at three abstraction levels (application, integration and development layers).

The *application layer* facilitates the capture and presentation of audio, video and images in multimedia applications on embedded and mobile devices. It includes the ability to create and control player and recorder objects and to connect them to configurable inputs and output objects including content readers/writers, audio inputs and outputs, display windows, cameras, analog radios, LEDs, and vibra devices. The *integration layer* defines a media component interface to enable developers and platform providers to integrate and communicate with multimedia codecs implemented in hardware or software. The *development layer* contains a comprehensive set of audio, video and imaging functions that can be implemented and optimized on new CPUs, hardware engines, and DSPs and then used for a wide range of accelerated codec functionality such as MPEG-4, H.264, MP3, AAC and JPEG.

7. Discussion

Using the GENESYS architecture, significant improvements concerning time-to-market, cost, and robustness for a wide range of applications – from mobile phones to safety-critical computers in airplanes – are possible.

Through its architectural principles (e.g., strict component orientation) and the generic architectural services, the GENESYS architecture contributes towards a cross-domain solution for the ARTEMIS challenges. In particular, GENESYS provides solutions to the following three challenges:

- ***Composability***: GENESYS supports the straightforward composition of system out of independently developed components. System components can realize the generic optional services, while application components offer the application services. Thereby, the GENESYS architecture facilitates the establishment of a component-market for multiple domains. For example, a given security or networking component might be deployed in avionic as well as automotive and industrial control applications. Thereby, the transition from separate sectoral, vertically structured markets to a horizontally structured market is supported. The cross-domain GENESYS architecture results in a high volume market for generic components that can be reused in different application domains (i.e., automotive, avionics, industrial control, medical).

Furthermore, the strict component orientation leads to systems that are decomposed into nearly autonomous parts with clearly defined interfaces. The GENESYS architecture reduces the complexity when integrating components by limiting the component interactions to occur exclusively via messages exchanged over the component interfaces (cf. basic communication services in Figure 2: periodic, sporadic and streaming communication). Also, the availability of a common notion of time as a core service enables the temporal specification of component interfaces in a way that the interactions of components in a distributed system can be temporally coordinated.

- ***Robustness***: GENESYS supports the robustness of embedded systems by establishing a framework for fault containment and error containment, the selective restart of components that have failed after a transient fault, and the masking of transient and permanent errors by the replication of components. The waistline architecture encompasses optional services for voting (e.g., in a Triple Modular Redundancy (TMR) configuration), thereby performing active redundancy transparently to the applications. Furthermore, the global time, the clear notion of state, and the basic communication services support determinism as a basis for voting on a bit-by-bit basis.

- ***Integrated Resource Management and Energy Efficiency***: GENESYS provides for energy efficiency by an integrated resource management that makes it possible to individually reduce the power-requirements of components or to turn off components completely that are not needed during a particular interval (power gating). In addition to the basic configuration services, advanced dynamic resource management services can be provided by dedicated resource management components.

• ***Technology Obsolescence Management***: In a number of scenarios (e.g., aerospace industry, process control) the embedding system (the airplane or plant) has a significantly longer life-cycle than the embedded system. The component-based style of GENESYS makes it possible to modify the technology of a component implementation without a modification of the message-based component interfaces and the environment. In addition, the well-defined core services abstract from the implementation technology of the platform and permit changes of the platform without an impact on the application.

8. Acknowledgments

This work has been supported in part by the European research project GENESYS under project Number FP7/213322 and by the project INDEXYS under the Grant Agreement ARTEMIS-2008-1-100021.

References

[1] ARINC, "ARINC Specification 651: Design Guide for Integrated Modular Avionics," Aeronautical Radio, Inc., 2551 Riva Road, Annapolis,Maryland 214011991.

[2] AUTOSAR, "AUTOSAR - Technical Overview V2.0.1," AUTOSAR GbR2006.

[3] K. Kronlöf, S. Kontinen, I. Oliver, and T. Eriksson, "A method for mobile terminal platform architecture development," in *Advances in Design and Specification Languages for Embedded Systems*, 2007, pp. pages 285-300.

[4] Semiconductor Industry Association (SIA), "International Technology Roadmap for Semiconductors - 2006 Update," 2006.

[5] M. Day and P. Hofstee, "Hardware and Software Architectures for the CELL BROADBAND ENGINE processor," in *CODES+ISSS Conference* Austin, Texas, IBM Systems & Technology Group, 2005.

[6] D. Albero, "Automotive Forum. Electronics Technologies and Trends in Automotive Field," Fiat Group, Torino 2008.

[7] ARTEMIS, "Strategic Research Agenda. Reference Designs and Architectures. Constraints and Requirements. Report Version 13. ," 2006.

[8] Robert Bosch Gmbh, "CAN Specification, Version 2.0," 1991.

[9] Khronos Group, "OpenMAX IL 1.1.2 Specification," 2008.

[10] H. Kopetz and R. Nossal, "Temporal firewalls in large distributed real-time systems," in *6th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS '97)*, 1997.

[11] F. Cristian, "Reaching agreement on processor-group membership in synchronous distributed systems," *Distributed Computing,* vol. 4, pp. 175-187, 1991.

[12] International Standardization Organisation, "ISO11898 - Road vehicles - Controller area network (CAN)," 1993.

[13] J. Erjavec and R. Scharff, *Automotive Technology: A Systems Approach*, 5th ed.: Delmar Cengage Learning 2009.

[14] B. Heppner and H. Brauner, "Assessment of whole vehicle behaviour by means of simulation," Daimler AG,2008.

[15] R. Obermaisser, "Reuse of CAN-based Legacy Applications in Time-Triggered Architectures," *IEEE Transactions on Industrial Informatics,* vol. 2, pp. 255-268, 2006.