



GENESYS

GENeric Embedded SYStem Platform

Report on Analysis for Architectural Requirements

D 6.1

Project	GENESYS	Contract Number	FP7-213322	
Document Id	6.1-001-1.0r	Date	31-03-2008	Deliverable D 6.1
Contact Person	R. Obermaisser	Organisation	TU Vienna	
Phone	+43 1 5880118216	E-Mail	romano@vmars.tuwien.ac.at	

Contributors

Name	Company	Part affect	Date
H. Kopetz, R. Obermaisser	TU Vienna	all	2008/02/04
Perez Jon, Antonio Perez	Ikerlan	Sect. 3 and 4	2008/02/14
Philippe Millet	Thales	Sect. 4.2	2008/02/15
Philippe Millet	Thales	Sect. 4.2	2008/02/19
Ricardo Sanz	UPM	Sect. 3 and Sect. 4.3	2008/02/18
Klaus Kronlof	Nokia	Sect 3 and Sect 4.5	2008/02/25
Roman Obermaisser	TU Vienna	Sect 4.4, Sect 4.5	2008/02/25
Simon Bliudze	VERIMAG	Sect 3	2008/02/27
Valentin Gherman	CEA	Sect 3	2008/02/29
Roman Obermaisser	TU Vienna	Sect 3, Sect 4.4	2008/03/04
Roman Obermaisser	TU Vienna		2008/03/17

Change History

Version	Date	Reason for Change	Pages Affected
0.1	2008/02/04	document creation	all
0.2	2008/02/18	input from Ikerlan, Thales (requirements and existing architectures)	
0.3	2008/02/25	input from UPM, incorporation of feedback from Nokia	
0.4	2008/02/29	input from CEA and VERIMAG	
0.5	2008/03/04	draft responsibilities included for all requirements	
0.6	2008/03/17	update of integration levels	

Contents

1	Introduction	6
1.1	Attributes of Requirements.....	6
2	Definitions.....	8
2.1	Component.....	8
2.2	Composability.....	8
3	Requirements	9
3.1	Precise Interface Specification.....	11
3.2	Stability of Prior Properties.....	14
3.3	Non-Interfering Interactions.....	18
3.4	Fault- and Error-Containment	20
3.5	Fault Masking.....	22
3.6	Requirements w.r.t. Composability for the Development Process	25
3.7	Components with Heterogeneous Technologies	26
4	State-of-the-Art and Analysis of Existing Architectures	30
4.1	Automotive Architectures	30
4.2	Avionics Architectures.....	30
4.2.1	Current state-of-the-art.....	30
4.2.2	Advances beyond the state-of-the-art	31
4.3	Industrial Architectures.....	33
4.4	Mobile Architectures.....	33
4.5	Multimedia Architectures.....	34
5	References.....	36

1 Introduction

This document elaborates on composability requirements of the GENESYS architecture. It is structured as follows. The first section explains the format for capturing requirements. In the second Section we define the basic concepts of a component and composability. In the third Section we list requirements for composability. In particular, we explain necessary requirements that must be satisfied by an architecture that supports composability. The fourth Section analyzes different existing architectures from the point of composability and looks to what extent the selected architecture meet the five composability requirements. The document closes with a critical evaluation of the findings.

1.1 Attributes of Requirements

We have chosen the following uniform form for capturing requirements: Each requirement has mandatory attributes which are "ID", "Name", "Responsibility", "Description", "Rationale", "Architectural Significance", "Application Significance", "Integration Level", and the optional attributes "Relations", "Domain", "Additional Classification", and "Additional Information".

The identifier (**ID**) is unique throughout this document and is defined according to the following format: <WP number>_<sequence number>.

The **Name** is also unique throughout this document, and acts as a synonym for the ID, which might be longer but easier to memorize.

The **Responsibility** denotes the work packages and partners that will be responsible for ensuring the satisfaction of the requirement in the GENESYS architecture.

The **Description** specifies the requirement. The description has to be expressed in way, that the requirement becomes verifiable.

Rationale is an explanation why this requirement is necessary to achieve a specific goal of the GENESYS project

The **Architectural Significance** of a requirement specifies whether the fulfillment of the requirement through the platform is necessary to achieve a desired goal. We distinguish the following Architectural Significance options:

- Must be guaranteed by the platform architecture by construction
- Must be implemented by the platform architecture
- Must be enabled by the platform architecture
- Not relevant to the platform architecture

The **Application Significance** of a requirement can also be *high*, *medium*, *low*, and *optional*. This attribute denotes the importance of a requirement for systems/products in the domain, but not for the platform. Still, the reference architecture template and the cross-domain architectural style must not be in conflict with the requirement, i.e., it must be possible to implement the requirement on top of an instantiation of the reference architecture template.

The **Integration Level** denotes the layer in a system-of-systems at which the requirement is expressed: chip, device, and system (open or closed).

The **Relations** describe how a requirement is related to other requirements (e.g., a requirement may require, imply or be in conflict with another requirement).

The **Domain** lists the domains for which the requirement is important or cross-domain in case of a universal requirement.

Additional Classification is used to classify requirements, e.g., as physical or logical.

Additional Information can be supplied in order to help the reader to understand the requirement and its context.

ID:	unique id
Name:	
Responsibility:	WPs + partners
Description:	
Rationale:	
Architectural Significance:	
Application Significance:	
Integration Level:	
Relations (optional):	
Domains (optional):	
Additional Classification (optional):	e.g., physical vs. logical
Additional Information (optional):	e.g., relation to standards

2 Definitions

2.1 Component

At an abstract level, there is general agreement on the notion of a *component*: a component is considered to be a self-contained building block that can be used to build larger systems. However, when it comes to a more detailed description in the context of computer system design, discussions emerge about the concrete properties associated with the notion of a component, particularly when the notion of a *software component* is portrayed.

Must a component have behavior?—*Behavior* relates to the exchange of information between a component and its environment, which can be in the form of messages. Behavior does have a temporal connotation. We call the intended behavior of the components its service [DSOS 2002]. What is the behavior of a *software component*?

Does a component contain state? According to Mesarovic [Mesarovic 1989], the concept of *state* can be explained as follows, p.45 : *The state enables the determination of a future output solely on the basis of the future input and the state the system is in. In other words, the state enables a “decoupling” of the past from the present and future. The state embodies all past history of a system. Knowing the state “supplants” knowledge of the past. . . . Apparently, for this role to be meaningful, the notion of past and future must be relevant for the system considered.*

The notion of state thus presupposes an *appropriate model of time* to be able to distinguish the past from the future. What is the notion of time of a *software component*?

The above raised questions disappear if a component is considered to be a hardware/software unit that accepts input messages and produces output messages. Such a hardware/software unit, we call it *system component*, is *time aware* and contains *state*. In the rest of this section whenever we use the term *component* we mean such a software/hardware unit.

2.2 Composability

The ARTEMIS SRA says on p. 9: *a framework that supports the smooth integration and reuse of independently developed components is needed in order to increase the level of abstraction in the design process* and formulates more concretely on p.24 under the topic composability: *Architecture instantiations that are derived from a generic platform must support the constructive composition of large systems out of components and sub-systems without uncontrolled emerging behavior or side effect.*

3 Requirements

In this Section we elaborate on *necessary* requirements that must be satisfied by an architecture for the support of *composability*.

A practical methodology for embedded systems design needs to scale and overcome the limitations of current algorithmic verification and synthesis techniques [HS 2006,2007]. These techniques are often limited by their exponential complexity in the size of the system under analysis. One route to circumvent this limitation and achieving scalability is to rely on composability properties such as compositionality, incrementality, and non-interference, which allow inferring global properties of the system from those of its components and the way these are integrated.

ID:	6.1
Name:	Composability
Responsibility:	WP6, all partners
Description:	<i>The architecture shall support composability in a way that larger systems can be composed out of smaller subsystems.</i>
Rationale:	Composability is a concept that relates to the ease of building systems out of subsystems. We assume that subsystems have certain properties that have been validated at the subsystem level. A system, i.e., a composition of subsystems, is considered composable with respect to a certain property if <i>this property</i> , given that it has been established at the subsystem level, is not invalidated by the integration. Examples of such properties are timeliness or certification .
Architectural Significance:	must be guaranteed by the platform architecture by construction
Application Significance:	high
Integration Level:	chip, device, system
Relations (optional):	
Domains (optional):	
Additional Classification (optional):	
Additional Information (optional):	source: ARTEMIS SRA

ID:	6.2
Name:	Compositionality
Responsibility:	WP6, all partners
Description:	<i>The architecture shall allow inferring the properties of the integrated system from those of its components.</i>
Rationale:	Compositionality relates to the ease of validation of system properties. In a compositional architecture, properties of the system can be inferred solely by the analysis of related properties of the components carried out independently for each component. This avoids exponential (combinatorial explosion of the) analysis complexity.
Architectural Significance:	must be guaranteed by the platform architecture by construction
Application Significance:	high
Integration Level:	chip, device, system
Relations (optional):	implies modularity properties (e.g. 6.17 Modular Certification)
Domains (optional):	
Additional Classification (optional):	
Additional Information (optional):	source: VERIMAG

ID:	6.3
Name:	Incrementality
Responsibility:	WP6, all partners
Description:	<i>The architecture shall allow for systems to be constructed by adding individual components, and this without restricting the order of composition.</i>
Rationale:	Incrementality relates to the ease of the analysis of systems, their validation, assembly, and maintenance. It means that components can be added, removed, or replaced independently, and that the rest of the system has an unambiguous meaning (can still be considered and analyzed). Incrementality also means that the result of composition does not depend on the order of integration, facilitating, among others, the assembly process.
Architectural Significance:	must be guaranteed by the platform architecture by construction
Application Significance:	high
Integration Level:	chip, device, system
Relations (optional):	Strengthens 6.2 Compositionality
Domains (optional):	
Additional Classification (optional):	
Additional Information (optional):	source: VERIMAG

3.1 Precise Interface Specification

A component is an abstract unit that provides its services across interfaces. The important *complexity-reduction* effect of *component-based design* can only be achieved if it is possible to use and reason about the services of a component at the system level solely on the basis of the precise interface specification of the component and if this interface specification is much simpler than the specification of the component's internal implementation. In our context we assume that the services of a component at the atomic core service level are offered across a *message interface*. Above the atomic core service level, component services can also be provided using different communication paradigms such as streaming services.

A *message interface* can be specified on two levels, the *operation level* and the *semantic level* [Kopetz 2003]. The operational level establishes *interoperability* and deals with the syntax and the timing of the input and output messages of a component. The semantic level is concerned with the assignment of meaning to the syntactic units of the message established at the operational level by the provision of an interface model. We call a component a *stateful* component if this interface model contains *state*, otherwise the component is a *stateless* component. In order to be able to reintegrate a stateful component into a running system, we must provide for *reintegration points*, where the internal state of the component is fully defined.

Many architectures do not provide the means to precisely specify the temporal properties of component interfaces, such as the send instants of messages, or the maximum or average rate of message production and message consumption which are needed for the allocation of buffers.

For a more detailed discussion of the subject of *component interface specification* we refer to [Kopetz 2003].

ID:	6.4
Name:	Linking Interface Specification
Responsibility:	WP4, WP6
Description:	At the atomic core service level, the architecture shall restrict interactions between subsystems of a system, to occur exclusively by the exchange of messages (defined data structures for inter-process communication) via the linking interfaces (LIFs) of the subsystems. These messages have to be fully specified in the value and the time domain in a linking interface (LIF) specification.
Rationale:	With a full specification of a subsystem's LIF a system designer can use a subsystem based solely on its LIF specification and can abstract over the inner structure of the subsystem. All issues related to the composition of a set of subsystems can then be investigated by referring only to the specification of their LIFs. Due to this advance of the level of abstraction, the cognitive complexity of the whole system is significantly reduced. Furthermore, an exact LIF specification is a prerequisite for independent development and reuse of subsystems.
Architectural Significance:	must be enabled by the platform architecture
Application Significance:	high
Integration Level:	chip, device, system
Relations (optional):	
Domains (optional):	
Additional Classification (optional):	
Additional Information (optional):	The challenge is to develop component interface specifications in the time domain and value domain that support the decoupling of the components under the constraints of minimal performance degradation. <i>The system should be robust even if the interface specification is violated.</i> source: ARTEMIS SRA

ID:	6.5
Name:	Minimalism
Responsibility:	WP6
Description:	To keep the LIF simple and understandable, only those objects and functions that are required for the intended emerging service should be visible at the LIF.
Rationale:	It is counterproductive for all other internal objects of a subsystem to be visible at the LIF.
Architectural Significance:	must be enabled by the platform architecture
Application Significance:	high

Integration Level:	chip, device, system
Relations (optional):	
Domains (optional):	
Additional Classification (optional):	
Additional Information (optional):	A subsystem can have multiple LIF specifications for different user groups. source: ARTEMIS SRA

ID:	6.6
Name:	Operational Specification
Responsibility:	WP6
Description:	The linking interface specification of a subsystem has to include an operational specification that captures the syntactical and temporal properties of the linking interface.
Rationale:	A specification of the syntactical and temporal properties of the LIF is a prerequisite to ensure interoperability. Interoperability means that the mechanisms for the exchange of information chunks among components are intact, however it is not concerned with whether the meaning assigned to these information chunks by the sender and receiver are compatible.
Architectural Significance:	must be enabled by the platform architecture
Application Significance:	high
Integration Level:	chip, device, system
Relations (optional):	
Domains (optional):	
Additional Classification (optional):	
Additional Information (optional):	source: ARTEMIS SRA

ID:	6.7
Name:	Meta-level Specification
Responsibility:	WP6
Description:	The linking interface specification of a subsystem has to include a meta-level specification that describes the meaning of the information that is exchanged over the interface.
Rationale:	The meta-level specification bridges the gap between the information chunks formed at the syntactic level and the user's mental model of the service provided at the interface. The concepts used in the specification must fit well with the internal conceptual world of the prototypical user of a given subsystem (i.e. it must be expressed in terms and notations that are common knowledge to the

	user). The meta-level specification may contain formal descriptions, but can also incorporate natural language, pictures, and diagrams that are commonly used in the user's domain.
Architectural Significance:	must be enabled by the platform architecture
Application Significance:	high
Integration Level:	chip, device, system
Relations (optional):	
Domains (optional):	
Additional Classification (optional):	
Additional Information (optional):	Research Challenge: Methodology and formalism for the representation of meta-level data. source: ARTEMIS SRA

ID:	6.8
Name:	Linking Interfaces of Reconfigurable Systems*
Responsibility:	WP4, WP6
Description:	A precise specification of linking interfaces in the temporal and value domain must be supported for reconfigurable systems.
Rationale:	Systems are composed of multiple subsystems, multiple sensor / actuator nodes, etc that might even not be known at design stage. In addition to this, the topology and hierarchy where subsystems are integrated might not be known a priori. Therefore, the definition of the LIF (Linking Interfaces) covering time domain, value domain, safety domain, etc. is of utmost importance.
Architectural Significance:	must be enabled by the platform architecture
Application Significance:	high
Integration Level:	chip, device, system
Relations (optional):	
Domains (optional):	
Additional Classification (optional):	
Additional Information (optional):	source: IKERLAN

3.2 Stability of Prior Properties

A component is a self-contained unit that can be designed and used on the basis of its interface specification. We call a specified set of relevant properties of a component before its integration into a system the *prior properties* of the components. If the *stability-of-prior-properties* requirement

* This requirement is pending, because the handling of the requirement needs to be determined. The handling of the requirement will be decided upon during the AB meeting in Helsinki on April 9/10, 2008.

is satisfied, the *prior services* of components may not be impaired by its integration into a larger system.

Take the example of an engine controller in a car that is connected to a theft-avoidance system by a message interface. The prior service of the engine controller is the control of the engine. If the arrival of a message of the theft avoidance system raises an interrupt in the engine controller that sporadically interferes with the execution of a time-critical task of the engine controller, than the prior services of the engine controller are impaired by the integration. The stability-of-priorities principle requires that such an impairment may not occur (In the above example the replacement of the *message push interface* by a *message pull interface* will solve the problem).

ID:	6.9
Name:	Stability of Prior Services
Responsibility:	WP3, WP4, WP5, WP6
Description:	The architecture has to assure that a validated service of a subsystem, both in the value domain and in the time domain, must not be refuted by integrating the subsystem into a larger system.
Rationale:	The stability-of-prior-services principle is essential to allow independent design, development, and verification of a subsystem.
Architectural Significance:	must be guaranteed by the platform architecture by construction
Application Significance:	high
Integration Level:	chip, device, system
Relations (optional):	
Domains (optional):	
Additional Classification (optional):	
Additional Information (optional):	Research Challenge: Definition of a framework that eliminates all unintentional side effects of integration without an excessive performance penalty. source: ARTEMIS SRA

ID:	6.10
Name:	Temporal Composability
Responsibility:	WP4, WP6
Description:	The architecture shall support composability with respect to the temporal domain.
Rationale:	Specified timeliness properties of subsystems that have been established at subsystem level shall not be refuted by the integration of subsystems into a larger system.
Architectural Significance:	must be guaranteed by the platform architecture by construction
Application Significance:	high
Integration Level:	chip, device, system
Relations (optional):	
Domains (optional):	
Additional Classification (optional):	
Additional Information (optional):	source: ARTEMIS SRA

ID:	6.11
Name:	Application-level temporal constraints*
Responsibility:	WP4, WP6
Description:	The architecture shall provide mechanisms to enforce application-level temporal constraints.
Rationale:	Some application-level properties depend on the timely execution of a distributed activity across several components. The architecture shall provides ways for design-time analysis and run-time enforcement and monitorisation of this property.
Architectural Significance:	must be enabled by the platform architecture
Application Significance:	high
Integration Level:	chip, device, system
Relations (optional):	
Domains (optional):	
Additional Classification (optional):	
Additional Information (optional):	See model of distributed activity in OMA, DRTSJ, COMPARE source: UPM

ID:	6.12
Name:	Development and test composability*
Responsibility:	WP5, WP6
Description:	The integration of tests should also consecutively proof / test the previously tested properties at sub-system level.
Rationale:	Each subsystem is usually composed of other sub-subsystems such as the inputs / outputs (analogue, digital, etc.) and processing sub-subsystems. Requirements and safety cases usually need to ensure that certain properties (e.g. time) are always met with independence of other sub-subsystems (e.g. processing).
Architectural Significance:	must be enabled by the platform architecture
Application Significance:	high
Integration Level:	chip, device, system
Relations (optional):	
Domains (optional):	
Additional Classification (optional):	
Additional Information (optional):	source: IKERLAN

* This requirement is pending, because the handling of the requirement needs to be determined. The handling of the requirement will decided upon during the AB meeting in Helsinki on April 9/10, 2008.

* This requirement is pending, because the handling of the requirement needs to be determined. The handling of the requirement will decided upon during the AB meeting in Helsinki on April 9/10, 2008.

3.3 Non-Interfering Interactions

In a distributed control system subsets of components have to interact to provide the required system service. For example, a central brake controller in a car has to send messages to the brake computers at the four wheels to achieve the desired effect. We call such set of components a distributed application subsystem (*DAS*). The *non-interfering interactions* requirement states that there must not be any interference between the interactions of different *DASes*. Such interferences can occur in the domains of time and space. Consider, for example, a distributed system where the information among the different *DASes* is exchanged by a shared memory. In such a system the memory allocation of one *DAS* can interfere with the memory allocation of another *DAS* (interference in space). An interference in time can occur if a shared communication medium is allocated according to message priorities (such as in the CAN system), since a sequence of high-priority messages of one *DAS* can lead to a starvation of the other *DASes*.

ID:	6.13
Name:	Non-Interfering Interactions *
Responsibility:	WP4, WP6
Description:	The architecture has to ensure that there is no interference between the interactions of different <i>DASes</i>
Rationale:	Interference between the interactions of different <i>DASes</i> would add an additional accidental complexity to the inherent complexity of an application. Hence, activities such as diagnosis, testing, and understanding of the system are complicated.
Architectural Significance:	must be guaranteed by the platform architecture by construction
Application Significance:	high
Integration Level:	chip, device, system (closed)
Relations (optional):	
Domains (optional):	
Additional Classification (optional):	
Additional Information (optional):	source: TUVI

* This requirement is pending, because the handling of the requirement needs to be determined. The handling of the requirement will be decided upon during the AB meeting in Helsinki on April 9/10, 2008.

ID:	6.14
Name:	Growth and Scalability
Responsibility:	WP3, WP4, WP5, WP6
Description:	The architecture has to ensure that, if n subsystems are already integrated into a system and the amount of available resources permits the integration of an n'th +1 subsystem, then the n'th + 1 subsystem must not disturb the correct operation of the n already integrated subsystems.
Rationale:	This constraint ensures that the cognitive complexity grows linearly with the number of subsystems that have to be integrated into a system. Furthermore it reduces the effort that is needed to extend an existing design.
Architectural Significance:	must be guaranteed by the platform architecture by construction
Application Significance:	high
Integration Level:	chip, device
Relations (optional):	
Domains (optional):	
Additional Classification (optional):	
Additional Information (optional):	Research Challenge: The interconnection fabric among components must not exhibit uncontrollable load dependent degradation of its services with respect to latency and jitter. source: ARTEMIS SRA

ID:	6.15
Name:	Hot replacement*
Responsibility:	WP4, WP6
Description:	The architecture shall provide mechanisms to enable the hot replacement of components
Rationale:	Some application domains require that some maintenance activities are performed on hot systems.
Architectural Significance:	must be enabled by the platform architecture by construction
Application Significance:	high
Integration Level:	chip, device, system
Relations (optional):	
Domains (optional):	
Additional Classification (optional):	

* This requirement is pending, because the handling of the requirement needs to be determined. The handling of the requirement will be decided upon during the AB meeting in Helsinki on April 9/10, 2008.

Additional Information (optional):	source: UPM
------------------------------------	-------------

3.4 Fault- and Error-Containment

The abstraction of a component must remain intact under fault conditions, otherwise the advantage of using components is severely diminished. This requires that the architecture must provide for *fault containment units* (FCU) and establish error propagation boundaries. An FTU is a well-defined subsystem that is considered a unit of failure at the system level. The *independence assumption* [Kopetz 2003] states that a single fault will directly effect only a single FTU. In a federated architecture a node is an FTU and fault containment is realized by the properties of the physical design. In order to eliminate the error propagation of a faulty FTU to an FTU that has not been affected by the fault, error propagation boundaries must be part of the architecture. For example, the CAN protocol does not provide for error containment. A faulty component that sends a continuous sequence of high-priority messages can cause all other components to die of starvation and miss their deadlines.

ID:	6.16
Name:	Mixed-Criticality Subsystems
Responsibility:	WP6
Description:	The architecture has to support certification of subsystems with different criticality levels.
Rationale:	This support allows certifying each subsystem according to its criticality level. Otherwise, every subsystem would have to be certified according to the criticality level of the most critical subsystem within the system.
Architectural Significance:	must be guaranteed by the platform architecture by construction
Application Significance:	high
Integration Level:	chip, device, system (closed)
Relations (optional):	
Domains (optional):	
Additional Classification (optional):	
Additional Information (optional):	source: ARTEMIS SRA

ID:	6.17
Name:	Modular Certification of Subsystems
Responsibility:	WP6
Description:	The architecture has to support modular certification of the subsystems of a system.
Rationale:	Modular certification is a necessity for cost-effective reuse of subsystems.
Architectural Significance:	must be guaranteed by the platform architecture by construction
Application Significance:	high
Integration Level:	chip, device, system (closed)
Relations (optional):	
Domains (optional):	
Additional Classification (optional):	
Additional Information (optional):	source: ARTEMIS SRA

ID:	6.18
Name:	Independent Certification of the Platform
Responsibility:	WP6
Description:	The architecture has to enable independent certification of the platform and the application.

Rationale:	The certification of the platform should be done only once, such that the already certified platform can be reused for many applications.
Architectural Significance:	must be guaranteed by the platform architecture by construction
Application Significance:	high
Integration Level:	chip, device, system
Relations (optional):	
Domains (optional):	
Additional Classification (optional):	
Additional Information (optional):	source: ARTEMIS SRA

ID:	6.19
Name:	Safety Composition*
Responsibility:	WP5, WP6
Description:	Safety composition rules defined by the IEC-61508 shall be considered.
Rationale:	IEC-61508 is the common base certification standard for multiple industrial domains (e.g. lift, PLC) and systems are usually composed of non-critical and safety-critical certified subsystems. Thus, there is a need for safety composability in order to ensure that the integration of these systems do not invalidate the safety cases. The "safety composition rules" defined by the IEC-61508, transparent redundancy management and DECOS like integrated support for diagnosis and fault isolation services might be of interest.
Architectural Significance:	must be enabled by the platform architecture
Application Significance:	high
Integration Level:	chip, device, system (closed)
Relations (optional):	
Domains (optional):	
Additional Classification (optional):	
Additional Information (optional):	source: IKERLAN

3.5 Fault Masking

In a high-dependability architecture it must be possible to mask the failure of a component by generic mechanisms. One such generic mechanism is triple modular redundancy (TMR). In a

* This requirement is pending, because the handling of the requirement needs to be determined. The handling of the requirement will be decided upon during the AB meeting in Helsinki on April 9/10, 2008.

TMR system three synchronized replicas of a component execute in three independent FCUs and produce independently three results. A voter compares these three results and selects the majority as the output of such a triad. One necessary precondition for the operation of a TMR system is the deterministic operation of the replicas. For a more detailed discussion on deterministic operation refer to [Kopetz 2007].

ID:	6.20
Name:	Sustained operation *
Responsibility:	WP5, WP6
Description:	The architecture shall enable the continuous operation (365x24) of components
Rationale:	Some application domains require non-stop operation of systems.
Architectural Significance:	must be guaranteed by the platform architecture by construction
Application Significance:	high
Integration Level:	chip, device
Relations (optional):	
Domains (optional):	
Additional Classification (optional):	
Additional Information (optional):	source: UPM

* This requirement is pending, because the handling of the requirement needs to be determined. The handling of the requirement will be decided upon during the AB meeting in Helsinki on April 9/10, 2008.

ID:	6.21
Name:	Replica Determinism*
Responsibility:	WP5, WP6
Description:	<i>The architecture has to ensure replica determinism of replicated components.</i>
Rationale:	A set of replicated components is replica determinate [Poledna 1994] if all the members of this set have the same encapsulated state, and produce the same output messages at points in time that are at most an interval of d time units apart (as seen by an omniscient external observer). Replicated components have to be replica determinate if voting should be done by bit-by-bit comparison of the component's output messages. The advantage of bit-by-bit comparison is that it can be systematically applied, and does not require any application specific knowledge about the values that have to be compared.
Architectural Significance:	must be enabled by the platform architecture
Application Significance:	high
Integration Level:	chip, device, system (closed)
Relations (optional):	
Domains (optional):	Process control industry, military, avionic
Additional Classification (optional):	
Additional Information (optional):	Source: ARTEMIS SRA

* This requirement is pending, because the handling of the requirement needs to be determined. The handling of the requirement will be decided upon during the AB meeting in Helsinki on April 9/10, 2008.

ID:	6.22
Name:	Temporal Order
Responsibility:	WP4, WP6
Description:	<i>The architecture shall ensure that all subsystems of a system see a sequence of events (e.g. reception of messages) in the same order.</i>
Rationale:	Since the temporal order of incoming events can have an influence of the internal state of a subsystem, a consistent view on the temporal order of events is generally necessary to achieve replica determinism.
Architectural Significance:	must be enabled by the platform architecture
Application Significance:	high
Integration Level:	chip, device, system
Relations (optional):	
Domains (optional):	Process control industry, military, avionic
Additional Classification (optional):	
Additional Information (optional):	Source: ARTEMIS SRA

3.6 Requirements w.r.t. Composability for the Development Process

ID:	6.23
Name:	Meet-in-the-middle
Responsibility:	WP3, WP6
Description:	<i>The architecture has to allow design methodologies where top-down and bottom-up design styles are combined.</i>
Rationale:	Many real product developments follow neither a strict top-down approach nor a strict bottom-up approach.
Architectural Significance:	must be enabled by the platform architecture
Application Significance:	high
Integration Level:	chip, device, system
Relations (optional):	
Domains (optional):	
Additional Classification (optional):	
Additional Information (optional):	source: ARTEMIS SRA

ID:	6.24
Name:	Intellectual-Property Software-Modules
Responsibility:	WP6

Description:	<i>The architecture shall support the integration of components based on the interface specifications, i.e., without having to understand the internals of the components (e.g., integration of precompiled software modules).</i>
Rationale:	Third party suppliers may wish to deliver only precompiled code in order to protect their intellectual property (i.e. the source code).
Architectural Significance:	must be enabled by the platform architecture
Application Significance:	high
Integration Level:	chip, device, system
Relations (optional):	
Domains (optional):	
Additional Classification (optional):	
Additional Information (optional):	source: ARTEMIS SRA

ID:	6.25
Name:	Component aggregation
Responsibility:	WP6
Description:	The architecture shall enable the aggregation of components to constitute compound components. All requirements of applicability to a simple component must be of applicability to a compound component,
Rationale:	Complex application design and management require scalability in the grouping, models and metadata of subsystems.
Architectural Significance:	must be guaranteed by the platform architecture by construction
Application Significance:	low
Integration Level:	chip, device, system
Relations (optional):	
Domains (optional):	
Additional Classification (optional):	
Additional Information (optional):	source: UPM

3.7 Components with Heterogeneous Technologies

ID:	6.26
Name:	Multi Vendor Environment
Responsibility:	WP4, WP6
Description:	The architecture must have the capability to support multiple standards (e.g., protocols, interfaces) that allow the integration of IP blocks from multiple vendors.

Rationale:	Since standards are constantly evolving, the architecture must be flexible enough to adapt to changing standards without a redesign of the core elements of the architecture. Standards should be encapsulated in subsystems that can be modified and exchanged without violating the established architectural style.
Architectural Significance:	must be enabled by the platform architecture
Application Significance:	high
Integration Level:	chip, device, system
Relations (optional):	
Domains (optional):	
Additional Classification (optional):	
Additional Information (optional):	source: ARTEMIS SRA

ID:	6.27
Name:	Heterogeneity
Responsibility:	WP6
Description:	The architecture has to support the integration of subsystems that are realized in different technologies.
Rationale:	Future SoC and system-in-package technologies, along with their associated integration capacities, may bring about the end of analog and mixed-signal design as a separate discipline. System elements from virtually all design domains will be co-designed for integration at either the package or the substrate level.
Architectural Significance:	must be enabled by the platform architecture by construction
Application Significance:	high
Integration Level:	chip, device, system
Relations (optional):	
Domains (optional):	
Additional Classification (optional):	
Additional Information (optional):	source: ARTEMIS SRA

ID:	6.28
Name:	Non-recurring Semi-Conductor Engineering Cost
Responsibility:	WP3, WP6
Description:	The architecture has to consider the fact that the non-recurring mask cost is a barrier.
Rationale:	The business model of the semiconductor industry has a major impact on the availability of components for the

	embedded systems market. The non-recurring costs (design, masks, test pattern ...) of a submicron SoC are continuously growing (tens of millions of dollars). Low-volume applications can hardly effort the design of a custom SoC. FPGAs may be a solution for small volume markets.
Architectural Significance:	must be enabled by the platform architecture
Application Significance:	high
Integration Level:	chip
Relations (optional):	
Domains (optional):	
Additional Classification (optional):	
Additional Information (optional):	Research Challenge: Development of a small number of flexible generic SoCs, possibly with FPGA subsystems, that can be deployed in many different application domains. Support of the development of an application of such a generic SoC by an appropriate tool chain. Introduction of platforms that make use of heterogeneous reconfigurability. source: ARTEMIS SRA

ID:	6.29
Name:	Distributed Services
Responsibility:	WP6
Description:	The architecture shall support the integration of distributed services that adhere to well established standards (e.g. Web services).
Rationale:	A distributed service (e.g. Web service) is a software system designed to support interoperable machine-to-machine interaction over a network (e.g. the Internet). The interface of a distributed service has to be described in a machine-processable format.
Architectural Significance:	must be enabled by the platform architecture
Application Significance:	high
Integration Level:	chip, device, system
Relations (optional):	
Domains (optional):	
Additional Classification (optional):	
Additional Information (optional):	source: ARTEMIS SRA

ID:	6.30
Name:	Support for Heterogeneous Data-Representations
Responsibility:	WP6
Description:	Heterogeneous data representations need to be supported.
Rationale:	The integration of multiple subsystems and communication protocols might lead to inconsistent data-representations that are usually handled with 'glue-code': e.g. endiannes, different basic data-type lengths, different units (e.g. meter / inches), etc.
Architectural Significance:	must be enabled by the platform architecture
Application Significance:	high
Integration Level:	chip, device
Relations (optional):	
Domains (optional):	
Additional Classification (optional):	
Additional Information (optional):	source: IKERLAN

4 State-of-the-Art and Analysis of Existing Architectures

4.1 Automotive Architectures

The Automotive Open System Architecture (AUTOSAR) [AUTOSAR 2006a] is a system architecture and development methodology for automotive electronic systems. Among the primary goals of AUTOSAR are the standardization of the basic software of an ECU (called standard core) and the ability to integrate software components from multiple suppliers. Another focus of AUTOSAR is the establishment of portability and location transparency for software components in order to accommodate future changes to the automotive electronic systems and facilitate reuse across product lines [AUTOSAR 2006b].

- **Precise Interface Specification:** AUTOSAR provides a meta-model for the model-based development of automotive systems. The meta-model supports the definitions of the syntax of signals exchanged between software components. In addition, the meta-model includes semantic properties (e.g., physical units) and temporal constraints (e.g., maximum transfer time and jitter). The meta-model currently does not, however, support the definition of dependability properties, interface state or more precise temporal specifications (e.g., a specification of send instants of messages w.r.t. a global time base or phase relationships of messages)
- **Stability of Prior Properties:** AUTOSAR supports protocols that do not ensure the stability of prior services, such as CAN [Bosch 1991]. For a deeper understanding consider an exemplary scenario with two subsystems. If the two subsystems share a common CAN bus, then both subsystems must be analyzed and understood in order to reason about the correct behavior of any of the two subsystems. Since the message transmissions of one subsystem can delay message transmission of the other DAS, arguments concerning the correct temporal behavior must be based on an analysis of both subsystems. In a totally federated system, on the other hand, unintended side effects are ruled out, because the two subsystems are assigned to separate computer system.
- **Non-Interfering Interactions:** The current version of the AUTOSAR runtime environment [AUTOSAR 2006b] supports priority-based mechanisms for communication and the scheduling of runnables. For example, a high-priority runnable of one software component could delay the execution of runnables of other software components. In general, it is thus necessary to analyze the execution behavior of all software components with higher priority on an ECU. Similar dependencies between software components can occur at the communication system when using a priority-based protocol such as CAN.
- **Fault- and Error-Containment:** AUTOSAR is not tailored to a specific type of communication infrastructure. Therefore, fault and error containment properties can only be assessed in conjunction with specific implementation technologies (e.g., communication protocol, operating system).
- **Fault Masking:** No fault masking mechanisms are defined in the current version of AUTOSAR.

4.2 Avionics Architectures

4.2.1 Current state-of-the-art

The current state-of-the-art of IMA, resulting from the research of the NEVADA, PAMELA, and VICTORIA projects, can be summarized as the solutions implemented on the A380, A400M and SuperJet100. This corresponds to the first generation (in Europ) of standardised IMA, so-called IMA1G, which is based on the following concepts:

- A large set of standardized electronic modules,
- A central aircraft data network (ARINC 664),
- Separation of the electronic module (including hardware and operating system) and applications through the use of a standardized application programming interface (API ARINC 653),
- Toolsets for programming applications but not for system integration e.g. overall allocation and system management across integrated resources,
- An incremental qualification process that allows (1) to separately develop and qualify the electronic module and applications (2) to independently develop all the applications. Any application can get qualification credit from the electronic module (within its defined usage domain). An application may change with the guaranty to avoid any impact on all the other applications hosted on the same electronic module.

Because of the emphasis on high levels of safety and the costs of the qualification and certification processes to support this approach the aircraft manufacturing industry adopts a prudent approach to the introduction of new technologies, even mature technologies. So, while benefits of IMA have been realized in this first generation implementation, the maturity and performance of the prevailing technologies dictated many of the technical solutions and limited the range of functions that could be hosted within the architecture.

4.2.2 Advances beyond the state-of-the-art

Significant evolution to the existing IMA1G will be introduced with the concept of Distributed Modular Electronics (DME) in the following areas:

- **Integrated set of multiple hardware resources** managing simultaneous execution of embedded functions instead of a set of independent computing modules allocated statically to individual applications.
- **Technological innovation in I/O standardization and reconfiguration capabilities** beyond those in IMA1G will achieve a minimum number of types of standardised electronic modules.
- **Support for extended range of avionics applications.** Incorporating state-of-the-art processing and communication electronics in the processing modules will allow to include in DME:
 - fast control loops,
 - High Performances Data Distribution / data flow processing functions,
 - standardized reconfigurable inputs/ouputs,
 - standardised power controllers or graphical modules.
- **Nearly-total (instead of partial) support of avionic functions.** Specific hardware will thus be limited to sensors/effectors, HMI and radio-frequency devices.
- **Greater independence of hardware and applications.** The introduction of the new platform middleware layer will enable the development of applications concentrating on their functional content, and will isolate them from underlying platform configuration (eg: initialisation/shutdown, time sharing, fault management, configuration, reconfiguration, communication, etc.)
- **Portability:** The further separation between functional application and underlying resources brought by the platform middleware abstraction layer will also favour functional component portability from one aircraft system design to the other.
- **DME System design toolset:** The advanced toolset, thanks to an extensive model-based approach, will enable to define and size the DME architecture, to allocate the resources (including network) to applications, to distribute the applications into the DME platform by considering performance, spare policy and reconfiguration needs.

The DME solution can separate the processing and I/O functions into different physical units, CPM (Core Processing Module) and IOM (Input Output Module), which will be located in the avionics bay of the aircraft, There will also be standardised remotely distributed I/O units known as Remote Data Concentrators (RDCs).

Some IO interfaces can remain closed to processing in order to allow fast control loops and low latency. Another alternative is to introduce some processing capability into RDC potentially closed to sensors and actuators.

This will allow the avionics architecture to be more easily scaled to meet the different interfacing and processing requirements of different aircraft types. Scalability will allow the capability to easily add (or remove) processing and I/O units in order to cope with Aircraft size and the quantity of avionics applications to be hosted. This approach will also allow flexibility in locating and relocating software applications, which is a pre-requisite for reconfiguration to achieve fault tolerance and very high availability.

Within the DME the services and network infrastructure will be designed to enable the separation of system applications from their signal interfaces. This provides the ability to easily reconfigure, statically during maintenance, but also potentially dynamically in flight, the allocation of resources to applications. Using this capability, the avionics designer will be able to optimise the avionics to produce the required system performance from the fewest number of components. In addition, the concept requires the use of Remote Electronics Units (REUs) for control of distributed equipment (e.g. motor control), and Remote Power Controllers (RPCs) for switching electrical power. The purpose of these remote units is to make substantial weight savings of the lengthy signal and power harnesses that run between the avionics bay and all areas of the aircraft. The elements of

the future DME architecture correspond to totally new components, based on the most advanced technologies.

4.3 Industrial Architectures

Industrial control applications share multiple common requirements / properties with other domains. For example, the architecture of industrial control is usually federated where systems are composed of non-critical subsystems, safety-critical subsystems (with safe-state) and operation-critical (no "clear" safe-state) subsystems.

On the other hand, something quite specific of industrial domain applications is that they must operate in "harsh environments" with extended temperature ranges, systems might be have a wide spatial distribution (e.g. sensors might be several meters away) and "strong" Electromagnetic Interference (EMI). These specific requirements are usually handled at the "hardware" level providing EMI protections with shields, "clean" power-supply mechanisms, surge protections, etc. This is required to ensure system level composability, by means of hardware mechanisms, that ensure the integration of sub-systems will not affect (e.g. electrically) the operation of the system.

Additionally large-scale industrial control spans several hierarchical layers and the composability requirements must be addressed in different levels of the scale (device, regulatory local control loops, unit subsystems, etc.).

An additional problem in industrial control architectures that set them apart from most embedded systems is the evolutionary aspect of many industrial systems. Reparatory maintenance is a common topic for a sustainable operation of all embedded systems but in the case of industrial plants, evolutionary maintenance is also a critical issue derived from the continuous change in the sublying plants under control. While the physical infrastructure of a vehicle does not change (at least from the design point of view) industrial plants may suffer continuous modifications of the core physical processes that imply an increased degree of adaptivity in the embedded systems in charge of control tasks. This has implications for the engineering processes and for the run-time capabilities of the embedded systems (for example requiring hot replacement of intelligent sensors or on-line integration of subsystems). Consider that many industrial plants operate in 365x24 regimes.

From the perspective of the platform software and communication technology, the common trend is towards the unification of platforms across all subsystems and layers; that has implications for deeply embedded devices that must support platform technologies that are non-specific to the embedded systems domain in non-interfering coexistence with embedded systems technologies.

The software component is the established approach to industrial control systems construction, but there is no component model of reference that is widely followed. Each control systems manufacturer has typically developed its own technology and while there are some standards – *de iure* and *de facto* – and some attempts to develop reference models, as of today, there is no established technology for industrial control system componentization (beyond some physically-bound components).

4.4 Mobile Architectures

Contemporary embedded systems development in the nomadic environments and private spaces is driven by pervasive trends such as increasing processing capability; scalable and flexible solutions; digital convergence and content.

Competition pressure and productivity improvements require decreasing reaction times and product development times, but these are increasingly difficult to achieve. Furthermore, it is quite difficult for many IP vendors to offer pieces of designs that seamlessly fit with proprietary embedded architectures.

The number of functions in embedded devices and nomadic wireless devices in particular is increasing rapidly. Most of the time these functions are implemented in software, implying that more and more of the system complexity resides on the software side. Taking into account that for practical product purposes a

system should encompass the joint operation of its constituent parts, it follows that a good embedded system architecture should address both the HW and the SW sides. This is currently not the case.

Contrary to a popular belief, current SW realizations are not easy and fast to change. Systems inherently gravitate towards more complexity over time. Therefore, also SW tends to become legacy. Complex things are typically difficult to change.

Furthermore, product technologies have their own life cycles. The embedded computing devices seem to be fated to follow the general-purpose computers to open, modular architectures and standardized sub-components. This ongoing process is also called horizontalization.

Progressing beyond the current state, modular architectures and design are an approach to answer the pressing design problems. Modularization exploits the traditional advantages of abstraction, i.e., design with larger entities and independent verification of modules, dividing the problem to manageable pieces. Moreover, there are system-level advantages that are not readily evident on lower abstraction levels. Interfaces remain persistent, allowing implementations to scale while retaining the architecture. Product versatility requires that heterogeneous architectures, technology, and sub-designs can be integrated from various sources. Interconnect-centric modularity is needed to make this a reality.

Designing the systems in a modular fashion allows better partitioning of the work; decreases development times; allows focused optimizations; makes integration and verification easier; and provides better understanding of the top-level operation of the system. The central technical enablers in achieving all these advantages are service-based system construction and building modules with semantic understanding of their own functionality and requirements.

Operating horizontally poses some unique technical problems. Particularly, it is not enough that the components encapsulate intellectual property, but the components must be able to seamlessly act as modules in a larger system, collaborating with each other. The convenience and error-free integration of these modules is extremely important. The additional integration work associated with assembling components from multiple sources must not overwhelm the advantages of the modularity.

Therefore, composability at the level of largely self-contained, independent modules with well-defined interfaces and a parallelism-friendly programming model is a critical factor of required architectural solutions.

4.5 Multimedia Architectures

An example of a standardized multimedia architecture is Universal Plug and Play (UPnP) [UPnP Forum, 2006a], which is an architecture for peer-to-peer interconnection of heterogeneous multimedia devices ranging from mobile phones, TV sets and DVD players to PCs. Its main aim is to support “zero-configuration” networking, which means to minimize the efforts required for installation and setup of device connectivity. For this purpose UPnP defines content providers (MediaServer), content receivers (MediaRenderer), and control units (ControlPoint), which are capable of automatic device and service discovery based on service specifications.

UPnP is based on well-established standards such as IP, TCP, UDP, HTTP, SOAP, and XML. The standardization of the UPnP architecture is driven by the UPnP Forum, which is led by a steering committee, including members such as Intel Corporation, Microsoft Corporation, Nokia Corporation, Philips Consumer Electronics, and Sony Corporation.

- **Precise Interface Specification:** The central concept in the UPnP architecture is the concept of a service. A service is modeled by a set of state variables describing its interface state as well as a set of actions [Microsoft, 2000]. Each service implements a control server, which receives incoming action requests, updates the state variables accordingly and returns a response. In addition, a service can implement event server functionality, which notifies subscribed devices about changes in the state of the service.

The UPnP forum provides standardized service specifications for typical multimedia and networking applications (e.g., audio/video transfer, UPnP printing service, etc). For those services syntax and semantic of state variables and actions is precisely specified using XML service descriptions. However, since the UPnP architecture is stacked on top of various protocols (e.g. HTTP over TCP/IP) the service specifications include no information about the temporal properties of the communication activities required to invoke and deliver the respective service.

UPnP is only used to exchange control information between media servers and media clients. The subsequent streaming of content is not within the scope of UPnP.

- **Stability of Prior Properties:** UPnP is designed to support easy integration of new devices into an existing system. Therefore it provides a discovery service enabling newly integrated devices to be automatically detected and to discover the services of the existing devices, respectively. However, this holds the risk that existing system properties (e.g., an available communication bandwidth between a video source and a video sink) are invalidated by the newly entered device.

For this purpose, the UPnP forum has released the service descriptions for Quality of Service (QoS). For all UPnP devices implementing this service, this enables the assignment of portions of the available network resources to dedicated *traffic streams* [UPnP Forum, 2006b] for bandwidth reservation. However, there is no guarantee that devices without implementing the UPnP QoS service will adhere to this reservation-scheme.

- **Non-Interfering Interactions:** Up to now, UPnP specifies no services for encapsulation of devices or device groups or for securing the access to particular devices by means of authentication. Therefore, every device entering the system can access the services provided by the other devices. For instance, consider an audio system consisting of a MediaServer (e.g. a digital MP3 music library) and a MediaRenderer including a ControlPoint (e.g. a stereo music player) [UPnP Forum 2006a]. The ControlPoint sets up a point-to-point connection between media server and media renderer for transferring the music stream. Any additional ControlPoint in the system is now able to disrupt the running system by also requesting a music stream from the media server, either until the capabilities of the media sever are exhausted, or the available bandwidth of the system is consumed.
- **Fault- and Error-Containment:** UPnP specifies the underlying protocol stack that is required for operation (e.g. HTTP over TCP/IP, SOAP, etc). However, the standardization does not define a specific communication infrastructure (e.g. switched Ethernet, WLAN, etc). Therefore, fault and error containment properties can only be assessed in conjunction with specific implementation technologies.
- **Fault Masking:** Fault masking as defined in Section **Error! Reference source not found.** is not in the scope of the UPnP architecture. Therefore, no mechanisms for fault masking are defined in the current version of the UPnP architecture standard.
- **Requirements w.r.t. Composability for the Development Process:** UPnP provides standardized templates for service specifications for different applications (e.g. UPnP printing services, audio/video transfer etc). The development of UPnP enabled devices or UPnP services is not further considered in the UPnP standard.
- **Components with Heterogeneous Technologies:** The UPnP architecture consists of several software layers on top of well established protocols, which are widely used in the Internet (e.g. SOAP for building web-services). Therefore, UPnP can be implemented on a wide variety of devices implemented with different technologies.

5 References

- [Mesarovic 1989] M.D. Mesarovic and Y. Takahara. *Abstract Systems Theory, chapter 3*. Springer-Verlag, 1989.
- [Kopetz 2003] H. Kopetz and N. Suri. Compositional design of RT systems: A conceptual basis for specification of linking interfaces. In *Proceedings of Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 51–60, May 2003.
- [Kopetz 2007] H. Kopetz. The Complexity Challenge in Embedded System Design. Research Report 55/2007. Technische Universität Wien, Institut für Technische Informatik. 2007.
- [AUTOSAR 2006a] AUTOSAR GbR. AUTOSAR – Technical Overview V2.0.1, June 2006.
- [AUTOSAR 2006b] AUTOSAR GbR. AUTOSAR – Specification of RTE Software V1.0.1, July 2006.
- [Bosch 1991] Robert Bosch GmbH, Stuttgart, Germany. CAN Specification, Version 2.0, 1991.
- [DSOS 2002] M.-C. Gaudel and V. Issarny and C. Jones and H. Kopetz and E. Marsden and N. Moffat and M. Paulitsch and D. Powell and B. Randell and A. Romanovsky and R. Stroud and F. Taiani. Version of the DSoS Conceptual Model. DSoS Project (IST-1999-11585) Deliverable CSDA1. December, 2002.
- [COMPARE 2007] Deliverable D2.2-1 User Requirements. Final Release. IST 004669 COMPARE Project. February 2007.
- [HS 2006] Thomas A. Henzinger and Joseph Sifakis. "The Embedded Systems Design Challenge". In FM 2006: Formal Methods, LNCS 4085/2006, Invited talk. Springer 2006.
- [HS 2007] Thomas A. Henzinger and Joseph Sifakis. "The Discipline of Embedded Systems Design". *Computer*, October 2007, pp. 32-40.