

This document has been prepared by the members of the ARTEMIS Strategic Research Agenda Working Group ("ARTEMIS SRA WG").

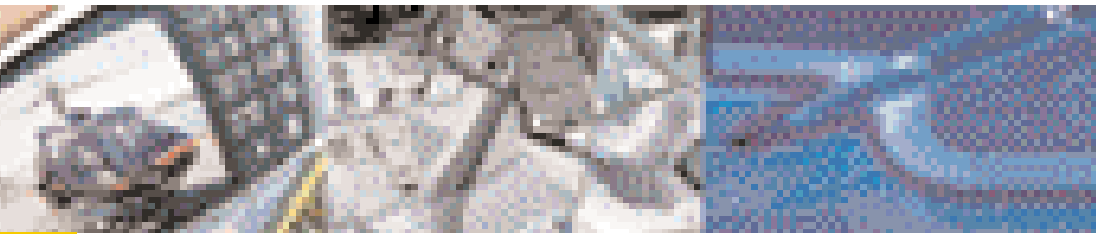
Neither the ARTEMIS SRA WG nor any person participating in this working group is responsible for the use which may be made of the information contained in the present document.

www.artemis-office.org

All rights reserved

© 2006 by





Strategic Research Agenda

Design Methods and Tools



Strategic Research Agenda
Design Methods & Tools

Preface

ARTEMIS (*Advanced Research & Technology for Embedded Intelligence and Systems*) is a 'European Technology Platform'. This is a public-private partnership led by European industry with the goal to establish and implement a coherent and integrated European research and development strategy for Embedded Systems.

Embedded technologies are becoming dominant in many industries, such as communications, aerospace, defence, manufacturing and process control, medical equipment, automotive, and consumer electronics. This trend is likely to continue, given the ever-increasing possibilities for new applications offered by ever-advancing communications, embedded computing devices, and persistent storage.

Industries using and developing embedded systems differ significantly in business and technical requirements and constraints. Development cycles of complex industrial equipment, such as aeroplanes, industrial machines and medical imaging equipment, but also cars, are much longer than the development-cycles of other high-volume, cost-dominated devices for private customers, such as DVD players, mobile phones, ADSL modems and home gateways. Safety requirements are different for an aeroplane, for a car and for a mobile phone. Security, privacy and data-integrity all pose differing requirements in different environments.

Industry of a specific domain is increasingly confronted with the integration of requirements from other industries. In cars, for instance, there are not only the traditional safety requirements, but also the requirements from the consumer and mobile industry, with the increasing integration of entertainment and mobile communication into the

total system. We see little cross-fertilization and reuse over the different industrial domains, as segmentation of markets with their differing requirements has resulted in a fragmented supply industry and research field.

One of the main ambitions of Artemis is to overcome this fragmentation, cutting barriers between application sectors so as to 'de-verticalize' the industry, sharing across sectors tools and technology that are today quite separate and establishing a new embedded system industry that supplies tools and technology that are applicable to a wide range of application sectors.

Embedded systems usually do not operate in isolation, but are often used in combination with other systems to realize an overarching function. Such larger systems are referred to as 'systems of systems'. Examples are a digital television that is integrated in the digital home, a medical diagnostic device that is embedded in the hospital workflow environment, and a car that interchanges information with other cars in its vicinity in order to improve safety. Also the large infrastructural systems, such as the air traffic control system, the electric power grid and of course the telecommunications infrastructure may be viewed as complex systems-of-systems. These systems are characterized by large-scale networked integration of heterogeneous and often intelligent components.

In the extreme, we see the formation of 'sensor networks', even aggregations of 'smart dust', where completely new challenges have to be addressed, related to such different research areas as low power communication, energy scavenging, micro devices,

sensor and data fusion, and controlled emergence of system properties.

Given that the environment of open systems of systems cannot be controlled and specified, and thus cannot be completely modelled, it follows that the reliability and performance of a system may be compromised by unforeseen environmental behaviour. Consequently, we will also require new and so far unexplored approaches to safeguard the safety, security, reliability and robustness of the embedded systems of the future. The use and integration of off-the-shelf components is also a challenge, as such components were not designed from the perspective of the decomposition of the system at hand.

The changeover from design by decomposition to design by composition raises some of the most challenging research and development questions in the embedded systems domain today.

This change, and the ambition for cross-sectoral commonality, inspires much of the specific research proposed in the Artemis Strategic Research Agenda.

The ARTEMIS Strategic Research Agenda (SRA), published in June 2005, outlines the objectives and the research topics that need to be investigated in the field of embedded systems. This present document is one of a trio of documents that amplify that original SRA with more specific research priorities. These three parts of the 'full SRA' are concerned with:

- Reference Designs and Architectures,
- Seamless Connectivity & Middleware, and
- System Design Methods & Tools.

The Reference Designs and Architectures SRA establishes common requirements and constraints that should be taken into account for future embedded systems, and will establish generic reference designs and architectures for embedded systems that can be tailored optimally to their specific application context.

The Seamless Connectivity & Middleware SRA addresses the needs for communication at the physical level - networks; at the logical level - data; and at the semantic level - information and knowledge. Middleware must enable the safe, secure and reliable organization - even self-organization - of embedded systems under a wide range of constraints.

The Systems Design Methods and Tools SRA sets out the priorities for research into the ways that these systems will be designed in future so as to accommodate - and optimise the balance in achievement of - a number of conflicting goals: system adequacy to requirements, customer satisfaction, design productivity, absolute cost, and time to market.

Each part of the SRA was produced by a group of experts that devised their own method of working. While the three Expert Groups liaised so as to achieve coverage and avoid inconsistencies, each of the three documents has its own structure and style, with this preface being the only common element.

All three of these parts of the SRA are living documents that will be continuously refined and updated as research results arrive and as the technological and societal environment changes during the coming years.

Contents

Introduction	6
DESIGN METHODS AND TOOLS: THE SCOPE OF THIS SRA	6
OBJECTIVES	6
METHOD OF WORK	6
STRUCTURE OF THIS DOCUMENT	7
The ‘context of use’ for system design methods and tools	7
The research agenda for system design methods and tools	8
TOOL REFERENCE FRAMEWORK	8
END TO END DEVELOPMENT PROCESS OPTIMIZATION	9
SUMMARY	9
Research priorities architecture	9
Research towards implementation of the tool reference framework	10
ARCHITECTURE TOOLS	10
Capabilities engineering	10
<i>Environmental Modeling</i>	12
<i>Functional Design Tools</i>	12
<i>System Architecture, Co-Design, Distribution</i>	14
Design, Implementation & Verification Tools	15
<i>Application Software Design & Verification</i>	15
<i>Hardware-related Software Design & Verification</i>	17
<i>Application Software Code generation, IDEs, Compilers...</i>	18
<i>Hardware Design & Verification: behavioural synthesis and signal integrity</i>	19
<i>Real-time Operating Systems</i>	21
Integration Tools	21
<i>System Integration & Testing</i>	21
<i>Simulation & Virtual prototyping</i>	22
Transversal Tools	24
<i>Certification, Safety Planning</i>	24
<i>Requirements and Traceability Management (including Use Cases) & Configuration</i>	
<i>Management, Methodology, & Life Cycle Management</i>	25
<i>Tool Integration, Frameworks</i>	27
End-to-end process optimisation	27
Model-Based Design Flow Optimisation	27
Model-Based Validation & Verification Flow Optimisation	30
Global HW+SW Solution Verification & Optimisation	32
Glossary	34
Contributors	36

Introduction

Design methods and tools: the scope of this SRA

The Artemis SRA identifies ‘Design Methods and Tools’ as an important area of research. Design methods and tools are essential for rapid design and prototyping, without which it is unrealistic to attempt development of such complex systems.

The objectives for research in this area are: design efficiency, systematic design, productivity and quality. For example, radical design and verification methodologies, including specific approaches for systems modelling and simulation, enabling both software and hardware instantiation from high-level descriptions with automatic co-verification, are required in order to achieve an order of magnitude advance in productivity. We will call the collection of novel tools and design flows to be developed the ‘ARTEMIS Method’.

The research required to establish the ARTEMIS Method will embrace:

- the development of specific tools, integrable into the core work-flow, that address key design problems of Embedded Systems. These design tools will address the needs of heterogeneous system structures;
- the management of the design process so as to accommodate the growing complexity, and provide product hierarchy, supply chain, and information flow management.
- interoperability between tools and procedures that are included in the ‘ARTEMIS method’;
- open interface standards, to ensure large acceptance of the ARTEMIS Method by securing the intellectual property rights of the specific tools developed to support it;
- methods and tools for systematic traceability of component properties and their attributes, including safety and dependability, during development and integration;
- methods and tools for simulation, automatic validation and proving, and verification and validation (V&V);
- methods and tools for developing product lines of embedded systems.

However, it is not enough to develop tools in isolation from each other. They must, in combination, work together in a coherent fashion to support complex design processes so as to achieve multiple - and often conflicting - objectives. This SRA therefore also includes the research required to achieve coherent interoperation and integration between tools and methods, in order to optimise the end-to-end development process.

Physical design of semiconductor is considered out of scope: it is discussed in other initiatives such as the ENIAC

European Technology Platform initiative. However, particular care has been taken to integrate the ARTEMIS method with tools reference frameworks and methods developed in ENIAC as well as within the two other expert groups of the ARTEMIS Platform dedicated to Reference Design and Architecture and Seamless Connectivity and Middleware.

During refinement of this Strategic Research Agenda cognisance has been and will continue to be taken of similar domain-specific methods and tools analyses already undertaken in initiatives such as ITEA, MEDEA+ and in the course of the 6th Framework Programme program and in preparation for the 7th Framework Programme.

Objectives

ARTEMIS research into Design Methods and Tools for Embedded Systems is motivated and guided by the following targets for embedded systems development:

- reduce the cost of system design by 50%
- achieve 50% reduction in development cycles
- manage a complexity increase of 100% with 20% effort reduction
- reduce by 50% the effort and time required for re-validation and re-certification
- achieve cross-industry reusability of Embedded Systems devices
- inter-operability, at a deep semantic level, between tools and procedures that are included in the ‘ARTEMIS method’.

For each of the research priorities that are described in the body of this document, the nature of their contribution to achievement of these objectives is indicated.

Method of work

The work of the Expert Group that prepared this part of the Artemis SRA was structured in two phases. The first phase established a ‘tool reference framework’, incorporating system design, application software design, hardware related software and middleware design, electronics system level design, hardware design, etc. within a coherent framework.

In the second phase, the Expert Group firstly referenced existing tools and methods and how they address design issues and challenges outlined by the ARTEMIS SRA. Then, based on the perceived gaps between present capabilities and the needs for ARTEMIS, the Expert Group identified a set of priorities for research and anticipated timescales for the results of that research.

In a parallel activity during this second phase, the Expert Group identified the priorities for ‘transverse research themes’ to achieve end-to-end process integration and optimisation.

The research priorities that were identified form the main body of this present document.

Structure of this document

Following this introduction are two short sections that provide an overview of the context that should be considered in the planning of tool and method research and an overview of both sets of research priorities - those that fill out the 'Tool Reference Framework' and those that will support the interoperability and integration of such tools.

There is then a table that illustrates the structure of the research priorities.

These introductory sections are followed by the main body of the document, which is the Artemis Design Methods and Tools SRA itself. This is in two parts:

- Research towards implementation of the Tool Reference Framework, and
- Research to enable end-to-end process optimisation.

In each part, each group of research topics has first an introduction that describes the 'landscape' of the area - typically the problems faced by industry at present; the present state of methodological and tool support; the desires and prospects for future methods and tools.

Next, for each group of research topics the landscape is followed by a table that sets out, briefly, each specific research topic; the objectives for the research into that topic; the ways in which the research should contribute to the Artemis objectives; and approximate expectations for the timescales in which results may be expected.

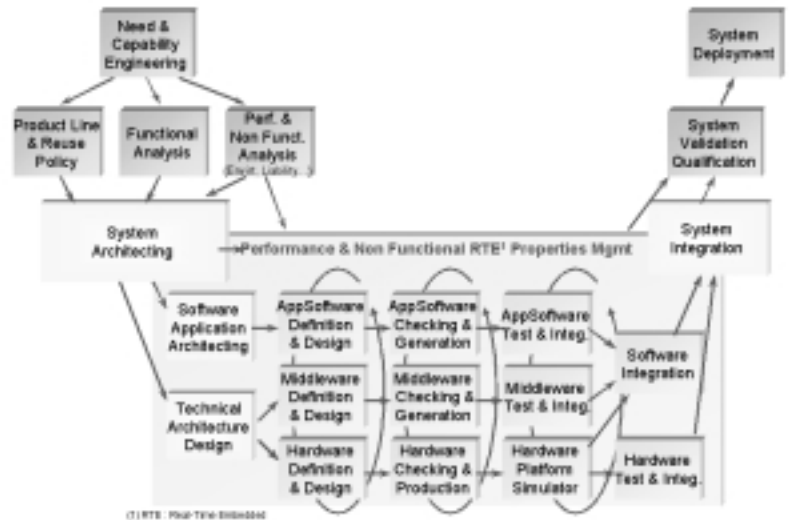
The 'Context of Use' for System Design Methods and Tools

A typical example of a design process for a real-time embedded system is illustrated in the following figure.

A first and important requirement is that the design process should enable the accommodation and reconciliation of different and potentially conflicting constraints.

These constraints include the needs to:

- realise and deliver expected user-level functions
- satisfy expected performance
- optimise performance-to-cost ratio
- satisfy stringent certification requirements
- ensure required dependability
- preserve safety for user and environment
- manage security issues

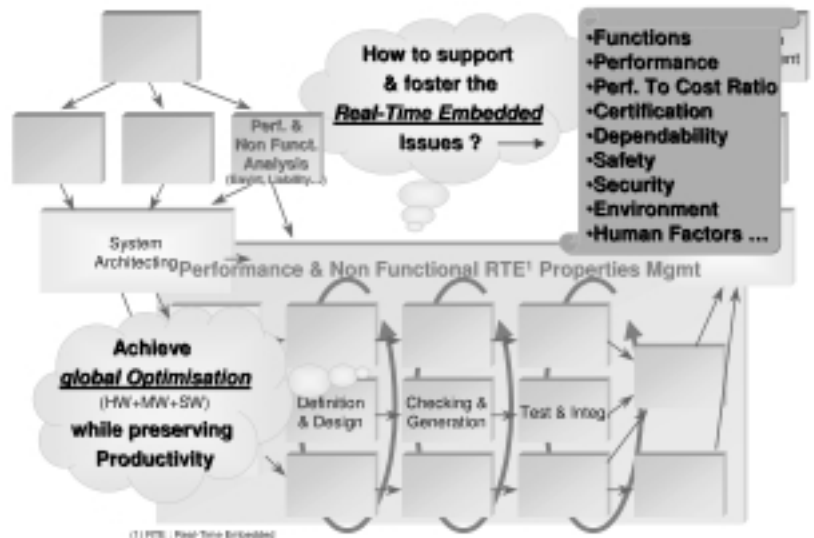


- preserve environment, adapt to it (e.g. temperature, shocks, power availability)
- optimise use via proper consideration of human factors
- and more...

A second requirement is to help achieve global optimisation between hardware and software items, behaviours and properties, while preserving productivity, achieving target costs, and satisfying those constraints.

Increasing complexity of needs, complexity of systems and complexity of interactions between systems and their environment make system design cost a crucial issue for industrial organisations. At the same time, there is constant pressure to shorten the time to market.

Most solutions to the achievement of these requirements need mixed hardware and software solutions, intricately intertwined. The final RTE System is the result of complex trade-offs and compromises, that constitute a significant source of complexity during system design, as illustrated below.



Frameworks, guidelines, aids to drive and tools to assist the design process are the keys to an efficient, cost-effective system development process.

A third requirement is to help improve system quality and design productivity through reuse and early validation support, while reducing risk in development and in failure of the system to meet the needs for it.

To achieve these aims there is a strong desire to be able to capitalise and reuse designs, to detect and correct defects early, and to validate systems early against the customers’ needs and the conditions of use.

The Expert Group considers that the way forward to satisfy all these requirements, and to deal with the complexity increase, the conflicting constraints on systems, and the need to maintain and increase productivity, requires architecture-centric, model-driven approaches.

This approach is summarised in the table below:

Purpose	Goals To be achieved	Required
Manage complexity; reduce time to market; capitalise	Formalise, model, architect	Methods, languages, meta-models, architectural patterns
Increase productivity by assisting & aiding engineering	Make solution emerge	Domain know-how; decision aids & techniques
Detect errors & misconceptions early	Internal checks	Domain rules & criteria; patterns; ‘check processors’
Detect inadequacy for the need; analyse impact	Assess, evaluate wrt need/definition	Traceability; model-mapping; evaluation tools
Ensure & secure conformance to non-functional properties	Prove; demonstrate	Formal languages; proof techniques
Increase productivity by automatic or assisted production	Exploit; produce	Model transformation, [code] generation, simulation...

Model-based design and verification has thus been identified as a fundamental research priority that should underpin more specific research. Models may address the user and/or procurer needs for the system and its properties, the environment of the system, the system architecture, functional assets, software applications, middleware, hardware, and many other facets of a system.

Model-based developments rely on the use of explicit models to describe development activities and products. Explicit process and product models allow:

- the definition and use of formal development steps that are a priori correct by design,
- the generation of proof obligations for a given transformation,

- the traceability of requirements,
- development process documentation.

This process is complemented by *a posteriori* formal verifications of the executable code (e.g. abstract interpretation) in order to demonstrate critical end-product properties, such as:

- semantic equivalence between source models and automatically generated binary codes,
- safety of numerical computation : even a well conditioned numerical algorithm (at design time) might have an unstable behaviour once coded, compiled and executed on the target computer,
- absence of Run Time Errors (e.g. overflows, division by zero, access outside an array),
- safe use of actual computing resources (e.g. the WCET worst case execution time computation cannot be accurately performed at design level in a ‘safe by construction’ approach)
- proof of functional properties on code not produced automatically.

The Research Agenda for System Design Methods and Tools

The requirements indicated above reveal two major, parallel, themes of research:

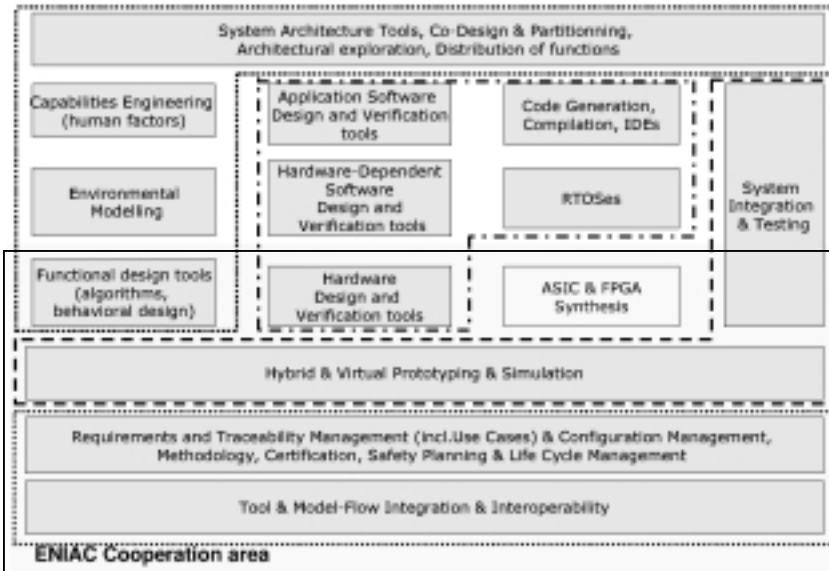
- the advancement of methods and tools, and
- the optimization of the systems design processes, and of the methods and tools themselves for use within those processes.

To address these needs, the Expert Group took a two-pronged approach to the identification of research priorities:

- identification of the full range of processes that need support of disciplined methods and tools - a **Tool Reference Framework**, to structure tool research *per se*.
- identification of a set of transverse research requirements for **End-to-end Development Process Optimization**, to enable tools to integrate and interact effectively so as to support the global processes of RTE system design.

Tool Reference Framework

The Expert Group established - and continues to refine - a ‘Tool Reference Framework’ that characterises and categorises the wide range of tools that support RTE system design and development. The structure of the first part of the ARTEMIS SRA for Design Methods and Tools matches the structure of this framework:



- inter-operation and sharing of semantically rich data between them all.

These concerns and ambitions motivate research into a range of transverse themes that are set out in the second part of the Artemis SRA for System Design Methods & Tools.

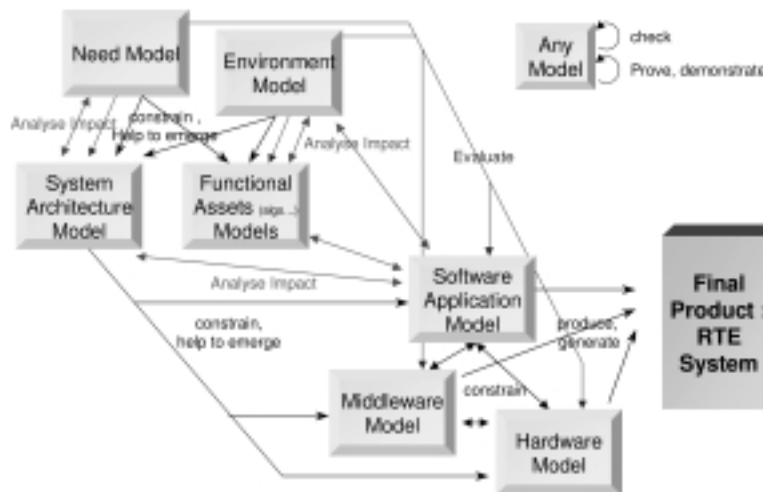
These transverse research themes will complement more traditional tool-oriented approaches, with a more global perspective, to achieve global end-to-end process optimization, and especially to:

- preserve a unified process for global, simultaneous, optimisation of combined RTE software, middleware, and hardware;

- establish model-driven and architecture-centric approaches that will provide global and unified solutions to the present challenges for RTE systems development;
- enable seamless integration with enterprise-wide extended system engineering processes (supply chain management, full product configuration management...)

End to End Development Process optimization

As indicated above, the links and transitions between models are at least as important and valuable as the development of techniques and tools for modeling of different aspects of systems design. The diagram below indicates just one set of such interactions. The emergent properties of these interactions have a major impact on the effectiveness of a suite of methods and tools.



Summary

The ARTEMIS approach to the realization of the objectives for system design methods and tools is to:

- advance methods and tools through implementation of the tool reference framework, and
- achieve end-to-end process optimization through realization of a model-based design flow.

The body of this document explains and describes the detailed research priorities for each of these major themes. These are summarized in the Research Priorities Architecture indicated below, which may serve as a structured index of the priorities.

Research Priorities Architecture

The table following indicates in more detail the structure of this part of the ARTEMIS SRA, with its two major elements - method and tool RTD to implement the Tool Reference Framework, and requirements for 'transverse' RTD to achieve end-to-end process optimisation by ensuring that the methods and tools interoperate effectively and synergistically:

Models, and the tools based upon them, must work together to enable a global approach that addresses the end-to-end development cycle in its entirety in order to support the complex design processes and multiple - and often conflicting - objectives that were described in the introduction.

RTE Systems methods and tools must, in combination, achieve:

- global optimisation between all 'system views' and constraints;
- harmonious and synergistic relationships between need, system, environment & user;

Research towards implementation of the Tool Reference Framework

■ Architecture Tools

- ◆ Capabilities engineering
- ◆ Environmental Modelling
- ◆ Functional Design Tools
- ◆ System Architecture, Co-Design , Distribution

■ Design, Implementation & Verification Tools

- ◆ Application Software Design & Verification
- ◆ Hardware-related Software Design & Verification
- ◆ Application Software Code generation, IDEs, Compilers...
- ◆ Hardware Design & Verification (Behavioural Synthesis & Signal integrity)
- ◆ Real-time Operating Systems

■ Integration Tools

- ◆ System Integration & Testing
- ◆ Simulation & Virtual prototyping

■ Transversal Tools

- ◆ Certification, Safety Planning
- ◆ Requirements and Traceability Management (including Use Cases) & Configuration Management, Methodology, & Life Cycle Management
- ◆ Tool Integration, Frameworks...

Research into end-to-end Process Optimization

■ Model-Based Design Flow Optimisation

- ◆ RTE Architecting Techniques & Patterns
- ◆ [Meta]-Modelling for RTE
- ◆ Engineering Continuum & Impact Analysis
- ◆ Model transformation to Reuse
- ◆ Use of heterogeneous & multi-domain models (continue efforts)

■ Model-Based Validation & Verification Flow Optimisation

- ◆ Early Design Validation Support
- ◆ Early Product Validation Support
- ◆ Formal Proof Techniques & Support
- ◆ Mixed Real and Simulated prototyping
- ◆ Validation Strategy optimization
- ◆ Automated connection between modeling and V&V tools

■ Global HW+SW Solution Verification & Optimisation

- ◆ Decision Aids for Solution Emergence
- ◆ Integration of behavioural synthesis with HW design tools
- ◆ Timing & Power & Resource Consumption Verification & Optimization
- ◆ Hardware/Software Optimization

Research towards Implementation of the Tool Reference Framework

Architecture Tools

Capabilities engineering

The term 'Capabilities Engineering' as used in this document encompasses major activities intended to ensure product to need adequacy, through a relevant analysis of what is expected from the system to be developed; how the system should answer these requirements; and how to support and ease these activities.

Some strong expectations from this engineering activity relate to

- ability to identify new potential capabilities for next generation systems, and correct assessment of the gap to fill in order to achieve these new capabilities, thus fostering innovation
- accurate knowledge of the real use conditions and necessary capabilities & functions of the system, in order

to reduce 'over-requirements', (i.e. requirements that are not compulsory for the real-life use of the system), and select relevant trade-offs (performance, functions versus cost...)

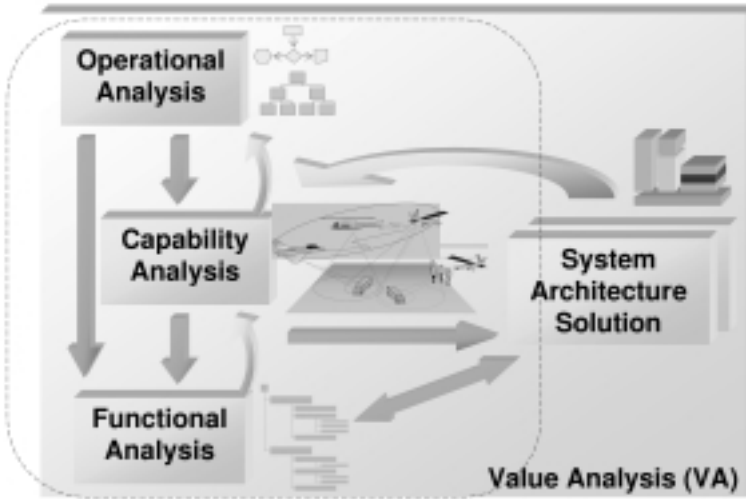
- realistic definition of operational scenarios to secure and support evaluation, comparison of solutions, dimensioning, simulation, qualification.

An example of a structured approach to capability engineering is illustrated in the following figure.

- Operational Analysis (OA) identifies operations & tasks involved in the organisation embedding the System, or individuals using it, along with situations to be addressed by the system;
- Capability Analysis (CA) deals with new capabilities to be offered by the system in these situations
- Functional Analysis (FA) defines the functions required of the System or Operators (From OA & CA).

A snapshot of the current state of the art would show: some approaches usually devoted to information systems (Zachman, DoDAF-like Architecture Frameworks), addressing domain specific issues (military, business process support...)

- some proprietary capability analysis approaches, also depending on domain (military, constrained systems such as traffic management...), and more developed, functional analysis methods (e.g. Functional Analysis with Value Analysis (e.g. AFAV)



- Dependability Analysis Support,
- Criticality Management,
- Performance Guarantee...
- environmental issues,
- mobility schemes & constraints,
- human factors analysis,
- non-functional requirements (liability, dependability, certification, ability to evolve...)

These are seldom taken into account in current practices.

Moreover, no method nor process nor formalism currently exists to ensure and secure emergence of a solution, especially in RTE systems, where finding the best compromise between contradictory

A great amount of work is yet to be done in the field of capability engineering, especially when targeting Real-Time Embedded Systems. In most cases, there is only an ad-hoc 'intellectual process', usually not formalised. This process, even when it does exist, is usually poorly tooled and, more problematically, mainly focuses on functional analysis, hardly addressing real-time embedded issues, such as

- Impact Analysis, Value Analysis (of the Solution versus Need),

constraints and necessary trade-offs (functional need, performance, cost, power consumption, weight, safety aspects) are of prime importance.

All this lack of formalised and tooled approach results in inadequate alignment of system design with the tasks to be performed by the user, and insufficient consideration of user requirements in the development and acquisition processes (specification of mainly technical functions instead of operational capabilities...)

Research topic	Technical objectives	Contribution to ARTEMIS objectives	Outputs of research and timescales
Define a technical process for RTE capability analysis & need mining supported by RTE-oriented models & tools	Model & <i>quantify</i> need incl. non-functional requirements : <ul style="list-style-type: none"> ■ Performance, Real-Time constraints, ■ Resource Consumption, ■ Performance To Cost Ratio; ■ Certification, ■ Dependability, Safety, Security; ■ Hardware/software co design & interleaving, ■ Environmental constraints, ■ Human Factors, MMI... Model dimensioning, dependability , human factors, criticality... Integrate usability engineering & use requirements analysis into system requirements engineering (human factors) Support relevant <i>quantitative</i> scenario creation & assessment Deliver explicit product policy support Define a customisable approach to adapt to different RTE domains	Managing Complexity Increase and Reducing Design Cost	Process & Models end-07 Prototypes end-08 First Tools end-09
Build methods & tools for solution assessment wrt need	Evaluate & compare need compliance, scenarios fulfilment <ul style="list-style-type: none"> ■ include performance, dimensioning, dependability, human factors, criticality... aspects ■ support quantitative bi-directional impact analysis for trade-off support ■ check usability aspects related to embedded specific context 	Managing Complexity Increase and Reducing Design Cost	Methods early version end-08; first tools end-10
Create an approach and supporting tools to aid solution emergence from need analysis	Need rationalisation, auto-organisation, factorising... <ul style="list-style-type: none"> ■ resolution of conflicts in performance, dimensioning, dependability, human factors, criticality... multi-criteria optimisation ■ allocation of requirements to product versus user, with respect to usability ■ support reconciliation of product policy & specific need 	Managing Complexity Increase and Reducing Design Cost & Cycle Time	Approach early version end-2008; first tools end-2010

Environmental Modeling

The environment of an embedded system will influence the system in three major ways. First the embedded system has to physically withstand the physical environment it will operate in i.e. temperature range, electromagnetic interference, humidity, vibrations. Secondly the system has to obey legislation regarding electromagnetic emission. Thirdly the system has to handle environment dynamics of sensors and actuators still providing correct system functionality. The three aspects call for different methodologies and tools to enable rapid, cheap and successful design and implementation of embedded systems. All aspects will be addressed in the following.

Withstanding the physical environment will include issues such as, temperature, vibration, electromagnetic interference, stress etc. These issues are currently modelled using general tools like FEMLAB, Matlab, Simulink, and special tools like Microwave studio. Simulations are based on physical modelling in up to full 3D. In general these tools are not compatible and interoperable with other major hardware and software design tools.

Enabling the embedded system functionality to handle a varying environment asks for capabilities to design for not all but at least most of the environmental aspects that are important for the intended product. This put the focus on, for example, sensor and actuator modeling, antenna and communication channel modeling. The current state of the art consists of many diverse and incompatible tools developed in Fortran/C/C++ or using Matlab. Here, as well, the available tools are not compatible and not inter-operable with other major hardware and software design tools.

The current main industrial issue, concerning both major aspects, is to enable reliable modeling and simulation of large complex systems up to complete product modeling. The second is the need for reliable “quick and dirty” tools for design and manufacturing checking. Finally modeling tools concerning both aspects have to be integrated into the hardware and software design flow and its suite of tools.

Research topic	Technical objectives	Contribution to ARTEMIS objectives	Outputs of research and timescales
Integration of environment modeling and simulation into the HW and SW design flow	<p>Obtain integration of environmental modeling tools in to the HW and SW design flow and their suite of tools like Spice, VHDL, UML etc.</p> <p>Enable environmental modeling supported HW-SW co-design</p> <p>Provide integrated design environments enabling efficient design and verification methodologies</p>	<p>Will shorten product development cycle.</p> <p>Will improve product quality, safety, security and dependability.</p>	<p>2009: Design environments integrating environmental modeling and simulation into HW-SW design flow.</p> <p>2013: Commercial tools for selected environmental issues</p>
Efficient modeling and simulation of environmental effects on embedded systems in large complex systems	<p>Efficient and accurate tools for modeling and simulation of environmental effects like electromagnetics, temperature, vibration, stress etc.</p> <p>Provide 3D modeling and simulation of large and complex systems eg. cars, airplanes, railway systems, power systems, etc.</p> <p>Robust and adaptable modeling and simulation tools for rapid validation of design ideas.</p> <p>Curricula and University networks for education of current and future engineers in modeling and simulation of embedded system functionality in large and complex systems.</p>	<p>Will shorten product development cycle.</p> <p>Will improve product quality, safety, security and dependability.</p> <p>Will provide directions for sustainable competence development in the field</p>	<p>2009: Curricula and University networks for education of current and future engineers in environmental modeling and simulation.</p> <p>2013: Accurate full 3D modeling and simulation for most major problems are possible to reasonable computational cost.</p>

Functional Design Tools

Due to the lack of methods and tools for describing system functionality and non-functional requirements at early stages in the design process, until now models have usually been based on unnecessarily low level concepts that lead to problems in validation and verification (no formal verification of the system functionally, low simulation performance). We even lack understanding of how to describe and manipulate such system descriptions.

Available functional modelling approaches have serious limitations:

- functional models (their design languages or at least their semantics) tend to be tool specific, severely limiting tool interoperability and reuse.
- domain-dependent Formalisms, Flows & Tools (e.g. Signal Processing, MMI design, Control & Command, System Modes Management...)
- little help for general purpose functional processing.

Techniques to integrate heterogeneous multi-domain functional models are inadequate. As a consequence, ad-hoc approaches

are adopted, most of them having serious deficiencies. Such integration capabilities would significantly contribute to first-time right designs, as a lot of the integration issues could be captured up-front and designs could take them into account.

There are some aids to express design solutions and a few aids to check solution, but none to help solutions emerge. Such tools would improve design productivity considerably.

Research topic	Technical objectives	Contribution to ARTEMIS objectives	Outputs of research and timescales
<p>Models and model manipulation: Define a modelling approach and framework with tools & methods support.</p>	<p>The approach and framework should:</p> <ul style="list-style-type: none"> ■ Start at a sufficiently high level: part of research is to define this level (paradigm, principle, concept, etc.) ■ Captures functional requirements and non-functional requirements ■ Has explicit and formal semantics (to allow automation and verification) ■ Allows (consistent) integration of heterogeneous multi-domain models ('meta'-models) ■ Allows consistency and interconnection with models at other (abstraction) levels (needs alignment between subgroups to avoid interoperability problems and to ensure that an integrated design flow is obtained) ■ Allows the back-annotation of lower level details into relevant/understandable parameters at higher levels ■ Supports semantics preserving transformations and refinement ■ Efficiently copes with legacy components <p>Formalize techniques for modeling, e.g. by the definition of new modeling language or extensions to current ones</p> <p>Tools/methods that allow (semi-) automated manipulation of the models enabling incremental optimisation</p> <ul style="list-style-type: none"> ■ Transformation inside the same abstraction level (with guaranteed or easily verifiable invariance of properties) ■ Refinement (propagation) of both functional and non-functional requirements to lower abstraction levels 	<p>Contributes to :</p> <p>Managing Complexity Increase and Reducing Design Cost</p> <p>Productivity increase by subsystem re-use</p> <p>achieve cross-sectoral reusability of Embedded Systems devices</p> <p>Interoperability between tools and procedures that are included in the "ARTEMIS method"</p>	<p>2009: Modelling framework available, no tools, nor methods support yet</p> <p>2011: Modelling practices and methods and modelling environments (tools) available</p> <p>2011: Techniques for manipulation (transformation and refinement) of models available</p> <p>2013: (semi-)automated tools for model manipulation (transformation and refinement) available</p>
<p>Validation and verification</p>	<p>Formal Proof of Design should:</p> <ul style="list-style-type: none"> ■ Identify key properties to be respected ■ Define formal, high abstraction-level languages to check ■ Define Tools to support and guide design according to these. <p>Validation</p> <ul style="list-style-type: none"> ■ a formal approach to capture the user requirements (part of model of research priority above) ■ Forward these requirements into a test sequence for automating the validation and verification of both functional and not-functional properties, will lead to significant improvements in design time. <p>High level estimates</p> <p>System simulation</p> <ul style="list-style-type: none"> ■ Tools should simulate the behavior of the embedded application in its entirety giving information at different levels: functional (linked to the modelling research priority above), logical, implementation ■ Efficient simulation engines 	<p>Contributes to managing complexity increase and reducing design cost</p>	<p>2009: Simulation framework (tool) available.</p> <p>2010: Estimation methods available, partly implemented in tools already.</p> <p>2011: First methods available (all manual) to support validation, building on the modelling framework.</p> <p>2013: First subset of formalism present in tools.</p>
<p>Complex systems and environments</p>	<p>Dynamic environment: Design methodologies must support flexible and adaptive systems that respond to external and internal triggers and hence to provide the necessary quality of service and error resilience. This requires an optimized dynamic runtime reconfigurability (i.e., the capability of embedded systems to auto-adapt or to be reconfigured to changing physical/context-aware conditions).</p> <p>Modelling and Design approaches to cope with incomplete specifications / unknown environments, ensuring that the system still behaves correctly.</p> <p>Approach to design systems based on unreliable components.</p> <p>Design methodologies must support heterogeneous systems with a unified design flow.</p>	<p>Contributes to managing complexity increase and reducing design cost</p>	<p>2010: Extension of modelling framework to support heterogeneous systems.</p> <p>2010: Extension of modelling framework with concepts for dynamic systems.</p> <p>2011: Extension of modelling framework with concepts for dealing with incomplete specification.</p> <p>2012: Extension of modelling framework to cope with unreliable components.</p> <p>2013: Tool support for the above.</p>

System Architecture, Co-Design, Distribution

The embedded systems architecting practices in the industry are quite divergent, which is a sign of an immature domain of engineering. System architects have to rely on add-hoc methods, as there is no established systematic engineering practice. Design automation cannot be achieved to any significant extent, because design is not based on true system modelling, or if it is, then the modelling approach (i.e. metamodels) is proprietary. As a consequence, embedded systems architecting is still largely a 'pen and paper' job. If tools are used at all, then they are quite simple, such as Excel for exploration and impact analysis or UML for drafting, in a quite informal manner. Architectural design decisions are largely based on experience of past designs, and this is difficult to apply to new situations. There is no true traceability from requirements to architectural solutions. Requirement management tools (e.g. DOORS) have too simplistic semantics for this purpose. Some system companies have formalized their own architecture reference so as to support product line policy, to organize development & integration, and to build a bid proposals.

Although the mainstream practices of the industry can be characterized as rather primitive, there are more advanced technologies available or emerging, such as:

- A few commercial tools for architectural exploration exist (e.g. CoFluent, ConvergenceSC, Virtio), but they cannot fully capture the complexity and the architectural variety of state-of-the-art Hardware-Software platforms.
- A few tools for HW-SW mapping exist (e.g. Matlab/Simulink FPGA tools, Accelchip), but they are mostly restricted to FPGA targets, and they are highly domain-specific (e.g. digital signal processing systems).
- Academic tools for architectural synthesis exist (non-practical, very restricted). Commercial tools are appearing but still in their infancy (Celoxica Nexus-PDK)
- Automatic or semi-automatic HW-SW partitioning is supported by a few academic tools (restrictive models and architectures)
- Work going on to define standard UML profile(s) for embedded systems engineering (SysML, MARTE rfp)
- Standards, such as SPIRIT, are emerging for integrating multi-vendor tool and IP design methodologies and making design-chain integration efficient
- Architecture Frameworks: DoDAF, MoDAF, Agate, ...
- Architecture Languages (ADLs): AADL, C2/SADL, ACME, OLAN, ...
- Architecture modeler: ACME studio, Arch Studio, STOOD, OSATE,
- Hardware/software transactors design tools: Esterel Studio, Spiratech....

The main problems caused by unsatisfactory embedded systems architecting methods and tools are as follows:

- Verification and validation of architectural design choices can be done only after implementation or prototyping. Thus design space exploration, or even design tuning imply very costly iterations.
- Re-use of architectural designs is limited, because tools lack of flexibility and impose severe restrictions to the specification of architectural IP blocks.
- Poor tool support for trading-off multi-processor and interconnect: most of the available tools for modelling, exploration and validation focus on processor modelling, and provide only very abstract and inaccurate models for communication resources and storage hierarchies.
- Current practice hinders the use of novel architectural solutions and inhibits organisations from addressing new challenges (new markets, new capabilities, new solutions, system of system, ...)
- concurrent development and subcontracting is poorly supported
- architectures are not robust to industrial constraints
- there is poor support for any product-line policies.

Research topic	Technical objectives	Contribution to ARTEMIS objectives	Outputs of research and timescales
Establish common modelling practice for embedded system architecture engineering	Metamodels that can be domain specific but should embody commonality across application domains. UML/SystemC profiles. Transactor design tools Techniques for automated manipulation of the models Re-use methodology	Achieve cross-sectoral reusability of Embedded Systems devices. Achieve interoperability between tools and procedures that are included in the "ARTEMIS method".	2008-2009: Basic concepts, metamodels and standards proposals emphasizing cross-sectoral reusability 2010-2013: Consolidated metamodels, standard profiles, tools that streamline creation, manipulation, composition and reuse of models

Research topic	Technical objectives	Contribution to ARTEMIS objectives	Outputs of research and timescales
Performance estimation tools	<p>Cover timing, energy/power, memory, etc.</p> <p>Support (according to common metamodel) the separation of application load models and computing/communication capability models.</p> <p>Support continuous improvement of estimation accuracy based on implementation experience and measurements.</p> <p>Support property extraction from various models on various design levels</p>	<p>Contributes to all ARTEMIS objectives, but especially to achieve 50% reduction in development cycles (i.e. iterations)</p>	<p>2008-2010: Modelling techniques, tool prototypes for analyzing full-system performance with respect to multiple cost metrics with a high degree of fidelity.</p> <p>2011-2013: Commercial tools based on the techniques and prototypes previously developed</p>
Practical architectural exploration and development tools based on common metamodels	<p>Automatic tuning of architectural parameters for targeted application workloads (e.g. cache size, bus-width...).</p> <p>Design space exploration with multiple objective functions (e.g. power performance trade-off curve exploration), including sensitivity analysis.</p> <p>Means to guarantee consistency of functional and non-functional properties across models at different levels (requirements, specifications, designs).</p> <p>Support for solution emergence from functional need: design rationalisation, auto-organisation, factorising, architectural patterns, design rules, know-how to be capitalised.</p>	<p>Reduce the cost of the system design by 50%.</p> <p>Achieve 50% reduction in development cycles</p> <p>Manage a complexity increase of 100% with 20% effort reduction</p>	<p>2008-2010: Modelling techniques, tool prototypes capable of (partially) automatic design space exploration, and providing a high level of support for rapid and correct-by-construction design instantiation</p> <p>2011-2013: Commercial tools based on the techniques and prototypes previously developed.</p>

Design, Implementation & Verification Tools

Application Software Design & Verification

Most of the issues and the associated costs in the activities concerning embedded Software Design and Verification are consequences of poor understanding of the software requirements, or may be related to lack and incompleteness of specifications. Requirements are often captured in natural languages and documented in lengthy, verbose documents, which are seldom kept updated during the project lifecycle, when better understanding of the specification emerges.

Specifications themselves are also often expressed in natural languages, lacking formality, verifiability and determinism.

On the other hand, embedded application software is generally developed in high-level procedural languages, such as C, C++, or in a few specific cases Java (most notably Java 2 Micro Edition). The development team has to translate requirements and specifications into the chosen program language and make sure that all the requirements will be satisfied by the developed code.

As a consequence, application software design often suffers from the so-called ‘impedance mismatch’: the software developers start designing their code based on static specification documents, which often are not complete and do not illustrate some critical use cases or conditions that will only be discovered later in the design cycle.

There are some circumstances where a standard graphical modeling language is used as a common language between analysts and application domain experts to mitigate the “impedance mismatch” effect. The most notable example is UML (the Unified Modeling Language) and its profiles which have been specifically developed for embedded system engineering, such as SysML, and UML-RT. UML models are mostly used in a quite informal manner, in essence to get an overview of the system, its main actors and interfaces. There are still many areas in which the details about the requirements and specifications are still expressed in natural language, by means of lengthy Word documents. Even though there are a number of attempts to adopt model-based design practices throughout the industry, these are currently being deployed in an ad-hoc manner, often confined to specific and limited application domains.

As an example, there are good modeling tools and formalisms that are currently used for the specification of Graphical User Interfaces or the behaviour of closed-loop control systems.

- Tools such as Matlab and Simulink, ETAS or Scilab, for instance, are used for simulating and implementing rapid prototypes of new control algorithms.

- Tools such as SCADE are often used as formal specification for data flows and state machines for safety-critical systems, enabling certified and automated code generation in combination with formal verification engines such as Prover, SRI, Offis or Verimag tools
- Currently available Integrated Development Environment (IDE) tools have a good capability and integration of the debugging environment with the source code editor. On the other hand, they are currently rather limited in providing a higher level view of the model during the application debugging.
- A series of point tools such as testing tools (Test RT, Logiscope), execution checking and MISRA compliance checking tools (Polyspace, LDRA...) as well as embedded GUI-design tools (IMAGE, Altia,...) are often used in combination with the abovementioned design tools.

An area where there is a great need of improvements is the capability to verify the coverage of the requirements. This activity is mostly carried out manually by the software developer, and there is limited support provided by automatic tools. This task therefore easily becomes tedious and error-prone and is often sacrificed when the project times or budgets are tight.

Based on the results of past projects there is a common understanding that a significant part of the project cost and time is spent during the validation and test of the full system when in production because some critical aspects of the actual environment where the system is deployed were not captured in the specification or the analysis phases of the project.

In the light of this analysis, the major industrial stakes are threefold.

- First, favouring low-cost development and reuse by implementing complete Model-Based Design software “factories” that enable verification activities previously performed at code level to be performed as much as possible at model level
- Secondly, mastering earlier in the lifecycle the quality of ever more complex software appears as being equally critical. It must be noted that only a few modelling languages deployed in the industry do allow to prove properties about models. Besides, every engineering speciality may use its own formalisms, making difficult the consolidation of models at system level
- Thirdly, qualification and certification issues pose a significant challenge, as even though domain-specific standards do exist (DO-178B for avionics, IEC 61508 for rail, industry and automotive, ITSEC for security-related matters and in particular smart cards...), there is little cross-domain fertilization of methods and tools.

Research topic	Technical objectives	Contribution to ARTEMIS objectives	Outputs of research and timescales
Implement complete Model-Based Design tool-chain (software “factories”)	Provide tool facilities to develop specific software from a common basis at low cost <ul style="list-style-type: none"> ■ further develop tools around model-based design approach (MBD) ■ develop model transformation facilities such as the MDA approach ■ Enable verification activities previously performed at code-level to be performed at Model-level ■ Integrate MBD tools into complete “Software Factories” 	Achieving 50% cost reduction in development cycle	Process and Model flows by 2008 Complete Software Factories by 2010
Provide tool facilities to detect problems earlier in the development lifecycle	Improve Software Quality by developing: <ul style="list-style-type: none"> ■ Tool-aided process and product control ■ Simulation, model assembling and formal model checking & verification facilities ■ Multi-disciplinary modeling tools (Functional modelling, Quality of Service, Interfaces modelling, Operational modelling...) ■ System/ software integrated development environments to avoid design flow gaps 	Reduce by 50% the effort and time required for re-validation and re-certification after change	First set of integrated verification processes by 2009 Complete verification chain by 2010
Provide capabilities to integrate easily specific process, meta-model, profile or architecture in development tools	Enhance Tools adaptability by: <ul style="list-style-type: none"> ■ developing multi-platform transformation ■ develop heterogeneous integration & validation infrastructure ■ develop certification, safety, and RTE profile implementation ■ develop meta-model plug-in facilities in development and integrated validation & verification tools 	Achieve cross-sectoral reusability of designs	First approach early 2008 Meta-modelling and multi-platform transformation by 2010

Hardware-related Software Design & Verification

The production of hardware-related software - this term encompasses device drivers, BIOSes and low-level elements of system kernels (protocols, schedulers, memory management...) - present specific challenges, especially from the point of view of the validation process.

Testing is time-consuming, because of the lack of efficient tools to observe the behaviour of low-level software (which, very often, has to execute before any debugging tools are loaded), especially on parallel architectures. Moreover, this testing may have to be done in a situation where the couple "new hardware + new software" is being validated.

Formal methods (static analysis, program proving...) may be used instead of testing; however, for these methods to be efficient, mathematical models (leading to algorithmic solutions) and usable software tools have to be available to take into account the "low level" semantics of the analyzed software. This semantics include, for example, interrupts, the use of specific registers or memory locations, the respect of strict resources limitations (buffer memory, clock cycles...), power consumption etc. Last but not least, in order to verify the adequacy between the behaviour of the software and the hardware to be supported, a widely used formalism would have to be available to describe the "hardware-side" needs for the software tasks.

In order to avoid the costly validation process, a different approach could be to systematically produce the hardware-related software from higher level models. This of course requires formalism to address not only the above mentioned issues (hardware requirements, low-level software behaviour) but also formalism to describe and specify all the necessary interactions (between different software levels as well as between hardware and software components). Widespread formalisms in this domain are still to be created.

Finally, a mid term we propose to address is the re-use of existing software elements. Experimentally, it appears that, especially in families of hardware components, the specific hardware-related software components are similar, but not identical. A similar observation is that low-level software elements targeted to the same hardware but intended to support different operating systems have to be different implementations yet share a large number of identical mechanisms. Hence, a very significant validation cost reduction can be obtained if the validation process is "factored out" for the common behaviour and has to focus only on the differences.

Therefore, the industrial issues we propose to address as priorities are, schematically:

Enrich the semantic models

This includes relevant formalisms and tools to describe and validate the interaction between hardware and software and the necessary software behaviour.

Focus on low-level software re-use

Methods and tools to develop and validate hardware-related software which can be used on a different operating system, especially in the case of proprietary platforms and when the hardware resources are very different from one platform to the other, as well as to develop & validate families of software base elements energy to be used on families of hardware components.

Observability and debugging

Run-time methods and tools to reduce the cost of the industrial "test and debug" phase.

Research topic	Technical objectives	Contribution to ARTEMIS objectives	Outputs of research and timescales
'Low-level' semantics: methodological, mathematical and software tools taking into account the 'low-level' semantics to model and validate low-level software aspects (buffers, interrupts, memory addressing ...) and the SW/HW dependencies.	New automated proof techniques. Formalism for 'low level' property specification and abstraction. Derive new analysis and debug tools.	Reduce the cost of system design Achieve cross-sector reusability.	2009-2010: New concepts (formalism and models for exact or approximate static analysis or other verification techniques) 2011-2013: New analysis and debug tools deriving from the new concepts

Research topic	Technical objectives	Contribution to ARTEMIS objectives	Outputs of research and timescales
Re-use ■ of low-level software elements; ■ of whole sub-systems; ■ of design and validation tasks.	New standards for software-software and hardware-software interfaces. New standards to formalize the “physical” properties of software (power consumption, bus bandwidth, memory use...). New standards to formalize hardware-software integration (meta-models for HW-SW systems needed). Factor out the design and validation tasks for families of similar low-level components (e.g. drivers for similar hardware) and families of different implementations (e.g. targeted to different OSes) of the same component. Develop meta model and generic simulation model for operating system including standardized interfaces between the OS and Middleware on one side and OS and Application on the other side. Synthesis of functional HW/SW Interface (HW side and SW side) from a single description. Develop Platform independent coding styles and tools, that validate the correct mapping (Formal Equivalence Check of C/C++ vs. Assembler and Machine code) Making formal methods for HW/SW Verification usable, i.e. develop and train methods as well as fully automate verification	Reduction of the cost of system design Reduction of the effort for re-validation or re-certification Enable platform independent reuse of “full IP”, i.e. incl. low-level SW Further Reduction of effort for platform independent reuse of “full IP” incl. low-level SW	2008-2010, models and standards proposals; 2012-13: new software tools relying on the models and standards to implement and support the re-use approaches in an industrial environment.
Observability and debugging	Tool facilities to observe and analyze the behaviour of the low-level system components at run-time	Reduction of the cost of system design Reduction of the effort for re-validation or re-certification.	2009: Facilities to improve the observability of the system 2012-13: Facilities to analyze the behaviour of the system and to identify problems
Joint Hw-Sw diagnostics	Develop methods for jointly debugging of HW and SW with focus on massively parallelism, real time analysis capability (as one representative of physical feature debugging)	Reduction of effort for Reuse of “full IP” incl. low-level SW	2009 .. 2010 : methods supported by pre-industrial implementations.

Application Software Code generation, IDEs, Compilers...

Application software represents indeed the core functionalities of systems. They are the visible part of the iceberg of embedded systems and thus deserve particular care and attention.

Like synthesis in EDA 10 years ago, application software code generation is clearly a very important trend in embedded systems design currently as it conceptually answers many issues about time to market, savings, productivity, quality, reusability, etc.

However, the current state of the practice is that code generation is used in production only in some domain-specific environments like aerospace, mostly in pilot project or rapid prototyping in other domains. On the other hand, there has been widespread use of IDEs and compilers for a very long time.

The current state of the art in code generation blends several complementary techniques, such as UML and Algorithm design tools-based code generation for rapid prototyping (Matlab/Simulink, Rhapsody, Artisan...), formal design tools that enable production code generation (SCADE, ETAS...) for data flows and state machines, and GUI-design tools for embedded graphics (IMAGE, Altia...)

The major industrial stakes for code generators typically fall into 4 categories.

First, implementing efficient code generation from application model to generated code requires the re-use of application models with various implementation patterns (fixed point, floating point, performance trade-offs..), the portability and modularity of application models, and an easy integration with execution platforms (RTOS, compilers, etc...)

The second requirement is, of course, to maximize productivity by ensuring ‘correct by construction’ coherence between design artefacts (requirements, models, and code, with full traceability). Few tools offer the assurance of permanent, automatic coherence between code and models.

Lastly, in order to achieve cross-domain fertilization, the capability to customize code generators through meta-models is a very important challenge.

IDEs and Compilers are more traditional items. However, they are still facing important industrial challenges, such as reducing the gap between design and debug, enabling “round-tripping” between code and design, mastering performance & quality issues (performance analysis and simulation, enabling WCET computations, simulating resources consumption in the case of new code releases).

Research topic	Technical objectives	Contribution to ARTEMIS objectives	Outputs of research and timescales
Certified Code Generation	Implement certified, efficient & safe Code Generation from application model to generated code: <ul style="list-style-type: none"> ■ re-use of application models with various implementation patterns (fixed point, floating point, performance trade-offs..) ■ ensure portability of application model ■ ensure modularity of application ■ facilitate integration with execution platforms ■ enable Certification of code generators ■ ensure correct by construction coherence between design artefacts (Requirements, Models, and Code, with full traceability) 	Achieving 50% cost reduction in development cycle “Right First Time, Every Time” objective Reduce by 50% the effort and time required for re-validation and re-certification after change	Integrated tool chain end 2009
Performance & resource analysis and simulation	Timing analysis such as WCET, power & memory estimation, starting from existing software being able to analyse CPU, I/O ... bottleneck or maximum peak and simulate performance impact of software evolution. Compiler as a support to load & resource estimation/forecast Compiler development languages	Management of complexity increase, especially in timing and performance area	First tools end 2008 Production use by end 2010
Develop Code generators parameterisable through meta-model	Develop code generators that can be parameterised in order to map the design application model to the targeted system	Cross-sectoral reusability of design	Approach by end 2010

Hardware Design & Verification: behavioural synthesis and signal integrity

The issue of mastering complexity of complex IP designs has become of paramount importance in order to master quality and time to market of hardware IPs design and verification, as well as hardware software partitioning and optimization.

Model based design has the advantage of raising the level of abstraction from the realization domain to the problem specification domain. In this way we may reason about general properties of the design rather than a specific implementation under a given (fixed) software/hardware partitioning. Given appropriate methodologies and tools, such models can be synthesized into efficient, reliable (correct, proven by formal methods) software and hardware realizations. Thus, model based design holds the potential to radically improve design productivity and quality.

The issue of signal integrity in embedded systems is critical both from a product quality and from a development time to market point of view. Thus design tools have to enable qualified analysis of signal integrity as frequencies go higher and voltages and currents get lower. This must also take account of the influence of environmental conditions such as temperature, radiation etc.

Although promising, methodologies and tools for model-based design is still a relatively immature field, offering a plethora of research challenges, spanning from specification of loosely coupled real-time systems and methods and tools for analysis, to realization by software and hardware co-synthesis.

Signal integrity

Modeling and analysing tools for signal integrity in many cases share the physics with EMC modeling. Such tools are presently used in industry.

The increase in frequency, use of lower voltages and currents and smaller feature sizes brings the current models and tools to their limit. This is mainly because of full wave effects, 3D effects, and relativity effects that are not reflected in the models. Thus more advanced models are required in the future and, in particular, further measurement techniques for tool verification become critical.

Behavioural synthesis

Novel approaches, known as ‘behavioural synthesis’ have been developed since ten years, in order to be able to express complex behaviours in a more compact and efficient manner than hand-coded VHDL or Verilog.

In particular, a clear distinction between algorithm-dominated designs and control-dominated designs has been made in order to develop domain-specific tools in order to handle the inherent nature of algorithm behavioural synthesis and control/communication synthesis.

For algorithm behavioural synthesis, C-based data entries, enabling data path algorithm optimization, through resource and clock cycle allocation permit RTL or specific co-processor generation. Tools such as Forte Cynthesiser, Mentor Catapult C, Celoxica or Synfora Pico represent this track.

For control/communication synthesis, FSM-based languages, enabling the direct specification of complex sequential behaviour, through visual or textual design, simulation, permit sequentially optimized RTL generation and corner bug detection with formal verification of executable specifications. Tools such as Esterel Studio or Bluespec represent this other track.

The ability to efficiently intertwine algorithm and control/communication behavioural synthesis is naturally an important issue.

Research topic	Technical objectives	Contribution to ARTEMIS objectives	Outputs of research and timescales
Full wave and 3D modeling and verification of signal integrity	<p>Development of signal integrity validation methodologies and tools that will scale with future silicon and HW system technologies. Examples of issues are smaller feature sizes, higher frequencies, lower voltages and currents.</p> <p>Integration of signal integrity modeling and tools in design flow enabling very early inclusion of signal integrity verification. Thus enabling even pre-layout checking enabling adaptive layout rule modification.</p> <p>Develop tools for checking of signal integrity for in-system generated EM environmental with accounting for system operation conditions like temperature, radiation etc.</p> <p>Develop integrated design environments enabling efficient design and signal integrity verification methodologies</p>	<p>Will shorten product development cycle.</p> <p>Will improve product quality, safety, security and dependability.</p>	<p>2009: Design environments integrating signal integrity verification into HW-SW design flow.</p> <p>2013: Commercial tools for selected environmental issues</p>
Measurement technology for verification of modeling tools.	Development of measurement technologies and instruments that enables the validation of signal integrity tools.	<p>Will shorten product development cycle.</p> <p>Will improve product quality, safety, security and dependability.</p>	<p>2010: Accurate measurement methodologies for validation of signal integrity methodologies and tools.</p> <p>2007-2013: Continuous releases of instrumentation enabling verification of signal integrity methodologies and tools.</p>
Model based specification languages and tools for system analysis.	Methodology for capturing soft and hard real-time properties of distributed (loosely coupled) system in a modular (composable and decomposable) manner.	Will shorten system development time by abstraction and module reuse.	<p>2009: Specification of semantics for modeling language, version 1.0.</p> <p>2007-20013</p> <p>Continuous releases of tools supporting formal methods for model based design</p>
Synthesis tools for model based design.	Tools for model based hardware/software co-synthesis.	Will cut production and maintenance costs by cross-layer optimization through repartitioning, (composition/ decomposition)	<p>2008-2013</p> <p>Continuous releases of synthesis tools and compilers for hardware/software co-synthesis.</p>
Behavioural synthesis tools enabling an order of magnitude gain of performance above hand-coding	Development of algorithm-based and control/communication centric behavioral synthesis tools and their integration	Will shorten product development cycle and improve quality	<p>2008: Behavioural synthesis tools enabling better performance than hand-coding</p> <p>2011: Gain of an order of magnitude in speed/area performance</p>

Research topic	Technical objectives	Contribution to ARTEMIS objectives	Outputs of research and timescales
Power-efficient behavioural synthesis tools	Native support of multi-clock design and massive power-optimization	Will improve design competitiveness and reduce time-to market	2010: Power efficient behavioral synthesis
Integration of behavioural synthesis tools with System-level simulation tool	Enable efficient generation of ultra Fast C/ SystemC code from behavioral synthesis tools for fast system-level simulation	Will shorten product development cycle and improve quality	2009: Gain of an order of magnitude for C/SystemC -based synthesis versus hand coding

Real-time Operating Systems

The subject of RTOS per se is addressed in the part of the SRA concerned with 'Seamless Connectivity & Middleware'. This section is concerned specifically with certain special tools associated with RTOS.

Real Time Operating Systems are in widespread use within the Embedded Systems community, with a mix of general purpose COTS, domain specific COTS or domain specific variants of COTS and proprietary RTOSes.

Various trends have been observed in the market, ranging from RTOS certification and/or standardization across portability layers (such as AUTOSAR RTE), RTOS configurability and network-based RTOSes, such as Flexray or TTA-based systems. Also, domain specific RTOSes have emerged as a key trend, such as those in the wireless telecommunications industries.

However, across these various types and usage models of RTOSes, some major industrial stakes can be described as being common across industrial sectors and in particular mastering system complexity, such as dynamic distribution of the operational capacities in the system and its observability; the reduction of costs and duration of development and allowing application software to be RTOS-independent; as well as a higher level dependability and/or predictability of the performance of a given RTOS or RTOS configuration

In order to meet those challenges, two main research topics are summarized below: enabling RTOS performance and execution modelling through system-level schedulability and observability analysis tools and enabling RTOS-neutral architectures through standardization of RTE layers APIs.

Research topic	Technical objectives	Contribution to ARTEMIS objectives	Outputs of research and timescales
Enable RTOS performance and execution modelling thru system-level schedulability and observability analysis tools	System-level schedulability tools RTOS performance and execution modelling tools RTOS performance analysis tools Support to assisted / automatic deployment (multi-processor...) while preserving RTE performance	Reduction of integration cost by definition & control of resources access and consumption.	Approach & Method : end-2008 First tools : end-2010
Enable RTOS-neutral architectures through standardization of RTE layers APIs	High level capacity of system parameter settings (blueprints) to take into account the system/integrator policies	Master increasing complexity. Reduce integration effort.	First tools mid-2008. Integrated with MDE end-2010

Integration Tools

System Integration & Testing

Systems are made of hardware and software components which have to be integrated to build an entire system. Usually, there are a lot of integration steps to follow, such as software integration, hardware integration, hardware/software integration and overall systems integration. During these steps integration tests are executed which concentrate on the interplay of the integrated components.

It is usual for manufacturers to build up systems with components delivered by suppliers. This applies, for example, to manufacturers of cars, airplanes, trains, mobile phones and others. Hence, the manufacturer acts as integrator and has to

concentrate on integration and testing since it is the manufacturer who is responsible for the quality (and maybe certification) of the entire system and final product.

For defining the integration strategy to follow a lot of constraints have to be taken into account. Besides testing, which has its own claims on the integration strategy, other factors are important such as the supplier component delivery dates, management decisions on integration sequences, communication structures btw components and risks of components. It is often not easy to derive a comprehensive integration strategy which satisfies all given constraints.

On a more technical level there are today powerful integration technologies available, such as CAN (www.can-cia.org/can) and MOST (www.mostnet.org) networks for the automotive domain.

However, what is really missing are powerful tools supporting integration strategy planning and integration test design. There is an urgent need for such tools because, as described above, manufacturers are acting as integrators and are responsible for the final product's quality. So far, there are only very high-level or very specialised proprietary tools available to support integration planning and testing. The high-level tools are mostly test management tools, which act on a very abstract level of testing (see e.g. TestDirector from Mercury). The proprietary tools are much too specialised, mostly supporting very special models and specification & programming languages. So there is still a need for integration (testing) supporting tools which could be used for a wide range of domains, models and languages.

Research topic	Technical objectives	Contribution to ARTEMIS objectives	Outputs of research and timescales
Advanced integration strategy planning for electronic systems consisting of many embedded components taking into account many constraints (including, e.g., delivery dates of suppliers, management decisions, ...)	<p>Methods & tools supporting the planning of effective integration strategies</p> <p>Methods & tools to integrate and validate system parts independently</p> <p>Rules that allow to split the system under test in independent parts</p>	<p>Managing complexity increase</p> <p>Improving design efficiency</p>	<p>2008: Method supporting integration strategy planning</p> <p>2010: Tools supporting integration strategy planning</p>
Advanced overall integration test planning. For integration testing the method has to concentrate on the interplay between the integrated components. Enable functional and non-functional validation & testing to start as early as possible in the life cycle	Methods & tools to support the planning of effective integration (& system) tests for electronic systems	<p>Improve quality & productivity.</p> <p>Reduce costs for testing & re-validation.</p>	<p>2008: Method supporting integration test planning</p> <p>2010: Tools supporting integration test planning</p>
<p>Advanced integration test design which are taking into account the following:</p> <ul style="list-style-type: none"> ■ Combine proofs & tests: methods and tools should be developed to enable information gained by static analysis (model checking, source code analysis) to be used by test generators and oracles ■ Generate tests from test objectives: the research activity on test generation from a set of objectives (expressed in an open formalism) and model test coverage needs to be strengthened. As an outcome of this research it should be possible to automatically generate a limited set of tests to a set of objectives ■ Explain tests: ease the cooperation between testing and design tools, by generating comments for a set of automatically generated tests, which could be understood by humans (functional objectives) & by automatic tools (e.g. model checkers) 	Methods & tools supporting the test design of effective integration (& system) tests for electronic systems, <i>especially</i> the design of <i>test cases</i>	<p>Improve quality, reduce costs for testing & re-validation</p> <p>Reduce overall development costs</p> <p>Manage complexity increase</p> <p>Ease interoperability between tools</p>	<p>2010: Method supporting integration test design</p> <p>2012: Tools supporting integration test design</p>

Simulation & Virtual prototyping

The system design of embedded systems is currently realized through a set of languages, methodologies and tools that lead to heterogeneous fragmentation of IP definitions and cyclic design flows including multiple validation, verification and evaluation phases whose final success and prototyping may fail at many layers. The time to market for new IPs is heavily

influenced by the design flow efficiency, and by lack of support for efficient modelling, analysis and implementation tools. Design costs for complex systems and the first-time success rate is influenced by the degree of modularity and IP reuse.

Support by scalable and efficient tools and integrated methodologies for multi-layered preliminary composition, verification and evaluation, may result in cost and time to market reduction, strategic manufacturing flexibility, adaptation to market demand and IP customization.

Moreover, embedded control applications use a large amount of information (going from a large number of sensors to a large number of actuators) to close their loops; this information flows through a shared communication network. For this reason the design of embedded systems is a hard task that involves notions of embedded electronic hardware design, real-time operating systems, network protocols and control systems. The embedded application must also satisfy functional constraints, real-time time constraints and implementation constraints. The designer should be able to rely on powerful simulation tools able to enlighten how all the different aspects of the embedded system influence the designed application.

At the moment there exist different theoretical tools to simulate and verify the different features of the system: functional/structural analysis tools and stochastic simulation tools for the logical constraints, numerical ODEs solution tools for the functional constraints, discrete event systems verification tools for the algorithmic constraints, hardware simulation tools to test the electronics, networked system analysis tools for the communication features. The gap to fill is the lack of unique simulation/verification tools able to take into account all the features of embedded distributed systems.

We therefore propose to address the industrial need for such ‘unique’ tools with three research priorities.

Simulators suited for strongly heterogeneous models

Because embedded software has to “live” in a real (physical) environment, the “feedback loop” (from actuators to sensors) may include continuous phenomena. Hence, the simulator must be able to handle “hybrid” (discrete and continuous) models. The whole system is designed by different specialists, from different aspects and the model of the system therefore often mixes different formalisms (for example, synchronous and asynchronous execution models). Finally, because the system may be only partly known or because some aspects are not being validated, the model to simulate may mix different abstraction levels. The simulation tools must be able to efficiently handle a model mixing at the same time several kinds of heterogeneity.

Efficient simulation

In order to be used in an industrial environment, a simulator must be able to produce an answer after an acceptable delay. However, the difference of brute power between the general purpose processors used to run the simulator and the processors used in the simulated embedded systems is narrowing (it may even be negative because of multi-core embedded systems). Moreover, the algorithms needed to simulate heterogeneous models are complex and require more CPU time. Solutions have therefore to be found to provide ‘reasonably fast’ simulators.

Answer the designer’s questions

Simulation is generally not the preferred way to validate a whole system: it is used to enlighten a particular point (for example, to check some resource allocation). The designer uses a simulator at different stages of system design, with different aspects in mind. It is important to ‘connect’ a simulated model to the designer’s model, so that the simulator is able to answer the question the designer asks at the level of abstraction and from the point of view being currently handled by the designer. If this is not done, a manual process has to be implemented to build the model to simulate and to understand the answer.

Research topic	Technical objectives	Contribution to ARTEMIS objectives	Outputs of research and timescales
Establish an automatic “forth and back” connection between modelling and simulation tools, at any level of abstraction.	Derive the object(s) to be simulated from the model of the whole system; Derive the simulation objectives and oracles from the model ; Use information from the refinement of the models to refine the simulator; Be able to translate the results of the simulation back into the semantics of the models.	Reduction of the cost of design and complexity management Cross-sector reusability; Interoperability between ARTEMIS tools.	2008-9: Methodological results and algorithms 2011: First academic software tools, to be tested in an industrial process. 2013: Software tools able to provide a realistic diagnosis.

Research topic	Technical objectives	Contribution to ARTEMIS objectives	Outputs of research and timescales
Simulate heterogeneous models	Simulators able to efficiently handle models which include, at the same time <ul style="list-style-type: none"> ■ different abstraction levels; ■ different mathematical formalisms (e.g. hybrid models); ■ different semantic formalisms (e.g. different execution models). 	Reduction of the development cycle Complexity management.	2009: Prototypes, validation of the new concepts. 2011: Contribution to standards. 2011: First realistic software tools, ready to be tested in real situations.
Faster software simulators: speed up software simulators to the extent that they can be used in real situations even for complex applications.	Exploit the complexity of the model: <ul style="list-style-type: none"> ■ to dynamically switch to different levels of abstraction, in order to use computing time only where details must be produced ■ to make use of parallel and/multi core CPUs. ■ to mix Hw/Sw simulation. 	Reduce the cost of system design; Cross-sector reusability.	2008-9: Theoretical results, academic publications. 2011: First implementations 2013: Pre-industrial implementations for totally heterogeneous models.

Transversal Tools

Certification, Safety Planning

The Current state of the practice is actually extremely diverse among embedded systems industries with respect to certification and safety planning requirements, even though safety requirements are in essence generally targeted as protecting human life from systems malfunctions.

Safety-related certification is in fact required by law in certain specific domains, such as in avionics: DO-178B for software, DO-254 for hardware, in rail transportation and “heavy duty industries” like nuclear systems through different variants of IEC 61508 and equivalent EN or ISO standards.

Security-related certification requirements are most observed thru IETF RFC 2828 “Common Criteria”, especially in IT security/smart cards industries, but we can see a tendency in some industries of a combination between safety and security requirements (like risks associated with the remote control of aircraft navigation systems from hostile ground forces, for instance).

Several European and international workgroups are dealing with these issues, and aim to improve the current state of the art, such as the DO-178C Workgroup in civilian avionics, who is currently working on integrating model-based design and formal verification techniques into the avionics standards, as well as the use of COTS or the reuse of existing products. Certification standards evolutions and the impact of model-based design flows are also being discussed in other vertical domains (for instance in automotive through the IEC 61508/ISO-FAKRA Workgroup...).

Major industrial stakes should be outlined in order to delineate key research priorities in this complex but strategic field.

First of all, certification induces an effort that is increasing proportionally even faster than typical system complexity increase. Moreover, additional recurring efforts are needed to maintain a product in service and to cope with obsolescence of component and tools.

Secondly certification induces design constraints that are potentially contradictory with business objectives and technological trends, resulting in development and modification/cost and lead time, or robustness issues in operational service.

Lastly, security assurance (protection against unauthorized attempts to confidentiality, integrity, availability of information, or proper operation of systems) comes in addition to safety assurance and certification, but may result in redundant or contradictory requirements.

Research topic	Technical objectives	Contribution to ARTEMIS objectives	Outputs of research and timescales
Tools for safety & certification	Tools offering continuity between the different model views including safety analyses, traceability impact, FMEA, coverage analyses and enabling formal proof as acceptable means of compliance. Qualification of tools themselves : improvement of cross-domain convergence, development of efficient qualified tools, tool FMEA...	Better consistency with operational need. More robust solutions	2009: automated formal proof techniques accepted as means of compliance for certification in place of verification activities 2010: consistent modelling toolset covering all activities, FMEA analysis, tool qualification objectives/methods
Introduction of new technologies and facilitation of reuse	Establish acceptable means of compliance with ultimate safety objectives and solutions enabling the use of new technologies and COTS so that cost of re-use should be less than 30% of redevelopment.	Design cost reduction Development lead time reduction Cross-sectoral reusability	2008: Fully automated certified code generation spanning logic, algorithm and graphics applications 2010: Certification basis and design solutions compatible with product line policy (internal reuse) and external COTS reuse in safety critical applications
Security assurance and certification	Define an acceptable process and means of compliance for security assurance and certification in RTES , in correlation with prior safety assurance and certification process , addressing also COTS and reuse capacity. Define supporting tools reducing the effort in the process and production of artefacts.	Manage increased complexity with reduced efforts Reduce development lead time Favour cross sectoral fertilization	2010: Certification basis for security in RTES consistent with safety assurance process and COTS reuse 2012: Specific toolset extension to support this process

Requirements and Traceability Management (including Use Cases) & Configuration Management, Methodology, & Life Cycle Management

Embedded systems currently range from single systems up to mass products. Single systems are very expensive to develop but meet most if not all of the customer's requirements. Mass products are cheaper but meet just a, sometimes rather small, fraction of the customer's requirements. In the past ten years product lines became an emerging way to tackle this problem with the idea of a customizable mass product. Similarities between products in a given domain are used to make up a core, which is common to all products. All products are based on this core and will be individually extended by components of the product line using well-defined interfaces. The products of a product line are based on a product line architecture which emerged out of product line requirements and is of vital importance for the product line. Thus, requirements engineering, being a key issue for single system development, becomes even more important for the development of a product line.

Development models like the V-Model-XT and current product line development methods, like FAST, FeatuRSEB, FODA or KobrA, use hierarchical models within their requirement engineering phase to capture variabilities. These models are stored in a database and finally handled by tools like Requisite Pro or DOORS. The methods support the analysis of common and variable parts within a product line, but are lacking complete traceability support between requirements, design model elements described with the UML, code elements and other development assets. In addition, requirements metrics and specific product line metrics for the analysis of the product line's requirements and architecture need to be pushed further, to be integrated in an industrial software development method.

Support for product line development is present only in a few tools. Requirements engineering tools, as stated above, don't support the product line idea itself, but emulate it by additional parameters for requirements. This is not sufficient for developing product line models (as in FODA) which are the de-facto standard for professional product line development. Such models use features to configure products to meet the customer's needs. First tools like pureVariants are on the market to model and configure a product line, but the integration of such tools into a seamlessly working tool chain is the subject of further research efforts, to enable full traceability from requirements to the code level and to ensure competitiveness in the market.

Today's world-wide distributed development teams require even more elaborate development methods, as well as tools, to meet the market demands for short development cycles and cheaper products. Here, methodological support for distributed embedded systems development is needed. There are several communication tools and team working environments available, but the integration into a well-defined development method is not yet present in the industry.

The research priorities to address these issues are divided into three main topics.

Round-trip requirements tracing

To enable complete traceability of development assets, starting with requirements, a research effort for an overall meta-model containing and describing all the assets is needed. Finally the meta-model has to be developed. It will be the origin for tool developments, comparable to the significance of the UML meta-model for most design tools. For ARTEMIS this meta-model will be the backbone of a new requirements engineering method, where traceability, especially among different development tools, needs to be addressed. Thus, requirements for the underlying tool-chain realizing the method will be an important research result.

Tracing of arbitrary development assets is an important issue, although of less value without the ability to perform consistency checks for the underlying model and the traces itself. Based on the meta-model rules for consistency checks need to be identified and formulated.

Product Line Requirements Engineering

Developing product lines results in a special requirements engineering method and corresponding tools. Research effort has to be put into the enhancement of the meta-model elaborated above towards embedded system development together with its development method. The requirements model has to be related to the ARTEMIS reference architecture to reflect product line specific issues, such as common / variable components. In effect the product line model (feature model) is the key issue for any configuration of applications.

In addition, the mapping of the feature model to the architecture has to be elaborated. Thus, the quality of the requirements model as well as the architecture will increase. Furthermore, metrics for extended quality measures within embedded software development need to be subject of further research.

Wide-scale distributed requirements management

Development of current embedded systems is often distributed among different locations spread all over the world. Within this context development methods are needed, to reflect this special working concept. Distributed requirements engineering from elicitation to modelling and traceability has to be integrated into a requirements engineering data model and a method.

Research topic	Technical objectives	Contribution to ARTEMIS objectives	Outputs of research and timescales
Round-trip requirements tracing	A method/process to elaborate and organize the necessary information to maintain requirements traceability under the impact of change and change propagation, based on: <ul style="list-style-type: none"> ■ a data-model ■ consistency rules to ensure 'stressless' system maintenance and update ■ traceability of arbitrary development assets from requirements down to code fragments 	Management of complexity	Phased realization of a requirements engineering method that enables a structured way of integrating traces among arbitrary development assets throughout all development phases: <ul style="list-style-type: none"> 2008: Pre-Study 2009: Documentation of Consistency Rules 2011: Traceability Meta-Model 2011: Documentation of Tool Requirements 2012: Requirements Engineering Method
Product Line Requirements Engineering	An approach that enables: <ul style="list-style-type: none"> ■ commonality and variability analysis ■ categorization and structuring related to the reference architecture, with its common core and variable components ■ requirements metrics for assessing the influence of requirements on project progress and product quality. 	Support for product line engineering	Phased realization of support for product line engineering through the usage of features as a central model for commonality/variability management. A new meta-model and new assessment metrics will lead towards a higher degree of component re-use within a product line development: <ul style="list-style-type: none"> 2008: Pre-Study 2011: Meta-Model (enhanced with product line specifics) 2011: Tool Requirements 2011: Metrics 2012: Reference architecture 2012: Family Patterns [Gamm 1995] 2013: Feature-Model Architecture Mapping
Wide-scale distributed requirements management	Consistency checking between requirements and executable specifications in the context of world-wide development teams	Management of complexity Increased productivity	2008: Pre-Study 2012: Method (enhanced with distributed development capabilities)

Tool Integration, Frameworks

Tool integration is quite often described as a ‘costly nightmare’ by most embedded systems tools users.

It must be recognized that the current state of the practice barely differs from ad hoc tool integration, through dedicated APIs, code-level integrations, a lack of tool integration standards, even though some exchange standards such as XMI do exist but are not always implemented in tools.

The lack of standards generates naturally a vicious circle with the proliferation of ad hoc integrations and the fragmentation of the market and the need to maintain costly and proprietary ‘tools workshops’ by end-users.

Nevertheless, the state of the art has been changing quite rapidly recently, with the development of model transformation technologies, a more widespread use of meta-models and profiling, and novel initiatives such as Eclipse.

Four major kinds of industrial requirements could be outlined in this area

The first one is to reduce costs through reuse, and in particular tools to support ‘product line policy’ (functional and technical, architecture + middleware...) through component-oriented designs.

The second major need is for companies to be able to adapt to ever-changing industrial organisations, via standardised repositories for multi-organisation/tools access, the organization of development data packages, and the development of standards for tools interoperability.

Thirdly, increasing quality by reducing complexity should be facilitated by better navigation on the whole development cycle (requirements, models, code, tests...) , traceability and navigation capabilities between all development artefacts, and the support of information workflows throughout the organisation and the development cycle.

Last but not least, even if more technical challenges are involved, many companies point out the need for trans-disciplinary tooling supported by common meta-model standardization among tools.

While implementing these requirements, it should be noted as well that the existence of standards has a value if and only if these standards are actually implemented efficiently in tools.

Research topic	Technical objectives	Contribution to ARTEMIS objectives	Outputs of research and timescales
Meta-model standardization	Common meta-model standardization among tools, based on user inputs ensuring standardised repositories for multi-organisation/tools access: <ul style="list-style-type: none"> ■ tools to support Product Line Policy (functional and technical, architecture + middleware...) ■ component-oriented designs ■ common meta-model standardization among tools 	Interoperability between Tools Achieve cross-sectoral reusability Horizontal contribution to all Design Methods & Tools research priorities	2010: meta-model standards for product-lines
Development cycle navigation	Ease navigation on the whole development cycle by: <ul style="list-style-type: none"> ■ ensuring traceability/navigation between all development artefacts ■ supporting workflow of information throughout organisation & development cycle 	Mastering development complexity	2011: tools conform with meta-model standards, enabling navigation.

End-to-end Process Optimization

Model-Based Design Flow Optimisation

Real-Time Embedded Systems (RTE) have to face specific functional and non functional constraints that burden the product design, such as

- Performance issues,
- Real-time constraints,
- resource consumption (memory, CPU, communications, gates & cells...),

- Power consumption,
- Performance To Cost Ratio;
- Certification,
- Dependability, Safety, Security;
- Hardware/software co design & interleaving,
- Environmental constraints,
- Human Factors, MMI...

The art of product design is the ability to fully identify and then reconcile these ‘constraint domains’ or ‘aspects’ through a relevant architecture and a rational approach to making trade-offs so that an optimised solution will emerge.

The current state-of-the-art in the domain may be sketched as follows (although there will be variations

depending on each specific RTE domain, such as palm devices, safety-critical equipment, home appliance...):

- The analysis and design process has until now been ad-hoc in each organisation & domain.
- It often suffers from incompleteness in encompassing all the aspects listed above: many of them are poorly addressed in a formalised way (e.g. power management, resources consumption)
- Even when most aspects are addressed; they are seldom formally woven together in order to help a global, near-optimal solution, emerge
- Most academic work deals with modelling & representation of some of these aspects, but much less work addresses the way *to build these models*, and *to make use of them* for solution building, i.e. the means to aid the “solution emergence intellectual process”
- In particular, the specific issue “how to reconcile (or “weave”) these aspects” is often poorly addressed by current research and industrial work and practise, and insufficiently supported by tools.
- Even when supported by tools, the tools do not integrate easily together at the semantic level, and do not support well industrial work-flows.

Furthermore, until now no clear process, methods & rules have emerged, that could drive the definition of better-adapted tools.

To support the analysis & design process, model-driven engineering is currently gaining prominence, especially in communication & information technologies. Yet model-driven engineering is today mostly devoted to and tailored for application domains without significant constraints (such as Information Systems).

Most industrial stakeholders in RTE development, that is much more highly constrained, therefore develop their own pragmatic approaches, but available languages &

modelling techniques do not satisfy their needs.

Yet Model-driven approaches are considered as promising in the field of RTE systems, but do not fully support today RTE specificities as listed above:

- Meta-Models (i.e. syntax, grammar rules & semantics) exist for functional and basic real-time aspects, but lack to express other RTE aspects and describe design artefacts & candidate solution (considering & modelling performance, human factors, safety, reliability, resource consumption, environmental... issues),
- Common or linked models addressing all these aspects in a common and unified manner (consistent modelling) do not exist
- There are not yet defined rules for merging and transformation of models in order to be able to:
 - reconcile modelled aspects
 - investigate mutual influence of each of them on the others
 - relate and interweave architectural work and modelling to solve RTE constraints
- Application of model-driven engineering in hardware is still at an academic level, and little deployed in the industry, due to the lack of global approach lack of integration with hardware design & test tools, and little perceived return on investment in the short-term.

Yet these are considered as keys to face the increase in complexity in future embedded systems, challenges in design and in cost-effectiveness of products.

The following Research Priorities address these issues. They target RTE systems, in a consistent, model-driven and architecture-centric approach, including Reuse and Product Line Management.

These transverse research themes are to be considered as inputs for tool-oriented themes.

Research topic	Technical objectives	Contribution to ARTEMIS objectives	Outputs of research and timescales
[Meta]-Modelling	Define Methods & Meta-Models to describe & address most RTE non functional properties & issues, including <ul style="list-style-type: none"> ■ Performance, Real-Time constraints, ■ Resource Consumption, ■ Perf. To Cost Ratio; ■ Certification, ■ Dependability, Safety, Security; ■ Hardware/software co design & interleaving, ■ Environmental constraints, ■ Human Factors, MMI... ■ Checkable, executable (simulation Means) ■ Favouring Separation of Concerns between non functional properties & Issues ■ Keeping links with functional properties & issues ■ Emphasizing cross-Dependency & mutual Impact between them, via layered, composite Views, “Influence Maps” from one on another, and mutual traceability ■ Dealing with Product Line Management, Model Evolution, Reuse 	Managing complexity increase Reducing effort Reducing integration risks	Methods supporting weaving of all RTE aspects: <ul style="list-style-type: none"> ■ Performance, Real-Time constraints, ■ Resource Consumption, ■ Perf. To Cost Ratio; ■ Certification, ■ Dependability, Safety, Security; ■ Hardware/software co design & interleaving, ■ Environmental constraints, ■ Human Factors, MMI... and emergence of optimal compromise between them: 2009-2010 Meta-Models specifically defined to describe former RTE Aspects above, and interweaving them in a consistent Model: 2010-2012 Models & Methods adapted to each RTE Segment or Domain (e.g. palm devices, critical/certifiable equipment, home appliance...): 2010-2012 Tools supporting these Methods & Models : see Tools above

Research topic	Technical objectives	Contribution to ARTEMIS objectives	Outputs of research and timescales
RTE Architecting Techniques & Patterns	<p>Define architectural Techniques & Patterns to architect & design RTE Systems Specificity:</p> <ul style="list-style-type: none"> ■ Implementing / supporting RTE-dedicated meta-Models above ■ Easing Separation of Concerns in Design (e.g. separate functional Vs Real Time architectures, temporal/spatial partitioning...) ■ Giving Means to reconcile these Concerns (e.g. “synchronising” purely functional flows with underlying Real Time Organisation; self-organised, self configuring, “rule-based” Architectural Approach) 	<p>Reducing system design cost</p> <p>Managing complexity increase / reducing effort</p>	<p>General Architecting Techniques supporting former RTE Models (e.g. resource & power management, dependability/fault tolerance, dynamic temporal & spatial partitioning...) : 2010-2012</p> <p>Architecting Techniques & Patterns dedicated to each RTE Segment or Domain (e.g. palm devices, critical/certifiable equipment, home appliance...): 2011-2013</p> <p>Middleware & Tools supporting these : see Tools above & SCM</p>
Engineering Continuum & Impact Analysis	<p>Define assisted or automated transition & ensure consistency between need, design/product, tests, simulation models.</p> <p>Develop a bi-directional impact analysis capacity between need models & design models</p> <ul style="list-style-type: none"> ■ to support trade-offs between requirements, usability, and product feasibility ■ to check for feasibility of a functional evolution ■ to ease requirement-driven development & integration <p>Develop capacity of impact analysis between functional & RTE non-functional views of the product for solution tuning & verification.</p> <p>Ease unified seamless integration & navigation on the whole development cycle (requirements, models, code, tests, ...) & control workflow</p>	<p>Managing complexity increase</p> <p>Reducing effort</p>	<p>Methods & Models Links for Consistency (e.g. autonomy requirements, use cases scenarios, power consumption models & simulation, autonomy tests benches & results) : 2009-2011</p> <p>Impact Analysis Aids between need models & design models for each aspect, and between different aspects models (e.g. performance Vs resource consumption,): 2010-2012</p> <p>Tools supporting seamless integration & navigation, along with Impact Analysis : see Tools above</p>
Model transformation to Reuse	<p>Define methods and tools to extract and capitalize designed components (requirements, models, documentation, products, ...)</p> <ul style="list-style-type: none"> ■ dealing with characterisation of non-functional properties (performance, certification, resource consumption...) ■ assisting in checking global functional/non functional compliance in reuse environment <p>Define meta-models to describe reusable component</p> <ul style="list-style-type: none"> ■ measures to collect (execution performance, development costs, ...) ■ component use preconditions (use cases, environment constraints, protocols) ■ component needs (platform, other components interfaces) <p>Develop novel reverse engineering techniques for legacy integration (from code)</p> <p>Relate reusable RTE components model to relevant architectural support (e.g. separating functional & real-time architecture, abstraction levels and platform independence, platform adaptability...)</p>	<p>Reducing system design cost</p>	<p>Methods to capitalise RTE Components supporting non-functional properties capitalisation & check (performances, use cases scenarios, expected environment, certification constraints ...): 2009-2011</p> <p>Meta-models for Reusable RTE Components supporting capitalisation of component features measurement, use conditions, constraints on environment...: 2009-2011</p> <p>Reverse Engineering Techniques predicting non functional behaviour of reused code (e.g. resources consumption, real-time behaviour or determinism): 2009-2010</p> <p>Tools supporting former Models & Methods : see Tools above</p>

Research topic	Technical objectives	Contribution to ARTEMIS objectives	Outputs of research and timescales
Heterogeneous & multi-domain modelling	<p>Establish the means to efficiently handle models which include, at the same time:</p> <ul style="list-style-type: none"> ■ different abstraction & detail levels; ■ different mathematical formalisms (e.g. discrete & continuous models); ■ different semantic formalisms (e.g. different execution models). <p>“Handle” here means</p> <ul style="list-style-type: none"> ■ modeling tools : use and check, at any level of details (including source code) ■ mathematical tools (proofs) ■ test generation, oracles, simulation 	<p>Reduction of the development cycle;</p> <p>Complexity management.</p>	<p>First multi-paradigm Models mixing different Formalisms: 2009-2011</p> <p>prototypes, validation of the new concepts 2010-2011</p> <p>First realistic tools, contribution to standards : 2013</p>
Design process improvement & support	<p>Solution emergence from functional need: for each specific Domain / problematics, create an approach and supporting tools to aid reconciliation or compromise between former RTE Aspects & constraints :</p> <ul style="list-style-type: none"> ■ Design rationalisation, auto-organisation, factorising... ■ Default patterns, rules, know-how to be capitalised ■ Automated / assisted generation of solutions & pruning / culling. <p>Improving design efficiency</p> <ul style="list-style-type: none"> ■ Support distributed design ■ Support composability in design ■ Assemble sub systems while guaranteeing overall system functionality (linked to research priority B on validation and verification) ■ Predictability of emerging behavior ■ Improve predictability, without compromising non-functional requirements as power consumption, area, cost, ... ■ Meet in the middle design 	<p>Managing Complexity Increase and Reducing Design Cost & Cycle Time</p>	<p>Decision Aids & tools to support reconciliation & solution emergence between RTE specific Aspects :</p> <ul style="list-style-type: none"> ■ Performance, Real-Time constraints, ■ Resource Consumption, ■ Perf. To Cost Ratio; ■ Certification, ■ Dependability, Safety, Security; ■ Hardware/software co design & interleaving, ■ Environmental constraints, ■ Human Factors, MMI... 2010 - 2011 <p>Design Aids for</p> <ul style="list-style-type: none"> ■ distributed design, ■ composability, ■ multi-core systems, ■ predictability, ■ hardware/software optimisation (meet in the middle) <p>Approach early version end-2009;</p> <p>First tools end-2012</p>

Model-Based Validation & Verification Flow Optimisation

Validation & Verification is a huge work in any system design, but takes greater importance (and relative cost) in real-time embedded systems, where many different issues have to be checked (incl. power consumption, influence on / from environment...) and where very strong reliability constraints must be respected (safety aspects, human factors, ...).

Current practises mostly favour design / test / integration / verification / validation approach, thus facing a lack of visibility on potential risk & design troubles, and large cost over-runs due to late identification and correction of defects.

The following research priorities intend to address these issues through early, model-driven, check and validation. The idea is to take benefit from models to check and validate early the design against the need, and to be able to test the resulting product as soon as possible (in early modelling stages).

Furthermore, the large amount of constraints to be taken into account in RTE systems leads to make an extensive use of simulation and experimentation, including real assets. To achieve this efficiently, a continuous, seamless mixed use of models and real assets is promoted.

These transverse research themes are to be considered as inputs for tool-oriented themes.

Research topic	Technical objectives	Contribution to ARTEMIS objectives	Outputs of research and timescales
<p>Early Design Validation Support</p>	<p>Develop executable design [simulation, prototyping] models & underlying representations (formal languages, graphical representation...)</p> <ul style="list-style-type: none"> ■ as a complement to formal and rule-based check ■ for behaviour, safety & performance simulation ■ emphasising on metrics/measurement collection ■ strictly & automatically deduced from design models, through model transformation ■ related to rule & model checking engines <p>Enable relation of simulation results to initial models and update of those models</p>	<p>Reducing system design cost</p> <p>Reducing effort and time for re-validation / re-certification</p>	<p>Techniques for executable Models derived from design models, including RTE specific aspects,</p> <ul style="list-style-type: none"> ■ Performance, Real-Time constraints, ■ Resource Consumption, ■ Perf. To Cost Ratio; ■ Certification, ■ Dependability, Safety, Security; ■ Hardware/software co design & interleaving, ■ Environmental constraints, ■ Human Factors, MMI... <p>seamlessly simulating performance issues from hardware to upper software layers, resource consumption behaviour: 2008-2009</p> <p>Formal Languages & meta-models Complements to support the former executable models, for Metrics Collection & Exploitation, Execution & Integration with Design Models: 2008-2010</p> <p>Formal Models Execution Tools : see Tools above</p>
<p>Early product validation support</p>	<p>Define Methods & techniques to relate & compare need, definition & design models:</p> <ul style="list-style-type: none"> ■ compliance rules, proof or test objectives ■ models transformation & mapping to check for compliance ■ simulation including check of operational scenarios fulfilment ■ not only on functional, but also non functional, RTE specific aspects. <p>Develop automatic test patterns generation from requirements, and automatic execution & results interpretation (e.g. based on need/design models analysis & transformation)</p>	<p>Reduce system design cost</p> <p>Reduce development cycle times</p> <p>Manage complexity increase / reduce effort</p> <p>Reduce effort and time for re-validation / re-certification</p>	<p>Methods & Techniques for Solution / Need Assessment : models mapping, formal compliance check, automated impact analysis, including RTE aspects: 2008-2010</p> <p>Modelling Techniques and Languages to support the former executable models: 2009-2012</p> <p>Automatic Test Generation & Interpretation Techniques including sizing, performance, consumption, environmental... testing : 2010-2012</p> <p>Tools : see Tools above</p>
<p>Formal proof techniques & support</p>	<p>Define formal languages, methods & techniques to demonstrate correctness of a design model wrt needs across the full design & verification flow:</p> <ul style="list-style-type: none"> ■ key properties to be respected ■ formal, high abstraction-level languages to check ■ relevant formalisms for each main RTE domain or aspect ■ transformation and abstraction rules to produce formal models from need or design models ■ plus a posteriori check of non formal approach <p>Special focus on RTE constraints : safety, security, performance, resource consumption, reliability...</p>	<p>Reduce system design cost</p> <p>Manage complexity increase / reduce effort</p> <p>Reduce effort and time for re-validation / re-certification</p>	<p>RTE specific Languages dealing with formal proof of performance issues, complex & multi-core real-time constraints, safety, certification issues, resource consumption, ...: 2008-2010</p> <p>Transformation Rules to produce former formal Models from need or design models: 2010-2012</p> <p>Languages adapted to some key specific RTE domains : 2008-2010</p> <p>Tools : see Tools above</p>

Research topic	Technical objectives	Contribution to ARTEMIS objectives	Outputs of research and timescales
Mixed real and simulated prototyping	<p>Define methods, formalisms & rules to easily use (transform) capitalized models and components to develop simulation & prototypes</p> <p>Allow seamless transition between fully simulated, partially simulated and real-system elements</p> <p>Define interfaces with real hardware systems</p> <ul style="list-style-type: none"> ■ Use these prototypes in simulation frameworks ■ to explore operational concepts ■ to capture and confirm requirements ■ to identify environment constraints, system usage conditions <p>Develop capacity to reuse these models for quick prototyping of future systems</p>	<p>Achieve a mutual agreement on system requirements</p> <p>Reduce evolution demands and qualification risks</p>	<p>Techniques to introduce real world devices in Models Simulation : 2010-2012</p> <p>Standards to ease transition & integration between Models & real devices : 2010-2012</p> <p>Tools & Simulation Frameworks : see Tools above</p>
Validation strategy optimization	<p>Develop methods and tools to model, decompose, allocate validation objectives in the context where validation steps are done early at different abstraction levels, with different models according to different concerns and lately on products</p> <ul style="list-style-type: none"> ■ identify from requirements the validation objectives ■ help to decompose these objectives across the concerns, levels, components, ... ■ help to qualify the validation portion made at each step ■ help to qualify the hypotheses or abstractions used and the ways to validate them ■ help to define the validation objectives early achieved on models and those left to reach on product 	<p>Manage complexity increase</p> <p>Reduce validation effort</p>	<p>Models for Validation Strategy & Objectives (including RTE aspects) : 2008-2009</p> <p>Methods to identify & allocate Validation Objectives, link them with Need & Design models : 2008-2010</p> <p>Model-driven Techniques to manage Validation Objectives (e.g. model mapping to check objectives, impact analysis, transformation of need models to extract validation objectives,): 2009-2011</p> <p>Tools : see Tools above</p>
Automated connection between modeling and Validation & Verification tools	<p>Forth and back connection between models and V&V tools, at any level of abstraction:</p> <ul style="list-style-type: none"> ■ develop consistent modelling and v&v tools ■ use information from the models to refine the V&V accuracy. ■ be able to translate the results back into the semantics of the models. ■ derive objectives and oracles from models (functional and non-functional properties), ■ develop model coverage tools ■ automated feedback of V&V results within design activities (test coverage, counter-example scenarios, resource consumption...) 	<p>Manage complexity increase</p> <p>Reduce validation effort</p>	<p>Methods & Techniques to link Design & V&V Models : 2008-2010</p> <p>Transformation Techniques & Rules from design to V&V (e.g.V&V test patterns, expected tests results, automatic comparison between experiment and expected results, mapping in design models for failure/defects localization...): 2008-2010</p> <p>see Tools above</p>

Global HW+SW Solution Verification & Optimisation

One of the specific challenges to be addressed by real-time embedded systems is the ability to find best balance, when allocating requirements, between hardware and software items.

Although some approaches and even tools may exist today, to address this “co design” issue, there is no real support to managing relevant trade-offs, finding the best compromise or helping the solution emerge. Most tools intend to give same description means for software & hardware, but little really help in designing.

The following research priorities intend to address these issues targeting RTE systems, complementing the tools-oriented approach with some design aids, more dedicated to RTE specificities : hardware/software intricacy, resource consumption, timing & performance issues...

Some key issues to be addressed are:

- Criteria and aids to allocate requirements and functional processing either to hardware or software items
- The means to support early experimentation and optimisation of software/hardware allocation, at design time
- Automation and abstraction techniques (e.g. models) to hide hardware and software complexity at designer level, while generating relevant & efficient software code or hardware compilable description
- Design and architectural patterns enabling support of RTE aspects management in hardware/software allocation: resource consumption, processing & hardware patterns (co-processors, multi-core, [massively] parallel processing...)
- Integration and cross-links of all these aids in coherent, model-driven, high-level abstraction environments, related to former need & design models & aids.

These transverse research themes are to be considered as inputs for Tool-oriented themes.

Research topic	Technical objectives	Contribution to ARTEMIS objectives	Outputs of research and timescales
Decision aids for solution emergence	Define aids & techniques to help in defining the solution: <ul style="list-style-type: none"> ■ identification and management of design rules, applicable to Model Transformation, including ■ to help in allocating requirements & design assets to hardware & software ■ to hide HW/SW implementation specificities ■ to transform according to HW/SW specific constraints ■ "what if" facilities ■ multi-criteria optimisation for trade-off, resolution of conflicts & reconciliation between performance, dimensioning, resource consumption, dependability ■ "reasoning engines" to apply former rules to models ■ algorithms to drive self-organisation, factorising, rearrangement, simplification... of models 	Reduce system design cost Reduce development cycle times Manage complexity increase / reduce effort	Rules-based hardware/software Allocation & Design Techniques (e.g. to choose what to implement by software or by hardware, how to interface both, best performance trade-offs...): 2007-2009 Self-organisation Algorithms & Techniques for efficient mapping of models on HW/SW : 2009-2011 Tools : see Tools above
Integration of behavioural synthesis with HW design tools	Start from SW description, e.g. in C (somehow model based design methods might apply, eventually) Offer alternative paths in the direction of Co-Processor, Accelerator, or Peripheral (Requires appropriate Meta Models) Automate Synthesis of algorithms, synchronisation, and wrapper generation Extend Synthesis Capabilities in the control domain (rule based synthesis)	Reduce system design costs	Design Aids & Model Transformation Rules to target hardware architectural patterns (e.g. coprocessors, accelerators, programmable devices, multi-cores...) in a transparent manner for the designer 2008... 2010

Research topic	Technical objectives	Contribution to ARTEMIS objectives	Outputs of research and timescales
<p>Timing & power & resource consumption verification & optimization</p>	<p>Design time analysis and mapping tools for specific needs or targets (e.g. multi processor target)</p> <ul style="list-style-type: none"> ■ high-level cost and performance estimation (needs predictability); ■ real-time constraints; ■ power constraints; ■ scratchpad memory management (important source of cost and unpredictability); ■ software controlled coherency (implemented by design-time tools); <p>Architecture components giving control to the application and thus enabling the use of design-time methodologies and tools</p> <ul style="list-style-type: none"> ■ component properties closely match the application characteristics ■ components have a predictable behaviour, or at least allowing a reasoning at design-time <p>Platform run-time management</p> <ul style="list-style-type: none"> ■ resource management that exploits design-time information; ■ run-time verification of the design-time requirements (resources, timings...) => default management. ■ flexibility with respect to platform architecture, with possibly some features loss. 		<p>Resource Management Models & Rules, aiding in prediction / recommendation / design choices on resource management : 2008-2010</p> <p>Architecture Components dealing with Resource Management on application/operational level criteria, with predictable resource consumption behaviour : 2009-2011</p> <p>Compliant Run-time Resource Management Techniques to support, check & secure former application-level resource management: 2009-2011</p> <p>Tools : see Tools above</p>
<p>Efficient implementation</p>	<p>Software Parallelization</p> <ul style="list-style-type: none"> ■ Method and tool support for making existing software parallel and to develop new parallel software to support multi-core systems more efficiently ■ HW Support for SW parallelization ■ SW parallelization must be compliant with the model in modelling research priority above. <p>Strategy to efficiently translate a set of applications into architectures.</p> <ul style="list-style-type: none"> ■ aggressive optimisation techniques to optimise the application code and to exploit the possibilities of the architecture ■ techniques to derive the optimal architecture for a set of applications. ■ Tool support to guide the designer through design choices (linked to the model, the estimation and simulation, ...) 	<p>Basic technology for SW Re-use, subsystem reuse, Software productivity, embedded SW design</p>	<p>Methods & Tools, optimisation techniques and model transformation rules for automatic or assisted parallelisation of software on multi-processor: 2010-2013</p> <p>Hardware and firmware/middleware support for transparent multi-processing wrt software application 2009-2010</p>

Glossary

(Abstract) Interface State: The (abstract) state of a system as viewed from a particular interface. It is a notional attribute of the interface that is sufficient to explain future behaviour of the system across this interface.

Abstract Interpretation: A special case of static analysis (q.v.) ; by abstract interpretation, it is possible to discover some (possibly approximate, but always true) properties of a program by analyzing its source or binary code.

(Abstract) State of a System: At a given instant, a notional attribute of the system that is sufficient to determine its potential behaviour.

AFAV (l'Association Française pour l'Analyse de la

Valeur): a French association concerned with the advancement and promotion of 'value methods'.

API (Application Programming Interface): the interface that a computer system, library or application provides in order to allow requests for service to be made of it by other computer programs, and/or to allow data to be exchanged between them.

Architecture: A technical system architecture (or architecture for short) is a framework for the construction of a system for a chosen application domain that provides generic architectural services and imposes an architectural style for constraining an implementation in such a way that the ensuing system is understandable, maintainable, extensible, and can be built cost-effectively. The architectural style consists of rules and guidelines for the partitioning of a system into subsystems and for the design

of the interactions among subsystems. The subsystems must comply with the architectural style to avoid a property mismatch at the interfaces between subsystems.

ARTEMIS ((Advanced Research & Technology for Embedded Intelligence and Systems): an ETP (see below) aiming to establish and implement a coherent and integrated European research and development strategy for Embedded Systems.

ASIC (Application-Specific Integrated Circuit): an integrated circuit (IC) customised for a particular use, rather than intended for general-purpose use.

Behaviour: A sequence of (perhaps time-stamped) send and receive operations of a system.

BIOS: Acronym for 'basic input/output system', the built-in software that determines what a computer can do without accessing programs from a disk. Operation of the BIOS is usually necessary to start the operating system.

Connection: A link between the interfaces of two or more interacting systems.

Dependability: The dependability of a system is the ability to deliver a service that can justifiably be trusted, where the service is the intended behaviour of the system.

Device: usually an accessory to a home or office PC - a printer, DVD/CDROM drive, keyboard - but could be a radar head in an aircraft or a suspension control unit in a car.

Device driver: A program that controls a computer device. Every device (e.g. a printer, disk drive, keyboard) must have a driver program. A driver acts like a translator between the device and programs that use the device.

DO-178B: a standard - 'Software Considerations in Airborne Systems and Equipment Certification' - for software development used primarily in the certification of avionics software.

EDA (Electronic Design Automation): the use of tools for designing and producing electronic systems, from printed circuit boards to integrated circuits.

EMC (Electro-magnetic compatibility): the ability of different electronic systems to co-exist without detriment to performance of any of them as a consequence of their electro-magnetic radiation.

ENIAC (European Nanoelectronics Initiative Advisory Council): an ETP (see below) in nano-electronics.

ESL (Electronic System Level): EDA support for complete system design at a pre-RTL abstraction level

ETP (European Technology Platform): is a mechanism to bring together all the stakeholders in a particular domain, from industry, academe, and the public sector, to develop a long-term vision for that domain; to create a strategy to realise that vision; and to steer that realisation of the strategy. The elaboration and implementation of a Strategic Research Agenda is a crucial part of the strategy.

Failure: A failure of a system occurs at an interface of the system at the instant when its behaviour starts to deviate from the intended behaviour at that interface.

Fault: A fault is the cause of an error.

Feature: Requirements of a system are typically modeled by features on a higher abstraction level, emphasizing the

variability of a SW/HW System. A feature is an aspect concept of value or a customer and is usually expressed by a single word. Features can be of various types: functional, interface, parameter or structural.

Flow Control: Establishes synchronization between the sender and the receiver(s), such that the receiver(s) can follow the speed of the sender(s).

FODA (Feature Oriented Domain Analysis): a method to elaborate a feature model holding the commonality/variability information of a system family with all dependencies between the features. The resulting feature model is the central representation model for system families.

Formal specification: Can be used to derive an unambiguous specification of a subsystem, an interface, or parts of those.

Formal verification: Yields an objective correctness proof for properties. Usually a formal verification process requires a preceding formal specification process.

FPGA (Field-Programmable Gate Arrays): an integrated circuit that contains programmable logic blocks and programmable interconnects so that the same chip can be used in many different applications. (Cf. ASIC: an FPGA may be a more cost-effective way to realise the same application-specific purpose.)

IDE (Integrated Development Environment): a suite of interoperable development tools typically encompassing design, build-generation (software or hardware), testing, diagnosis and configuration management.

IEC 61508: an international standard (under the auspices of the IEC - The International Electro-technical Commission) concerned with the 'Functional safety of electrical/electronic/ programmable electronic safety-related systems'. Originating in the process industries, it has provided a framework for standardisation in other sectors, such as rail, automotive, medical and nuclear.

Interaction: A sequence of message exchanges between connected interfaces.

Interface Model: the model of the concepts a user has in mind when he/she relates the meaning of the chunks of information in a message (which are the results of the syntactic specification) to his/her conceptual world.

Interface: A point of interaction between a system and its environment.

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.

IP: Intellectual Property

IP Block: typically a sub-system, but could be a component of a system, in which there is some element of IP where the IP might be either 'protected' (through patents, etc.) or freely available (as open source silicon design, for instance).

ITEA (Information Technology for European Advancement): one of the main EUREKA cluster programmes, focussing on embedded software-intensive systems and services. After a successful first round, ITEA has been followed by ITEA2.

ITSEC (Information Technology Security

Evaluation Criteria): a structured set of criteria for evaluating computer security within products and systems.
Kernel (or System Kernel): The most central module of an operating system. It is the part of the operating system that loads first, and usually remains in main memory. Typically, the kernel is responsible for memory management, process and task management, and input/output management.

MBD: Model-Based Development

MDA (Model-Driven Architecture): an approach to specification and design that attempts to separate system architecture from design and from realisation technologies, so that architecture and design can evolve more or less independently.

MEDEA+: another major Eureka cluster concerned with 'system innovation on silicon'. It builds on the earlier initiative MEDEA (Microelectronics Development for European Applications)

Message: A data structure that is formed for the purpose of communication among computer systems.

MMI: Man Machine Interface

MPSOC (MultiProcessor System on Chip): a chip comprising or containing multiple processors.

Product Line: the business term for a suite of products in the same domain, usually embodying some kind of evolution.

Protocol: A set of rules that specifies the interactions between two or more systems across connected interfaces.

RTD: Research and Technology Development (sometimes Research, Technology development and Demonstration). This has been the main scope of the Framework Programmes, thus far.

RTE: shorthand for 'real-time embedded'

RTOS (Real-Time Operating System): is a class of operating system intended for real-time applications, typically on embedded computing systems.

Service Specification: The specification of the set of intended behaviours of a system.

SoC (System on Chip): Complete functional system implemented in a single chip

Static Analysis: An approach to discover the properties of a program by analyzing its source code, binary code or formal model, without executing the program.

System: An entity that is capable of interacting with its environment and may be sensitive to the progression of time.

System Family: this is the technical term for a set of products out of the same domain (see Product Line). Here the commonalities and variabilities of the products in terms of the underlying architecture are in focus. All the products share an architecture with a common core and specific variabilities based on the features of a product.

Testing: The process of executing system functions with the intention to find errors (e.g. software or hardware bugs).

UML (Unified Modeling Language): a modeling and specification language with a standardized graphical notation that is used in software engineering to create an

abstract model of a system.

V&V: Verification & Validation (see below)

Validation: The process of checking if something satisfies a certain criterion. Validation aims at answering the question: "Are we building the right system?"

Variability: within a system family, variabilities refer to parts of a system architecture that are selectable based on the features desired by a customer.

Verification: The process of proving the correctness of a system with respect to a certain specification. Verification aims at answering the question: "Are we building the system right?"

VHDL (Very High-level Definition Language): a hardware description language, commonly used for FPGAs and ASICs.

WCET (Worst Case Execution Time): the maximum time that a task may take to execute (on a specific hardware platform).

Contributors

Domonkos Asztalos	Ericsson
Eric Bantegnie (chair of the Expert Group)	Esterel Technology
Luca Benini	University of Bologna
Ivo Bettens	Thales/Artemis Office
Ed Brinksma	ESI
Christian Cantaloube	Thales
Francky Catthoor	IMEC
Daniel Chevalier	Thales
Petronilla Cifarelli	Pirelli
Jerker Delsing	Lulea Univ. of Technology
Alun Foster	STMicroelectronics
Olivier Gerard	Philips
Laila Gide	Thales
Jean Jourdan	Thales Aerospace
Jens Herrmann	Daimler Chrysler
Anders Johansson	Lulea University
Klaus Kronlef	Nokia
Marc de Krock	IMEC
Gianpaolo Macario	COMAU
Bob Malcolm	ideo
David Morera	European Software Institute
Ton Peerdeman	Thales Nederland
Ian Phillips	ARM
Sylvain Prudhome	Airbus
Alain Rioual	Thales
Lothar Schrader	Infineon
Hans Schurer	Thales Nederland
Jan Stransky	CEA-LIST
Detlef Streitferdt	ABB
Silvia Vecchi	University of Bologna
Jean-Luc Voirin	Thales Aerospace
Johann Vounckx (co-chair of the Expert Group)	IMEC