Challenges and Work Directions for Europe

Bruno Bouyssounouse, Joseph Sifakis

http://www.artist-embedded.org/
ARTIST IST-2001-34820

Session 7A: Major Projects

Embedded Systems

Challenges and Work Directions for Europe

Bruno Bouyssounouse[1], Joseph Sifakis[2]
VERIMAG Laboratory http://www-verimag.imag.fr/

The Artist FP5 Accompanying Measure in Advanced Real-Time systems workplan includes roadmapping work for strategic directions: Hard Real-Time Development Environments, Component-based Design and Integration Platforms, RTOS and Middleware. These roadmaps are available in draft form:

http://www.artist-embedded.org/Roadmaps/

The roadmaps will be updated and extended to include a new section on Execution Platforms. They will be published in a single volume in mid-2004, via Springer Verlag's LNCS series.

[1] ARTIST Technical Coordinator

[2] ARTIST Scientific Coordinator

1. Introduction

1.1 Embedded Systems

Embedded Systems are components integrating software and hardware, that are jointly and specifically designed to provide a given set of functionalities. These components may be used in a huge variety of applications, including transport (avionics, space, automotive, trains), electrical and electronic appliances (cameras, toys, television, washers, dryers, audio systems, and cellular phones), process control (energy production and distribution, factory automation), telecommunications (satellites, mobile phones and telecom networks), security (e-commerce, smart cards), etc. We expect that within a short timeframe, embedded systems will be a part of virtually all equipment designed or manufactured in Europe, the USA, and Asia.

Their extensive use and integration in everyday products marks a significant evolution in information science and technology. A main trend is the proliferation of embedded systems, which should work in seamless interaction while respecting real-world constraints.

Embedded systems have a number of specific characteristics, which play a role in structuring the technical domain, and in determining the relevant areas for research and industrial development:

- Criticality – Embedded systems are often critical. The degree of criticality depends on the consequences of deviation from a nominal behavior that can impact safety, security, mission completion, business. As an example, safety criticality means that failures can damage human lives, or cause major catastrophes. Safety critical systems include flight control systems, railway signaling systems, and industrial process management systems. As a second example, business criticality is where major financial stakes are associated with proper operation. Business critical systems include services deployed on an embedded infrastructure such as cell phones, power distribution, traffic management, distributed entertainment, etc.
- Reactivity – Embedded systems are deployed in the physical environment, and as such they have a continuous interaction with it. This is a central characteristic of embedded systems. Reactivity implies that embedded systems are subject to real-time constraints relating their execution speed to that of their environment.
- Autonomy – Embedded systems need to be autonomous, that is, to fulfill their functions without human intervention for extended periods of time. Autonomy is needed, especially where humans' reactions may be too slow or insufficiently predictable.

1.2 Economic Stakes

Embedded systems are of strategic importance in modern economies. They are used in mass-market products and services, where value is created by supplying either functionality or quality. Functionality is defined as the service rendered to the user. Quality for a given functionality characterizes extra-functional properties of the product or service, such as performance, or dependability. For instance, a cellular phone offers functionality for mobile communication, while quality is characterized by audio fidelity, battery life, etc.

Embedded technologies confer advantages to system and service developers, in generating added value and enhancing competitivity. The relative weight of software in the value of embedded systems is constantly increasing. Increased use of software allows new, complementary services, and competitive advantages through differentiation.

Embedded systems are the fastest growing Information Technologies sector.

Europe currently has leading positions in sectors where embedded technologies are central to growth. These sectors currently include avionics, automotive, space, consumer electronics, smart cards, telecom devices, energy distribution, and railway transport. It is anticipated that they will also include distributed services such as e-Health and e-Banking.

Europe has a leading position in civil avionics where fly-by-wire technology provides an overwhelming competitive advantage in the cost of operating aircraft. Europe is also well-positioned in the space sector, specifically for launch vehicles and satellites. In the automotive sector, European manufacturers and their suppliers enjoy a leading technological advantage for engine control, and emerging technologies such as brake by wire and drive by wire. Railway signaling in Europe relies on embedded systems, and allows faster, safer, and heavier traffic. Embedded technologies will be extensively used to make energy distribution more flexible, especially in view of the coming market liberalization. Embedded technologies are strategic for the European telecommunication sector, which is still well-positioned - in spite of the recent difficulties in deploying UMTS technology. Finally, Europe is also well-positioned for e-Services (e-Banking, e-Health, e-Training), based on the leading edge in smart cards and other related technologies.

1.3 Trends in Embedded Systems Engineering

In the last ten years, certain spectacular changes have been observed in systems engineering. The number of embedded system applications was relatively low, mainly used for control applications in aircraft, trains, and production plants. These applications being intensively safety-critical, development costs and times were extremely high. The development costs were nonetheless a small portion of the overall system cost.

Coming generations of mass-market embedded systems products will be characterized by the following factors:

- A great variety of component types. Components will be tailored to fit specific technical and economic requirements, with differing associated costs and levels of service. For instance, specific components must be developed for mobile phones, for each different automobile sector, for home appliances, etc. This is markedly different from the situation ten years, ago, where general-purpose processors were prevalent.
- Integration of heterogeneous components in a real-world environment. A main requirement for embedded systems is their integration within the larger embedded environment. The objective is to obtain smooth and harmonious cooperation with other components – embedded or not - to provide global services.

- Market Constraints. Market constraints require careful positioning between cost/quality for a given functionality. For example, mobile communication is a functionality that can be provided at different optima of cost and quality, depending on the market segment sought. This requires predictable engineering techniques allowing estimating costs as a function of design choices (e.g., processor speed, hardware versus software implementation). *Market constraints will deeply transform engineering of computerized systems.*

These factors imply the need for system development technologies allowing to jointly consider functionality, quality, physical implementation, and market constraints:

- Functionality is the capacity to deliver the given service.
- Quality covers a set of properties related to performance, and dependability. Performance properties include dynamic aspects of the system behavior, such as: throughput, jitter, speed, efficiency, response time, latency, etc. Dependability properties include safety, security, availability, reparability, and all properties characterizing the system's capability to provide a service in the presence of faults, errors, overload, and any type of incident susceptible of disturbing nominal behavior.

  Amongst dependability properties, safety and security are the most important. Safety means resistance to faults or errors. This is an essential property, particularly for transport and process control applications.

  Security means resistance to attacks and active hindrance. This is essential for networked applications, such as banking and other commercial applications.
- Physical implementation constraints deal with the use of resources and the system's deployment context, such as weight, physical size, resistance to vibration or radiation, etc. For embedded systems it is crucial to make the best use of resources, either to minimize costs (e.g., memory), or to improve autonomy (e.g., minimizing power and/or energy)
- Market constraints cover aspects related to the product quality and time to market, for a given functionality and target consumer profile.

Currently, we lack methods and tools allowing to jointly take into account these different constraints. Achieving the capacity to build systems of guaranteed functionality and quality, at an acceptable cost, is a major technological and scientific challenge.

1.4 Current Technological Limitations

It is important to have a fairly clear idea of what is feasible under the current state of the art. We identify two main technological approaches and related know-how, with their associated application areas:

- Hard Real Time and Safety Critical applications are extensively used in controllers, particularly for transport and process control. Such applications must meet deadline requirements, to guarantee that the system will be fast enough with respect its physical environment. They are composed of strongly coupled/coordinated components, integrated into a dependable, highly constrained architecture.

- Distributed Soft Real-Time applications with a good level of reactivity and dependability. Such applications are used in telecommunications, networked applications, etc. They are characterized by their quality of service, rather than by hard real-time requirements. Quality of service expresses dynamic properties other than strict deadlines, such as throughput, jitter, mean or statistical values for time-related parameters.

We currently have the capability to build relatively small hard real-time applications on the one hand, or large distributed applications on the other hand. In the coming years, it will be important to extend the applicability of hard real-time and safety-critical technologies to other areas (e.g., automobile). This implies certain accompanying technical adaptations, and that these technologies are made available at compatible costs.

In the longer term, we believe that it will be necessary to achieve large-scale, dependable distributed real-time systems, to allow applications such as automated freeways, and next-generation air traffic control.

2. Scientific Challenges

2.1 System-centric Approach

The need to jointly consider functional and extra-functional constraints in the design of embedded systems leads to the concept of system-centric development. Here, the main focus is the end result: *a system as the combination of hardware and software, in interaction with its physical environment*.

System-centric approaches are necessary to determine trade-offs between cost and quality, taking into account specific features of three factors: hardware, software and the environment.

In principle, hardware compared to software is faster, less flexible, slower to develop, and more costly. Taking into account the dynamic characteristics of the physical environment – when known – can drastically simplify the design and optimize the components chosen and the dynamic resources used. For example, knowledge of the physical environment for embedded signal processors in cellular phones (specified by international standards) allows optimal choice of hardware.

Current methods and tools do not allow system-centric approaches for designing embedded systems. In fact, these approaches raise difficult, fundamental research problems, which are the basis of an emerging theory that should bring together information and physical sciences.

Information sciences consider models of computation based on abstract notions of machines (e.g., automata, complexity and computability theory, algorithms, etc.), that do not take into account physical properties of computation (e.g., execution times, delays, latency, etc.). From this point of view, software is abstract and is essentially characterized by functional properties. There is no unified theory allowing to predict the behavior of an application software on a given execution platform. The latter is composed of physical elements whose characteristics determine the execution speed and other dynamic properties of the application.

Clearly, a unifying theory is needed for encompassing concepts of computation that take into account physical properties of the underlying platform. Its absence raises a major scientific challenge, and seriously limits the state of the art in systems engineering. For software there exist theory and models for the verification of functional properties, whereas these are not available for implementations. Extra-functional properties of implementations are validated essentially via testing. This makes system validation costly, and less amenable to analysis.

2.2 Grand Challenges

A system-centric approach raises two grand challenges common to all the activities of system development. The first is theory and tools for rigorous component-based engineering. It has to do with our ability to build complex systems from simpler ones by mastering their complexity. The second is intelligence, a long term vision for systems that are able to analyze and adapt their behavior to changes of their environment.

2.2.1 Component-based Engineering

Component-based engineering is of paramount importance for rigorous system design methodologies. It is founded on a paradigm which is common to all engineering disciplines: *complex system*s can be obtained by assembling components (building blocks). Components are usually characterized by abstractions that ignore implementation details and describe properties that are relevant to their composition.

Composition is used to build complex components from simpler ones. It can be formalized as an operation that takes in components and their integration constraints. From these, it builds a new, more complex component.

We lack a general theoretical framework for component-based engineering. This is the main obstacle to mastering the complexity of heterogeneous systems. It seriously limits the current state of the practice, as attested by the lack of development platforms consistently integrating design activities and the often prohibitive cost of validation.

Theoretical frameworks for component-based engineering should include satisfactory solutions to two problems: The first is theory for composing heterogeneous components. The second is theory for establishing correctness by construction, to cope with complexity.

Heterogeneity of components

Heterogeneity of components is a main obstacle for systems interoperability.

There exist hardware and software components, components may differ in their interfaces, communication mechanisms, execution speeds etc. There exist, in our opinion, two specific sources of deep heterogeneity: interaction and execution.

- Heterogeneity of Interaction

  Currently, there exists no formalism jointly supporting all these types of interaction:

- Atomic or non atomic. For atomic interactions, the behavior change induced in the participating components cannot be altered through interference with other interactions. Synchronous languages and hardware description languages use atomic interactions. On the contrary, languages with buffered communication (SDL) or multi-threaded languages (Java, UML), generally have non atomic interactions.
- Strict or non strict interaction. Strict interaction can occur only if all the participating components are ready, for instance atomic rendezvous used in Ada. In synchronous languages, interactions are atomic and non strict in the sense that output actions can occur whether or not there is a matching input ready. Nevertheless, inputs are blocking until a matching output occurs.

- Heterogeneity of Execution

  Currently, there exists no formalism jointly encompassing both synchronous and asynchronous execution:

  - Synchronous execution is typically adopted in hardware, in synchronous languages and in time triggered architectures and protocols. It considers that a system run is a sequence of steps. It assumes synchrony, meaning that the environment does not change during a step, or equivalently "that the system is infinitely faster than its environment". In each execution step, all the system components contribute by executing some "quantum" computation. The synchronous execution paradigm has a very strong assumption of fairness built in: in each step all components execute a quantum computation defined by using either quantitative or logical time.
  - The asynchronous execution paradigm does not adopt any notion of global computation step in a system's execution. It is used in languages for the description of distributed systems such as SDL and UML and programming languages such as ADA and Java. The lack of a built-in mechanism for sharing computation between components can be compensated by using scheduling. This paradigm is also common to all execution platforms supporting multiple threads, tasks, etc.

.

Correctness by Construction

It is desirable that frameworks for component-based modeling provide results for establishing correctness by construction for at least a few common and generic properties such as deadlock-freedom or stronger progress properties.

In practical terms, this implies the existence of inference rules for deriving system and component properties from the properties of lower-level components, before composition. In principle, two types of rules are needed for establishing correctness by construction.

- **Composability** rules allowing to infer that, under some conditions, a component will meet a given property after composition. These rules are essential for preserving previously established component properties. For instance, it could be used to guarantee that a component without internal deadlocks would remain deadlock-free after composition.

Composability is essential for incremental system construction as it allows building large systems without disturbing the behavior of their components. It is the stability of component properties when its environment changes by adding or removing components. Property instability phenomena are currently poorly understood e.g. feature interaction in telecommunications, or non composability of scheduling algorithms.

A theoretical framework for composability is badly needed.

- **Compositionality** rules that allow inferring a system's properties from its components' properties. There exists a rich body of literature for establishing correctness through compositional reasoning. Nevertheless, most of the existing results deal with preserving safety properties.

2.2.2 Intelligence

For modern systems engineering, intelligence is advocated as the means for improving system quality (performance and dependability). The current vision differs from the one of "Artificial Intelligence" popular in the 80's. The purpose is not to automatically synthesize algorithms from abstract specifications, but rather to control an existing system's behavior so as to achieve a desired property. This vision of intelligence connects systems engineering to control theory.

Intelligent behavior seems to be the last resort for ensuring quality. In fact, well-known limitations of the current state of the art in systems engineering make various kinds of flaws in complex systems impossible to avoid: design errors, faults, failures. Building systems enjoying properties which characterize the behavior of living organisms such as autonomy, self-organization, self-repair, resilience, survivability, resource awareness etc., is considered to be a means to cope with possible defects. *This is certainly a very attractive and sensible idea that will mobilize considerable R&D effort in the future.*

Nonetheless, these problems should be tackled with pragmatism, without underestimating their inherent hardness. It is important that the research community does not to give way to hype and over-ambition.

Two properties characterize system intelligence:

- Reflexivity – the capability to analyze its own state, and to act on it. It can be seen as a simple form of self-awareness. For instance, reflexive systems should be able to perform auto-diagnosis, to detect failures or errors.
- Adaptability – the capability to adapt its behavior according to given quality objectives (performance, dependability). For instance, adaptable systems should be able to adjust their behavior so as to cope with intrusion or to adjust their scheduling strategies.

Ambient Intelligence as described in the IST work programme is closely related to this concept. Developing theory and practices for engineering intelligent systems is a long-term objective.

3. Overview of Technical Trends and Work Directions

3.1 Systems Development

Ideally, the systems development cycle is a sequence of steps moving from *abstract* to *detailed* descriptions. It is important that the transition from one description to the next be supported by tools. These may be either automatic translation tools (e.g., code generators, compilers, assemblers), or verification and validation tools when automatic, guaranteed-to-be-correct translation is not possible.

The requirements describe system interaction with its environment, in principle they abstract out implementation details. Ideally, the requirements are expressed in the form of models, which can allow formal verification.

Apart from requirements, there are typically two other levels of description in system development: the application software derived from the requirements, and the actual implementation on a given platform. The transition from requirements to application software is usually not fully automated. It is the result of a design process which integrates constraints that cannot be entirely solved through synthesis. The transition from the application software to a specific implementation is largely done through the use of automatic translation tools such as compilers, assemblers and linkers.

In the current state of the art, the transition between levels requires verification and validation tools to check that the system obtained meets the initial requirements. Depending on the domain, a large part of the development effort is dedicated to V&V activities.

We provide below an overview of technical trends and work directions for embedded systems development.

3.1.1 Model-based Development

The objective for model-based development is to automatically generate, from some formal model of the system requirements, the application software, and even the implementation. For instance, this approach is applied with varying degrees of success, through the use of tools such as Matlab/Simulink, for the development of real-time controllers. Model-based development is also the motivation for standards such as UML and SDL.

Model-based development supposes the availability of environments for modeling and early validation of complex heterogeneous systems. These environments should allow the automatic construction of models for complex systems, from:

- Libraries of models of heterogeneous components – hardware components, software components, environment models e.g., continuous dynamic systems
- Architectural descriptions e.g., interaction and execution models

3.1.2 Programming and Implementation Technologies

A current trend is to combine expressive programming languages with semantically aware programming tools. These allow code analysis by various means (abstract interpretation, model checking, etc.) to reduce errors before execution.

A second important emerging trend is the extension of programming languages for modeling system features. These aspects may include models of the target platform, dealing with architecture, scheduling (from WCET estimations and QoS), and security. The application software enriched with a target platform model is compiled to generate the implementation glue, in addition to the application's object code.

For instance:

- From the architecture model, the compiler would synthesize code for messaging protocols, application-specific handlers, and interaction between tasks.
- From the WCET and QoS models, the compiler would synthesize code implementing scheduling policies to meet the QoS requirements. Such an approach supposes the availability of accurate timing analysis tools to estimate WCET of code segments, for specific target architectures.
- Finally, code implementing security control mechanisms can be synthesized from security models.

This second trend brings programming closer to modeling, and it is anticipated that at some point the distinction between the two will disappear. Most existing modeling tools allow importing external functions and data from programming languages. Programs are a form of executable models – they can be enriched to describe models (e.g., via timing information).

3.1.3 Operating Systems and Middleware

The main trend for operating systems and middleware is to adapt existing technologies to the constraints of embedded systems. Existing technologies are often more complex than necessary, undependable, and with hidden functionalities. This leads to unmanageable complexity and difficulty of use.

For embedded OS and middleware technologies, strong requirements are:

- Modularity. To reduce resource consumption, it is necessary to use minimal OS and middleware configurations. This can be achieved by using modular architectures.
- Adaptivity. According to the system's dynamically changing requirements or activities, it may be necessary to reconfigure the system and optimize the use of available resources. This implies giving more direct control over the hardware to the applications. One direct consequence would be to move much of the resource management functionalities out of the kernel.

  Adaptivity is particularly needed for flexible scheduling. This is becoming an important topic for the efficient use of resources, particularly for meeting QoS requirements. The idea behind flexible scheduling is to replace traditional fixed priority scheduling by strategies. These apply policies for adapting to dynamic changes in the execution and external environments. Adaptive scheduling raises theoretical problems such as the composability of scheduling policies, which remain largely unexplored and open.

Efficient implementation of scheduling strategies requires observation and control of low-level state variables (clocks, registers, etc.) in real-time and with a low overhead.

- Dependability. Current operating systems and middleware do not provide adequate features for dependability.
- Domain specific OS and Middleware. It is absolutely necessary, in view of the specific constraints for embedded technologies, to have commercially available domain-specific operating systems. This is confirmed by the emergence of domain-specific standards, such as OSEK, ARINC, JavaCard, and TinyOS. For cost-efficiency, these should be built through composition of COTS operating system modules.

Operating systems and middleware are strategic for embedded technologies, and we believe that Europe should make an exceptional effort to reduce the gap in this area.

3.1.4 Control for Embedded Systems

Automated control applications are central to embedded technologies. They are used to implement piecewise continuous control laws, using algorithms which are discrete by nature. These applications are usually called 'hybrid' in the sense that they combine continuous dynamics and discrete state changes (e.g., when switching from one control law to another). They are needed for solving typical control problems appearing in applications such as flight control, unmanned vehicles, and process control for manufacturing. They also appear in applications where adaptability is sought, such as network traffic control and pricing policies, or adaptive scheduling.

The design and implementation of automated control applications raises the following hard theoretical and practical problems stemming from their hybrid nature:

- Modeling and Verification of Hybrid Systems
- Design of Hybrid Controllers
- Specific techniques for implementation of distributed automated control applications, which take into account the influence of delays, jitter, aperiodic sampling on performance.

A well identified problem in this area is the lack of a theory integrating multi-disciplinary aspects: automatic control, numerical analysis, and computer science.

3.1.5 Verification and Testing

Validation technologies are essential for the development of embedded systems. Verification means checking correctness with respect to given requirements on a system model. Testing usually means exercising the software on a given platform, and checking correctness with respect to given test purposes. The frontier between the two activities is fuzzy. The notable difference is that testing implies observability and controllability restrictions, which are not present in verification which works on a model.

Existing verification and testing theory has been developed on relatively abstract and high level models, upstream in the development cycle. In the coming years, it will be essential to move downstream, developing theoretical and practical tools for testing and verification of implementations. This implies taking into account all the dynamic aspects of execution on a given platform. In particular, it requires techniques for faithfully modeling the real-time behavior in runtime environments.

The following problems deserve careful consideration:

- Execution platform modeling theory and tools,
- Verification and testing of timed systems theory and tools,
- Composable and compositional testing and verification methods

3.1.6 Dependable Embedded Systems

Embedded technologies are poorly suited to traditional methods which improve dependability through redundancy. This is because, in principle, such techniques generally lead to costly solutions. Dependability properties should not be an “add-on” to undependable implementations.

Dependable embedded systems should be designed with dependability in mind from the beginning. Dependability should be achieved by application of a set of specific techniques applied throughout the development process. This has impacts on architectural choices, programming style, documentation for traceability, version control, etc. It also implies the use of well chosen fault-tolerant mechanisms, and extra features for observability, controllability (e.g., for testing and masking errors).

The existence of methodologies for developing systems having guaranteed dependability levels (e.g., safety, security, availability) is crucial. In the various application domains, there exist standards defining such methodologies. For instance, “DO-178B Process Control Software Safety Certification” for real-time avionics and the “Common Criteria for Information Technology Security Evaluation”.

In the longer term, technologies for certification will be needed. Such technologies should be based on the use of verification and testing techniques, which need to be adapted and applied to dependability properties. These are difficult to describe, as they should formalize all situations that could potentially endanger dependability.

4. For a European R&D Strategy

4.1 Development Factors

Policies aiming to support R&D in embedded systems should strive to improve the academic and industrial research potential, the mechanisms for transfer to industry, education and the capacity to produce well-trained engineers. The ultimate objective is to reinforce the following key factors for systems engineering:

- Know-how and Human Factors. Complex systems are developed on the basis of the collective expertise and skills in large engineering teams. As examples, the development of a new processor mobilizes a few thousand highly qualified engineers; the development of an operating system requires the skills and knowledge of a few hundred engineers. In the development process, the important factor is the collective body of knowledge and the application of rigorous methodologies. More generally, the system development requires mastering a know-how which is usually very hard to formalize. Know-how and human factors are the critical element for building complex systems in the current state of the art.

- Components. Proprietary or commercially available hardware and software components are important for the productivity of the development process. Components embody design knowledge and effort. Their well-reasoned use can bring significant gains. Currently, component-based engineering for hardware is mature and well supported by methods and tools, while for software this is not yet well mastered.
- Tools. Tools are integrated into system development environments to support the various activities, so as to improve productivity and quality. Tools are used either to translate descriptions (e.g., synthesizers, compilers, interpreters, etc.), or to analyze them (verifiers, static analyzers, testing tools).

The balanced and harmonious development of the above three factors is essential for systems engineering. Use of sophisticated tools by poorly qualified engineers may result in lower efficiency. Skilled engineers can compensate for the lack of tools.

Any efficient strategy aiming to strengthen systems engineering should seek a balanced development of all three factors.

4.2 The European Position

Europe has traditionally been strong in producing high-quality engineers with a broad scope and a common culture that is grounded in theory. The availability of such engineers confers a significant advantage to industry in a variety of sectors, such as automobile, space, avionics, and telecommunications.

For hardware components, Europe enjoys a strong basis and is well positioned to become a main player. Through continuous support at national and European levels, the microelectronics industry has grown spectacularly and can now be considered to be a main player worldwide.

For software components, Europe has not yet been able to take up the challenge, in spite of an excellent research potential. In fact, many basic concepts for software originated in Europe, including languages (Pascal, Ada, Prolog), synchronous languages, formal methods, concurrency theory, etc.. This imbalance is mainly with respect to the USA, and for embedded systems mainly concerns operating systems and middleware.

For tools, the imbalance is similar to the one for software components. The main CAD tool providers (Cadence, Synopsys, Mentor Graphics) are in the USA. For system development tools, Europe previously had a significant position with tools for SDL, such as ObjectGeode, Tau – distributed by Telelogic. This position is threatened by the widespread use of UML and related tools commercialized by companies in the USA, such as Rational and I-Logix. For synchronous systems development, the USA clearly has a very strong position with products such as Matlab/Simulink and Matrix_X. Lastly, the USA also dominates the test and debugging tools market (Testware and Purify by Rational Software).

4.2.1 Recommendations

Education

Embedded Technologies require engineers with a broad spectrum of competencies – reflecting the multi-disciplinary nature of the area. Currently, with few exceptions, this is a weakness worldwide. We lack engineers with a solid background in electrical engineering, automation and control, hardware and software engineering, etc. It is necessary to set up, promote, and implement curricula specifically for embedded systems, in Europe.

Furthermore, the arrival of embedded technologies in new sectors of application has brought the need for new competencies. This can be compensated either by hiring engineers with appropriate skills, or by providing additional training.

Critical Mass in Research

Currently, European research is characterized by fragmentation, low reactivity, and the lack of frameworks for transfer to industry (legal and incentives). We need to build large centers of excellence comparable to those in the USA (e.g., Carnegie-Mellon, MIT, Stanford, and Berkeley).

In Europe, this can be achieved through the integration of competencies spread across many European institutions. Setting up Networks of Excellence in the 6th FP can be a first step towards such integration. For this to be meaningful, it is necessary to integrate the best teams for each of the work directions identified in section 3. These teams will need to work together on the basis of a joint programme of activities.

R&D Projects

It is important to tackle projects focusing on system-centric approaches, where the emphasis is on systems as a combination of hardware and software and their effective use. These projects should be characterized by:

- Commitment to extend the state of the art in a chosen application area by involving the main players from industry and academia. The quality of a project depends mainly on the strength and weight of the partners, as well as their willingness to achieve the objectives. If the aim of the IST programme is leadership in embedded systems, then we should launch ambitious projects involving the best European teams. Clearly, the technical quality of a project description does not by itself imply success if the consortium is weak.
- Adequate level of funding. Sufficient resources must be mobilized in proportion with those available in the USA for similar initiatives. Also, a structured approach to achieve critical mass and momentum on chosen topics should be adopted, rather than striving for uniform thematic and geographic coverage.
- Tools and components. Focus on enabling technologies, which are strategic for European embedded system developers. Over the last fifteen years, we have seen a number of European startups and high-tech companies in CAD, OS, testing and verification absorbed by larger US companies. This reduces the leverage in Europe to develop solutions that are well adapted to European industrial needs.

This requires a well-considered long-term, multi-faceted European strategy, including measures to promote transfer of research results through the creation of startups, incentives and appropriate legal and institutional structures. As direct users of components and tools, embedded system developers should be strongly associated in this strategy. The application of innovative development techniques should be encouraged through standards for quality, and R&D projects.