Embedded Systems Roadmap 2002

Vision on technology for the future of PROGRESS

edited by

Ludwig D.J. Eggermont[1]

30 March 2002

1.For authors and other contributors see page 9

CLP - gegevens koninklijke bibliotheek, Den Haag

Roadmap

Embedded Systems Roadmap 2002

March 30, 2002 / [ed. Ludwig D.J. Eggermont]

Utrecht: STW Technology Foundation/PROGRESS

ISBN: 90-73461-30-8
STW-2002

Keywords: Embedded systems, technology roadmap, embedded system design, PWA, Personal Well-being Assistant, roadmapping

Published by:
Technology Foundation (STW)
P.O. Box 3021
3502 GA Utrecht
The Netherlands
Phone: +31 30 60 01 268
E-mail: progress@stw.nl

Project leader, editor and process facilitator:
Ludwig D.J. Eggermont,
Eggermont Consultancy
De Speldenmaker 19
5506 CE Veldhoven
The Netherlands, E-mail: ldje@iae.nl

Executive Summary

Opportunities and threats

The importance of embedded systems is undisputed[1]. Their market size is about 100 times the desktop market. Hardly any new product reaches the market without embedded systems anymore. The number of embedded systems in a product ranges from one to tens in consumer products and to hundreds in large professional systems. An average household employs easily 50 embedded systems nowadays. This will grow at least one order of magnitude in this decade. Professional systems will see a similar growth. Besides, many distributed systems will rely on embedded systems for an ever larger part of their functions.

The strong increasing penetration of embedded systems in products and services creates huge opportunities for all kinds of enterprises and institutions. At the same time the fast pace of penetration poses an immense threat for most of them. It concerns enterprises and institutions in such diverse areas as agriculture, health care, environment, road construction, security, mechanics, shipbuilding, medical appliances, language products, electronics, etc., etc. They all need to respond timely in mastering the following technological and market challenges:

- Wide diversity and increasing complexity of applications
- Increasing number of non-functional constraints
- Increasing degree of integration and networking
- Increasingly multi-disciplinary nature of products and services
- Growing importance of flexibility and software
- Shrinking time-to-market

The current situation is especially threatening for small and medium-sized enterprises. More than half of them will disappear in the next decade unless they find means and ways to absorb and develop further the necessary know-how for the embedded systems in their products and services. Great efforts will be required for technology development, but this will not be useful if, at the same time, not enough money is spent to obtain a sufficient number of persons with the right level of education. The current prospect is worrying in this respect.

Purpose and scope of the Embedded Systems Roadmap (ESR)

Obtaining a clear picture of the essential technology developments for embedded systems and finding the related technology gaps is therefore an essential task and the purpose of the ESR. The scope is restricted to technologies for embedded systems incorporated into information processing, possibly networked, embedding systems. A certain focus on System-on-Chip related technologies can be

1.Report Task force ICT-en-kennis (2001), Citing from the management summary: 4c) The market for embedded systems will grow exponentially also in the next decade..... Every company faces the challenge of changing over to embedded and distributed systems to not get off track.
See also pp. 33/34: Competing with embedded and distributed systems.

observed. This was not so much the intention but more the result of the selection of the experts involved. When other areas become more important the roadmap needs to be adapted to that situation.

Target groups

The Program Committee of the Dutch PROGRESS (PROGram for Research in Embedded Systems and Software) chartered a Core-Team of persons representing the Dutch embedded systems community with this roadmapping task in March 2001. This roadmap document is the result of their efforts. Major goal of the document is to enable program management in the next phase of PROGRESS. The main target group is therefore the PROGRESS PC, the program managers of collaborating consortia, as well as their government contacts. But the roadmap will be equally useful for industrial R&D strategy and marketing managers and for group leaders of research groups at universities, institutes and industry.

Positioning of a roadmap

To explain the position of a roadmap (and especially this one) with respect to vision documents and research agendas or programs of projects the following picture may be helpful:

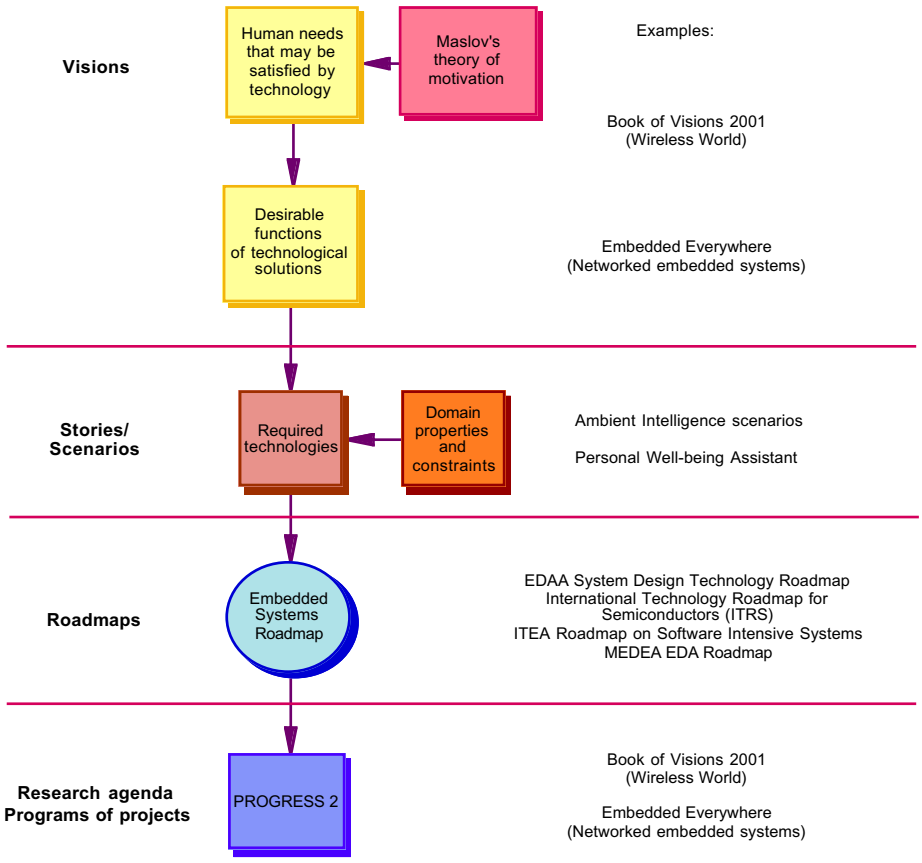


Figure 1: Positioning of the Embedded Systems Roadmap

As shown in the figure above the Embedded Systems Roadmap came about in a creative learning and search process that consisted of three phases.

In the first phase a vision on human needs in an application domain was created on the basis of what motivates people most. These human needs were subsequently interpreted as desirable functions of potential technology oriented solutions. From this resulted a vision on desirable technologies over time.

In the second phase this was further worked out in scenarios of rendez-vous of several technologies. The scenarios combined the potential technology needs with a view on the development of driving embedding system applications that could fulfil a selection of representative user needs. The resulting domain paper 'Personal Well-being Assistant: creating a society of well-being' is added to the ESR in Appendix 3. Geared towards future needs of designers of embedded systems is the domain paper 'Domain of the Embedded Systems Designer' in Appendix 4. This paper presents scenarios of rendez-vous for designer needs.

In the third phase a roadmap structure was extensively discussed and agreed upon. Technology information from the scenarios was extracted, refined and mapped into the roadmap structure. In the latter part of this phase a consensus process was entered in which extensive reviewing took place and in which also interviewing of experts was employed in an international evaluation workshop. This workshop served to reach further agreement on the ESR contents with the international embedded systems community. It also created a wider view on the field and served as an independent audit of the results.

Appendix 6 explains the roadmap process more comprehensively.

The roadmap structure

The Embedded Systems Roadmap has been segmented in two major parts:

1. The first part deals with the expected and desired developments in the major characteristics of embedded systems. Apart from general aspects that influence embedded systems, interaction and information processing are seen as the two most important technology areas for embedded systems.
2. The second (and larger) part deals with the design of embedded systems. The following areas are distinguished in this part:
   - From idea to executable specification
   - From executable specification to implementation
   - Platform design
   - HW/SW design
   - Verification and validation
   - Test, debug and integration

In each of these areas are high-lighted as much as possible the aspects that are specific for embedded systems.

In an earlier version a separate chapter was devoted to embedded software. Although there are many software-related problems in embedded systems, it was found that they are so closely coupled to many other design issues that they were best treated together with the other issues. This is not to say that software-only problems are not important but only that many of them are not embedded-system specific. And in the Embedded Systems Roadmap the focus is as much as possible on those aspects that are specific for embedded systems.

Major challenges

The major technology trends signalled in the Embedded Systems Roadmap create the challenges given below.

- An increasing degree of heterogeneity and networking of embedded systems can be expected to lead to a further complication of embedded systems design.
- A larger variety of information sources to interact with could easier be accommodated with fewer, but better co-ordinated standards on sensors and actuators.
- Working out the consequences of ever more integration of all kinds of technologies in virtual and physical implementation is a central theme of information processing in embedded systems.
- Designing the right system on target, without over- and without under-specification, is the major challenge for high-level embedded systems design.
- Moving from executable specification to implementation, a characteristic trend is that both transformations from software into hardware and those from hardware into software become increasingly important to consider.
- In platform design a strong need originates to derive more instances from a given platform to increase the cost efficiency or optimality of a platform.
- In hardware/software design a major challenge is to solve the design time problem and bring features faster to the market.
- Verification and validation of designs remains a significant bottleneck to design on target, while it is expected that for quality control reasons the burden of testing will move closer to the end-user.

Opportunities for action

Six major issues of the Embedded Systems Roadmap have been prioritised by the Core-Team. They constitute major opportunities for action and are summarised in the following paragraphs in the given order of priority.

1. *Promote, develop and facilitate the reuse of IP (Intellectual Property) blocks* on a broad scale, as an important enabler to increase design productivity. The publicly available certified IP blocks should contain models, which can be used to simulate, verify, debug, and test the embedded system including the IP blocks at different levels of abstraction. IP blocks are to be easily integrated in the embedded system by means of standardised interfaces. Note that IP blocks may comprise both hardware and software.
2. *Compilers and translators.* There is a need for at least two kinds of compilers. The first kind is retargetable towards various hardware designs in order to deliver code efficiently executing on these designs. The second kind of compilers derives the hardware architecture from the behavioural specification based on several cost functions and simultaneously generates the software executable for this hardware.
3. *Specification.* An urgent need exists for methods and tools that capture ideas in models in analogy to the virtual reality modelling as has become accepted in other areas such as the automobile industry. A knowledge base of re-usable parts and metrically quantified design experience must be developed to aid the designer in balancing constraints to obtain the final specification. Methods for

closing the gap between requirements and specification are lacking and are urgently needed.

4. *Design-Space Exploration (DSE).* To develop methods and tools to evaluate design decisions concerning the allocation of computation & communication (tasks) to resources with the purpose of obtaining high-quality solutions. This evaluation can be performed by co-simulation of executable models of heterogeneous building blocks at the appropriate abstraction levels.
5. *Verification & validation.* Apart from the highly desirable but difficult to realise correctness-by-construction, formal verification and validation are the only ways to solve the ever-growing simulation burden to verify the correctness of design steps. Although formal verification and validation are not feasible for all designs, they can be applied successfully in many cases. For this purpose it is necessary to educate the designers early in these techniques and to design more user-friendly verification tools by separating the verification and validation functionality from the underlying mathematics.
6. *Test.* The large diversity of hardware and software that comes together in an embedded system in various shapes and quantities, needs to be addressed from a single unified view on testing. The testing challenge is further aggravated by the increasing role of redundancy and heterogeneous and polymorphic parts. Increasingly we will need on-chip measures and dedicated test functions to allow for an acceptable fault diagnosis, isolation and repair over the life cycle of the product.

More detailed recommendations for action in each of the ten sub roadmap areas may be found at the end of each sub roadmap section.

Acknowledgement and concluding remarks

This roadmap could not have made without the support of the organisations that made their experts available for this project. This is therefore the right place to acknowledge the contribution of:

- Philips Research
- Eindhoven University of Technology
- University Twente
- Groningen University
- Delft University of Technology
- Leiden University

In this document the ESR is presented as the roadmap for embedded systems technology. It is based on extensive meetings of experts and on reviews from many persons that have commented in one capacity or another.

This is not the end of the story, but just the beginning! A next step, as shown in Figure 1, is to translate this roadmap into a research agenda or program of desirable R&D projects. Depending on the focus of such a program certain areas will have to be explored more in-depth. Besides, the many non-technology areas that constitute critical success factors need to be addressed urgently.

Finally, this roadmap introduces the current state of trends and user needs and the interpretation of their consequences. It will therefore be necessary to regularly update the vision presented.

Authors, Core-Team members and other contributors

Editor	Affiliation	Remarks
Ludwig D.J. Eggermont	Eggermont Consultancy	http://www.iae.nl/users/ldje

Authors	Affiliation	Remarks
Bart Mesman	TUE-EESI/Philips Research	
Ben Spaanenburg	RUG	
Ed Brinksma	Univ. Twente	
Ed Deprettere	Univ. Leiden-Liacs	
Eric Verhulst	Eonic	
Floris Timmer	BGenT	
Hans van Gageldonk	Philips Research	
Ludwig D.J. Eggermont	Eggermont Consultancy	
René van Leuken	TU Delft-Dimes	
Thijs Krol	Univ. Twente	
Wim Hendriksen	Route 67	

Core-Team Members	Affiliation	Remarks
Bart Mesman	TUE-EESI/Philips Research	Since 2001-03-27
Ben Spaanenburg	RUG	Since 2001-10-18
Ed Deprettere	Univ. Leiden-Liacs	Since 2001-03-27
Hans van Gageldonk	Philips Research	Since 2001-03-27
Irek Karkowski	TNO-FEL	Until 2001-05-03
René van Leuken	TUDelft-Dimes	Since 2001-03-27
Thijs Krol	Univ. Twente	Since 2001-03-27
Wim Hendriksen	Route 67	Since 2001-10-18

Project management and process facilitation	Affiliation	Remarks
Ludwig D.J. Eggermont	Eggermont Consultancy	Project leader and process facilitator Since 2001-02-01
Floris Timmer	BGenT	Facilitator until 2001-07-02
Stephan Eggermont	Sensus	Ass. project leader and DTP Since 2001-03-27

The authors like to thank the following persons for their contribution in improving earlier versions of this roadmap:

Ahmed Jerraya	Jos Huisken
Arnold van Ardenne	Jozef Hooman
Dirk Giesen	Kees Goossens
Donatella Sciuto	Kees Vissers
Eric Dortmans	Lambert van den Hoven
Francky Catthoor	Lars Philipson
Frans Beenker	Loe Feijs
Frits Greuter	Maarten Boasson
Frits Malotaux	Marc Duranton
Frits Vaandrager	Marc Engels
Ger van de Broek	Marco Diepenhorst
Gerard Vos	Martin Elixmann
Giovanni de Micheli	Martin Rem
Hans Duisters	Patrick Dewilde
Hans Toetenel	Peter Marwedel
Harro Jacobs	Pierre Paulin
Henk Corporaal	Rix Groenboom
Herman Beke	Roelof Hamberg
Huib Pasman	Rolf Ernst
Jac Goorden	Siebren de Vries
Jan Broenink	Steven Luitjens
Jan Madsen	Ton Zengerink
Jef van Meerbergen	Wolfgang Nebel
Johan Lukkien	Yervant Zorian
Jos Baeten	

Scope

The Embedded Systems Roadmap focuses on the characteristics of electronic information processing embedded systems and their design challenges. A focus on embedded SoC can be observed in some parts of the ESR. This relates clearly to the way the ESR came about. The contents of the ESR is to a large extent determined by the interests, capabilities and competencies of the experts of the Core-Team, the groups of Workshop participants and the reviewer community involved.

Depending on the needs of the users of the roadmap, future versions of this technology roadmap may cover more technologies like board design, packaging, mechanical reliability, large scale embedded systems.

This document is intended for the following target groups:

1. Program managers of collaborating consortia and their government contacts
2. Group leaders of research groups at universities, institutions and industry
3. Individual researchers in the technology areas covered
4. Strategy and marketing managers of the embedded systems industry

It brings for each of its target groups the following:

	Quantified technology trends	Technology priorities over time	Technology gaps	Technology dependencies
Program managers		x	x	x
Group leaders		x	x	x
Researchers			x	x
Strategists/ marketing mgr.	x	x	x	x

Figure 2: Target groups and deliverables of the ESR

The ESR depends for its realisation on many developments outside the direct application domain of embedded systems:

1. General trends in society related to individualisation, globalisation, mobility, safety and security, fashion sensitivity, changing composition of households and population
2. General trends in business and business models: flat organisation, focus on core-business, multi-site/multi-company co-operations, e-business, shift to services
3. General trends in technological areas necessary for the development and production of Embedded Systems: ITRS: Moore's law (semiconductor technology: CPUs, memories, ASSPs, FPGAs, etc.), network and communication capacity growth, databases, display technology, packaging technology, sensor/ actuator technology, MEMS technology, etc.

These aspects are not treated extensively in this roadmap, but reference is made, where appropriate, to consequences or to documents related to these aspects. And some of these aspects come into play in the domain papers, especially in the domain paper on the Personal Well-being Assistant, in Appendix 3.

As many reviewers signalled a wide variety of non-technological aspects relevant for embedded systems in one way or another, Appendix 5 contains a write-up of many of these aspects.

Contents

List of figures

List of sub roadmaps

1 On Embedded Systems

Embedded systems are highly specialisable, often reactive, sub systems that provide, unnoticed by the user, information processing and control tasks to their embedding system.

1.1 The importance and impact of Embedded Systems

Embedded systems are omnipresent nowadays. But as they are hardly noticable, their importance and impact are often underestimated. They are applied as sub systems in a wide variety of applications for an ever larger diversity of functions. That their market size is 100 times as large as the desktop market can be easily understood when one oversees some of their application domains:

- In consumer products traditional mechanical controls are since long replaced by electronic embedded systems. Many further enhancements and numerous new home control, kitchen appliances and white-good products have been designed based on sensor/actuator signal processing by embedded systems.
- In audio and video consumer products embedded systems are used for control processing in the user interface, the internal infrastructure control and for more and more advanced audio and video signal processing, storage and I/O.
- In communications not only mobile phones but also the infrastructure depends heavily on the use of standards implemented in embedded systems.
- In desktop and mobile computers embedded systems are indispensable for computing, storage, communication, I/O and display functions.
- In professional areas like medical systems, traffic control, environment, security, driving and car control, health care, airborne equipment, plant control, agricultural equipment, etc., embedded systems make possible the creation of systems with a functionality that can not be provided by human beings. In these areas often an extensive infrastructure depends on distributed embedded systems. In many areas e.g. health care, embedded systems may help to alleviate manpower problems,

This extremely wide variety of applications of embedded systems implies that our society has become to a large extent dependent on the proper functioning of embedded systems.

1.2 Characteristics of Embedded Systems

The pervasion of embedded systems derives in the first place from the economy of solution they provide while meeting a plethora of constraints. They obtain this economy often by using to a large extent specific hardware and/or software components, even for high-volume consumer-electronics applications. Here one would expect custom solutions mainly. Also the possibility to share an embedded systems platform over many different applications in a domain gives economy of scale to the solutions in which they are used. The fact that they create a high degree of flexibility in solutions by programmability and/or (re) configurability makes them very attractive. Especially when standards have not been completely

frozen this kind of flexibility is essential. Besides, in the past embedded systems have shown to be able to make profitably use of the increasing economy of all needed technologies in their evolution over time.

Figure 3: Embedded systems will be everywhere, but mostly unnoticable or invisible, to enhance the functionality of the devices and equipment shown

Embedded systems are characterised by the following properties:

1. They are an information processing sub system of their embedding systems.
2. They provide specific and highly specialisable information processing services to their embedding systems.
3. They are reactive, i.e. they interact with their physical environment often in a continuous mode at a speed imposed by the environment.
4. They provide usually a complex functionality to their embedding system with a combination of hardware and software specialised to meet a wide variety of non-functional constraints.
5. They are mostly not visible or directly accessible by the users of the embedding system although they are often used to increase the user-friendliness and awareness of an embedding system.

Embedded systems interact with their environment via sensors and actuators or via communication interfaces. Often they make use of interfaces to standard communication infrastructures when used in distributed applications. Also ad-hoc networking may be used.

Embedded systems may incorporate embedded systems themselves as shown in the picture below.

1.3 What makes Embedded Systems special

Embedded systems are mostly reactive systems, which means that they react continuously to their environment at a speed imposed by the environment. This in contrast with interactive systems that respond to external stimuli when they are ready with calculating their response i.e. at their own pace, and with transformational systems that process a block of input data into a block of output data. Reactivity imposes often real-time capabilities. This results in special requirements for the hardware and software architecture of the platform to be used.

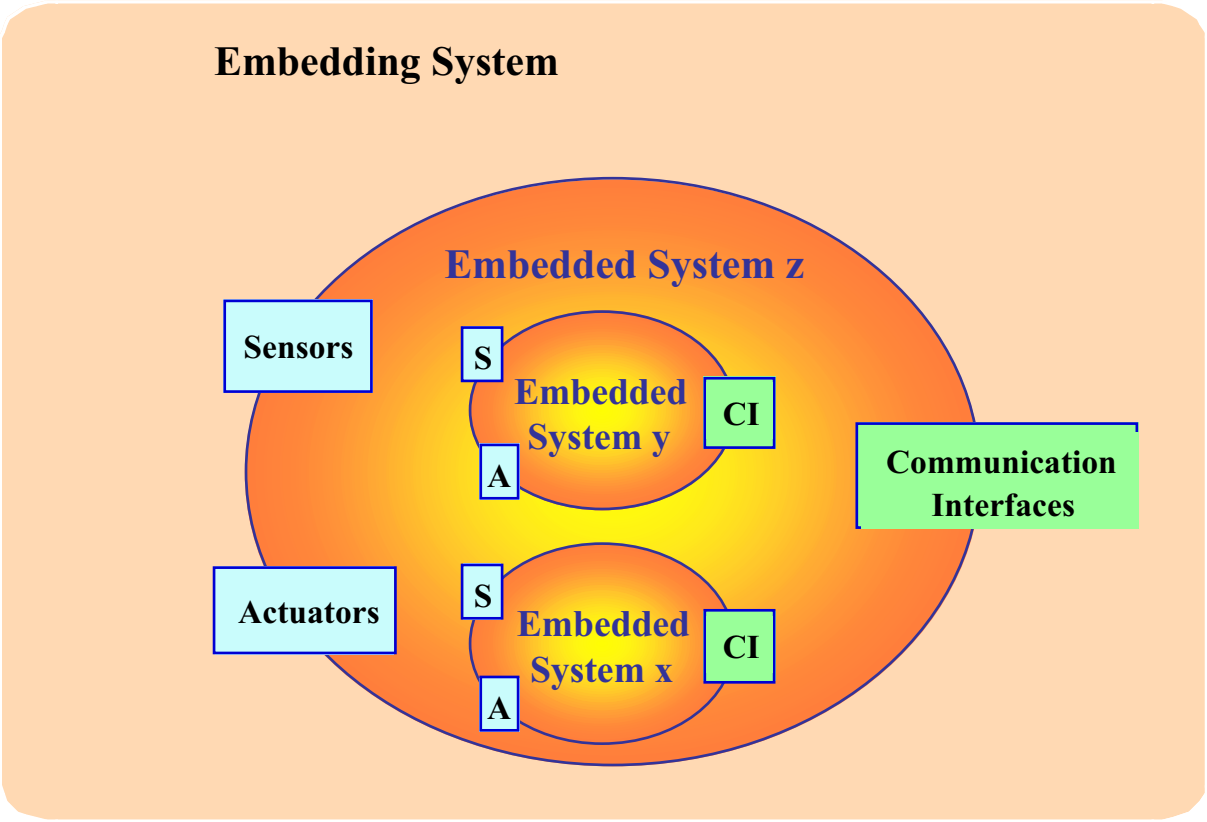


Figure 4: Embedded systems in their environment

Many non-functional constraints have a strong influence on design objectives and architecture of embedded systems. Low cost is an inherent requirement as embedded systems are not visible to the user, so their cost should be minimal. Often being used in mobile or wearable appliances low power is also a standard constraint for embedded systems. EMI and EMC: electromagnetic interference and compatibility requirements are important due to the different environments of which embedded systems can be part of. Hard timing constraints are imposed often as a consequence of a required real time response e.g. when processing A/V signals or when controlling an air plane by software only, instead of a mixture of software and direct mechanical control by the pilot. Reliability, robustness and safety constraints derive from situations where restart is impossible and a certain degree of autonomous behaviour should be possible. Size and weight are usually heavily constrained for embedded systems. System-on-Chip implementation is

also often required. All these difficult requirements and constraints make the design of embedded systems especially complex as well as demanding.

Special design related challenges come from the specialisation and customising of target platforms in their use for embedded systems. It becomes possible by detailed application know-how and the challenge is how to maintain some degree of flexibility, with the related wish to increase the reuse of hw and sw components.

A more recent trend which characterises many embedded systems and which poses several new design requirements is the design of distributed co-operating embedded systems.

The use of many disciplines and the heterogeneity of applied technologies are the last but not the least important factors in making the design of embedded systems special.

1.4 The world of Embedded System designers

The design of embedded systems has become a quite complex matter. As a consequence not a single person can master the complete trajectory from idea to testing and system integration. With the increase in complexity rises the necessary number of abstraction levels and more different specialists need to be involved to come to ‘optimal’ overall solutions.

The design flow shown below illustrates this growing complexity of embedded systems design by colouring the major specialist areas differently: systems architects, platform architects, board and System-on-a-Chip architects, (co-) verification specialists, hardware and software designers, test engineers, system integrators, etc. The world of the embedded system designer who, as a systems architect, deals with understanding the domain and the idea of the principal, eliciting the requirements of his solution and converting this into an executable specification, is quite different from that of the embedded system designer who works on architecting a platform for a specific application domain. And this again is quite different from the challenges of the world of the embedded system designer who works as a board or System-on-a-Chip designer to design a cost-efficient solution.

It will be understood that the large number of disciplines involved gives easily rise to mis-communication with the next person in the design flow. To address this problem special attention should be paid to proper use and definition of the terminology in all phases of the design and between all parties involved. Therefore, one of the appendices of this Embedded Systems Roadmap is a paper with terminology definitions.

If hidden semantic problems are not already bottleneck enough, the design situation is aggravated by the many different cultures that come together in embedded systems design. The fact that these cultures are different is not the bottleneck but the lack of understanding of the other specialists’ problems and solutions is.

Notorious in this respect is the communication, or better, the lack of communication between hardware and software worlds. This still hampers the evolution of hardware/software co-design. Part of the difference in culture stems from the different ways of dealing with design abstractions. Hardware complexity has grown

over the years. In line with Moore's Law it has become necessary to add design abstractions to master the exponential growth in available design elements at a level: about one level is added every 8 years since 1970. A hardware designer usually takes at most two levels into account in his design: the level at which he specifies and solves his problems using the building blocks and their available communication mechanisms from one level below. The efficiency of lower levels is taken for granted by him from a perspective of design cost and time-to-market. Software designers have similar complexity problems, but don't have similar physical forces leading to a shared view on how to deal with abstraction and hierarchy. This difference in visibility and structuring of hierarchy poses significant co-ordination problems in projects with combined hardware/software design.

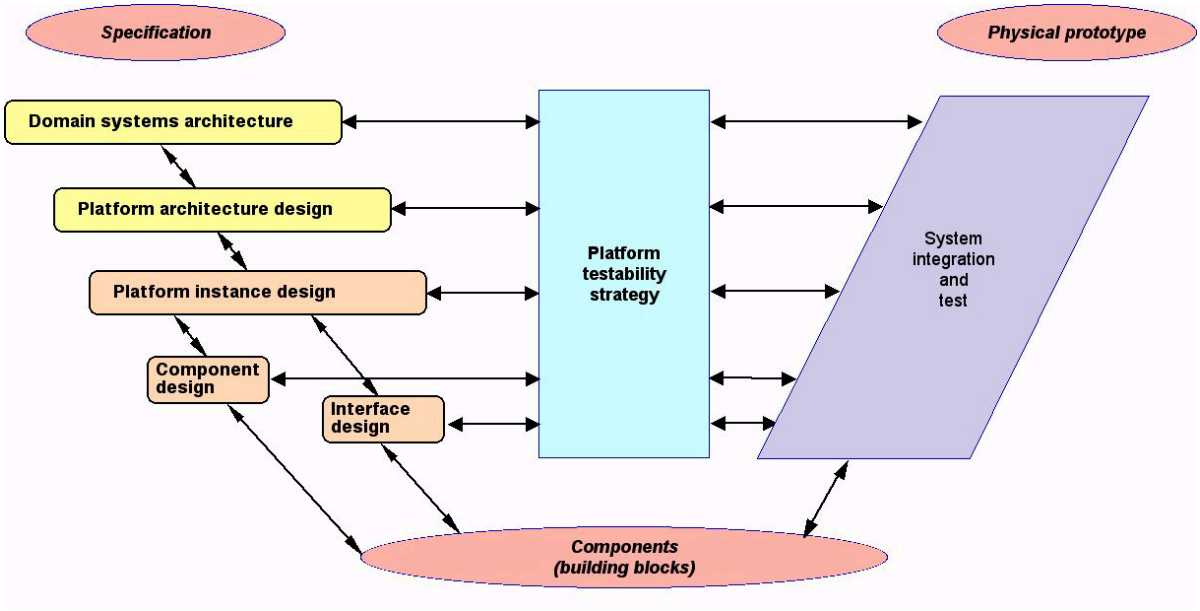


Figure 5: The overall design flow for embedded systems design

Also the worlds of system architects at the highest level (yellow in the figure) and of those involved in more implementation oriented design aspects (light-brown in the figure) are quite different. The first ones deal with concepts and are satisfied with executable specifications that simulate properly the product concept, while the latter start from executable specifications that should incorporate sufficient lower level information to simulate properly the actual hardware and software behaviour of a product. What executable specification means can therefore differ significantly.

In conclusion: apart from expressing the iterative process of specification, construction and verification for hardware/software co-design, a design flow for embedded systems represents also a communication and co-ordination flow between the different types of designers involved.

1.5 How to read the visual representations of the roadmap

The visual representation of each sub roadmap shows which technologies play a role at which moment in time in a specific technology sub domain in a sub roadmap. The division of the sub roadmap in technology sub domains is quite important as it signals what the major areas of attention in a sub roadmap domain should be. The order in which the technologies need to be developed as well as some of the dependencies between technologies within the same technology sub domain, within the same sub roadmap as well as between sub roadmaps are indicated.

A start is made to classify technologies in different categories:

1. Evolution of state of the art: known problem, state of the art solution available, work on necessary evolution is going on: green arrow. It is assumed that the state of the art is known (but not necessarily by us!) and available when the arrow starts and that the end point indicates when the subject is sufficiently mature to not longer require R&D activities of the same nature as before.

2. Technology gap: known future problem, idea on how to obtain a solution also known, indicated is the time at which working on the solution should or can start (but not by whom or where) and how long it might take to make the solution available (also without attempting to estimate the required R&D manpower or critical mass to develop a useable solution!): yellow arrow. This is in fact the development of technologies to realise the rendezvous described in the scenarios of the domain papers.

3. White gap: known future problem, no idea yet how to solve it: white arrow with thick red line. It is often also not known if people are already working on the technology (or even have found a solution already!).

A further refinement of this classification might be necessary in the future, depending on the required use of this type of roadmaps.

In the roadmaps arrows may overlap. Arrows may contain other arrows to indicate a hierarchical relationship. Successive arrows in line indicate a time dependency in the development of the technology. Dependencies may exist, symbolised by normal black arrows, that necessitate that certain technologies be first developed before it makes sense to start developing the indicated technology.

Each of the sub roadmaps of the Embedded Systems Roadmap uses the same set of graphical symbols. The remaining symbols used and the semantics of these symbols are as follows:

1. Time axis:

The time period over which we want to discuss and predict which technologies are needed for desired products and services and their design.

2. Essential trends:

The most important technical, economic or social factors that have a significant influence on this sub roadmap. Indicated are the name of the factor, a short description of its major result or consequence and a quantification of the result or consequence of the trend.

3. Challenge:

A short message (with the character of a quote) that summarises the most important future development or requirement of this sub roadmap domain

4. Technology sub domain:

Major structure element representing a technology sub domain of the sub roadmap, presenting the possibility to make a further refinement in technological problem areas that need to be tackled.

2 Embedded systems

2.1 General Aspects

'Increasing heterogeneity'

2.1.1 General trends and user needs

To a large extent embedded systems are influenced by the same trends that apply to other systems that use advanced technologies from the electronics, optics and micro-mechanics areas.

In the first place this implies that embedded systems will continue to follow a number of existing trends. The complexity of embedded systems will grow exponentially, with a size increase of the micro-electronics components of a factor of two every 18 months. Board solutions will follow similar complexity increase. To fulfil the related need for increase in flexibility a significant growth in embedded software will have to be accommodated as well. This may easily mean that software cost will dominate embedded systems design cost.

Besides, the number of specification points of an embedded system will increase at least one order of magnitude. This stems, apart from the increased complexity and its accompanying functionality increase, also from the expected growth in the number of different technologies needed in building an embedded system. Herewith also the number of different professions involved will increase proportionally. This number will continue to rise from an average of three now to about nine in 2011.

To be able to design these heterogeneous systems at board and IC level and stay within the time-to-market requirements that become more and more constrained, the amount of re-use of existing hardware, software and hw/sw components has to increase drastically from a current 20% to at least 80%. This aspect is coupled to the general need to increase design productivity significantly to be able to cope with the increasing complexity timely. Besides, a reduction of a factor 10 in the non-recurring engineering (NRE) costs needs to be obtained.

Some of the relevant trends are captured in existing roadmaps that therefore should be taken into account when discussing embedded systems development trends. To these belong the International Technology Roadmap for Semiconductors (ITRS, formerly called SIA roadmap), the Roadmap for Software Intensive Systems (ITEA), the EDAA Systems Design Technology Roadmap, Finkelsteins' Software Roadmap, etc. Where and when these trends create dependencies for embedded systems technologies will have to be indicated with linkages in the Embedded Systems Roadmap.

Apart from the needs that originate from the application of advanced technologies as discussed above, embedded systems have to take into account user needs[1] that derive from general trends in society related to aspects like individualisation,

1.See the domain paper in Appendix 3: Personal Well-being Assistant: creating a society of well-being

globalisation, mobility, safety and security, fashion sensitivity, changing composition of households and population. Increasing individualisation leads e.g. to more diversity in products and services, and therefore to the need for more flexibility in design, which in turn leads to an increasing software content. Globalisation of products and services necessitates multi-site design teams and increases the need for standards. Growing needs for safety and security in electronic transactions and mobile communications leads to functionality extensions that must be designed-in. Increasing fashion sensitivity results in shorter product or service life cycles, and thereby in a necessarily shorter time-to-market leading to shorter available design time. All these trends signal the ever growing societal impact of embedded systems.

2.1.2 Technology requirements

The embedding systems into which embedded systems have to be incorporated share in all application domains the overriding requirement of shortening the time-to-market (TTM). The major bottleneck in this respect is currently hardware/software co-design. The next hurdle to be taken is the easy integration of subsystems of acquired IP blocks. Qualification and certification of IP constitute a major part of the easy integration challenge together with the development of the necessary standards to make this feasible. Subsequently, the degree of integration will have increased to such a level that integration of what today are called systems, becomes possible and poses its specific problems.

The systems architecture of embedded systems will be more and more based on platforms, both board and IC level. Initially these will be proprietary, both for systems houses and for silicon platform providers. But the growing opportunity to integrate more than one company can design will force the origination of standards to enable easy IP exchange. This in turn will open the opportunity for multi-vendor platforms.

In hardware design technology one new level of abstraction needs to be introduced over the period of the Embedded Systems Roadmap to allow designers to cope with the growing complexity. This prediction is in line with the developments over the first three decades of micro-electronics development where it proved necessary to introduce about every 8 years a new level of abstraction. In software design a reconsideration of how to deal with levels of abstraction is urgently required as also embedded software reuse in ICs depends on it. And this is the first step to be taken to come more easily to higher levels of component integration. Also here the standardisation efforts mentioned above are necessary to be able to make the step from captive to merchant component integration.

The increasing heterogeneity will require that several technologies of implementation can be combined on a carrier technology. Especially in sensors and actuators the need for wireless communication will create a drive for a further integration of different technologies.

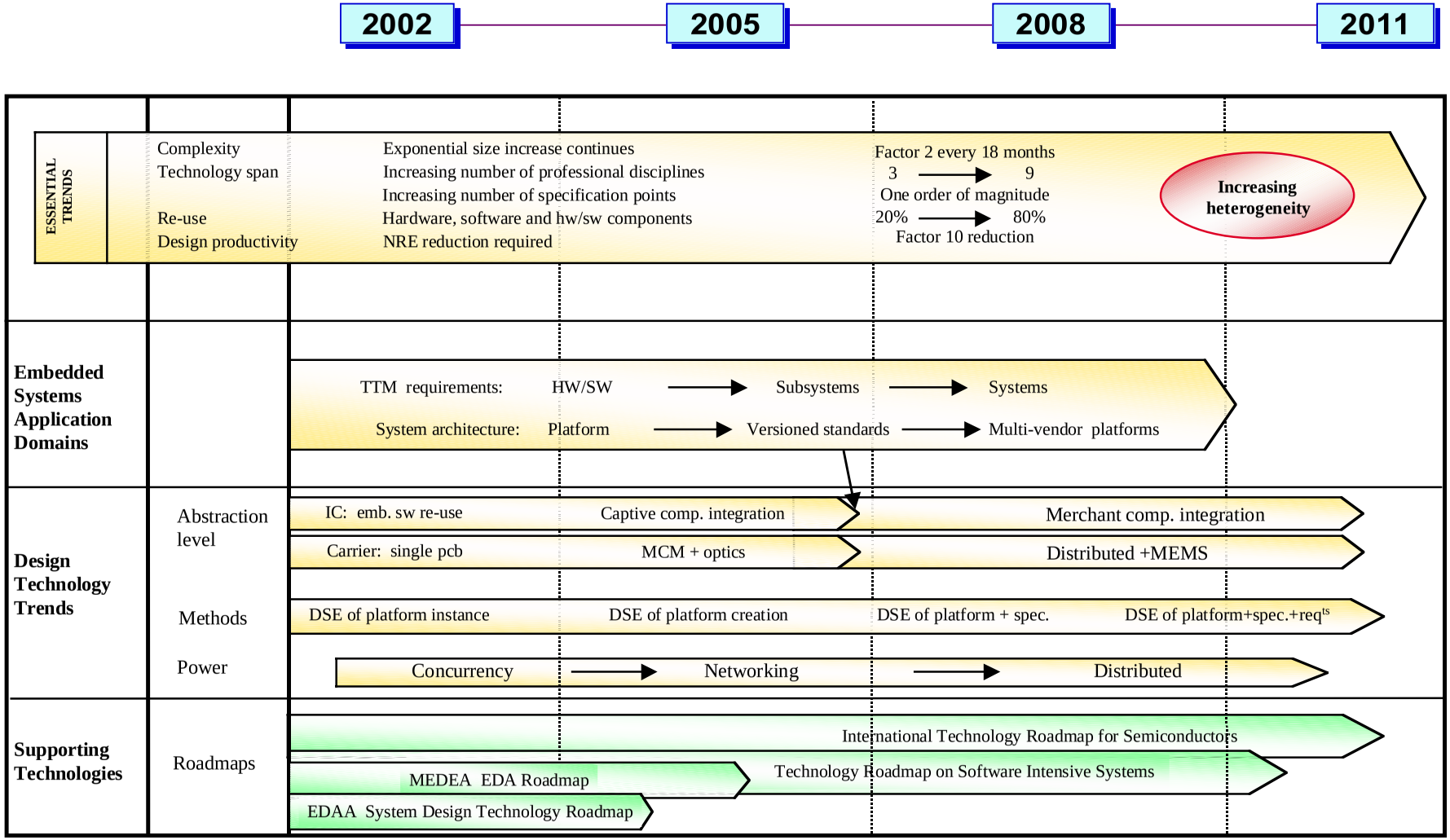
Major progress in the design of embedded systems has to come from the future evolution of design space exploration (DSE) methods and tools. Initially their scope will be restricted to design and evaluation of instantiations of a platform. But soon DSE has to be available in the platform creation phase, thereafter to be

extended to the design phases where specifications are written, and ultimately also playing a role in the early evaluation of the consequences of embedded system requirements.

An overriding design constraint has become power consumption of embedded systems. Major developments here should support the power analysis and minimum dissipation architectures for the construction of embedded systems that operate concurrently, later on to be followed by support for analysis and optimisation of power dissipation of networked and distributed embedded systems.

For required developments of enabling and supporting technologies reference is made to available roadmaps.

EMBEDDED SYSTEMS ROADMAP: GENERAL ASPECTS



2.2 Interaction

'Fewer standards, more information'

2.2.1 Introduction

The most important activity in this roadmap is standardisation. Individual sensors need specialised drivers, which may easily lead to a huge programming effort and/or an uncomfortably large amount of small IP cores. It is mandatory to limit the huge number of 'standard' interfaces (>60) to a few robust and generic ones. An IEEE committee has already started work in this area, but such is not likely to be a one-time effort as the interaction between the sensory/actuating parts and the embedded system will change with the addition of 'intelligence', i.e. the development of smart–sensors and actuators. The increased local autonomy brings self-sufficiency aspects that, together with functional redundancy, will give a data-driven broadcast nature to the co-operative communication between self-supportive parts and the embedding system.

2.2.2 General trends and user needs

The following trends and user needs relate to interaction:

Intelligence. This will lead to an increase in adaptability of sensors. Where sensors and actuators have been conceived as analog devices at the extreme ends of the information processing channel, more and more data will be processed locally by the increased use of digital techniques. Currently we see already the Virtual Peripheral around smaller analog parts, while software-driven systems (such as software radio) are just around the corner promising a factor 10 increase in product design time. With increasing intelligence, we will achieve adaptable systems that even reduce the service needs.

Service-oriented. When a service layer is added to smart-sensors and actuators, it will enable them to be re-used in a large diversity. The embedded systems will easily personalise to a changing environment. The overall system might seem more vulnerable from such a close interactive coupling between the parts, but intelligent coupling can also serve to detect internal mishaps. In case of need, malfunctioning will not be catastrophic as either the parts can be re-programmed or other parts may take over the role. This will facilitate plug-and-play and will on a longer term allow for hot swapping.

Industrial design. When more and more sensors are used in ever tinier area networks, as for instance house-holds, offices and malls, and the local intelligence grows, the role of interactive communication will grow to dominate the architecture. As fixed lines are decreasingly affordable and will be replaced by e.g. short-range wireless connectivity, the embedded function will not enforce a physical shape for the product. Form and function become separated, and industrial design will find new degrees of freedom.

Fault-tolerance. The growing market for sensors and actuators can only be created by a decrease in price. To simultaneously increase the quality, sensors and actuators should become more fault-tolerant. On the other hand, the sensing and actuating plethora will change the architecture from resource-limited to quality-driven. As the idea of ad-hoc network shows, this is based on a way of redun-

dancy that involves more than the, in retrospection, primitive N-version scheme. The quality of the embedded network (on the chip or distributed) will by definition not be dependent on the weakest node.

So far we have depicted the successive future trends as a gradual migration of concerns from sensing/actuating devices towards an integrative communication infrastructure. Therefore the sub roadmap falls apart in two parts. The first one is Interaction with sensors and actuators which consists of the sub domains Interface and Character/property. And the second one is Interaction with the communication infrastructure. This consists of the sub domains Means, Purpose and Complexity. Only some of the potential meeting points in such a scenario are shown as otherwise the picture would become too blurred.

2.2.3 Technology requirements

Standardisation of data formats and data protocols is the first important requirement. Only thereafter can collaboration of networked sensors become a reality. Standardisation will govern the availability of IP cores on a range of abstraction levels. Though for networks on chip, short range proprietary switched communication will remain to be of value, most of the interaction will become based on public packet-based standards.

The development of smart-sensors and actuators includes the local processing of data, such that data streams to controller systems will be at a higher level of abstraction. This development may impose constraints on the embedded system in the near future (e.g. analog/digital design, higher level protocol communication, etc.). By their adaptability, smart sensors will bring more freedom to the design space, but by the proper choice of platform this freedom need not overly complicate the design effort as detailed personalisation can be achieved by learning at any time. However, it does complicate the test, debug and integration issue as parts may change their behavioural details over time.

The primitive sensor will deliver only pure measurement data. At this level it is arbitrary whether such values are represented in the digital or the analog domain. The coming of digital technology implies not only a standardised protocol on the data transfer, but also the control of the collective behaviour. First such tactical control will be shaped in terms of reactivity and agility. Later, when nodal intelligence comes to bear, this will bring the communication to a strategic level. This interplay between tactical steering and strategic monitoring will create self-directive interaction along the lines of communicating agents.

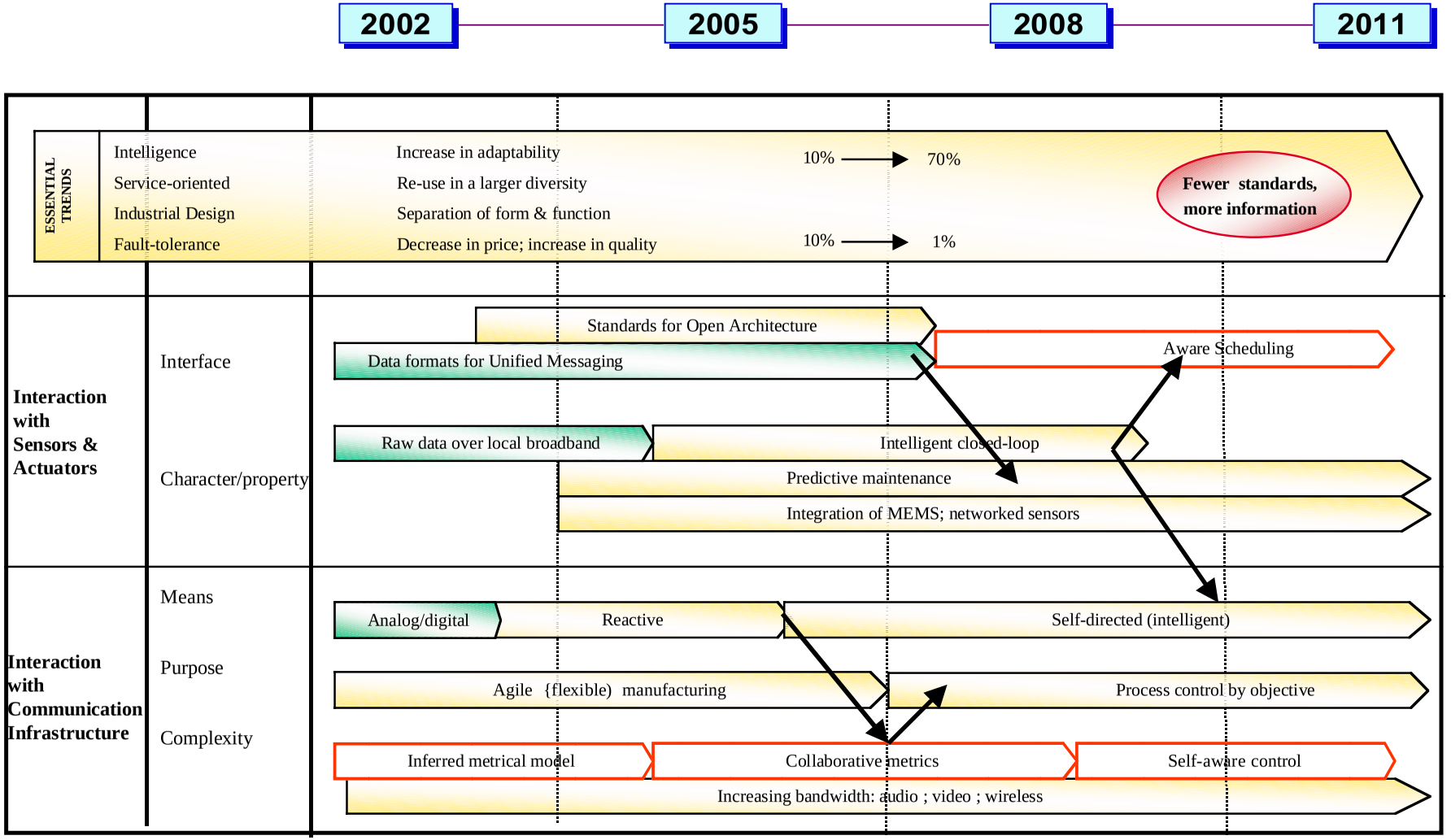
Distributed and networked sensors and actuators will start to behave as intelligent agents. The complexity of these systems will increase and without proper precautions they will require increasing bandwidth for audio, video and wireless communication. Breakthroughs in peer-to-peer communication will be required together with novel scheduling algorithms for resource-unrestricted architectures. Such local knowledge extraction gives only part of the solution, as this reduces the mere data amount, not the communication intensity. Despite the past abundant research on scheduling and control, the design metrics of collaborative adaptive systems will be in need of a long and gradual development.

Overall, there will be two aspects that may have a strong impact on the validity of the time-line as set out in the sub roadmap. The first is the assured operating conditions (as supply of electric energy). Embedding systems will become strongly dependent on the functionality of their embedded parts. More and more we see specialised systems that can not work properly without embedded systems, e.g. air planes but also combustion engines. If embedding systems become a commodity, dependability will be ruling. Assured energy supply is becoming a major factor in large-scale networks; embedded networks will not be an exception. Energy consumption comes into play to establish the integration means for the embedded parts. Where in the past decade energy consumption per household has increased with 3% per year in the Western hemisphere, this might easily set a limit to the future complexity and spread of embedded systems.

2.2.4 Recommendations.

1. Participate in international efforts to standardise sensorial interfaces. Reduce the today many sensor interface standards (approx. 60) to one or two and include in the new standards the emerging new communication protocols (CAN, fast serial, etc.)
2. Foster development of smart sensors. Improve and expand the sensor ‘intelligence’ by integrating data processing capabilities and high-level communication protocol.

EMBEDDED SYSTEMS ROADMAP: INTERACTION



2.3 Information processing

'Ever more integration'

2.3.1 Introduction

For this sub roadmap we have taken the application as starting point. Therefore this sub roadmap has the character of technology pull rather than of technology push. It describes the characteristics of information processing in embedded systems in the future; how to design these information processing parts of embedded systems is treated in the next chapter.

An overall trend is towards more integration. The scope of applications for embedded systems is broadening all the time, because of the ability to do more integration. Therefore this trend is taken as the theme and major challenge of this sub roadmap: 'ever more integration'.

We distinguish two aspects of information processing for embedded systems: behaviour and structure. For the behaviour we see a clear trend towards increased intelligence of the devices. Embedded systems will show ever more intelligent behaviour for the benefit of the embedding system and/or the user. Another clear trend is towards more mobility and connectivity. Embedded systems are envisioned to communicate with each other in order to gather information and to show more intelligent behaviour.

When we consider the structure of embedded systems, constraints are typical parameters that the designer has to deal with. On the other hand, advances in technology offer all kinds of possibilities to actually build useful embedded systems.

We have divided the area of information processing relevant for embedded systems into two parts, following the above reasoning. First we consider behaviour, and look at intelligence and connectivity as themes. We then consider structure, with sub items constraints and possibilities. First we look at the overall theme: ever more integration.

Note: an important aspect of information processing is the point of view of services. Who is going to deliver what services to which devices? Other roadmaps and research agendas elaborate on these issues, for example [Embedded Everywhere, 2002] and [Book of Visions, 2001]. In this sub roadmap we highlight some important trends of information processing in the embedded environment, without the claim of being complete. We will not elaborate on the issue of services in this sub roadmap. Also aspects in the legal area, like (copy) rights on content and software, and digital rights management (DRM) are not considered here.

2.3.2 General trends and user needs

For *integration* we see the following trends:

- The integration is visible at the technology side: ever more electronics, hardware and software, and sensors and actuators are integrated into one single unit that implements the embedded system. Integration with storage of data (for example optical storage and hard disks) and database functionality also takes place.

- Integration of embedded functionality takes place with other (non-technical) environments, like clothes, diapers, etc.

2.3.3 Technology subdomain: Behaviour

Essential trends

For the *intelligence* of embedded systems, we consider the following trends:

While systems show re-active behaviour nowadays, they will show active and even pro-active behaviour in the future.

There is a strong trend towards personalisation of devices. For example, there are hard-disk VCRs on the market that construct a personal profile of the user and start recording those broadcasts that fit into that profile. Numerous embedded systems in other application areas can be listed that have some capability of ‘learning’ or ‘intelligence’ in this respect. The number of personal parameters will grow.

Considering *connectivity* and *mobility*:

There is a trend towards ever more functionality both in base-stations as well as in terminals. There is no clear shift between the functionality on the one side or the other; it depends on the constraints on the implementation (e.g. power dissipation) and the bandwidth of communication between the two where the functionality will be implemented. Dynamic scheduling of computation over base station and terminal will become necessary to better utilise the system's capacities. This has consequences for the services for terminals, for example the question who will provide a service on the move. In this sub roadmap we will not elaborate on the issue of services, but we refer to other sources, e.g. [Book of Visions, 2001].

Technology Requirements

For the *intelligence* part of behaviour we have the following:

User interfaces will be a driving factor for functionality of embedded systems. While we currently have to limit ourselves to the keyboard, speech and sound recognition will become more and more used to communicate with the (embedding) system. A lot of processing is involved in getting from the recognition of simple commands to connected speech and even (natural) language communication. Friendly user interfaces are key for disabled people to work with devices containing embedded systems. One can think of gesture recognition followed by animated sign language for deaf people, for example.

For the audio domain a lot of processing will be involved in the interpretation of sounds. It is then a gap to identify and implement useful ways to use the resulting information in the system. Sound recognition might be a next step. The technologies developed in the audio domain are key to the aspects described in the user interface domain. Sound interpretation is essential for connected speech, while sound recognition is a prerequisite for doing language recognition in the user interface.

In the video domain we can also distinguish steps in functionality. While we can currently detect movement in a video stream, it is not yet feasible to do proper localisation of objects and people. Interpretation of images is a capability that is even further away in the future (after 2011). These technologies are important before we can do gesture recognition and animated sign-language in the domain of user interfaces.

Another strong activity in the video domain is moving from 2D to 3D. Already a lot of research is going on in this area. Examples are 3D reconstruction from 2D images and video streams, and stereo vision. 3D will be a next step in the area of entertainment, both in graphics (games) and video. Further, trends towards integration of these two domains are starting, as we can see from standardisation efforts like MPEG4.

Considering *connectivity*:

Devices will become more and more connected in some kind of network. As a first step we envision that devices will detect which other devices are nearby and can be of any use, for example in getting information or delivering a service to the embedding system or the user. In this first step the device can construct a personal profile of the user. Striving for more intelligent behaviour, we need agents to talk to each other and negotiate in a next step. For this, communication of the embedded system to the embedding system and to the environment is necessary.

Closely related to networking devices, they need to access information available in the network to do their job. While embedded systems nowadays access local data, in the future this data might be retrieved from elsewhere. Web-connectivity is envisioned as a strong driver to get information from a global network. This strongly relates to the negotiating and independent agents mentioned previously.

In the area of transactions, for example in banking and money transactions, we need standards that have to be subsequently implemented in systems. These systems will contain a lot of embedded systems. They will influence the people's lives considerably.

Considering a network of connected devices, information management in the network will become a serious issue. As a first step data will be migrated through the network depending on the need of the various embedded systems. Migrating the application over the network is a next challenge to be solved in striving for more efficient information processing of embedded systems.

2.3.4 Technology subdomain: Structure

Constraints are a key characteristic of embedded systems. The complexity and functionality of information processing in embedded systems are limited by constraints on the implementation of these systems. On the other hand, advances in technology allow for possibilities in building complex embedded systems with new kinds of functionality.

Essential trends

Much of the functionality of embedded systems will be implemented in hardware and software. Therefore constraints on technology side of hardware and software will heavily reflect on trends in embedded systems. For hardware we have the usual parameters like chip area, cost, and power dissipation. As more and more software is incorporated into embedded systems, also software will claim influence on these parameters.

For *constraints* we see the following trends:

Cost is an important parameter. In consumer electronics, for example, chip prices in the range of a few dollars are common. These prices tend to stay the same or even decrease over time.

The chip area will be approximately constant, but as more transistors can be implemented in 1 cm^2, more functionality can be implemented using such chips. The increase in software is reflected in more memory embedded in the system, both for the program as well as for the data. Currently, a mobile phone contains a few mega-bytes of software, and this amount is expected to increase over the coming decade.

The power budget of such hardware also tends to stay the same over time, but again as the functionality increases, we have more computational power per Watt. For wired applications this is in the order of Watts; for mobile battery-powered applications in the order of milliWatts; and for devices without a power supply (e.g. smartcards) in the order of microWatts. Both the hardware and the software influence the power dissipation in an embedded system. Here we encounter the traditional trade-off between hardware and software: doing more in dedicated hardware will result in a strong reduction in power dissipation at the cost of flexibility of design. As the sub roadmap on Hardware/Software Design will explain, the future use of reconfigurable hardware will offer a new dimension to this trade-off, also to designers who do not have the facility to make their own chips and hardware.

Transistor technology is not enough to bring the desired computational power; therefore clever hardware and software techniques need to be researched. The embedded systems design roadmaps highlight these aspects. For information about transistor technology we refer the reader to the International Technology Roadmap for Semiconductors [ITRS, 2001].

Bandwidth for both on-chip and off-chip communication will increase in the future. For example, for wireless communication in mobile telephony the bandwidth will increase from about 10 Kbit/s to the order of 10 MBit/s. A characteristic of applications is that they will always use the available bandwidth.

Backward compatibility with other systems has always been an important constraint, and will remain so, as we build up more and more legacy.

Considering *possibilities*:

A glimpse of possibilities offered by embedded systems in the future is already shown in the section on behaviour.

Display technology is an important driver in realising embedded systems everywhere. In this roadmap we do not discuss a roadmap for display technology but we refer to the ITEA roadmap to obtain more information on this subject [ITEA, 2000].

Technology requirements: Constraints

Energy supply is key for the widespread usage of embedded systems. Battery-operated devices tend to become more and more energy-consuming, while the roadmap for batteries shows only a few percent improvement per year, which is by far not enough to supply the ever more complex operation of embedded systems. Therefore we need to look at other energy-sources as well, like induction, fuel cells, and solar energy.

Electro-magnetic radiation of electronics will increase when the functionality and complexity increase. This poses two problems. First, we need to shield some embedded systems from others, because the operation is disturbed by the radiation (EMI, electro-magnetic interference). Shielding is only partly a solution. Therefore we need to research embedded systems and devices that have low electro-magnetic radiation. We do not know how to do this; therefore it is considered a white spot. Asynchronous hardware (i.e. hardware without a central clock) is known to deliver some solutions in this area, and might be an interesting avenue for research. Apart from this technical aspect, the implications of radiation for human health are unclear, and therefore considered with scepsis by society. It is a governmental task to put standards in this area.

Dependability is an important issue. Fail-safe operation, robustness of systems etc. can be key constraints on the design of embedded systems. A failure in the controller of a toaster, for example, is not allowed to leave the system in heating state after failure. One can imagine a wide spectrum of issues that are relevant in the dependability area.

Technology requirements: Possibilities

For the hardware in embedded systems we see ever more integration: from multi-chip solutions, via the integration of smart sensors (it is considered a gap how to do this), to single chip solutions. Even MEMS comes in sight for integration into embedded systems. The sub roadmaps on embedded systems design highlight the design issues for this kind of hardware and integration.

The data that the embedded system processes will have to be stored somewhere and communicated to and from the system. Databases are becoming more and more common in embedded system design. Currently we have hard disks and solid-state storage in our devices. Optical storage is a cheap way to distribute large amounts of data or content. Small and cheap optical storage is a promising alternative for bulky storage and is appropriate to distribute content, but we need standards for this kind of storage. This is considered a gap to come to the integration of optical storage in embedded systems. From the technical point of view, small discs require energy friendly hardware and software, especially in the portable domain. Of course the development of solid-state memories and hard disks will improve considerably.

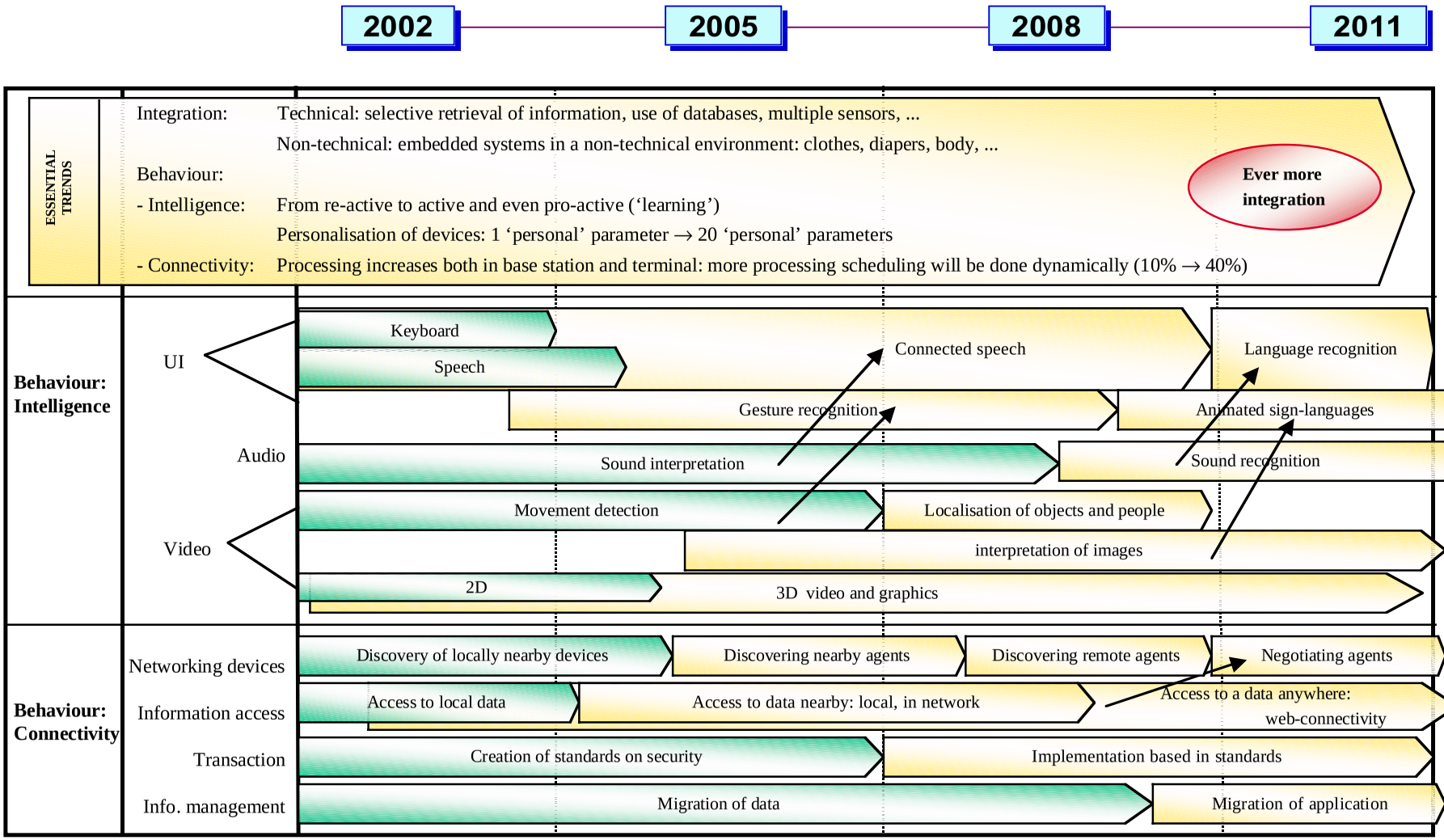
For components in embedded systems we will see a trend towards constructions of components with reuse as a means to achieve higher efficiency. This is described extensively in the sub roadmaps on embedded systems design.

Efficient algorithms will become necessary to allow for the envisioned intelligent behaviour of embedded systems. Though a lot of research in this area has been done already, we need to make the step towards applying the results in the context of the constraints of embedded systems.

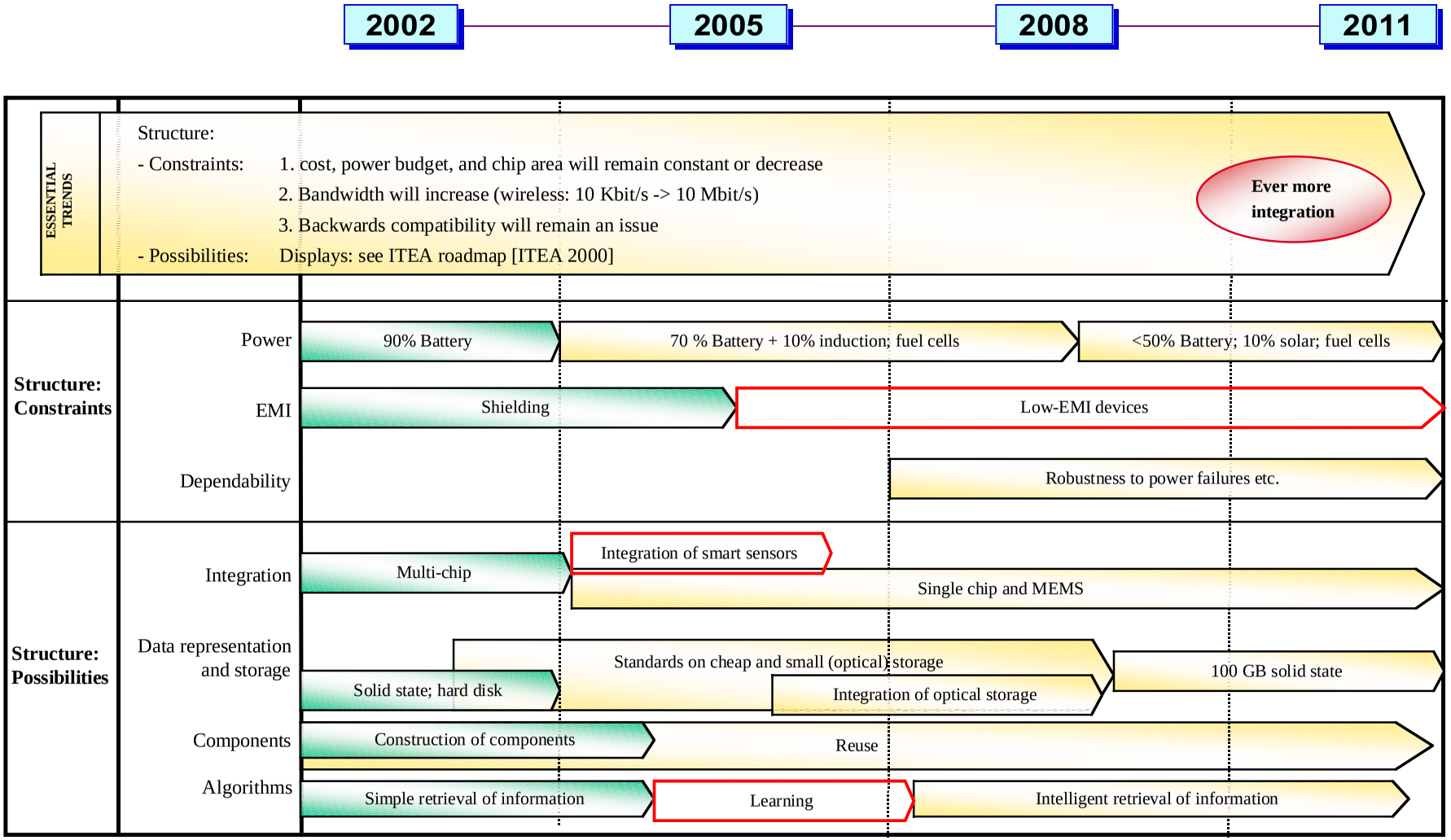
2.3.5 Recommendations

1. Power is considered as the most important constraints in embedded systems. Therefore we have to use alternative power sources besides batteries, and alternative ways of charging the batteries besides direct wiring to the power grid. The reason for the former is that batteries have a limited capacity and may contribute considerably to the weight and volume of the apparatus. The reason for the latter is the lack of access to the power grid in many circumstances (e.g. mobile), and the trend or wish to charge the batteries invisibly to the user. These alternative power sources and charging methods may derive from movement, solar energy, electromagnetic induction, etc.
2. We envision the use of portable embedded systems connected to a base station. Many processing tasks can be either performed by the portable device or the base station. Because the portable device may be subject to extremely low power restrictions, the allocation of processing is essentially determined by the energy consumption (on the side of the portable device) of the processing vs. the communication of data to and from the base station. Because the latter changes under different circumstances (e.g. distance to base station), the allocation of processing should be performed dynamically.
3. When the portable device is retrieving information from a (distant) source (e.g. a database), the selective and intelligent processing can be performed as a service by the source itself, rather than the power constrained portable device. Research in this kind of information management is necessary to get the best out of the embedded systems of the future.

EMBEDDED SYSTEMS ROADMAP: INFORMATION PROCESSING 1



EMBEDDED SYSTEMS ROADMAP: INFORMATION PROCESSING 2



3 Embedded systems design

3.1 From idea to executable specification

'To design the right product'

3.1.1 Introduction

A crucial and challenging aspect of embedded system design is the steadily rising internal and external complexity. The embedded system has grown considerably since the early days of the simple micro-controller board, but it has also diffused into a multi-disciplinary world. This intimate coupling to other disciplines, who often defy a precise mathematical modelling, and the restrictive operating conditions that should be derived from this environment make it hard to establish a specification that can serve as input for a subsequent transformation to a suitable implementation.

This makes it mandatory to pay special attention to tools and techniques that can support a designer to think and communicate about the problem at hand in a firm effort to derive an agreed starting point for the remainder of the system development. More often than not, getting from idea to executable specification has been an art rather than a craft in the past. In other words, we are looking at unchartered territory.

In order to be able to profit from more professional discipline, it is proposed to explore several directions. First of all we need better metricity to quantify alternative specifications. Second we have to create modelling techniques that allow to study a proposed embedded system within its future environment. Lastly we need design space exploration styles that allow for requirement analysis in a mixed technology framework.

3.1.2 General trends and user needs

As stated above, almost all products, whether consumer or professional, tend to become increasingly complex. This is, in the first place, due to the general belief that there is a need for more sophisticated products, and that technology is fairly well capable of dealing with this increase in complexity. The growth in complexity may originate from requirements related to performance, quality, accuracy, safety and the like, or from an increasing demand for more functionality, a larger variety of technologies and related disciplines, or a wider domain of applicability.

The complexity inflation finds expression in the fact that products are conceived as embedding systems that are compositions of embedded systems. The TV receiver of tomorrow will contain embedded systems that the receiver of today does not contain, turning the passive device of today into the (inter)active one of tomorrow. This is just one example, and many more could be given. If this trend goes on, and it will, then the design of an embedded system will become as challenging as was the design of a complete system in the past. And surely, the design of a complete embedding system with all its embedded subsystems makes the designers face many tough problems.

Now, given the optimistic attitudes at both the demand and supply sides, it is predictable that more and more ideas for more and new functionality in new and existing systems or products will be proposed in the years to come.

An idea is the expression of a concept, often stated in non-technical terms e.g., 'We want to charge cars for the use of the public road infrastructure' or 'The remote control of our TV receivers has to become more user friendly'. Such an idea has to be translated into a set of requirements. Some of these requirements have to do with functionality and others have to do with conditions and constraints. The requirements may very well be incomplete and even conflicting as e.g., privacy and fraud resistance are in a smart road system. Requirements capturing and requirements analysis will remain a major challenge in this phase of the design of embedding and embedded systems alike.

Designing and constructing small scale prototypes may be useful but will in general not be scalable as the large scale system may behave differently than an upscaled prototype. Similarly, the large scale system may have to reuse components that do not appear as such in the prototype. Therefore, a trend in the design of an embedded system is to see the translation from requirements to specifications as part of the design process: from idea to specification, followed by from specification to implementation.

The first part: from idea to specification has, of course, been always part of a design trajectory; however, the intuitive and ad hoc approach that has been commonly taken so far will no longer do for several reasons. Firstly, increasing the complexity of a system must not lead to an increase in design cost. This condition cannot be satisfied when the design is based on repetitive prototyping because prototyping is expensive. Secondly, if specifications are not the result of a well-defined and systematic approach, then there is no guarantee that the resulting systems will obey the initial requirements because the specifications may already fail to do so. Thirdly, even an expert who has been deriving specifications for many years may fail to see all relations between requirements and specification, and even more so between requirements and alternative specifications. At least traceability between requirements and specifications has to be provided to make relations explicit. But that is certainly not enough.

Thus, deriving specifications from requirements in a sound and methodological way is sort of an emerging discipline that has to be given much attention in the coming years.

3.1.3 Resulting technology requirements

Deriving specifications from requirements is a process that, like any other process has inputs, outputs, and state. The process itself is specified in terms of relations between outputs and current inputs and state, and between next state and current inputs and state. The process at hand here is iterative and exploratory by nature. The relations between input, output, and state quantities are extremely difficult to qualify and quantify. It is, therefore, necessary to rely on models. One can even conceive of two models: an unconstrained model and a constrained model. The unconstrained model consists of a chain of abstract components and is a means to express the requirements in such a way that one can reason about them, that is,

that the decision making can be supported. This implies that dependency and sensitivity analysis of the model must be possible, locally and globally, at component level, at chain level and at the level of emerging system properties. The constrained model is less abstract in that it incorporates two kinds of requirements that the abstract model doesn't: component (re)use requirements and requirements imposed by the second step in the design process, that is, the specification to implementation step. A specification process that is based on these two models will also have to have a method for the mapping of the unconstrained model into the constrained one. For this approach to work, the specification process and toolbox must have access to a component library. Moreover, components can only be included in the library when they are specified and described in such a way that their properties can be imported in the model with a high degree of (blind) confidence. Hence, the formalising of component properties is something that has to be done, if not already done. Once the library is available, components may be easily imported. However, it is likely that more than one component in the library is a candidate that can be imported. Therefore, a method must be provided to search for a set of components that are jointly somehow a best choice with respect to the fulfilment of (emerging) system properties.

Now, the specification-deriving process being an iterative exploration process, means to quantify decisions must be provided. Formally as well as practically, this means that metrics-based analysis and exploration is the preferred approach. It is, however, not known what the structure is of the metric space of embedded systems. As a consequence, no metrics are known and hence the usage of metrics is not common practice. This is a major drawback that must be resolved in the first place and with high priority.

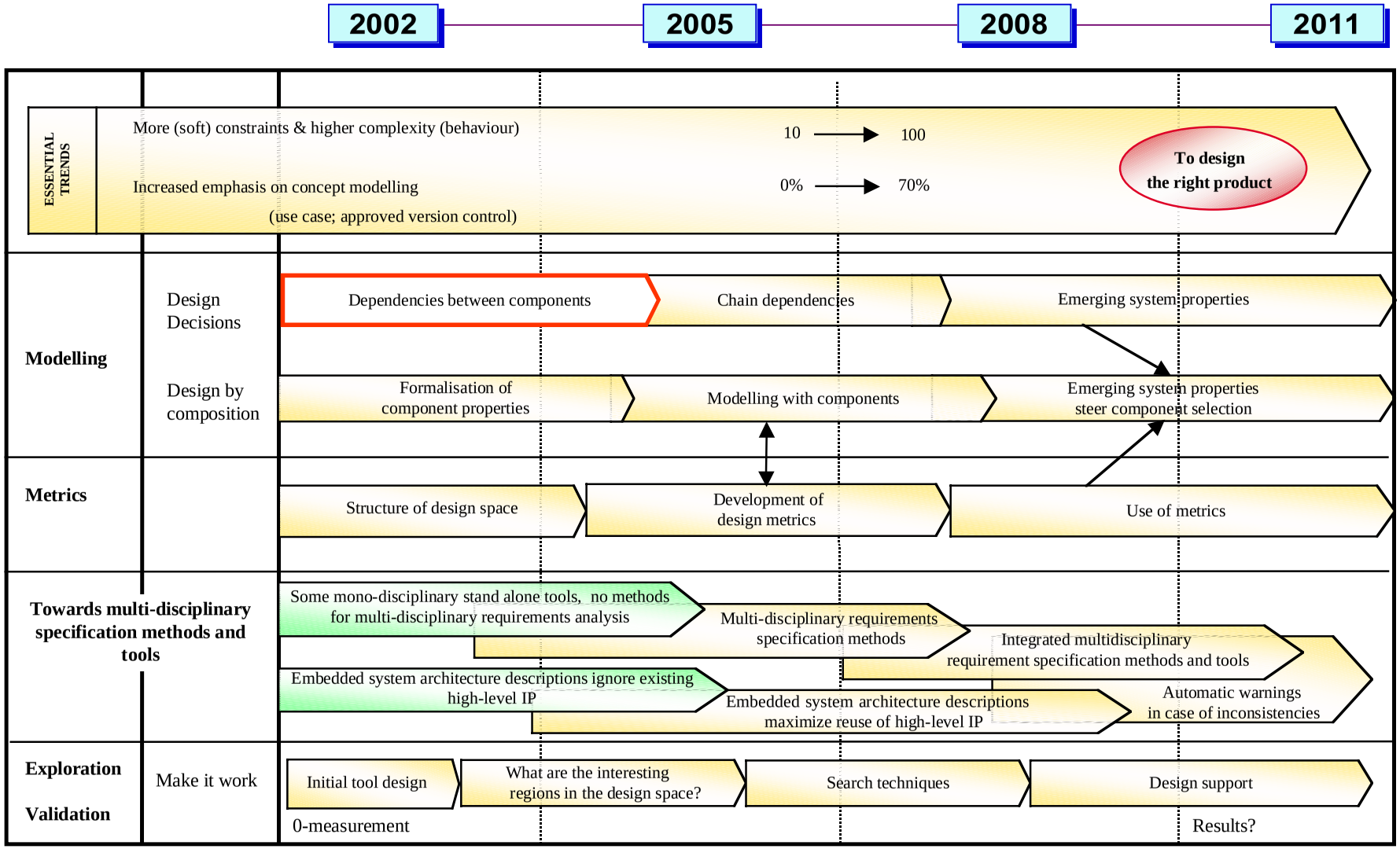
Recalling that the translation of requirements to specifications is an iterative exploration process, this process has to be implemented and it has to be made to work. This implies that exploration tools have to be designed, that candidate specifications in the specification space have to be identified, that search techniques have to be developed and, of course, that all of this has to be made effective to support the specification task efficiently.

3.1.4 Recommendations

The idea-to-specification roadmap depicts the technologies required to implement the following recommendations that have been identified for future R&D programs:

1. Propose research programs to resolve the specification problem of embedded systems by
   - relating emerging system properties to individual system components,
   - quantitatively evaluating specification decisions based on specification space metrics,
   - obtaining specifications through iterative exploration of the specification space.
2. Propose a validation program to evaluate the impact of the proposed methodology.

EMBEDDED SYSTEMS ROADMAP: FROM IDEA TO EXECUTABLE SPECIFICATION



3.2 From executable specification to implementation

'Mapping behaviour on hardware'

3.2.1 Introduction

The design flow from idea to final product design can be divided into two parts, the first being the flow from idea to some form of an executable specification and the second being the flow from executable specification to the final design. Using the same term 'executable specification' in both parts of the design flow does not mean that these executable specifications are based on the same model and are described at the same level of abstraction or detail. The design flow from idea to executable specification is elucidated in section 3.1; the design flow from executable specification to implementation is the subject of this sub roadmap. A clear border between these two parts of the design flow does not exist since pure top-down design is not possible and there always exist a strong interaction between these two parts of the design flow. In this sub roadmap, an 'executable specification' is an executable specification of (a part) of the behaviour at some abstraction level. In practice it will be annotated with additional design constraints like timing, power, throughput, etc. The level of abstraction and detail can vary over a wide range. For example, it might be an executable Petri-net description, in which only the communication between the different modules from which the system is built, is described. On the other hand it might be a detailed VHDL description that describes the behaviour in full detail while providing a strong implementation suggestion. In practice the first executable specification in the design process will never be complete. Hence, although the design flows described in this section and the design flow described in section 3.1 are quite different, there always will be a strong interaction between these design flows.

In principle, an executable specification describes the desired behaviour and therefore the design flow from executable specification to implementation can be summarized as the challenge 'mapping behaviour on hardware'.

3.2.2 General trends and user needs

Tools and representation

According to Moore's law the complexity of ICs in terms of the number of transistors on a chip, will double every 18 months. The production cost of ICs per mm^2 will hardly change. So in order to keep the design cost reasonable with respect to the production cost, the design cost per transistor will have to decrease with a factor two every 18 months. For ICs that have a high degree of repetition, like memories, this can be accomplished. But, for complex systems we will have to count on improved tools that semi-automatically translate (compile) the desired behaviour into the final design. This often will be on the account of less efficient implementations. So tools for automatically mapping behaviour on hardware will become extremely important. Moreover, from the preceding remarks we may conclude that ever more systems will be implemented on programmable and reconfigurable hardware.

Furthermore, in future it will become impossible to guarantee the correctness of a design by just simulation, hence verification and validation of the design in the various stages of the design flow will become ever more important, cf. the sub roadmap on verification and validation, section 3.5. Furthermore, much more attention is to be given to the correctness of the design tools, formalizing the design process and improving the simulation tools. All three themes need attention in order to keep up with the increasing size and complexity of future designs.

Correctness of the design tools

When we start from the assumption that the executable specification correctly reflects the desired system behaviour, then the correctness of the final design only depends on the correctness of the automated and manual design steps in the design process and the consistency of the design flow.

Correct tools transform, optimise or refine an intermediate design description such that the behaviour of the resulting design is implied in the behaviour of the original design. Hence, the correctness of the result need not to be verified by simulation and thus, correct tools lessens the simulation burden. How the correctness of these tools is obtained, by formal verification or by exhaustive testing, is not important. The only thing that counts is whether the tools can be trusted. For example, an ordinary C-compiler is correct; it is trusted although it has not formally been verified. Clearly, validation of tools could be more expensive than validation a design or verifying a design step. However, tools have to be validated only once and each design must be validated separately. So, eventually validating tools is cheaper. Furthermore, tools are validated naturally during its extensive usage, as is the case with the C-compiler mentioned before.

For the user it will be impossible to assess the tools that are on the market beforehand. So these tools need to be certified. If an international organisation for the certification of design tools is started in the next few years, it must be possible to have 30% of the (basic) design tools certified within ten years from now. Although we think that this is possible, we fear that it is unlikely to happen because of the structure of the CAD market, the complexity of the tools and the desire of the users to use the latest tools.

Correctness of the design flow and the design representations

Due to the ever-growing complexity of the designs more emphasis will be on the correctness of the design flow, on the capabilities of the design representations and on the correctness of the semantics of these representations and therefore on the formalisation of the entire design process. This in particular holds for the design flow from executable specification to the final design. The different steps in the design flow should be fitted together in a more appropriate and less error prone manner. Manual or ad-hoc translations of representations cannot be accepted any more in the near future.

Specifications and design representations need to be augmented with means to express properties that go beyond behaviour and structure. These languages must make possible to express different time models and quantities such as real-time constraints, throughput, power dissipation and required or available silicon area.

These developments will put more emphasis on the formal semantics and models on which the design flow and the design representations are founded.

Moreover, at the level of executable specification (system level) we need an integral representation of physics, mechanics, etc. together with behaviour and structure.

Simulation and emulation

Simulating or emulating the embedding system on the, or in the embedded system will become a necessity. Examples are a virtual reality model of the embedding system in which the executable specification of the embedded system is included, or for example a simulation environment that simulates the specification of a processor and emulates the operating system mounted on it.

Compilers

The complexity in terms of the number of gates available for a design will continue to grow with a factor two every 18 months. Moreover designs will become more ‘difficult’ due to the increasing technological possibilities.

In order to keep design cost in proportion to the production cost, the design cost per transistor must follow a negative exponential curve as has been explained in the introduction to this sub roadmap. Moreover more designs will have to be made with roughly the same amount of designers. High-volume production will be needed and thus in future there will be fewer different ICs, which however, will be more programmable and will become reconfigurable. This all will require further automation of the high-level synthesis process, design re-use and standardisation of the target architectures. Therefore, we foresee an increased usage of compilers in its widest meaning. Notice that when we talk about compilers in this sub roadmap, we do not only refer to the classical compiler that translates some language into object code, but also to any translator that translates one representation into another representation including synthesis and optimisation. But, a compiler always operates on languages that express some form of behaviour (function or algorithm); this behaviour is preserved in the compilation process.

The classical compilers, which map on a fixed architecture, will become more important. In particular compilers that optimise on the basis of different criteria and compilers that map on new classes of architectures such as VLIW and reconfigurable architectures will be needed. Retargetable compilers, which start from a parametrisable architecture, i.e. a class of architectures, will become mature for the classical architectures and compilers for reconfigurable architecture will leave its research stage. The ultimate goal is a compiler that automatically maps the behaviour expressed by an executable specification, on hardware such that the compiler determines both hardware and software.

Ten years from now it will be possible to automatically derive an optimising compiler from the architecture description.

Specifying behaviour by means of an imperative language or some functional language does not suffice. Currently, all kinds of specification methods are developed based on a model of concurrent processes, for example a Kahn model in which the processes are described in C. We foresee that compilers will be needed

that operate on these higher levels of behavioural description. For instance, interactive compilers that support the composition and decomposition of these processes and are able to map these behavioural descriptions on heterogeneous, possibly reconfigurable, architectures.

The requirements for a language in which an executable specification can be expressed are often conflicting. User friendliness for the specifier often means inefficient execution. Therefore many compilers are needed that translate well-readable executable specification into a specification that executes efficiently. Typical examples are:

- (Inefficient) SDL to efficient C.
- (Inefficient) MathLab to efficient C.
- UML-RT to efficient C++.

3.2.3 Technology requirements

Tool assessment and certification

Assessment and certification will be an essential part in solving the validation (simulation) burden. Assessment of tools will be an expensive process. Therefore it can only be done by an international organisation supported by the tool providers and tool users. The initiative should be taken by industry; universities can give support by studying the assessment process. Raising such an international organisation will take several years. In order to gather experience, the first assessments should be tried out on existing well-know tools. Thereafter newly developed tools can be taken up. Unfortunately, the structure of the CAD market, the complexity of the tools and the desire of the users to use the latest tools make successfully setting up such an assessment organisation rather unlikely.

Design flow representation and formalisation

The design flow will become increasingly automated and therefore increasingly dependent on its correctness. This can be solved by formalisation of the design process and the development of suitable design languages. Furthermore standardisation of tools and languages will contribute to the correctness requirements.

Further developments in the area of design representations (design languages) will be needed. Currently, no design language or representation does exist that can be used throughout a large part of the design process. Many tools use a different representation, which requires a lot of ad-hoc or even manual translations. Furthermore, the current design languages are not able to express design constraints such as real-time, required throughput and power dissipation. In the end, such design languages should contain constructs that make it possible to extract architecture parameters from a design description.

In the first place, design-languages need to be developed that are able to model different aspects of time and are capable to express real-time constraints. At the same time research can be started in which other design constraints can be expressed such as throughput, power, etc. Prototypes can be expected after six years from now.

Typically for embedded systems there is a need for design representations that not only express behaviour, structure and geometry, but also are able to provide a representation in which mechanical and physical behaviour are integrated.

Modelling the embedding system

Often, an embedded system is correct in relation to its specification. However, embedded in its embedding systems it finally turns out to be incorrect. So the specification of the embedded system was incorrect. It is therefore important to have tools in which the specification of the embedded system can be simulated as a part of the embedding system. This is already common practice in the automotive industry. A simulation environment that supports simulation of both the embedding system and the embedded system will be needed.

Moreover tools are needed to emulate efficiently complex software on a hardware oriented design description.

Compilers

A highly automated design process will be needed in order to keep the design cost in proportion to the production cost. For this reason, better compiler techniques have to be developed.

We distinguish between classical compilers, retargetable compilers, compilers of the second kind, compiler generators (common knowledge) and compiler generators of the second kind.

Notice that our emphasis is not on the front-end of the compiler but instead the back-end.

Classical compilers map behaviour expressed in some language on a fixed architecture. The complexity of these classical compilers depends on the kind of target architecture, such as single-processor, VLIW and reconfigurable. Compilers for VLIW need further improvement and compilers for reconfigurable architectures are still in its infancy. Moreover, these classes of compilers should be able to optimise on the basis of different criteria, such as code size, throughput, power consumption, real time constraints, etc.

'Locality' is a relatively new constraint for optimisation. In the current and future IC technology, delay and power are no longer determined by the standard cells but instead by the length of the interconnect. Moreover, in the near future it will take many clock cycles for transmitting a signal from one side of the chip to the other.

Retargetable compilers get as input a specification of the behaviour, expressed in some language (the source code) together with a description of the architecture. Current retargetable compilers can only manage a very small class of architectures. The distinction between the different architecture is given by only a number of parameters. Based on the source code and the description of the architecture, object code is generated that is optimised according to some of the criteria mentioned above. Future retargetable compilers should start from larger classes of architectures and eventually from an arbitrary architecture.

Compilers of the second kind start from a specification of the desired behaviour (expressed in some language) and one or more optimisation criteria. The compiler delivers both the hardware design and the object code. In these compilers the high-level synthesis is fully integrated in the compilation process.

Compiler generators of the first kind start from a syntax description of a language and derive a compiler from it for a fixed architecture. These compiler generators might be considered as common knowledge. However in the area of optimisation still a lot of work needs to be done.

Compiler generators of the second kind start from a description of the target architecture in terms of hardware resources, interconnectivity, instruction set etc. and derive a compiler from this data for a fixed language.

All the compiler types that are mentioned above will be needed.

The development of these compilers is mutually dependent and partly depends on common techniques of which we will mention a few.

A suitable representation of the target architecture will be needed for retargetable compilers, but also for compilers for VLIW in which the number of resources may be parameterised. Clearly, a suitable representation of the target architecture will be needed for compilers of the second kind.

In most architectures, similar plural resources will be available, hence combined resource allocation and scheduling will become a challenge.

Techniques to derive from the desired behaviour and the optimisation criteria the architecture parameters, such as which and how many functions are needed, the number of registers required, the required busses and even the choice of the optimal architecture, need further research and development. These techniques are needed for controlling the design process and will be part of compilers of the second kind.

3.2.4 Recommendations

1. Start an international organisation for the assessment and certification of design tools.

   Incorrect tools require that the results of these design tools need to be verified by means of simulation. Due to the ever-increasing complexity of the designs this practice will soon become infeasible.

   Correct functioning design tools make lengthy simulations superfluous and increase the reliability of the final design. Correct functioning design tools makes possible ‘Correct by construction’. The assessment of design tools should be based on testing and on formalisation of the underlying models.
2. More attention is to be spend on research on languages for- and representations of specification and design.

   Currently many specification languages and methods are available for expressing behaviour, however, only few are able to express time in an appropriate way and we are in general unable to express design constraints like real-time requirements, power requirements, area, etc.

   Different tools used in a design flow often have different representation formats with unclear semantics. A common representation format (language) based on clear semantics and a model that relates the language to the presumed

reality will considerably improve the quality and the efficiency of the design flow.

It is important to get involved in international high-level language standards working groups.

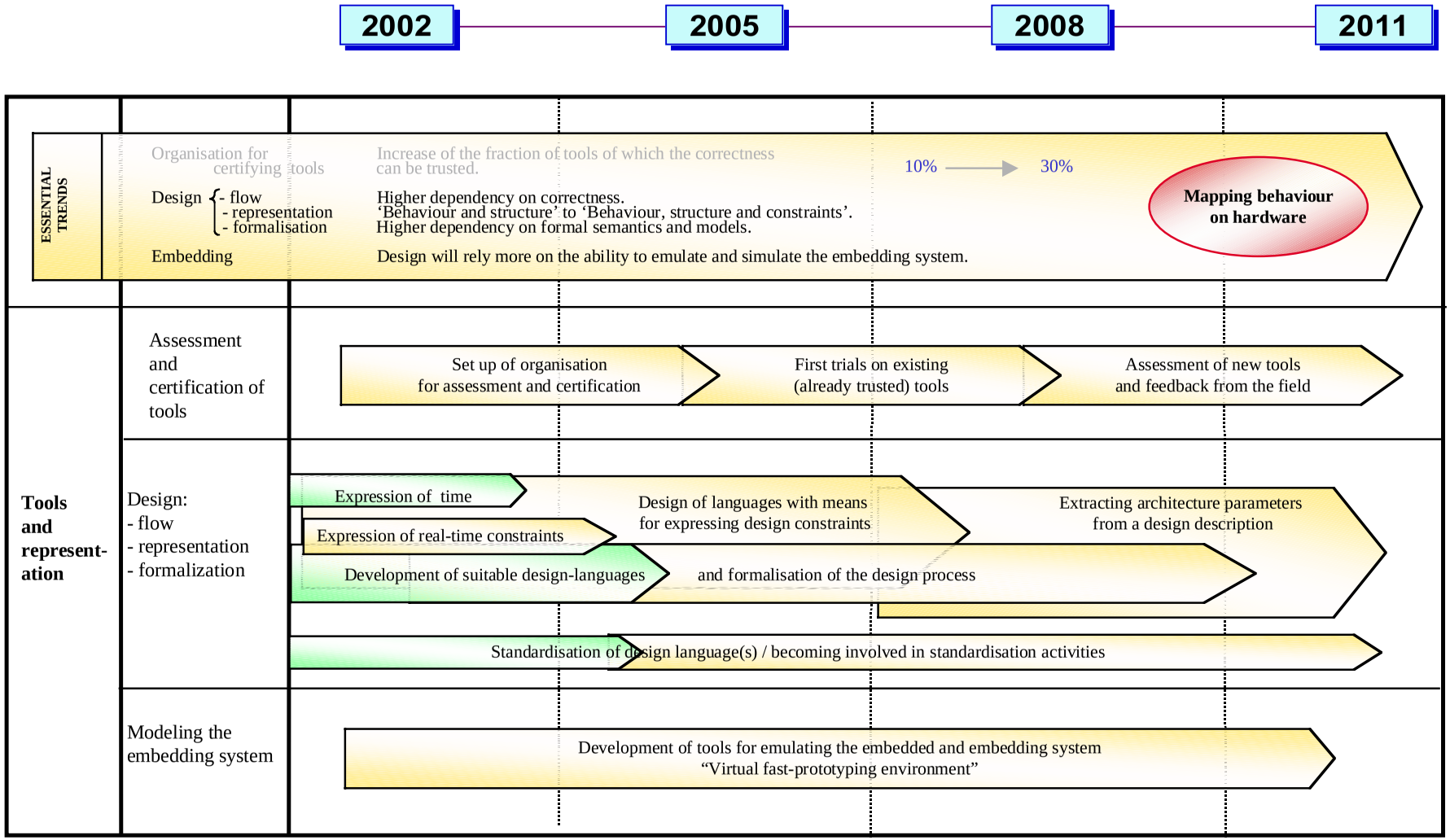
3. Research on compilers, translators and compiler generators must be intensified. Due to the new architectures, such as VLIW and reconfigurable architectures, there exists an urgent need for better compilers, translators and compiler generators

   There is a need for two kinds of compilers. The first kind is retargetable towards various hardware designs in order to deliver code efficiently executing on the programmable blocks in the design.

   The second kind of compilers derives the architecture from the behavioural specification based on some criteria and simultaneously generates the executable.

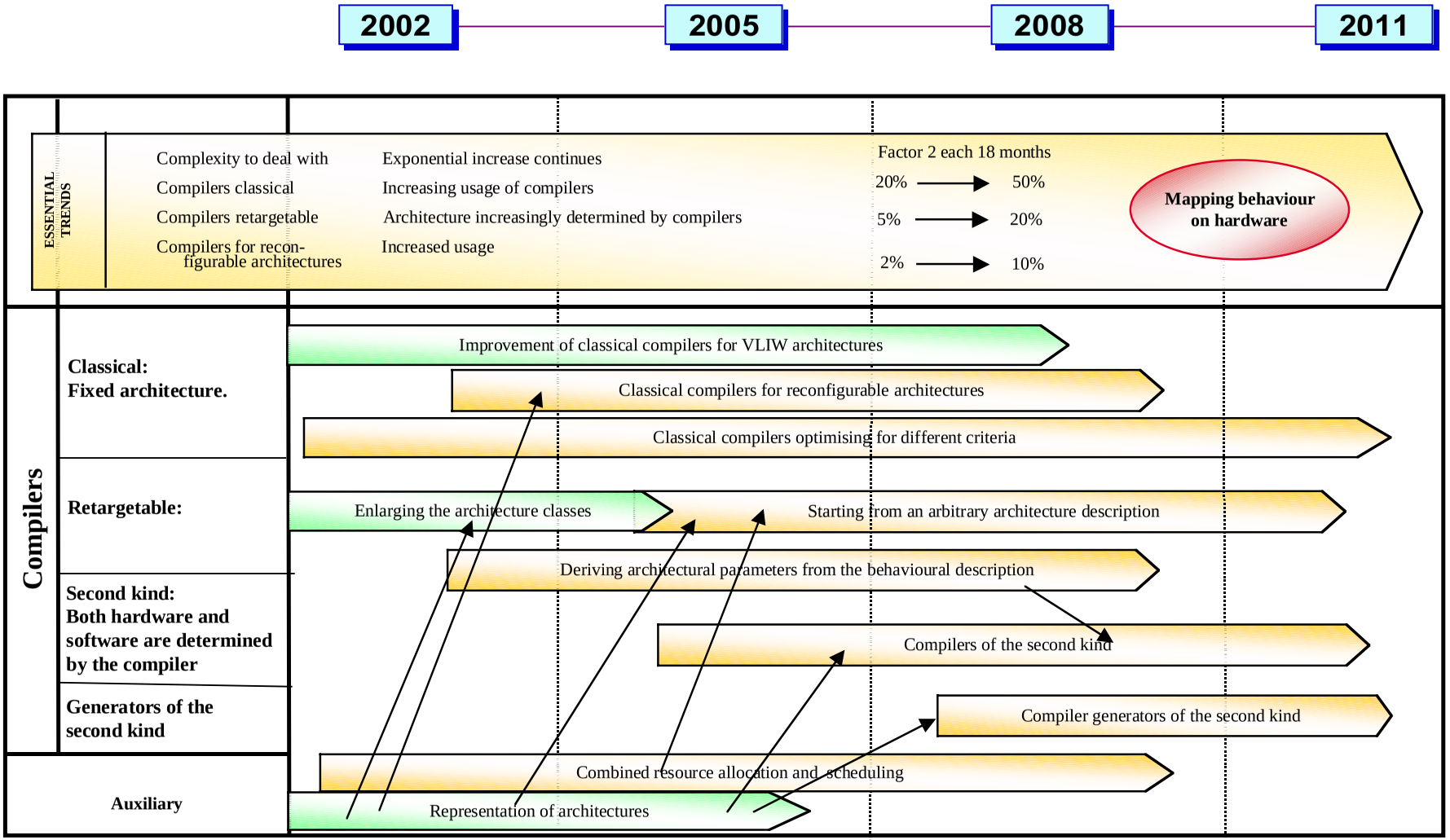
EMBEDDED SYSTEMS ROADMAP:

FROM EXECUTABLE SPECIFICATION TO IMPLEMENTATION 1



EMBEDDED SYSTEMS ROADMAP:

FROM EXECUTABLE SPECIFICATION TO IMPLEMENTATION 2



3.3 Platform design

*'More **from** the same'*

3.3.1 Introduction

First an introduction of definitions and explanations:

- A *platform architecture* is the maximal (and preferably optimal) superset of functions and blocks that are part of the platform, designed with a certain application domain in mind. The goal is to find the commonalities between various designs, while still being able to create differentiating products.
- *Platform design* is the activity of defining a platform architecture plus design environment to be able to create instances based on the same theme. Therefore, it's more from the same!
- A *platform product* is a product instantiated from a platform.
- A *platform* describes the material realisation (architecture) and the way to create platform products, but also the support aspects: coding rules, test benches and documentation standards. A platform is not equivalent to a system-on-a-chip (SoC). In other words a platform is the combination of components, communication architecture, rules, tools, test benches, and documentation. Hence, platform includes board level realisations.

Note that IP blocks in the following text means both hardware and software IP blocks.

3.3.2 General trends and user needs

The following trends and user needs relate to the necessity for the creation and usage of platforms:

- The complexity of implementation and functionality shows an exponential increase over time (a factor 2 every 18 months).
- Electronic products will behave like fashion; the lifetime of a specific product will decrease to about one month. As this cycle is too short to design new products, these products need to be designed on a common basis: the platform.
- In this respect, time-to-silicon needs to be reduced to one month. On the other hand, suppliers in a competitive market want to differentiate their products from those of the competitors.
- The number of application domains that result in products based on a platform will increase from currently about 4 to 100.
- "Meet in the middle": the kind of blocks that are used in platforms will get a higher level of abstraction.
- The lifetime of a platform will double over the next decade.

The sub-roadmap falls apart in two parts:

- Platform family selection and creation, which coincides with the scenario 'Platform Design Creator' in the ESD domain paper. It consists of three subdomains: design space exploration, standardisation, and design languages.
- Platform unstantiation, which coincides with the scenario 'Platform Design Instantiator' of the ESD domain paper. The keywords in this part of the roadmap are design space exploration and component integration

Design space exploration

Design space exploration is important both for the platform creator and for the platform instantiator. To put the scene, consider tools for design space exploration. Currently it is possible (i.e. coming out of the research labs) to create IP blocks in a platform fashion. It is described which blocks can be included in an IP block, for example register files, functional units, and communication between the two in a hardware DSP. Further, a compiler can be retargetable, to translate an application written in a language like C to machine code (an executable) running on an instance of this family of DSPs. A way to achieve this, is to let all tools be steered by a machine description, which gives values for all parameters in the platform. The following figure illustrates this:

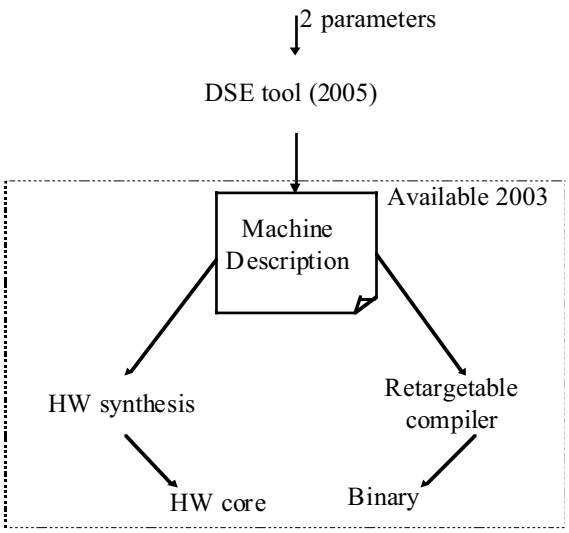


Figure 6: Steering DSE tools by a machine description

Once this system is obtained, experimenting with different machine descriptions can start design space exploration at the level of IP blocks. This is depicted by the DSE tool (not available before 2005) in the above figure.

The next step to take is enable design space exploration in a structured and (partly) automatic way. Which values of the parameters should be described in the machine description to satisfy the constraints in the specification?

When such a method for structured design space exploration has been developed, structured DSE at the IP level becomes a reality. A platform consists of many IP blocks put together to perform the embedded system's functionality. A way to execute DSE at platform level is to exploit the DSE methods for the various IP blocks separately and combine them at the platform level. In other words, the same trick applies one level of abstraction higher, as is shown schematically in the following figure:

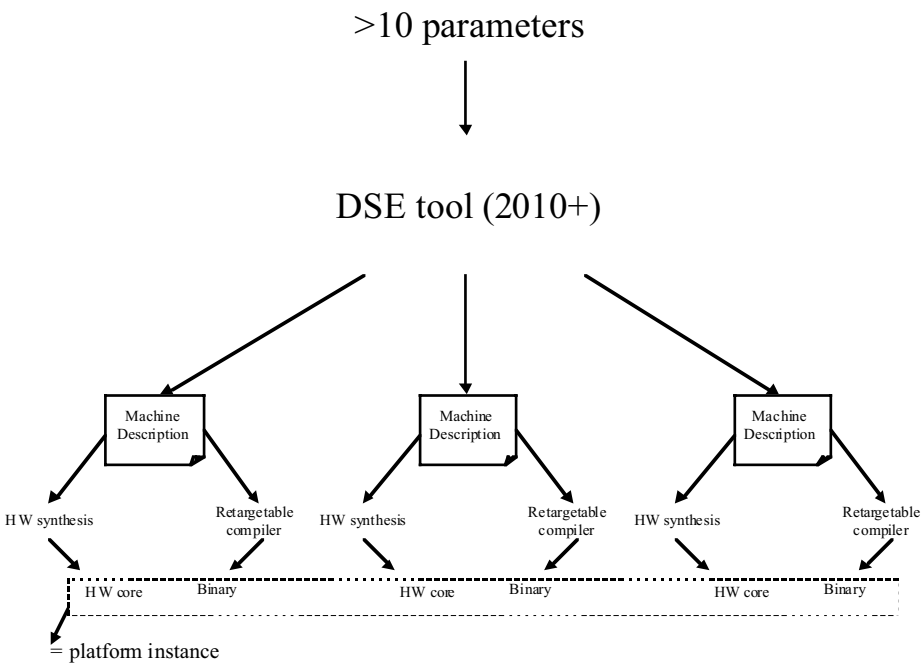


Figure 7: Steering DSE tools by machine descriptions on platform level

Component integration in SoC: Networks on Chip (NoC)

For Systems on Chip, a platform also comprises the communication protocols and components. Currently, practical methods for designing the communication infrastructure are bus-based and synchronous. Bus-based communication is not considered very scalable however, whereas scalability is a prerequisite for the re-usability of the platform. Synchronous communication at the system level is also becoming a burden, given the many on-chip clock domains in future SoCs. Furthermore, due to the increasing clock frequency it is expected that within ten years it will take up to 30 clock periods to transfer data from one side of a chip to the other. These problems with bus-based synchronous communication suggest an approach where communication is Globally Asynchronous, Locally Synchronous (GALS). An emerging communication paradigm in SoC that favours scalability and GALS is the Network on Chip (NoC) approach. This approach is based on the 7-layer OSI data communication protocol designed for general networks. A distinguishing feature in SoC is the predictability of the task executions and the arrival of data from the environment. SoC designers will exploit this feature to make SoCs more cost-efficient.

3.3.3 Technology sub domain: Platform family selection and creation

This sub roadmap of this sub domain is about the people who create the platforms and define which blocks and communication structures should be included in a platform. Design space exploration, standardisation, and design languages are key ingredients.

Technology requirements

For design space exploration a trend from single-processor systems to multi-processor systems is visible. More generally, design teams of embedded systems will incorporate not only hardware and software people, but also people from other disciplines. Compilers are key in doing design-space exploration. The above text shows that retargetable compilers are needed, as are compiler generators and ultimately compiler generators for multi-processor systems.

Re-use of IP blocks is essential for the creation and usage of platforms. To make IP blocks re-usable in embedded systems standardisation of interfaces and functionality of blocks is required. In addition, models and other properties of IP blocks (e.g. power dissipation, area) should be included into these standards. The sub roadmap on hardware and software design also highlights these issues. Automatic IP-wrappers are an essential pre-requisite for the usage of IP blocks in platforms. Not only the blocks, but also the communication infrastructure to glue the blocks together needs to get attention in the standardisation. Test and debug of systems composed using platforms should be standardised in such a platform.

Models to express properties of IP blocks are not essentially all the same. Still it is necessary to be able to reason at the system level about the composition of these blocks. Therefore, methods and tools that collaborate in this sense are needed. A vision is, to go from single language design systems (e.g. like SystemC or UML based), to cooperation between them, where a small number of languages seamlessly operate. This is in view of the section on design space exploration above. Integration of other blocks to result in heterogeneous embedded system makes the relevance of this case even stronger.

A general gap in this sub-domain is that the possibility to provide for systematic generation of platforms given an application domain, is not yet available. The issue here is how to express a platform in terms of the parameters that are relevant in the application domain.

3.3.4 Technology sub domain: Platform Instantiation

The key concept in this sub domain is design-space exploration in deriving products from platforms.

Technology requirements

Proper methods and tools are the key parts to perform structured design space exploration. Tools that can take ever more design parameters into account without a sacrifice in user-friendliness and computational complexity should be devel-

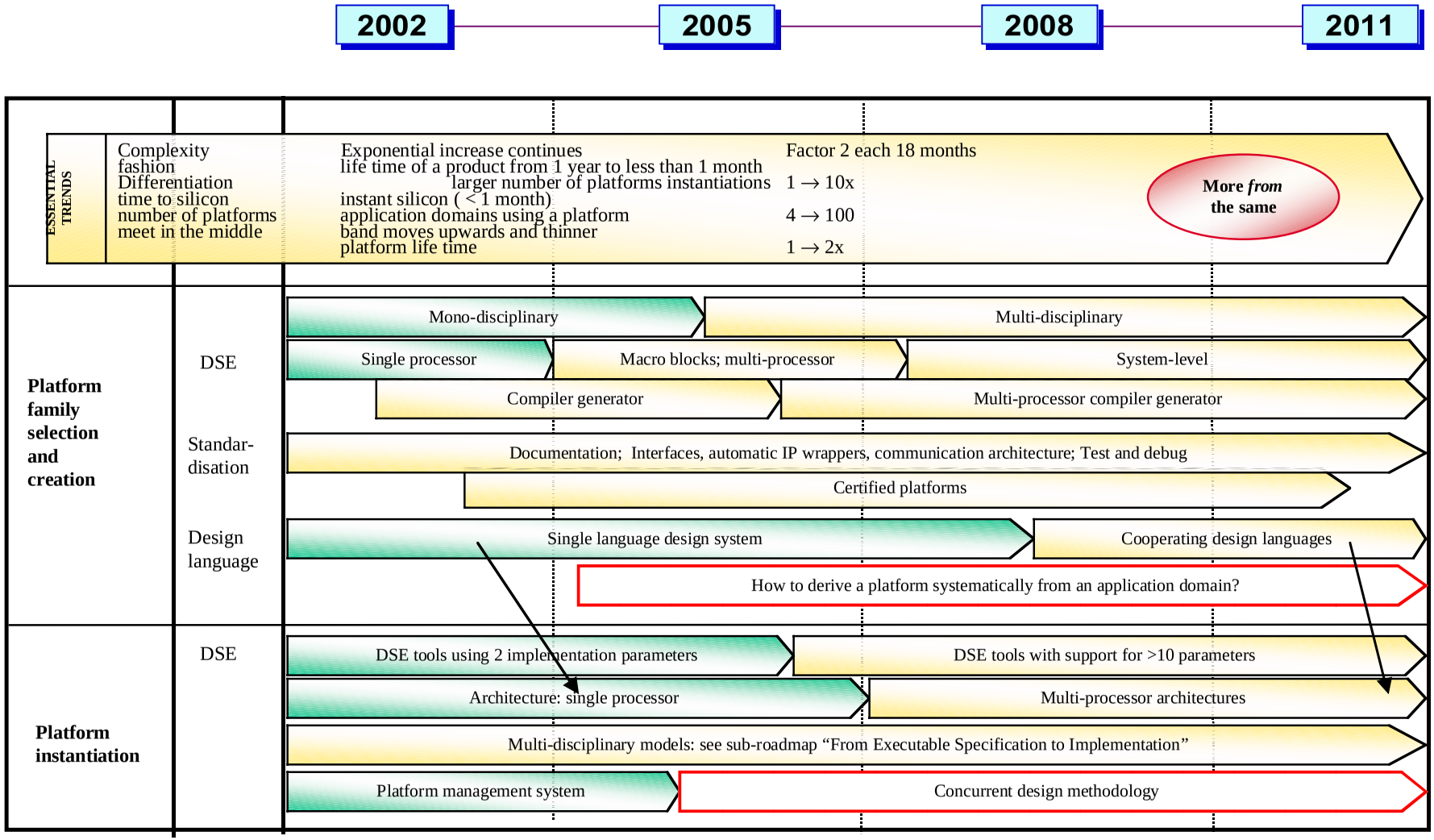
oped. Multi-disciplinary models are essential to do platform instantiation; these models are dealt with in the sub roadmap ‘From Executable Specification to Implementation’.

A platform management system and concurrent design methodology are aids necessary for the platform instantiator, to systematically use the strengths of platforms. Such a management system includes aspects like coding standards and documentation standards that are to be used for platform-instantiated products. Another essential ingredient is to be able to identify that an instance is indeed compliant to the platform it was instantiated from. For Systems on Chip, a considerable part of the design effort in platform instantiation is spent on making HW-SW trade-offs (at component level) and the integration of components in a communication infrastructure (at system level). Both are discussed in the section on HW-SW design.

3.3.5 Recommendations

1 Start a large demonstrator project for a GNU-like ES platform. The purpose of such a demonstrator project is to facilitate the implementation of the next recommendations:
2. To enable and promote the development of IP blocks and communication architectures, a project should be started to establish guidelines and working practices to be used as a starting point for a (concurrent) platform design methodology. This demonstrator project should also include a standardisation effort.
3. Establish the requirements of a platform management system and of a concurrent design methodology.
4. Start work on hard & soft co-simulation for on-time applications. Tools to support heterogeneous/hardware and software co-simulation including timing and communication signals should be developed.
5. Development of high-level models for IP interfacing. IP blocks at the moment usually lack high-level descriptions. The availability would improve the insight of the behaviour of the IP as well as improve simulation efficiency.
6. Facilitate multi-disciplinary teams to meet on a regular basis. These meetings serve as the exchange of ideas and common practice different teams and facilitate a common understanding. The result of these meetings could be the definition of a ‘cooperating design languages’ design system.
7. For Systems-on-Chip, migrate from bus-based communication towards packet-switched networks-on-chip, with HW routers.

EMBEDDED SYSTEMS ROADMAP: PLATFORM DESIGN



3.4 Hardware/software design

'Design effort versus cost-efficiency'

3.4.1 Introduction

HW/SW design in Systems-on-Chip (SoC) comprises the tasks of partitioning the application and determining on what type of HW each part of the application will execute. If the platform contains a homogeneous set of processors, distribution of the algorithm over the processors is also considered part of this task. The HW/SW designer is also responsible for integrating the implementations of the parts to a working system. The HW/SW designer is driven by two main forces.

Firstly, silicon technology allows more and more integration, which makes HW/SW design more complex, and cost of non-recurring engineering cost (NRE) higher. For example, chip-mask costs have risen from $100K to $1M in 4 process generations.

Secondly, the market demands a short design time and cost-efficiency. With regard to the latter, energy consumption is becoming an increasingly more important criterion in the context of both battery-operated devices and devices that generate a lot of heat. The issues of design effort and cost-efficiency are often conflicting, and the trade-off between them is the main concern of the HW/SW designer when exploring the design space of implementations for each part of the application. For large systems in a small market (e.g. wafer stepper, medical systems) however, the emphasis is on the software engineering aspects together with the real-time constraints typical of embedded systems

3.4.2 General trends and user needs

Three main design trends result from this trade-off: reuse (IP blocks), design tools, and the availability of an increasingly wider spectrum of possible implementations. In fact, the distinction between 'HW' and 'SW' will blur into a grey area, as will the distinction between their corresponding design flows. In order to allow flexible reuse, it may even be advantageous to delay decisions regarding implementation (details). In this implementation spectrum, (re-)configurable components are a developing trend with potentially large industrial implications, because they enable HW/SW design without fabricating silicon! This allows smaller companies to process chips with state-of-the-art technology and make their own HW/SW trade-offs. One distinguishes two levels of performing HW/SW design: the system-level and the component level. We also distinguish large complex systems with a relatively small market, because the typical HW/SW trade-offs are not representative for the design of these large systems.

3.4.3 Technology requirements

System level HW/SW design in SoC

At the system level, the HW/SW designer partitions the application into tasks that can be executed on components, and integrates the components to make a working system. In order to do this partitioning efficiently, the designer should have at his disposal sufficiently accurate cost/performance models of available (IP) com-

ponents (HW and SW!), both for computation and communication. Desired but lacking is a method to quickly estimate the relevant design criteria corresponding to the composition of the computation and communication blocks. In order to do the integration efficiently, the interfaces on the components have to be well-defined and standardised. The designer should also be familiar with the communication protocols implied by the platform, and have a simulation environment to parameterise the communication and storage infrastructure and to validate the integration of components. For this validation also behavioural models of the IP blocks are required. Scalability of the communication infrastructure will be an issue given the expected improvements in silicon technology. It is expected that more attention will be paid to the development of communication infrastructure due to the poor scaling of current bus-based communication, and to sub-micron effects leading to unreliable communication, like signal cross talk, and timing errors due to uncertain propagation delays. Therefore we expect communication to be supported by packet switched networks similar to those found in computer networks. The storage infrastructure will also gain in emphasis, because applications are becoming more data intensive, and more components will exhibit some degree of programmability, requiring memory to store the program. Because of the increasing dominance of memory in SOC cost, designers will feel the pressure to try and reuse memories among the different tasks and/or processors.

Task-level parallelism (TLP)

Because the components can run in parallel, an important goal in HW/SW design at the system level is the exploitation of this task-level parallelism. We consider two situations: systems-on-chip (SOC), where task executions are relatively predictable, and systems (servers) in a dynamic network, characterized by unpredictable arrival of tasks.

In SOC design, the exploitation of TLP is largely statically determined and dominated by the task of clustering towards or identifying tasks in the application. Besides the goal of enhancing the opportunities for TLP execution, an important criterion is to minimize the amount of communication between the tasks. This communication overhead can be a major obstacle for the predictability and scaling of the system performance (in the number of processors). With the current design practice (mainly in C) and all the available legacy code it is convenient to automatically identify opportunities for task-level execution in arbitrary high-level (C) code, maybe just for quick estimation of implementation cost. In the longer term this need can be expected to fade away given the current development towards more modular, object-oriented system specification methods. Also the increasing use of IP blocks necessitates early specification of the opportunities/constraints of using IP blocks in the design. Because of the increasing complexity of SOC and the incorporation of more data-dependent and control behaviour, there is a trend towards less predictable system behaviour. In a static schedule, worst-case assumptions are made to guarantee valid system behaviour. If the control of the system is performed run-time, the system can more efficiently cope with statistical and unexpected events by dynamic task scheduling and allocation of bandwidth-, memory-, and processor resources. The main draw-

back of these dynamic control mechanisms is that their functionality needs to be validated and simulated together with the HW and SW. This is more complex than validating a static schedule.

Component-level HW/SW design in SoC

At the component level, the HW/SW designer makes processors, either for use in his project, or as an IP block that can be incorporated in many designs. IP blocks are available now in the form of programmable microprocessors or DSPs. The market for these flexible processors is sufficiently large to justify the design effort. More specialised IP blocks are being generated by large companies for in-house use. SW IP is still hardly visible. The component designer has an increasingly wider spectrum of implementation choices to make trade-offs between cost efficiency and design effort. In this implementation spectrum, (re-)configurable logic is a clear trend. One expects that the concept of (re-)configurability will be drawn to higher levels of abstraction, from gates via arithmetic components to processor architectures (storage & communication, instruction set). This will have an impact on the design effort, but also on the number of (instruction-)bits required to configure and control the underlying HW. This will result in smaller configuration times, less configuration/program memory area, and less power consumption while fetching the configuration/instruction bits. The trend towards configurability is not just true for HW, but can also be observed for the corresponding mapping tools: retargetable and parameterisable compilers that also allow convenient tool reusability and design-space exploration (DSE).

Tool support for partitioning and HW allocation will develop from profiling tools to help the designer identify suitable targets for HW acceleration (current situation) towards tools that automatically identify such targets and eventually, tools that perform the partitioning and allocation process (semi-)automatically. Current tools are somewhat limited in design flexibility (and therefore reuse), because they typically have one language as a design entry, one implementation as a result, and often optimise to a single design criterion.

Instruction-Level Parallelism (ILP).

Another important development is the tool support for the detailed mapping of tasks to components once the implementation paradigm is chosen. The main design objective at the component level is the exploitation of ILP. In the current situation, compilers can efficiently exploit ILP for architectures (e.g. VLIW) that are not very cost efficient (code size), whereas cost-efficient processors usually require the designer to write assembly code to exploit ILP for the time critical parts of the algorithm. In this situation, the processor architecture and the compiler are developed independently. One expects that in the future the processor architecture and the compiler are developed coherently to make a practical trade-off between code size and compilability (cost-efficiency and design effort).

Large SW systems in small markets

These systems include the control of an air plane, a wafer stepper, a medical imagery system, and a telephone exchange server. Issues like safety-criticality often play a large role in the design of these systems. The dominant issue however, is the design effort. Advantages on both issues are offered by the re-use of existing software. This software can be borrowed from previous projects or bought from third parties. Some reasons for using existing embedded software in future products are:

- Up to 50% of embedded software code is related to exception handling and error recovery, while less than 10% of its architecture is related to these items.
- Very specific requirements on optimization of cpu cycles, memory, power etc. in hardware related software is difficult to design top-down.
- Software for embedded systems contains very detailed knowledge of the hardware it controls. This knowledge is delivered bottom-up. So it is hard to generate this code with a top down code generation tool.
- One of the prerequisites for re-use is that methods and tools for defining the embedded software architecture support re-use. Else the architecture of the product will go its own way without bothering about the past.

Some huge roadblocks prevent re-use.

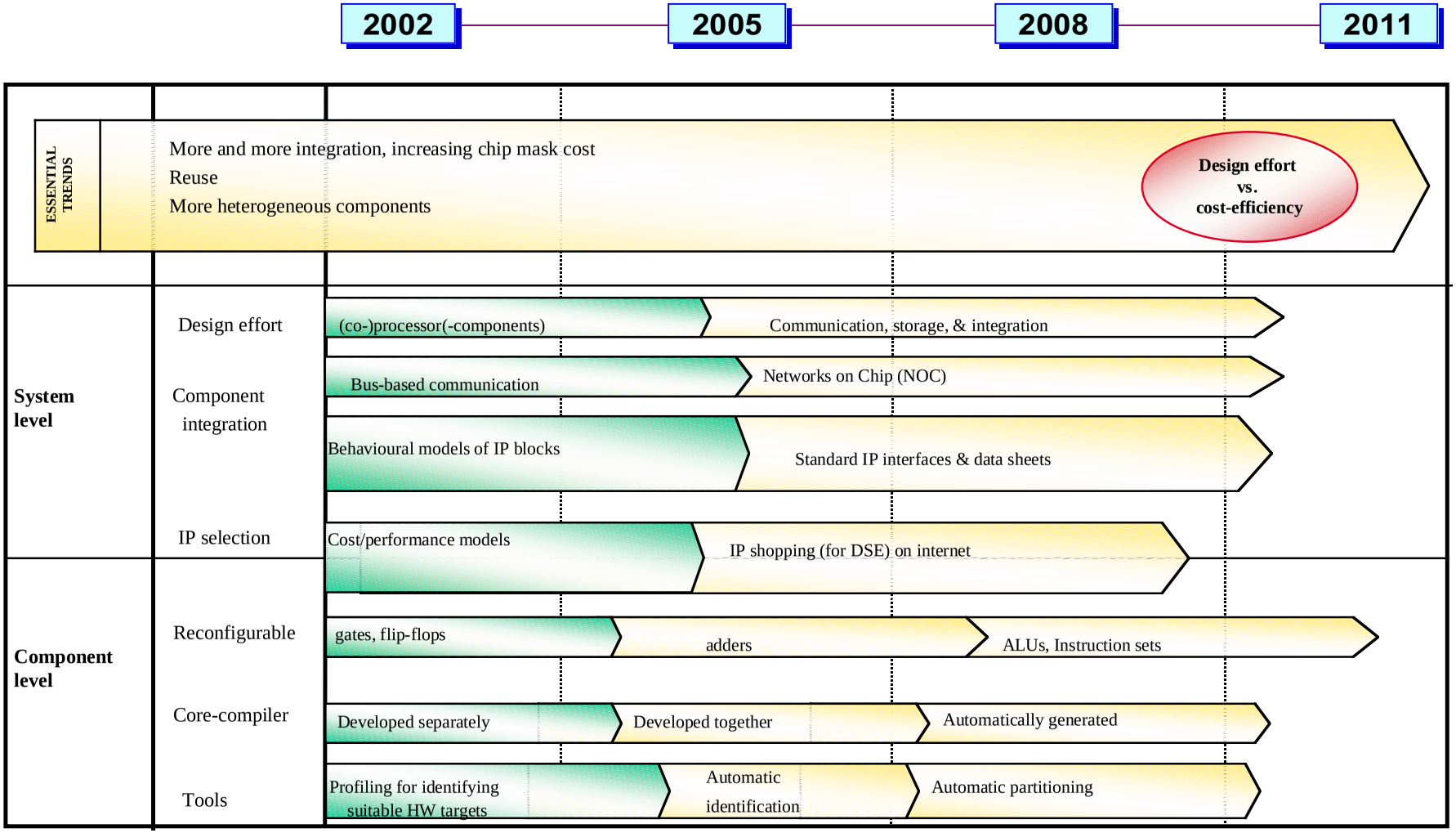
- Embedded software architectures are not re-used. Every project starts with inventing its own architecture. Therefore third party software will not be compatible with this architecture.
- Current software design tools don't help with using old code. In the best case these tools cross-compile code at the statement level.
- Embedded software development environments are very diverse. Think about computer language, compiler, configuration management, test environment, hardware, real time operating systems of target system, operating system of host system, communication protocols.
- Universities only teach students how to build new systems from scratch, not how to start with existing software.
- Current embedded software methods and tools do not support re-use. The embedded software expert therefore has a need for an embedded software IP market, where IP components, consultancy, maintenance and support can be sold. And where consumers and producers meet to predict future needs.

3.4.4 Recommendations

1. Provide (certified) standardised models of IP blocks regarding (cycle/bit-true) behaviour, cost, and performance at different levels of abstraction.
2. Develop HW architecture and mapping tools coherently to obtain more advantageous trade-offs between design effort and cost-efficiency. Examples are processor core and compiler, or network-on-chip topology and tools for determining the routing of the data.
3. Increase the level and grain of reconfigurability to accommodate high-level design decisions. Examples are reconfigurable instruction sets, memory organization, and communication infrastructure.

4. Designers should be trained in the use of (higher-level) tools; the distance between embedded system designers and tool designers should be bridged.
5. Embedded system designers need to be aware of the increasing spectrum of implementation paradigms. This is especially true for companies targeting a relatively small market, because they have no ‘tradition’ in HW design required for the emerging paradigm of reconfigurability.
6. Standardize API, architecture and external behaviour of an IP component
   a. Intra process, inter process and inter processor communication.
   b. Exception handling
   c. Debugging facilities
   d. Intra component verification and validation
   e. Inter component verification and validation (JTAG like)
   f. Interfacing with Real Time Operating system
   g. Hooks for hot-swappable software

EMBEDDED SYSTEMS ROADMAP: HW/SW DESIGN



3.5 Verification/validation

'To design on target'

3.5.1 Introduction

In each and every step in the design flow of embedded systems it is important to check whether the design implements the intended functionality. To this end we distinguish various techniques: formal techniques and non-formal techniques. Formal techniques try to either prove the system correct or check the entire state space a design can be in. Non-formal techniques check only a part of the complete state space of the design with the benefit of speed of verification.

Both for formal and non-formal verification the designer of an embedded system needs to consider where he or she wants to use it for. Formal verification has been researched extensively for control-like applications, while it has not yet been used for data processing on a large scale. Timing and performance are important aspects of embedded systems, thus these aspects need to be considered for verification. Classical compilers do not deal with these aspects. Further, formal verification relies on a certain model of reality: the designer of embedded systems needs to have confidence that this model truly describes the reality!

Designs of embedded systems that are currently created in industry have such large state spaces that the current state-of-the-art formal verification tools are not capable to verify them completely. Therefore embedded system designers rely for a large part of their verification on techniques like simulation and emulation. As designs are expected to grow in complexity the need for simulation shall be continuing its growth also.

On the other hand, as is also stated by the ITRS roadmap [ITRS, 2001] in its section on Design, simulation does not scale as designs grow; one can only cover a part of the design space. Therefore a breakthrough is necessary to cope with the design verification issue. This breakthrough is expected from the shift from non-formal to formal verification techniques.

According to the ITRS roadmap the main near-term challenge is to make formal and semi-formal verification techniques more reliable and controllable. Capacity (i.e. the sizes of designs that can be formally verified), robustness, and verification metrics are points of attention for the next five years. After 2007 new techniques are necessary according to the ITRS. Design for verifiability, coping with higher levels of abstraction, human factors in specifications (which need languages and specifications), and broadening the scope of formal methods to analog/digital and hybrid systems are mentioned in the ITRS.

The ITRS does not detail the formal verification methods necessary. In the roadmap for embedded systems we focus more closely on formal verification as a means to cope with the verification challenge for the next ten years.

3.5.2 General trends and user needs

The world of verification and validation, as well as the world of embedded systems are undergoing rapid changes. We see a few trends:

- Moore's Law states that the complexity of systems will increase continuously with a factor 2 every 18 months;

- Dunn's law states that we have to improve the analytical power of verification and validation tools for embedded systems with a factor 4 every 18 months;
- Thus a factor 2 improvement every 18 months has to come from methods, algorithms, data structures, and implementation techniques;
- The increasing complexity of embedded systems will necessitate the use of formal methods during the design of embedded systems;
- Verification and validation will not only be needed at low-levels of abstraction (e.g. hardware) but also at higher levels of abstraction (e.g. architecture; (hybrid) systems);
- It is essential that the designer of an embedded system can use verification and validation techniques in his design trajectory, without bothering about the underlying mathematics. Further, the tools for verification and validation need to become user-friendly.

Algorithms and data structures are the basic ingredients to do verification and validation of embedded systems. They have been studied in this area for a long time and many techniques are already available. There are also numerous tools available to do modelling, simulation, model checking, equivalence checking, consistency checking, real-time and stochastic analysis.

The main problem is that these algorithms, data structures, and tools are only practically applicable to small toy-size examples nowadays. It is important to be able to do verification and validation on ever-larger examples. However, for an example of a certain size some techniques are applicable while others are not (yet). In about ten years from now the goal is to be able to verify systems with an algorithmic complexity corresponding to that of explicit state model checking of state space of the order of 10 tera-states. Given the anticipated use of symbolic state space representations this includes the verification of large classes of infinite state systems.

We have divided the Embedded Systems Roadmap on verification and validation into three parts: formal verification, non-formal verification, and the integration of formal verification techniques with the design flow for embedded systems.

3.5.3 Technology sub domain: Formal verification

In order to comply with Dunn's law we have to develop fundamental algorithms, efficient data structures, and implementation techniques to improve the performance of tools for verification and validation.

Technology requirements

In general, for formal verification we need proper formalisms (languages) to be researched for two reasons:

1. What kinds of properties do we need to express in such a language;
2. How do we make the link with the (top-level) specification.

There are many formalisms which consider designing either bottom-up or top-down. SystemC e.g., takes the bottom-up approach: it allows high-levels of detail to be described in the design. UML is an example of a high-level formalism, which misses the power to describe the semantics of low-level details. An embedded systems designer wishes formalisms to bridge these two levels.

A prerequisite for formal verification techniques are the algorithms they use.

One of the major problems is that we need algorithms for the efficient exploration of state spaces a design can be in. Nowadays we can traverse discrete state spaces; in the near future it will be necessary to traverse symbolic state spaces.

Optimal search algorithms need to be developed.

A way to deal with more complex designs is to introduce hierarchy in the complexity of the design. For this we need proper abstraction algorithms.

We distinguish two main areas of techniques that are important to be researched for formal verification: model checkers and theorem provers.

The usability of theorem provers relies partly on proper decomposition techniques. Decomposition allows for hierarchy in the design; theory of composition is necessary to prove properties of the composed system.

Model checkers are foreseen first for functional system models; after that we need model checkers for soft/hard real-time systems. Also model construction, model simulation, and test-case generation need to be considered for these two stages. Static analysis techniques are important aids in doing model checking.

Ultimately it is desirable to integrate the theorem proving and the model checking techniques.

3.5.4 Technology sub domain: Non-formal verification

Due to the growing complexity of designs, non-formal methods will remain important to cope with the verification challenge. We focus on simulation and emulation as non-formal verification techniques as they traverse only a part of the design space.

Currently we can simulate single (IP) blocks. In the near future we will need simulation techniques of compositions of blocks; platforms should provide guidelines in these. Currently there are already activities in the area of system simulation; they will become more important as designs grow in complexity. When hybrid systems come into sight we will need simulation techniques to cope with these systems and models.

Emulation has always been important when the hardware (and the software) is already there, for example in prototype format. One of the purposes is to detect bugs related to timing that cannot be detected in simulation due to the abstraction of the (timing) models. As systems grow from single blocks to multiple blocks to hybrid systems, emulation techniques need to keep up.

The links with the Platform Design sub roadmap are obvious when we will be able to simulate or emulate systems at the block level or at the system level.

3.5.5 Technology sub domain: Integration

This sub domain focuses on the integration of formal verification and validation techniques with the design flow of embedded systems. The ultimate goal is that designers will think it a natural thing to verify their designs.

First we need to get the tools and techniques for formal verification out of the academic world into the industrial design world. Interfaces are one thing; even more important is to teach the embedded systems designer to use formal verification in the design flow. The only way this can be done is bi-directional. Current

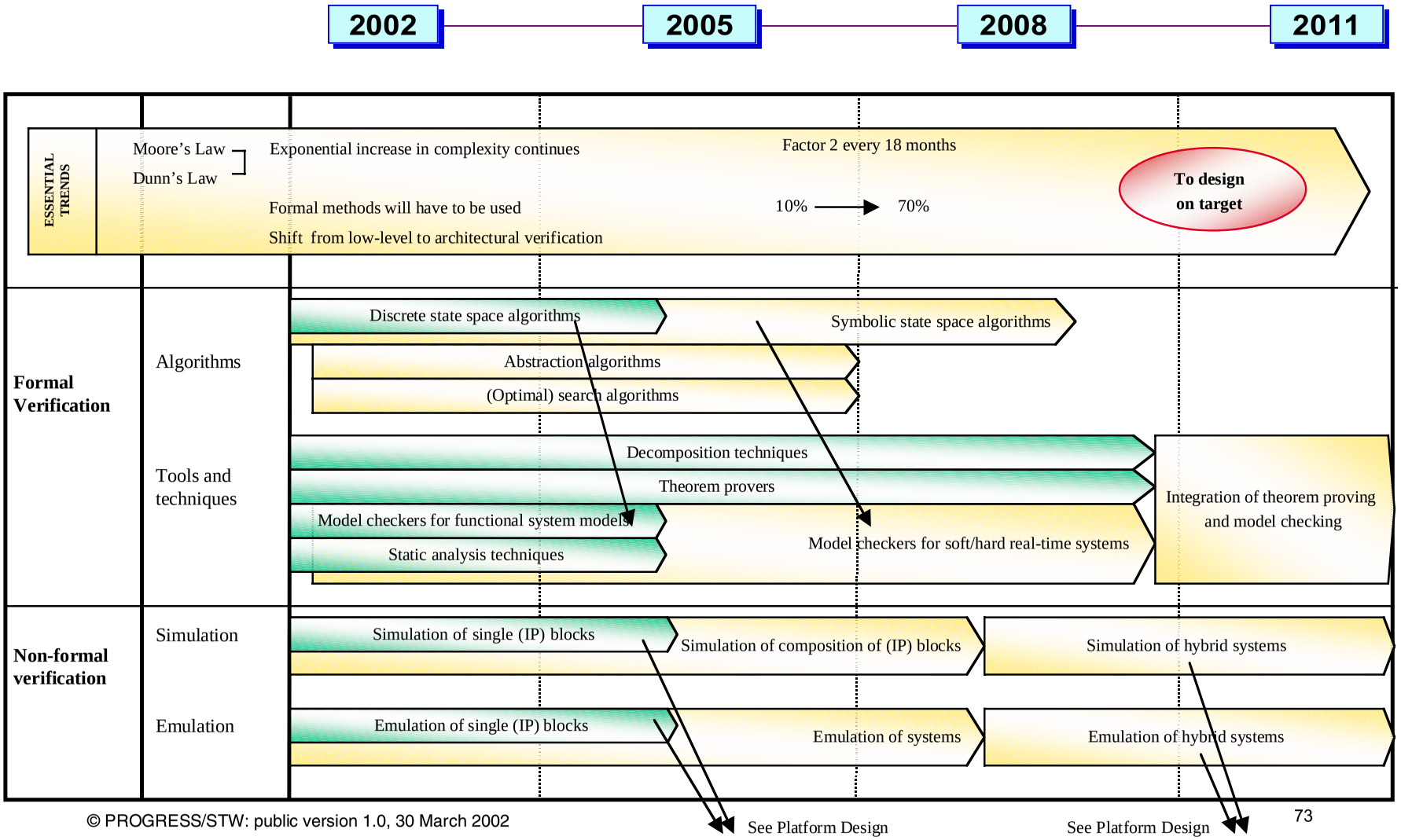
experts on formal verification need to adapt their tools and (user) interfaces to the industrial design flow, and need to take large designs from industry as cases. On the other hand, embedded system designers need to learn how to use these techniques in their design flow. Configuration management and application-oriented tool interfacing also are important issues in this context.

A next step is to actually integrate verification and validation in the design flow, with seamless integration as a goal. To verify systems that will be built in ten years from now, verification techniques should be able to deal with multi-core and even hybrid systems.

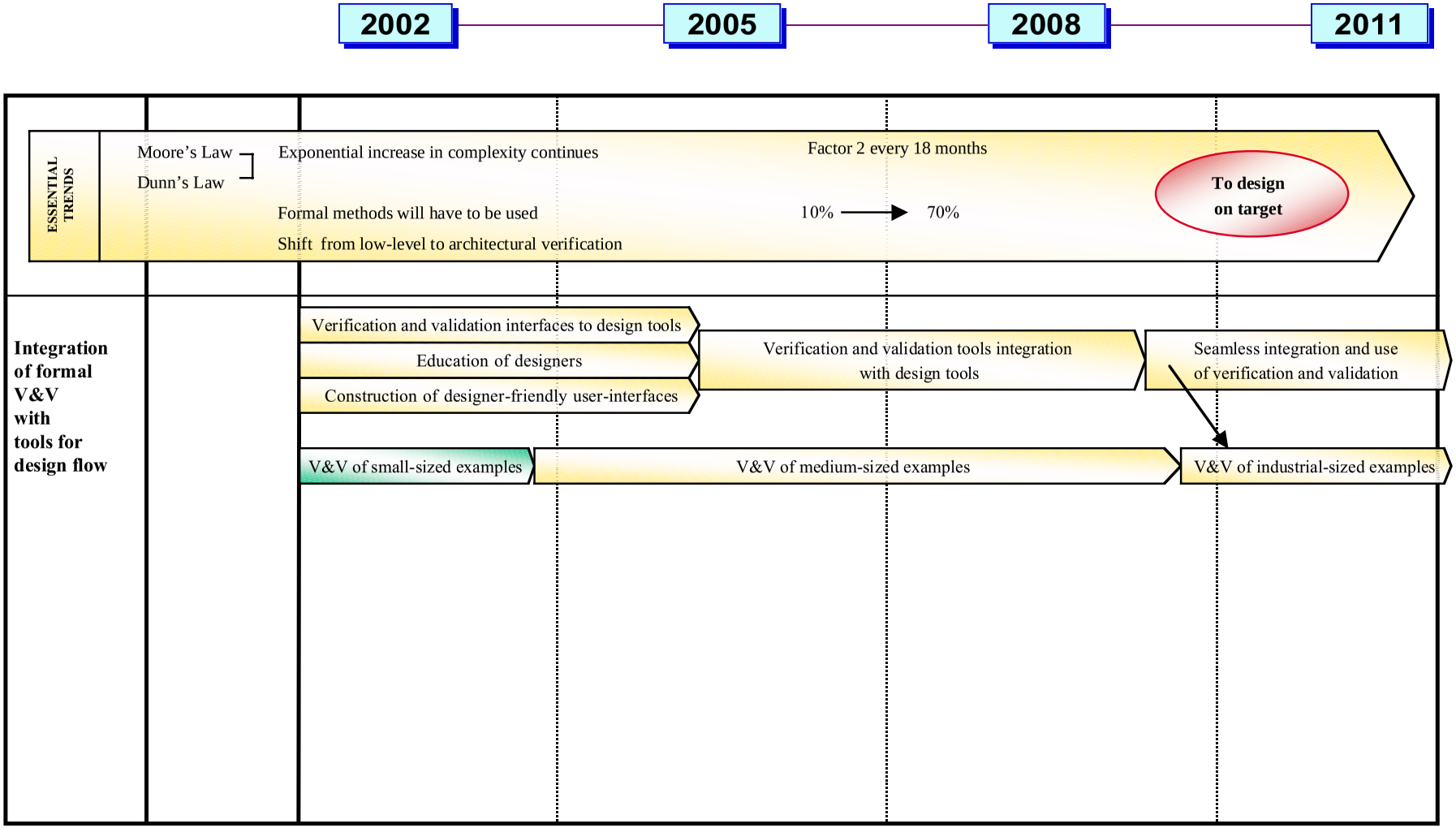
3.5.6 Recommendations

1. First a classification is needed when to do formal verification. In well-described domains verification is possible, but simulation or prototyping can be good alternatives. As verification comes at a cost (e.g. computational complexity), designers of embedded systems have a serious trade-off to make.
2. Verification of heterogeneous systems is key to future development of embedded systems and should be further investigated.
3. Representation formalisms (languages) on which formal verification operates should be studied. The relation with the top-level specification is of utmost importance to allow for integration of verification techniques in the embedded systems design flow.
4. The problem of exploring very large state spaces in a manner that is computationally efficient is one of the first prerequisites to make verification feasible in the design flow for embedded systems. Hierarchical design is a very important research topic to be studied to be able to verify large designs. Vendors of (trusted) components for embedded systems need to express their information for verification on the right levels of abstraction, in a formalism that the embedded systems designer can deal with.
5. Verification and validation should be made usable for every designer of embedded systems, including tool support and courseware. In order to integrate verification and validation in the design flow of embedded systems, it is essential for computer scientists working in this area to study realistic cases. These are needed to improve and adapt verification and validation techniques for industrial relevance. These cases can serve as benchmark to monitor the progress of methods and tools.

EMBEDDED SYSTEMS ROADMAP: VERIFICATION/VALIDATION 1



EMBEDDED SYSTEMS ROADMAP: VERIFICATION/VALIDATION 2



3.6 Test, debug and integration

'Quality control will stretch from factory to field'

3.6.1 Introduction

Many worlds come together in test, debug and integration.Testing means to establish the quality of the product. Most, if not all, of the design and fabrication steps are followed by a separate test. If a part is found to be incorrect, the nature of the defect may be discovered by debug. Test and debug are ingredients of a 1-technology quality control scheme. Unfortunately parts from different technological domains (micro-electronics, mechatronics, biotronics etc.) are integrated in non-trivial embedded systems, raising the problem of quality assurance (QA) and fault diagnosis and isolation (FDI) to unknown high levels.

The world of embedded systems is computer dominated. Though the embedding system features many different technologies, the attraction of the embedded systems world is largely caused by the digitally programmable embedding core. Hence we will assume an electronic embedded system with proper models of the embedding and external world. This is not a major restriction, but the simple admission that we will always test from an environment in which a software test program can run and the unification of test views on hardware and software will already keep us busy during the roadmap period.

3.6.2 General trends and user needs

Where originally an IC was fully tested, this is already not economically feasible and will in the future even become impossible. This is caused not only by the sheer impossibility to test a complex design in a short time, but also because the 'System on the Chip' will have additional characteristics that pose new testing problems such as:

- increased heterogeneity (more parts in different test technologies);
- increased programming diversity (more ways to structurally change the part function);
- more in-system support to the needs to test a system in/off/on line;
- raised polymorphy as caused by the additional reconfiguration potential.

Next to the improved (or autonomous) test of the part, the half-fabricate character of the manufactured chip may lead to delayed testing: testing only when a function is programmed. And as programming can occur also at the moment of product fabrication or even at the moment of instalment, testing may largely be offloaded to a later stage in the life cycle.

Web technology may even allow the ultimate test to be performed under supplier control at the user's site. The user can either be the product manufacturer (or even the local shop), where the chip is assembled into a (consumer or professional) product or the end-user, where the product is applied in connection to other products. If the business model assumes the local shop to be fully responsible for the service to the end-user, all debug and test will take place at the shop floor. The end-user will only need a failure indication, while the shop needs full support from the manufacturer.

In the long-term business model, a growing part of the test will be applied in use. A web-like support will be mandatory. This is partly true because one can not burden the average house dweller with the need to test an electronified house part. But having the product as part of a digital network, the added advantage is that any supplier can constantly monitor the life of delivered goods. Moreover it seems that this will become a necessity rather than just a business model. Still, debug support at the shop remains necessary as a trusted third party.

3.6.3 Technology requirements

In the past one has seen an advance in multi-level, multi-mode simulation because both analog and digital hardware must be handled at varying levels of abstraction. With the rise of heterogeneity and polymorphy together with the delayed commitment of functions to silicon the requirements will be raised. Further developments will be urgently needed.

On the factory floor, the basic functionality of the ‘system on a chip’ is validated. The on-going development of process technologies will regularly change the dominating fault model. Especially dynamic faults are posing problems, both in communication protocols as in the IP core themselves. At higher abstraction levels such problems will re-occur, for instance as a degree of non-functional interaction between IP-cores.

Also software design will mature to the recognition of testability as a development goal. Software engineering needs to have testability in mind, but so far hardware and software have totally different views on the testability issue and use a totally different terminology. More commonality seems required. An example is the software built-in self test (BIST) to facilitate self-test on-chip or in-product. Another issue is the sensitivity of software patterns on the production fault profile of a specific platform. It seems also that the canyon between TDO and verification & validation must still be closed.

At the shop floor, automated debug facilities are growing in importance because of the shift in the moment when the specialising functions are finally committed. On-chip test facilities will help to ease the debug effort, quantified in operator skill and in test patterns communication needs. Fabrication errors will be pushed away by reconfiguration and replacement.

The end-user expects total quality. This pushes the needs of self-test and self-repair to the limits. To limit the additional on-chip test structures, a degree of resource sharing will become mandatory. The test structures should not reduce the testability and the quality of the overall system.

As a consequence, the life cycle phases of the system will become apparent in a separate test view. This indicates a test process that permeates every aspect of the overall endeavour. Such an outgrowth of the test impact from a design view with some additional test measures over local pattern generators to an intertwined process will require a number of innovations, as

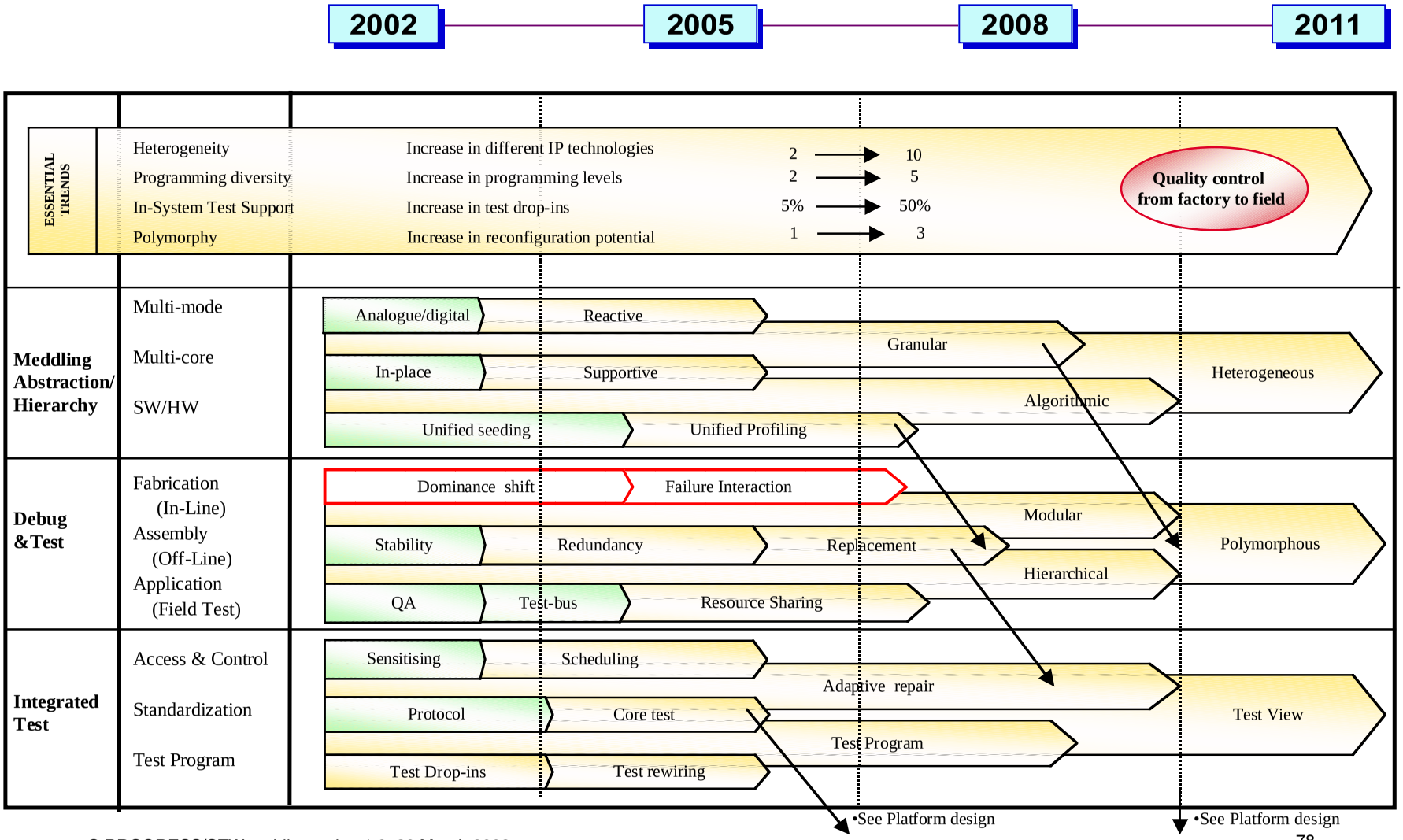
- built-in test for hardware/software combinations;
- hierarchical propagation of compacted test results.

3.6.4 Recommendations

From the sub domain roadmap, the following recommendations on future R&D activities can be derived:

1. Improved control management of tests for fabrication, product and application. As the test sequence is potential a set of time-dispersed activities of changing target and complexity, this process must be carefully monitored.
2. Unification of hard- and software test. With the increase in flexibility in implementation and the number of abstraction levels, the allocation of the fault to hard- or software becomes more difficult. In the current state-of-the-art, testing has overlapping points of strength in hardware and software. Consequently a more unified view is necessary to make function tests independent of the actual morphology.
3. Development of an extensive set of on-chip test measures (drop-in, re-wiring, program). Increasing system complexity and delayed function tests will require more potential to test on the chip.
4. Integration of test for heterogeneous, polymorphic architectures. Each logic technology as standard logic, reconfigurable logic, logic-enhanced memory or pure memory needs different test algorithms. Mixing such different parts in the same system and changing the actual implementation form of a function over night requires additional attention to monitor the pluriformity in the larger system

EMBEDDED SYSTEMS ROADMAP: TEST, DEBUG & INTEGRATION



Appendix 1. References

[Szyperski 1998] Clemens Szyperski (1998) Component Software, Beyond Object-oriented Programming, Addisson-Wesley, ISBN 0-201-17888-5

[EDAA 1998] EDAA (1998) System Design Technology Roadmap, http://www.iae.nl/users/ldje/edaa.html

[ITEA 2000] ITEA(2001) Technology Roadmap on Software Intensive Systems, http://www.itea-office.org

[Edwards 1997] Stephen Edwards, Luciano Lavagno, Edward A. Lee & Alberto Sangiovanni-Vincentelli (1997) Design of Embedded Systems: Formal Models, Validation, and Synthesis, Proceedings of the IEEE, Vol85, No. 3, March 1997, pp. 366-390

[De Micheli 1997] Giovanni De Micheli, Rajesh K. Gupta (1997) Hardware/Software Co-Design, Proceedings of the IEEE, Vol85, No. 3, March 1997, pp. 349-365

[Paulin 1997] Pierre G. Paulin, Clifford Liem, Marco Cornero, François Naçabal & Gert Goossens (1997) Embedded Software in Real-Time Signal Processing Systems: Application and Architecture Trends, Proceedings of the IEEE, Vol85, No. 3, March 1997, pp. 419-435

[Finkelstein] Anthony Finkelstein [ed.] (1998) Software Engineering: a Roadmap

[Goossens 1997] Gert Goossens, Johan van Praet, Dirk Lanneer, Werner Geurts, Augusli Kifli, Clifford Liem & Pierre G. Paulin (1997) Embedded Software in Real-Time Signal Processing Systems: Design Technologies, Proceedings of the IEEE, Vol85, No. 3, March 1997, pp. 436-454

[Schlett] Dr. Manfred Schlett (1998) Trends in Embedded Microprocessor Design, Extension from paper for IEEE Computer, Vol. 31.,No 8, pp.44-49, August 1998

[MEDEA 2000] MEDEA (2000) EDA Roadmap

[ITRS 2001] International Technology Roadmap for Semiconductors

[Embedded Everywhere 2002] http://books.nap.edu/html/embedded_everywhere

[Book of Visions 2001] Wireless world research forum (2001) Book of Visions 2001

Appendix 2. Terminology and abbreviations

Term	Description
Abstraction level	A design can be described at different abstraction levels that are characterised by data types and timing concept. Higher abstraction levels have compact descriptions that hide the details typical to lower levels.
Analogue behaviour	Behaviour of a system or component measured or described in a continuous domain (time, amplitude, frequency)
Analogue HDL	Hardware Description Language for modelling amongst others
API	Application Programmer Interface -- a set of functions that facilitate programmers in using today's complex software programmes
Application domain	see Domain
Architecture	Overall design of a system. An architecture integrates separate but interfering issues of a system, such as provisions for independent evolution and openness combined with overall reliability and performance requirements. An architecture defines guidelines that together help to achieve the overall targets without having to invent ad hoc compromises during system composition. An architecture must be carefully evolved to avoid deterioration as the system itself evolves and the requirements change. [Clemens Szyperski, Component Software]
Asynchronous	Different activities in a system are not synchronised by a common clock signal that generates the exact time instances of computation.
ATM	Asynchronous Transfer Mode
Behaviour	Describes the relations between inputs and outputs of a (part of) the design. The behavioural aspects concentrate on what must be designed, free of implementation aspects
Behavioural compiler	A behavioural compiler interprets the behaviour of a system that is described in a formal language, generating an implementation at a lower abstraction level.
Bit-error rate	The non-negligible probability that an unwanted bit reversal occurs during transmission and (de)coding of a message in a digital communication system
Bus	An element in the architecture, which allows communication between components that are sending data and other components that receive data. Many components may be sending on the same bus, but not simultaneously.
Capturing	The process of collecting all information that is required.

Term	Description
CASE	Computer Aided Software Engineering
Co-simulation	Simulation of a system by co-operating different simulation kernels for the different semantic domains which are part of the system model.
Complexity	The number of independently interacting items or possibilities in a given context, often the number of elements in a set that can be described independently.
Component	A component is a unit of composition with contractually specified interfaces and explicit context dependencies only. Context dependencies are specified by stating the required interfaces and the acceptable execution platform(s). A component can be deployed independently and is subject to composition by third parties.[Szyperski]
Component-oriented programming	Encapsulation+Polymorphism+Late binding+Safety
Computational model	The type of mathematical relations or formulae used as a basis to constitute models for describing (sub)system behaviour.
Concurrent engineering	The act of designing a system by several independent parties acting simultaneously or quasi simultaneously and whose contributions have to be co-ordinated.
Configuration management	Keeping track of different implementation options, with their mutual dependencies and partial results.
Content	The data that is of direct value to the user
Contract	Specification attached to an interface that mutually binds the clients and providers (implementers) of that interface. Contracts can cover functional and non-functional aspects. Functional aspects include the syntax and semantics of an interface. Non-functional aspects relate to quality-of-service guarantees. [Szyperski]
Core	An implementation in silicon, programmable, hardwired or anything in between that allows it to be used as a building block of a system.
Dataflow	Computational model which can execute completely on the basis of the availability of data to its operations.
Design environment	The entire suite of software and hardware used by system designers, ranging from workstation operation systems to dedicated design tools and libraries
Design flow	Keeping track of and providing guidance for the execution of different management design steps in a suitable order
DFL	Data Flow Language. A language from Frontier Design Company oriented towards data flow. The language combines functional as well as procedural elements

Term	Description
Discrete behaviour	Behaviour of a system or component measured or described in a discrete domain (clock cycle count, digital word values, finite state graphs, etc.).
Discrete event	A time-value pair describing a change in value of a signal of a system at that instance of time. In discrete event systems value and time must be countable
Distributed processing	The processing is handled by multiple resources, normally operating in parallel and of which the geographical location is arbitrary
Domain	(roadmap) An area of interest in which products/services share certain characteristics
Domain papers	(roadmap) The vision written down and worked out in scenarios of rendezvous
Driving application	A product/service that challenges technology capabilities in a domain to the utmost, even to the extent that it might imply the need for not-yet existing technologies
DSE	Design space exploration
Dynamic data flow	The production and consumption of data values (tokens) and the execution of operations depends on the actual data occurring at run time. Normally this originates from "if-then-else" constructions in the behavioural specification and complicates the scheduling problem
Embedded memory	Memory implemented on the same chip as the processing elements (as opposed to off-chip memory) and fabricated in a process technology optimised for embedded logic circuits
Embedded logic	Logic which is fabricated in a process technology optimised for memory circuits
Embedded software	Software belonging as integral part to a system, which is normally not configurable or even visible to an outside user
Embedded system	*Embedded systems are highly specialisable, often reactive, sub systems that provide, unnoticed by the user, information processing and control tasks to their embedding system*
EMC	Electro Magnetic Compatibility: the immunity or susceptibility of the system for undesired electromagnetic field interference either due to mutual internal effects or interaction with the environment
Emulation	Replacing part of a real system by a simulation of its model while maintaining the communication with the real system

Term	Description
Encapsulation	Enclosure of a part of the state space of a system such that only operations enclosed together with that part can effect state changes on that part. Typical units of encapsulation are objects, classes, modules and packages [Szyperski]
EPROM	Electrically Programmable Read Only Memory
Estimation of performance	Getting data on the physical behaviour of a system or device based on timing, power consumption, heat dissipation, signal propagation, etc.
Event	(roadmap) Something that happens in some point in time
Expertise span	The number of different disciplines required to be covered in the design process
Finite state machines	Behaviour is described using states and transitions between states and
Framework	A particular architecture for a system.
Hardware	Implementation of a system into a physical device
Heterogeneous	A composition from parts with a different technological origin, as for instance a different logic or physical structure.
HW acceleration	Acceleration of system simulation by using dedicated hardware support for a fast execution of the system model
Intellectual Property	The legal ownership of the knowledge incorporated into a specification or (partial) design, which can represent significant commercial value
Interface	Abstraction of a service that only describes the operations supported by that service (publicly accessible variables, procedures, or methods), but not their implementation. [Szyperski]
IP	see Intellectual Property
JAVA	An object-oriented programming language of which the execution is possible on a variety of operating systems due to the concept of virtual machine compilation
Latency	The time delay between an input event and its corresponding output reaction. In this context caused by computational, by on-chip communication as well as by storage delay
Library	A collection of design descriptions, possibly at different levels of abstraction, which are not specific to one particular design project, but often specific to the technology of the supplier

Term	Description
MATLAB	A high-level software package in which applied mathematicians, signal processing- and control engineers can input and test their ideas
Module level	A module is a system or circuit with a single, well-defined and unified function
Multi-processing	Distributing the workload over several processing elements which can operate concurrently
Multi-processor architecture	A system in which there are several processors capable of running software independently. Different architectures are often distinguished in the interconnection scheme to enable inter-processor communication
Multi-rate	Different parts of a synchronous digital system operate at different clock speeds and where the different clock signals must have a fixed phase relationship with each other
Multi-tasking	Various tasks requested by a single user are executed at the same time
Object oriented software	An approach to computer programming that emphasises data and attaches procedures as "methods" to the classes of data for which they are relevant. In object oriented software everything in sight is an object which belongs to a class. Objects are classified in a hierarchy where each object class may be a subclass of a higher one. The class definition consists of a list of constituent objects called "attributes" and a collection of procedures that can be applied to these objects. Encapsulation+Polymorphism+(implementation) Inheritance
OMT	Object Modelling Technique: a methodology for designing and implementing a (software) system in an object-oriented way. It may replace SDL
Operating system	The software a computer runs to manage its resources: display,
Polymorphism	The ability to view different kinds of entities through a common projection [Szyperski 1998] A function shaped in one from many ways; e.g. either in software or hardware
Platform	The computer system and architecture used to design software (sometimes used to specifically indicate the operating software)
Platform architecture	The maximal (and preferably optimal) superset of functions and blocks that are part of the platform, designed with a certain application domain in mind. The goal is to find the commonalities between various designs, while still being able to create differentiating products.

Term	Description
Platform design	The activity of defining a platform architecture plus design environment to be able to create instances based on the same theme. Therefore, it's more from the same!
Platform product	A product instantiated from platform
Power modelling	Devising mathematical formulae or algorithms to predict the energy
PROGRESS	Program for Research on Embedded Software and Systems
Protocol	Describes the mechanism used for communication. Many different levels of abstraction are possible
QoS (Quality of Service)	The non-functional aspects guaranteed under a contract [Szyperski]
Re-targetable compiler	A compiler for software that transforms a program expressed in a higher level programming language to instructions of a given machine that is highly dependent on the machine architecture
Re-use	Modules (or cores) that are not designed for a particular design project, but are obtained from a library or from other designs
Real-time	The ability of a system to guarantee that actual latencies remain
Real-time kernel	A part of the operating system that is responsible for (run time) scheduling, resource management and for synchronisation between tasks in such a way that real-time deadlines are met
Rendezvous	(roadmap) Event where technologies meet that are necessary for the emergence of a new generation of a product/service that fulfils a user need
Run time scheduling	Scheduling is deciding the execution moment for subtasks or operations as part of the containing task. The decision freedom is normally constrained by a prescribed partial ordering and by limited resources. In run time scheduling this is decided during the execution of the containing task
Scenario	(roadmap) Sequence of events
SDL	Specification and Description Language: a general purpose description language for communication systems, standardised by the ITU (International Telecommunication Union) and widely used in telecommunication
Simulation	Creating behavioural traces of a system by interpreting or executing its model
Simulator	A software utility capable of performing simulations of properly described systems

Term	Description
Single processor system	System in which all computation is performed sequentially on a single processing element
Software	Specification of behaviour or procedures in an appropriate textual format (code) and which is to be interpreted, compiled and executed by a computer-like system capable of interpreting the software code independently
Specification	Each step in the design process starts by defining the specification. A specification must present all the information necessary to execute the relevant step and must cover both behaviour as well as architecture (partial or complete)
Spectral techniques	Analysing system behaviour by representing the system data as well as transformations on these in the frequency domain
Static data flow	The production and consumption of data values (tokens) and the execution of operations does not depend on the actual data values as encountered with run time scheduling, and thus can be completely analysed at compile (design) time which results in a fixed (static) schedule
Sub-system	A set of modules that are interacting in an independent way and which form but one of the constituents of a larger system
Synchronous	All actions of and state changes in a system occur at points in time indicated by a single (global) clock
Synchronous data flow	At each firing of an operation a well-defined number of tokens is produced or consumed (allows multi-rate digital signal processing)
Synthesis	The act of transforming a specification into a more detailed specification keeping the overall external behaviour and meeting physical requirements
System characteristics	Aspects of the implementation of a system, decided upon during its specification or design phase and valuable to reach the overall functional and/or performance criteria
System technology span	The set of design and implementation techniques used to build a system, normally requiring knowledge from different scientific disciplines
Throughput time	The rate at which the system can process input data to output data
UNIX	Multi-user and multi-tasking operating system available to a wide range of computer platforms of different vendors. Originally developed and freely distributed as university software, oriented towards such areas as software development and network interconnection

Term	Description
User needs	The expression, in non-technical terms, of the wishes of (target groups of) end-users that may motivate a search for fulfilment of these needs by product/service solutions in which technology may play an important role
User interface	The part of a system that allows the user to input and understand data, as generated by the system, in an easy way. For the user the interface is the tool
Validation	Checking whether system description satisfies specified properties
Verification	Determining whether two different system descriptions of the same design (possibly at different levels of abstraction) are conformant with respect to relevant functionality
Vision	(roadmap) The description of the common view on the practical evolution of the major function characteristics of product/service solutions that fulfil (some of) the user needs
Web	Documentation stored in central data base accessible through the http internet protocol, allowing for cross-references through links and using text, images, sound, movies, programs, etc. to communicate information.
Windows NT	Operating system developed by Microsoft allowing for a multi-tasking environment
Yield	The percentage of correctly functioning devices after fabrication

Abbreviations

2D	two-dimensional
3D	three-dimensional
A/V	audio/video
ALU	arithmetical and logical unit
ASIC	application specific integrated circuit
ASIP	application-specific instruction set processor
ASSP	application specific standard product
BIST	built-in self test
CAD	computer aided design
CAN	car automation network
COTSES	Common Off The Shelf Embedded Software

CPU	central processing unit
DSE	design space exploration
DSP	digital signal processor
EDA	electronic design automation
EDAA	european design and automation association
EMC	electro-magnetic compatability
EMI	electro-magnetic interference and isolation
ESR	embedded systems roadmap
FDI	fault diagnosis
FPGA	field programmable gate array
GALS	globally asynchronous, locally synchronous
GNU	GNU's not unix
GP	general purpose
HW	hardware
I/O	input/output
IEEE	institute of electrical and electronic engineers
ILP	instruction-level parallelism
IP	intellectual property
ITRS	international technology roadmap for semiconductors
MCM	multi-chip module
MEDEA	micro-electronic developments for european applications
MEMS	micro-electro-mechanical system
MoC	model of computation
MPEG	Motion Picture Experts Group
NoC	network on chip
NRE	non-recurring engineering
OSI	open systems interconnection
pcb	printed circuit board
PWA	personal well-being assistant
QA	quality assurance
RC	reconfigurable computing
RT	real-time
SDL	system description language
SIA	semiconductor industry association

SW	software
TDO	timed digitiser option
TLP	task-level parallelism
TTA	time-triggered architecture
TTM	time to market
UI	user interface
V&V	verification and validation
VCR	video cassette recorder
VHDL	VHSIC (very high speed integrated circuit) hardware description language
VLIW	very large instruction set wordlength
VLSI	very large scale integration

Appendix 3. Domain paper: Personal Well-being Assistant: creating a society of well-being

3.1 Domain description

3.1.1 Introduction

'PWA (Personal Well-being Assistant or Persoonlijke Welzijn Assistent) is the name of the first domain to be exercised for its driving potential with regard to the technological evolution of embedded systems. In future other domains may be identified to fill in missing technology areas.

It is a new domain that aims at supporting human beings in their strive for well being. As such it provides interesting opportunities to link fundamental research on human behaviour and motivation from psychology with advanced technological research topics from a.o. physics, micro-mechanics and micro-electronics.

The PWA concept is developed here in the context of embedded systems, but this is not an inherent limitation to the PWA concept. This context originates from the current assignment to facilitate an Embedded Systems Roadmap.

The PWA itself is not an embedded system but an embedding system. From an embedding system can be derived the functionality and the non-functional constraints of the embedded system(s) which are to be incorporated into it.

3.1.2 Purpose of this domain study

The purpose of this document is:

- To create input material for the process of construction of the technology roadmap for embedded systems;
- To investigate the domain in order to help find the most important needs for technology evolution and to identify potential gaps in that evolution for embedded systems;
- To identify important rendezvous of user needs and technology development. Such a rendezvous implies the convergence of needs and technologies leading to the possibility of a new generation or class of products and/or services.

Further purposes are:

- To stimulate development of and experience with relevant technologies for embedded systems;
- To facilitate the discussion between experts on technology roadmaps for the domain;
- To create a common understanding of the impact of embedded systems on the domain.

3.1.3 Rationale for the PWA-concept

The main reason for developing the PWA-concept in the context of the assignment to facilitate an Embedded Systems Roadmap is the following. A product (or service) example in an application domain of embedded systems is needed with sufficient potential for driving and challenging technology evolution and which preferably also has some relevance for society. It is also necessary to develop a product concept that is not forbidding at some point in time any of the participants in the roadmapping process to stop contributing because of fear of having to show any company confidential future product plans.

Looking into evolution of existing appliances proofed to be difficult without a general concept to steer their further development. This now has been found by taking as a starting point the fulfilment of a very basic human need: the desire for well-being. Investigating what this might imply at the personal level for different age groups in different situations has been shown to be a powerful mechanism for discussing the needs in technology progress.

A further rationale for the PWA is the need to make a large leap forward in time and be able to discuss user needs from the far future, 7 to 10 years ahead. This requires a mental reframing that becomes easier by having a concept like a PWA to start from.

The PWA-concept, further explained in the sequel, as a quite general concept, implies that a nearly unlimited amount of functionality could be accumulated within it. It is, however, more probable that a limited number of different versions originate, each one focused on its own target group and its specific needs and desires. It is also most probable that PWAs will not originate from scratch, but will be based on currently existing appliances. These might then be re-focused to serve a well-being purpose of a specific target group.

In the extensive exercises described below, the work context of target groups has not been explored. This might be an interesting topic for a future follow up study.

3.1.4 Relation to Embedded Systems

As indicated above, a large variety of PWA-appliances can be imagined, each of them connected with the outside world and the person wearing it, and performing a specific set of functions for its user. PWAs have to react to all kinds of sensor signals and control all kinds of actuators. PWAs have to be able to communicate speech, sound and pictures to other PWAs and to and from a local or global communication infrastructure. Apart from this the PWAs have to perform all kind of processing functions: from interpreting sensor signals, generating actuator signals, processing speech, audio and video to accessing large databases and processing the therewith related transactions to perform the required PWA functions. This implies a huge variety in architectural needs of PWAs. Some functions need to be realised in hardware, some in software. A large variety of hardware, software and reconfigurable modules can be envisaged to fulfil these needs. The modules or compositions of modules in such PWAs are embedded systems. For cost and efficiency reasons there will be a drive to define as many common modules for a range of PWAs.

3.2 PWA characteristics

3.2.1 The concept of personal

A major characteristic of the PWA is that it is oriented towards the individual. It will be small enough to be wearable or portable and certainly lightweight. Also there might exist a kind of base or docking station to interface with a local or global communication infrastructure. When used in a car the docking station may be an outgrowth of the current hands free phone infrastructure.

It is important that the individual himself/herself is in control. This means switching in and out all the features of the PWA as it is a fundamental aspect of well-being that oneself can control the degree of penetration into the personal privacy at any time. This implies that the individual determines the highest privacy level on which external influence can be executed. And also what level of security is desirable for the different kinds of transactions that can be performed from the PWA. All of this applies not only for configuration at set-up time, but also later on it must be easy to change all kinds of feature preferences.

Another characteristic is that the PWA will mostly be connected to an infrastructure, be it a potential special PWA infrastructure at home, at work, in the car, in hospital, or to a global "general purpose" infrastructure like the mobile phone network or a localisation system like GPS.

3.2.2 The concept of well-being

Well-being is a quite broad concept. It is a term that basically combines in one word a generally felt most elementary human need. It touches upon fundamental existence issues of individuals and covers the range of needs from caring for elementary living needs for oneself and his beloved ones up to and including the self-realisation of the successful professional.

It has therefore widely different implications for individuals, for families, for associations, for society as a whole. This means that it is a good, socially relevant topic to use as underlying principle for the development of appliances that make sense. And this will be reflected in the origination of different types of PWAs.

There will also be a strong cultural element in the identification and priority setting of needs to be supported by a PWA. This might lead to a strong geographically oriented development of different PWAs.

Through the wide scope of the subject it is guaranteed that also a sufficiently wide coverage of technologies will be possible. It will depend upon us to exercise our creativity to specify interesting and challenging problems for technology that might help to increase the well being of individuals and groups in our society.

3.2.3 The concept of assistant

Providing assistance by a PWA will be constrained by some desired properties:

1. The user must be in control: a PWA must be customised to user wishes. It will not be acceptable that there are features in a PWA that cannot be switched off or on by the user himself. An exception to this may be made when special functions are performed e.g. when a PWA is used as a measuring device to establish costs involved in using scarce resources like roads or other common infrastructure facilities.
2. Help or assistance must be available on request. This again implies that the user is in control of telling if he/she wants to be disturbed by its PWA or not.
3. Assistance should not be patronising, but should provide help with respect for the person concerned. This poses quite some social challenges in relation with the elderly.
4. Assistance should be provided in a context-sensitive way. This creates major technological challenges in defining and updating automatically the status relevant for a person with a PWA.

The type of help that can be provided by a PWA may have specific characteristics:

1. Monitoring/measuring.
   Objects can be monitored: gas burner on or off, is a certain person present. A generalisation is monitoring the status of an environment e.g. for safety purposes. Person monitoring may extend from observing absence or presence to monitoring of behavioural aspects. Also health monitoring falls in this category. This may take extremely widely different forms from external observations and measurements to intra-brain measurements with wireless transmission to an infrastructure. And, of course, it would be quite valuable if one could monitor the degree of well being of a person in his/her environment. A further type of help in this category could originate if the PWA gets also an environment measurement function e.g. related to access and payment for use of facilities like roads and other infrastructure facilities.
2. Reminding.
   This subject starts with a simple extension of the current generation of personal organisers to remind people about their appointments. The MediMinder might be a PWA devoted to reminding and actively supporting taking medication in time and in the right dose.
   This could be extended in various directions: reminding what day and time it is now (especially relevant for very old people), reminder for the context: show where somebody is on a map, or indicate by an arrow which direction to take to get home.
3. Advising.
   This will be an area where it is important to show respect for the person being advised. This will be imperative when older people are advised to use a PWA and they cannot fully oversee the consequences of accepting this. They must then be reassured that their privacy is not invaded unsolicited.
   One can think about an advice not to drive if the PWA senses a dangerous situation e.g. finding to high a level of alcohol in the air in the car. Or an advice to no longer continue with a tennis match if the PWA measures a body overload situation for too long a time. A further extension of the advising function might be, in connection with coupling to a service provided over a network, notification that roadblocks are coming up, and then suggesting an alternative route.
4. Intervening.
   The most far-reaching type of assistance is having the PWA taking an intervening action. In the future medication taking might be actively controlled from a PWA. Much simpler functions are already now within practical reach: opening and closing of doors e.g. for disabled, as soon as their PWA gives the appropriate signal. Actuators can be controlled from a PWA, e.g. to switch of the gas if it burns without a cooking device above it. It must be possible to set all kinds of conditions by the user to ensure the user that no unwanted actions will take place.
   In the car the PWA can be used to take action when the driver loses attention for the road e.g. by some unfavourable change in his medical condition. This might be signalled by the PWA to the safety system of the car, which tries to

get the drivers attention before braking automatically. The signal may even be communicated to the safety systems of neighbouring cars.

3.3 PWA classification

PWAs can be classified according to their target group characteristics

1. Age. A number of age ranges can be distinguished that allow a useful clustering of functions, and for which an existing personal appliance exist which could evolve further with a focus on well-being as a PWA:
   - Teenagers: game console
   - Young parents: baby phone
   - Sportsmen: hart rate measuring appliance
   - Young urban professionals: personal organiser
   - Vital 55+: mobile phone
   - Supersenior (70+): wearable electronic alarm

   Other categories of users with specific needs will undoubtedly be formulated over time, certainly when one would also look into other cultures than our Western European culture. But even within Europe cultural differences are so large that different types of PWA may evolve in different countries, be it already with different language support. A major technological challenge will be to optimise (=minimise) implementation diversity.
2. Needs taken care of:

   In relation to well-being a natural hierarchy of needs was presented by Maslow:
   - Elementary living needs: food, water, sleep, sex
   - Safety: protection from violence and natural disasters, health
   - Love: for and from others, belonging
   - Esteem: respect for self, from and for others, influence
   - Self-actualisation
3. Geographical working area

   A distinction can be made with respect to the geographical area where a PWA is supposed to work:
   - Individual wearing a PWA with only personal functions without the need for attachment to a communication infrastructure
   - Home, hospitals, university, disco, company building, sports complex, senior citizen service flat. Each of these may have its own infrastructure for communication with the PWAs of its inhabitants/visitors.
   - Global working PWA, most probably this implies a wireless connection to the global mobile phone networks and their successors.

3.4 User Needs, Technologies and Rendezvous

3.4.1 Overview of PWA rendezvous

The following picture presents an overview of the different types of PWA to which we have paid attention until now. The light blue ones have been worked out in some detail. Many others can still be devised, and may be should be, to better serve the goal for which we have developed them: to find drivers of significant technological change.

A first analysis has been performed to see which technological problems would originate from the realisation of the sequence of rendezvous. This has led to the remarks below. They are not to be seen as an exhaustive enumeration of technological problems. But as a limited list of problems signalled, mainly with the purpose to trigger you, the reader of this domain paper, to read this carefully, and give additions and further comments based on the viewpoints of your own expertise. Our theory is, that by involving a representative group of experts, we can create a reasonable accurate picture of what needs to be tackled in the future, and more or less in what order this needs to be done as well.

As you will see the three PWAs are discussed in three different ways. This will help to stimulate taking different viewpoints and approaches to finding the most important technological problems and gaps related to the implementation of the embedded systems of this variety of PWAs.

The horizontal axis represents the time line over which we want to make the roadmaps. On the vertical axis are displayed for various user groups the starting points of the evolution followed by a sequence of three rendezvous between user needs and technological capabilities, covering the whole period until 2011, or even extending beyond that in the last shown rendezvous. This extension relates directly to the estimated time for developing the complex algorithms and other technologies needed for implementation of the PWAs at that point in time.

From the PWAs shown three families are further worked out in the sequel: the first one as generations of a Parent-PWA, the second one is the generation line for the Yup-PWA and as the third one the Supersenior-PWA has been investigated. Names have been given to some of the PWAs in a generation line to emphasise the evolution in functionality within such a PWA-family.

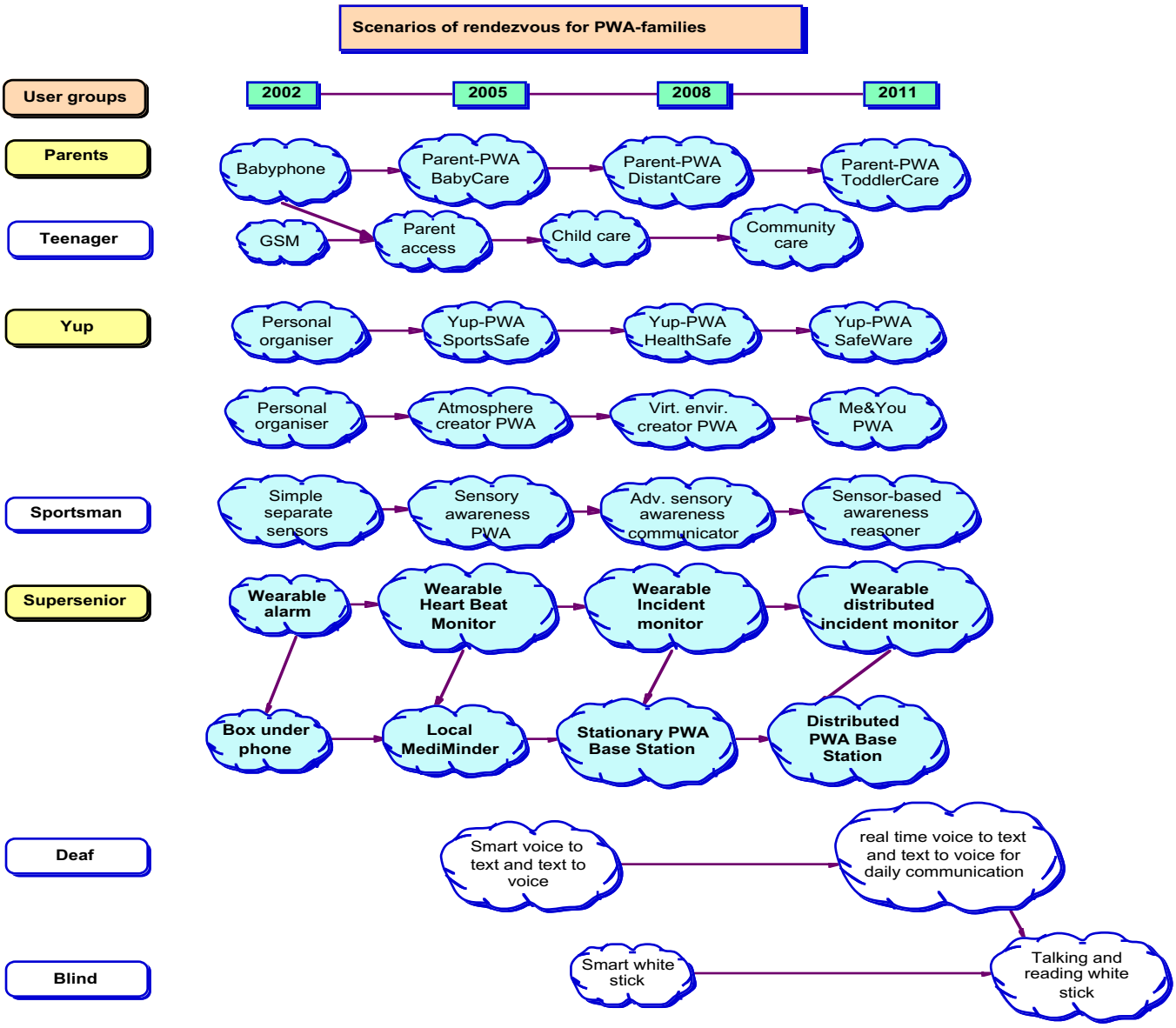


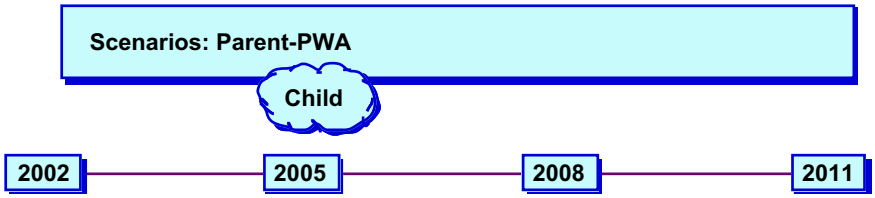
Figure 8: Scenarios of rendezvous for PWA families

3.4.2 The Parent-PWA rendezvous

Introduction

The Parent-PWA describes the needs of the (young) parents that want to be supported in their care taking of a child(-ren). Parents want to be able to monitor their children. There is a clear separation between the device the parent is using

and that of the child, making necessary some kind of communications method. The personal aspect of who is in control shows a gradual shift from parent to child as the child grows up. Though it is oriented towards the individual, it has a number of neighbourhood aspects (baby-sitting), requiring an infrastructure. The well being of the child shows a development of the needs over time that has to be addressed.



User Needs/Functionalities

Young parents have the following needs:
- Monitor the safety of their babies and children while at sleep and playing in bed (vital functions check, food testing & advice, measure simple and useful things to keep their children well cared)
- Keep a (distant) eye on their first movements around the house and in the neighbourhood
- Influence the learning and development process, gradually transfering control of the process to the child, preventing unwanted influences, educational games, archive 'first steps'
- Communicate in a parent community about common problems and solutions
- Make it easier to raise children, combine career with care

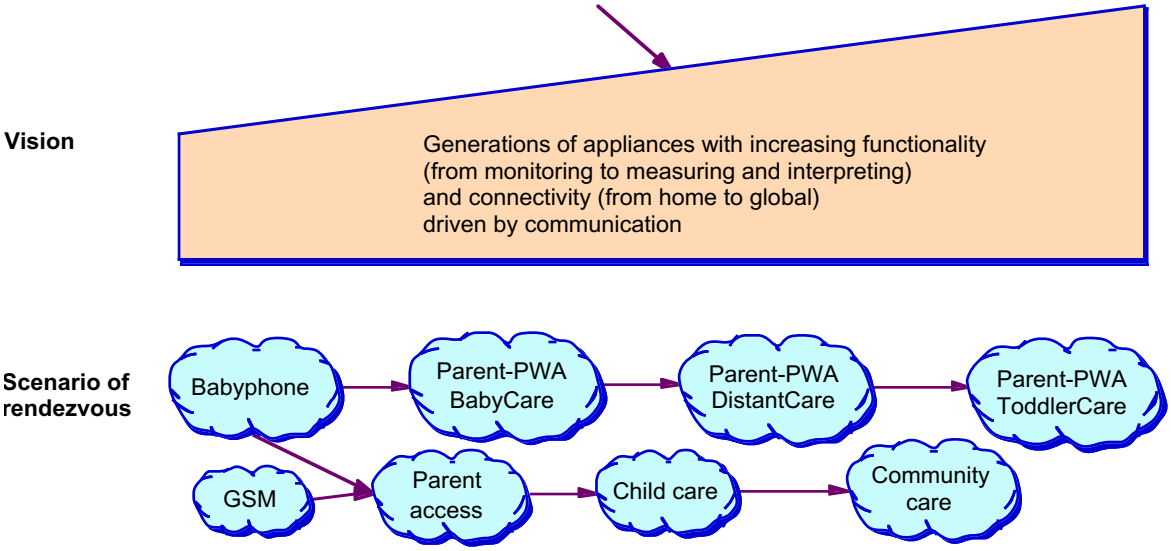


Figure 9: Scenarios for Parent-PWA

	2002	2005	2008	2011
Major functions				
	Increase communication between children, parents and members of the local community. Make help and information more accessible to parents and children. Vital functions check. Tracking & tracing.			
Technologies				
Sensors/actuators	Sound recording	Humidity measurement Warning generated	Body measurements (respiration surveillance)	Emotion measurement (stress,...) In-body
Speech/sound	Amplification Warning = loudness	Sound interpreted Warning signals	Child recognition	Play friends recognition
Image/video	No	Movement detection Movement interpretation	Baby-sitting at a distance Child localisation	Child playgroup supervision Advanced video processing
Info transport/storage	At home Wireless	At home Wireless Low data rate	At home and neighbourhood Wireless to global network Higher data rate	At shopping mall Wireless to global network Video data rate
Software content		Small Configurable	Large Adapting	Very large
Gaps	Security Reliability Perceived radiation effect	Endurance		
Other	Handicapped parents & children?			

'I do not support technology used to encourage fear of strangers or isolation of individuals or family units. Parents and children should be given more opportunities to seek help from those around them. In isolation there are increases in child abuse and under-development' [PROGRESS workshop 2001, author unknown]

Short discussion of application aspects of the Parent-PWA rendezvous

Remarks about problems and technology challenges in some detailed applications in the Parent-PWA rendezvous:

Wetness detection

- Integrate with the baby phone
- The sensor has to be very low cost and very low power.
- The market is extremely large.

- Communication is most likely to be RF-based. The part of the system in the diaper is most likely to be active. The amount of power (electro-magnetic field) needed to allow the sensor to be passive is socially not acceptable.
- We don't know how difficult the integration of the production with diaper production is going to be
- There is also a professional market, aiming at diapers for persons in hospitals and homes for the elderly, allowing higher cost and lower volume
- 'Wetness' is depending on the skin condition of the baby, the capacity of the diaper and leaking. The diaper has to be adapted to these different aspects
- Electrical resistance gives a good indication, but a zero risk is needed before parents will adopt it
- Infrared image is also usable. Very advanced processing is needed.

Location detection

- Access to parts of the house based on the age of the child asks for an infrastructure which is unlikely to be widely available soon (electronic locks, 'ambient intelligence')
- Location detection of people by camera pattern recognition is a long-term issue, there is too much processing power needed for this moment (2020). Integrate other information & sensors.
- Recognition without contact sensors or carrying an apparatus needs integration of multiple sensors and databases to reduce the needed processing power (recognising one out of four people known to be in the building is much easier than having to select them from all Dutch citizens)
- Put a chip in the child's bike (power and space available), integrate with speedometer
- Recognition of who's handling a phone, game boy through finger/voice print is both needed to adapt the behaviour of the apparatus and to identify who's in a room. How to handle privacy? In your own house it is easy, but you don't want everyone's identity broadcast.
- For this application the wireless range needed is limited. Integration of different networks can handle the WAN aspects. The video bandwidth that is needed can be limited by making the frame-rate dependent on movement.
- There's a sub-division between built-in and stand-alone devices. Built-in devices might have to be customised (child's bike is used for a few years and then sold to the neighbours). We need standards for this reconfiguration, both to new users and to upgraded hard/software. (>2007)
- Identification of all devices makes for excellent theft prevention (and a privacy problem)
- Trend: processing moves from 'base-station' to near the sensor.
- All data has to be encrypted (not like the current wireless ether-net), otherwise everyone can listen in on wireless communications. Mechanisms are needed for selective, fine-grained disclosure. The neighbours can look when they are baby-sitting.
- When the video system in the shopping mall is capable of tracking children it is also capable of tracking customers. The number of camera's needed suggests to build them into a lamp.

- When all these devices need a power adapter the number of adapters needed might start to be a barrier to adoption of further devices. A standard would be welcome.
- Children are already carrying a handy (to call mom and 112). A much smaller transponder would be needed for smaller children.
- The localisation problem basically consists of:
- The child is lost or
- Has to come home (for dinner) but is somewhere in the street or
- Is playing with a friend at home somewhere or
- Is not allowed to play in dangerous areas
- A transponder that has to be carried can be ruled out. It should be so small that it can be put in all garments. Then we need ultra-low power and ultra low data rate sensors, using movement as a power source, resilient against dropping/breaking, and finally usable for all items.
- Transponder use is largely an infrastructure question; they can already be built into shoes.
- Security has to be very tight, and adapt to emergency situations

Head lice detection

- Very high resolution camera needed
- Build into each classroom?
- Other sensors possible?

Sudden Infant Death Syndrome (SIDS)

- The detection if a baby is still breathing is difficult. Lots of processing power is needed to do it with (infrared) cameras. Recognising how the baby is lying is critical.
- Parents want to know if a baby lies on its belly. Use +/- 5 camera's to create a 3D image
- The temperature in the close neighbourhood of the baby shouldn't be too high.
- Characterisation of the baby's behaviour is difficult. Individual differences are large and time-dependent
- Recognition has to be very fast, response time < 1.5 minute.
- The life-saving aspects are too difficult for a home-situation. First application in hospitals. Re-animation is an option in a hospital, but not at home. The preventive aspects are worthwhile.

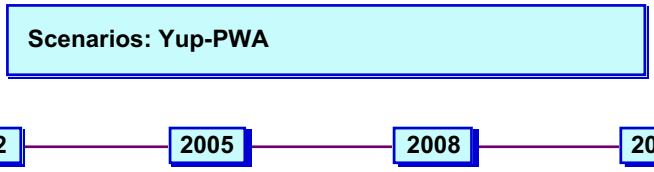
3.4.3 The Yup-PWA rendezvous

Introduction

The Yup-PWA rendezvous are intended to cover the evolving needs of the single, young urban professional (Yup). It is the age range of the young grown-up who has left the parental home, has finished his higher studies successfully and just started a professional life with a bright career ahead. Needs relate to a large extent to meeting the right person, to start building an own lasting social contact environment, to have interesting leisure experiences in sports and entertainment, and to build an interesting and profitable professional life. In terms of well being there is a constant need for safety: feeling safe in contacts with other people, in going out and in more personal appointments, in driving and in having sex.

Even in the desire for excitement and challenges the need for safety will surface in the form risk analysis. Being on the safe side helps building self-confidence and trust in own capacities, and is therefore important for well being. The picture of the rendezvous takes the existing personal organiser as the status quo, and sketches a potential future development thereof with emphasis on providing support for well being of the Yup in terms of providing help in improving safety in all kinds of different situations.

	2002	2005	2008	2011
Technologies				
Sensors/actuators	Touch screen UI Mini-keyboard	Larger colour display, sound+ Stress detection to prevent RSI Sports load measurements Kitchen safety (dustbin monitor) Fertile period indication	Speech-based UI Personal stress measurement Extended body measurements Safe food date labelling Health check of potential parent	Disco emotion measuring for safety Alcohol percentage processing Food safety processing advice Gene check of potential partner
Speech/sound	Single-word recognition	Short sentence input Single word translator output	Connected speech input Text-to-speech output	Sentence translation (1 language) Music recognition
Image/video	No	Web-cam Limited person recognition Personal profile matching when entering disco	High-resolution camera Sport stroke improvement Advanced dating services	High-resolution video Distant street safety check Picture and location display of profile matching person
Info transport/storage	Global via GSM	Global via GPRS Info exchange with local infrastructure	Global via UMTS Info exchange with other PWAs	Info exchange with other types of PWA
Software content	Large	Large	Very large	Very large



User Needs

Single, young urban professionals (Yups) have the following needs regarding:

1. Physiological functioning:
   - Advice on safety in foods, eating and drinking, especially when going out or during exotic travelling
   - Reassurance of safe home and environment upon entering neighbourhood
2. Safety:
   - Safety in car driving
   - Safety in scheduling and meeting interesting people
   - Safety in doing exciting things
   - Sports performance without overload
   - Smart advice on safely dealing with financial situation
   - Support for clever shopping
3. Love and belonging:
   - Safety in having sex
   - Guidance for finding interesting partners
   - Obtaining emotion support from peers
4. Esteem:
   - Participate in any activity with trendy visibility
   - Obtain peer recognition by fashion
   - Assistance to improve social status
5. Self-actualisation:
   - Support of learning processes (Improving self-knowledge, self-trust, self-image and influence)
   - Assistance in self-improvement: communication skills

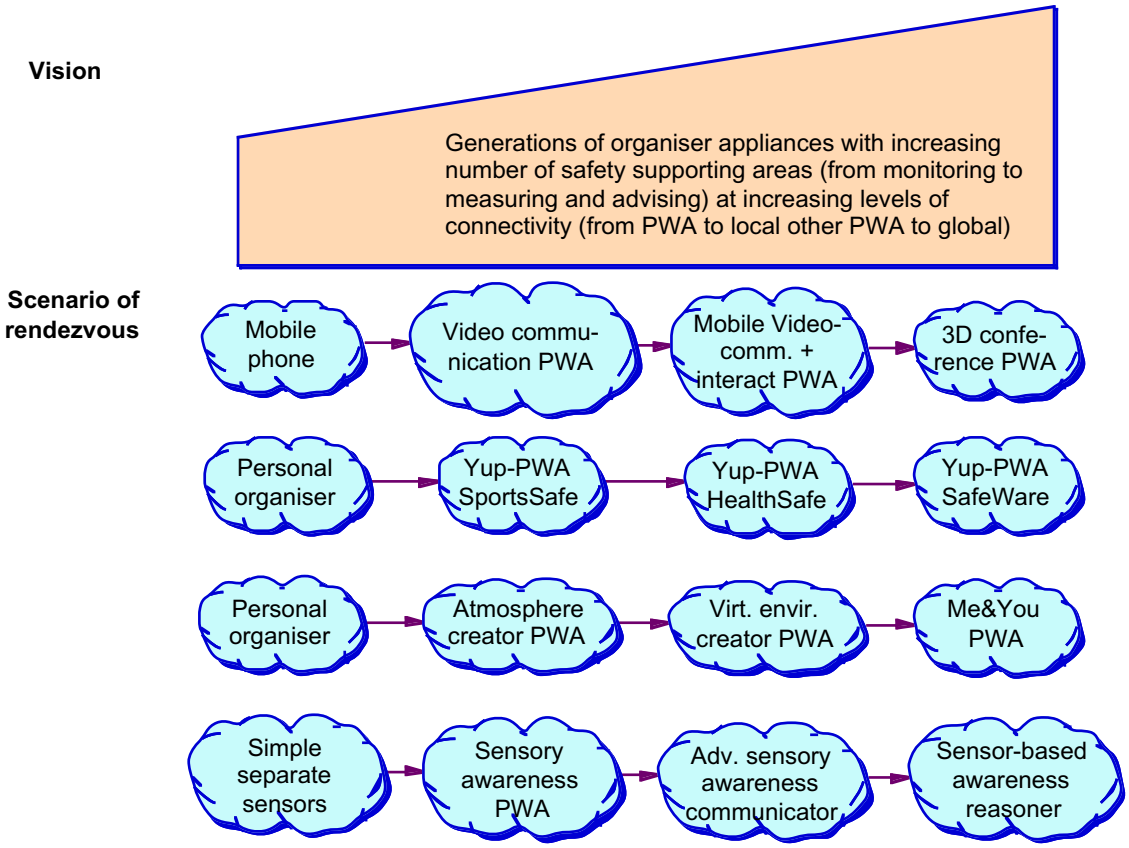


Figure 10: Scenarios for the Yup-PWA

Short discussion of several aspects of the Yup-PWA rendezvous

Remarks about problems and technology challenges in the Yup-PWA rendezvous:

User Needs

Focus of this PWA is strongly on safety. An interesting combination might be found in the not unusual desire for challenge and excitement in this age category e.g. in the context of safety during travel in high-risk countries. This would put mainly high demands on the infrastructure for diagnosis at a distance, access to massive knowledge databases, etc.

A further explicit user need might be help in assessing risks of whatever nature they may be.

Sensors

Major technology problems stem from smallness, light weight and low power needs:

- Sensors need to be on the body or in clothing: attachment problems
- Sensors which need to measure continuously pose a problem with how to attach to what part of the body and with their power supply
- Sensors need to become wireless for connection to the PWA
- Will there be sensors that transmit to more than one PWA or to the PWA-infrastructure?
- Sensors need a unique identification of themselves and of the PWA to transmit to; this to avoid abuse of private information
- For security reasons sensor communication to its PWA needs to be protected by encrypting the measured signals
- Sensors will develop in two directions:
  1. Lower cost, very cheap stickers, one time programmable, major problem is sufficient power
  2. Smarter: more processing within the sensor
- MEMS can be inserted into the veins for diagnostic purposes to reach optimal performance in sports by measuring sugar content in blood, thickness, acidification, etc. Not yet available/feasible for consumers, but there is a professional market for people doing sports professionally.
- A major problem is that the development of sensors takes place at a rather low pace. This may become a technology gap that hampers further evolution of the PWA domain.
- Measuring stress is probably still a research topic: it is not clear what to measure, and how to process the measurements to a useful feedback signal
- Lowering stress requires first recognition of stress from physical signals (probably measured directly in the brain) and building up a database with results. These can than be used to give advice to help prevent stress. Anticipation of stress in social contacts from measurements is still far away
- Emotion measurements can probably be derived from analysing voice behaviour, from gesture recognition and brain activity. All these are still major unsolved problems

Speech/sound

- Major problem is still to acquire a high quality sound without acoustical interference from the environment and from which disturbing sounds (noise) can be removed. Environment compensation is non-existent in its widest

	scope; some noise cancelling processing is available for fixed acoustic environments • Major problems still exist in speaker recognition: recognition of the same speaker under a wide range of conditions (normal voice, voice when having a cold, voice when tired, voice when gasping for breath) is not yet existing, and is essential for user friendliness. • Idiom translation seems to be possible. It is however not clear if this can be done for a sufficiently low price • Many functions require advanced ways of pattern recognition • Music recognition is seen as a problem of pattern recognition and fast access to massive parallel databases. Strategies for making smart problem partitioning are not yet available
Image/video	• Major problems are, apart from the necessary algorithm investigations, may be mostly in supporting technologies like the optical creation of 3D images • Person recognition is seen as a combination of pattern recognition and fast massive parallel database access • Stroke improvement is mainly a pattern recognition and interpretation problem, for which probably not yet algorithms exist • Distant street safety warning signals is a subject that is in its automatic version far out: it is a problem of pattern recognition over time and the corresponding interpretation algorithms. There is currently not much known about this area • When developed street safety warnings might develop further as a service for which a subscription fee can be bought for a certain region • Many PWA functions might be offered as subscription services • Safe driving may give rise to an interesting set of person-oriented measurements: alcohol, degree of tiredness, sleep, in general decreasing attention to what goes on at the road. But also more general interpretation of other driver's behaviour in traffic might generate useful (=well-being increasing) signals: entering a road giving figures about degree of loading of the road, percentage of drivers with anomalous (=dangerous) behaviour.
Information transport	• All information transport needs to be wireless. Significant extensions of the current capacities are needed to make PWA networks possible • Sensor communication to the PWA also needs to be wireless, with a reasonable range, speed, reliability and price. This poses major challenges for all types of sensors • Low power communication will be essential • Many of the proposed functions require access to massive databases to search through with a high speed. This poses enormous demands on the communication infrastructure
Software size	• Will increase; reliability a major problem

3.4.4 The Supersenior-PWA rendezvous

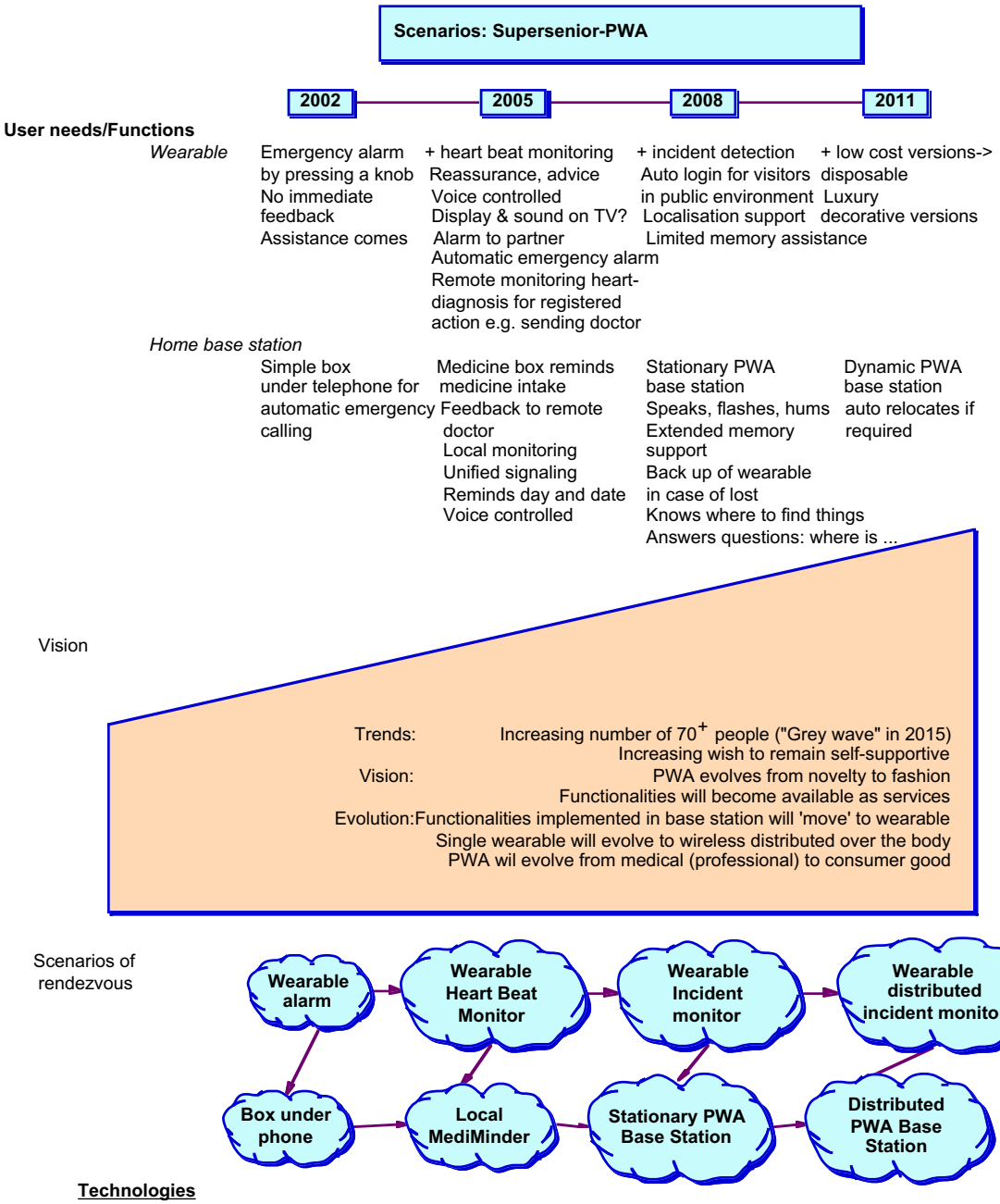


Figure 11: Scenarios for the Supersenior-PWA

Introduction

The general purpose of a Supersenior PWA can be summarised as follows:

- To aid elderly people to live longer on their own
- To support them in a non-patronising way
- This is especially important in the Netherlands because a kind of Delta Plan is needed for the 'Grey Wave' coming around 2015

Description of the Supersenior-PWA rendezvous

2001

Present situation: Emergency alarm by pressing a knob on the device that hangs on somebody's neck. The device sends a radio signal to a box under the telephone, which dials a predetermined number to a service. A human person at this service recognises the phone number and calls one of the neighbours, who has a key, with the requests to take a look. If this phone call fails, the service warns the police or sends somebody to have a look at a certain cost.

Supervision on medicine intake is difficult in the Netherlands for people that live on their own.

At present a significant part of super senior society refuses to actually wear an alarm device. Some are of the opinion that they will be able to walk to it in case of need, which is in many situations not the case.

2004

Wearable Heart Beat Monitor

This device is characterised by:

- Day and night wearable e.g. around the neck
- Water resistant
- Voice controlled (Dutch + dialects, later for other languages as well)
- Simple large emergency knob and voice control for help
- Connected via wireless body LAN to heart beat sensors (low weight)
- Possibility to set a level for automatic alarm
- The device speaks via small loudspeaker or the television which is automatically switched on to dedicated channel
- Gives feedback in natural way, reassurance at predetermined intervals or on verbal request: 'Heart OK?' answer: 'Yes'.
- Asks in case of alarm, permission to call a dedicated service or the nurse or physician. In case of no reaction, it will make the call
- Informs, in case of alarm, partner who may be disabled (deaf or in a wheel chair).
- The power of the device will come from weak electromagnetic field e.g. during sleep? (Battery exchange is out of the question)
- Could be extended with blood pressure measurement.
- Can be extended with diet monitoring (you say what you eat and it warns you if it is not good for you or too much, or you should drink another glass of water.

Local MediMinder®

- Medicine box that provides, monitors and reminds medicine intake
- Voice controlled in mother language (Dutch + dialects, later for other languages as well)
- Connected to service to indicate irregularities, physician or chemist to indicate low medicine levels
- Indicates day of the week date and time in mother language
- General signalling: provides audible and or visible signal (also on TV) that something requires attention like:
  - Fire alarm
  - Medicine intake time
  - Something burning on the kitchen cooker (again)
  - Front door bell rings
  - Email arrived
  - Phone call

2007

Wearable incident monitor

Same functions plus:

- Incident detection: may be in combination with TV cameras that analyse images and detect incidents. No storage of images.
- Auto login for visitors in LAN in elderly peoples home
- Auto login of GSM based PWAs of owners that want to be under surveillance also in public areas.
- Localisation support to find direction to home. A compass like needle points direction to go
- Limited memory assistance
- Extendable with unit for early epilepsy warning

Local Stationary Base Station

MediMinder has evolved into Local Base station. Same functions plus:

- Speaks. Simple conversations in real time. Can wake you up, or tells a joke, reads the headlines of your favourite newspaper via Internet or a bible text.
- Back-up for wearable, if lost in toilet
- Extended memory assistance:
  - ‘’Where are my keys’? You need to tell it were you put things before or vital things like keys could be labelled and localised by the Base Station who gives direction indication.
  - ‘Who is having birthday today?’ or ‘What is the weather?’

2010

Wearable disposable incident monitor

Same functions plus

- Distributed over the body
- Low cost versions
- Some parts are disposable

- Luxury decorative functions (in 2001 you have ballpoints for 0,1 Euro and 500 Euros)

Distributed Stationary Base Station

- Extended with simple natural language conversations
- Helps to find things

Short discussion of several aspects of the Supersenior-PWA rendezvous

Remarks about problems and technology challenges in the Supersenior-PWA rendezvous:

User Needs	• Dynamically configurable functionality. Configurable to circumstances and development of the needs of the user: Requires special user interface and dialogues
Sensors	• Battery powered in the beginning, Later with low-power ICs powered by external EM fields • Acceleration detector in wearable • For unified signalling (all relevant messages come from one place): front door bell, phone, fire alarm, cooking plate, email arrived
Speech/sound	• Simple conversation requires knowledge and handling • User friendly may require new technologies for sound interpretation • User friendly may require new technologies for speech interpretation. To be solved are: distinguishing words, recognising words taking into account subject and context, real time • Missing today: Reliable voice control with natural feedback • Recognition of emotions e.g. panic (from pitch?)
Image/video	• Continuous design problem: what to realise in the wearable and what in the Base Station? • Video processing to analyse images and recognise patterns. Much to be done on image pattern recognition • Problem: how to offer visual information: required adaptability by the user and use of standard television as display NB: retina projection will come far after 2010 • Pattern recognition: distinguish between someone has fallen down on the floor and someone who is cleaning the carpet
Information transport	• Continuous design problem: what to realise in the wearable and what in the Base Station?
Other	• Auto test with feedback ‘Wearable is OK’ • Base station could become distributed as well • Auto login (when in another elderly people home) challenges: • No protocols exist to login in arbitrary network • Important to choose for an existing network like GSM

3.5 A day in the life of William in 2011: a letter to an old friend

John, you asked me to describe a day of my life, so here I am at 76.

Today is June 12th 2011. My PWA base station greeted me cheerfully this morning with the voice of my wife. You know, when Dora was still with me she programmed it for fun that way. But now I don't know whether she was aware that she might pass away before me ……

I took the hart pill that the PWA offered me and prepared to go for a walk.

These PWAs are so clever today. I don't know how, but it knows - perhaps because I moved my stick - that I wanted to go out. 'Darling', it said, 'I advice you to take my small assistant with you when you go out'. So I took the small wearable PWA out of it and put it around my neck. 'Are you fully loaded?' I asked and it said, 'Yes'.

This assistant is very small and lightweight and it monitors my hart. It speaks to me every ten minutes to assure that every thing is OK. In the past men refused to wear these things, but it looks like a piece of jewellery. And when I call 'Dora', I see her photograph in its display for 10 seconds, but that is enough. I get immediately access when I call 'Help', 'Doctor' or 'Misses Pride' from the elderly peoples home were I live. I feel very safe. Before I got it, I was quite afraid and limited by the painful remembrance of a severe hart attack.

Another nice feature is, that when I call 'Home' an arrow appears in the display window to indicate the right direction. It is like a compass needle. Its indication is independent of how you direct the device. It is not very precise but good enough to return into the neighbourhood to where I recognise the houses.

I went home to drink coffee in the lounge, after the nice walk. The PWA watches my diet too. You say to it what you eat or drink and it responds. At lunch I met Peter, an old acquaintance. We spoke about holidays and he wanted to see some photographs. So we went to my apartment and I put the small PWA back into the robot. I know, that the doctor said that I should remain under vigilance but I feel sometimes a little childish with it. I asked the PWA base station: 'Where are the photographs from holidays in 2000?' He answered: 'In the electronic photograph display'. Just by asking some questions we found the photographs from Kautenbach in Luxemburg.

After he left I took a nap. In the afternoon I awoke by the unified signaliser function of the PWA: it signalled that there was something to pay attention to. It was a phone call from one of my sons. Unified signaliser is an improved version of the previous unified messaging. It calls my attention to gas that remains burning sometimes (!), TV programs I could want to see, who is at the front door, and what medicines to take.

When I took my glass of sherry, the PWA reminded me of my diet, but I said 'Shut up' and took another glass. The nice thing of it is that it never loses its temper, like the cleaning woman does, who accuses me of being dirty in the kitchen.

After dinner I watched the television program with the Elisabeth Concourse for piano. I have recorded it in my home mass storage and the PWA can help me to find it back.

Nothing can replace Dora, but I am a lucky man not to be too much dependent on my family or others at present. By the way, it seems that Sony is preparing a PWA in the shape of their well-known dog. It follows you everywhere in the house. At night it will sleep at the side of your bed. It would be nice if it could fetch my slippers.

3.6 Reference

1. A.H. Maslow, ‘A Theory of Human Motivation’, Psychological Review 50 (1943): 370-96.

Appendix 4. Domain of the Embedded Systems Designer

4.1 Domain description

4.1.1 Introduction

We discuss in this paper the design of Embedded Systems. In terms of the needs of this field three areas where needs originate, may be distinguished:

- Needs that originate from the nature of the Embedded Systems themselves
- Needs that stem from performance demands and non-functional constraints placed upon Embedded Systems
- Needs that come from the wishes of designers of Embedded Systems and which relate to methods and tools desirable and/or indispensable for the design of Embedded Systems

In this domain paper we will focus on the last mentioned category of needs. The PWA domain paper is intended to cover the second category of needs. The first category of needs is not yet covered.

Apart from the needs that originate from within the field of design of Embedded Systems, we will have to take into account needs that derive from:

- General trends in society related to individualisation, globalisation, mobility, safety and security, fashion sensitivity, changing composition of households and population
- General trends in business and business models: flat organisation, focus on core-business, multi-site/multi-company co-operations, e-business, shift to services
- General trends in embedding systems: more functionality, higher complexity, more technologies involved
- General trends in technological areas necessary for the development and production of Embedded Systems: Moore's law (semiconductor technology: CPUs, memories, ASSPs, FPGAs, etc.), communication capacity growth, databases, display technology, sensor/actuator technology, MEMS technology.

Where and when these trends create dependencies for Embedded Systems technologies we will have to indicate that with linkages in the Embedded Systems Roadmap.

4.1.2 Why Embedded Systems exist

Embedded Systems are a new term for and an outgrowth of the compositions of individual components that still make up most of today's (sub-)systems-on-a-board. The components used are standardised in several aspects and are often programmable. This makes them widely applicable and reliable for use in (sub-)systems design. The term Embedded System has come up to distinguish the flexible, reactive electronic data processing part of a total system solution (the embedding system) from its other subsystems. Technology progress enables to implement complete hardware/software subsystems on a single board or chip. High-volume Embedded Systems applications are often implemented nowadays as Systems-on-a-Chip (SoC).

Making a Roadmap devoted solely to Embedded Systems implies that the Embedded Systems area represents a special field of technical activity that deserves a special treatment. Embedded Systems thank their existence to the Embedding Systems that incorporate them. The importance of the Embedded Systems field derives from:

- Economy of solution by using to a large extent standard hardware and/or software components, even in high-volume consumer-electronics applications
- Economy of solution provided by platform sharing over many applications
- Flexibility of solution by programmability and/or (re)configurability
- They have shown in the past to be able to make profitably use of the increasing economies of all needed technologies

4.1.3 Classification of Embedded Systems

Embedded Systems can be classified according to several criteria. For the purpose of a Roadmap the following classification helps in reaching a useful structure for the Roadmap:

- Application domain: speech, audio, video, control
- Implementation technology: hardware, software, hardware/software
- Functional performance: real time throughput, etc.
- Non-functional constraints: latency, power, cost, etc.

4.1.4 Purpose of this domain study

This domain study serves as a trigger for discussions in the Core-Team meetings and the Embedded Systems Roadmap Workshops about what are major stumbling blocks for increasing the design productivity of Embedded Systems designers. This subject has, of course, to be broken down in much more detail to come to practically useful statements about what needs to be done in R&D. For that purpose we will discuss the design flow of Embedded Systems in general. Subsequently we will draw conclusions on the needs for specific tasks at specific levels in the total design trajectory. This should be further analysed as to what that means for linkages between Roadmap subjects.

4.2 Characteristics of Embedded Systems Design

4.2.1 Characteristics of Embedded Systems

Embedded Systems are characterised by the following properties:

- They are a subsystem of their embedding systems
- They provide information processing services to their embedding systems
- They are reactive, i.e. they interact with the physical environment of their embedding systems
- They provide usually a complex functionality to their embedding system
- They are mostly not visible or directly accessible by the users of the embedding system

The following sketch serves to make our choices clear with respect to what belongs to the Embedded System and what not. As there is some arbitrariness in these choices the purpose is also to avoid extensive discussions on the topic of Embedded System definition in our Workshops on the Embedded Systems Roadmap.

There are many application domains for embedding systems, and embedding systems may contain many embedded systems. Embedded systems themselves may also contain other embedded systems.

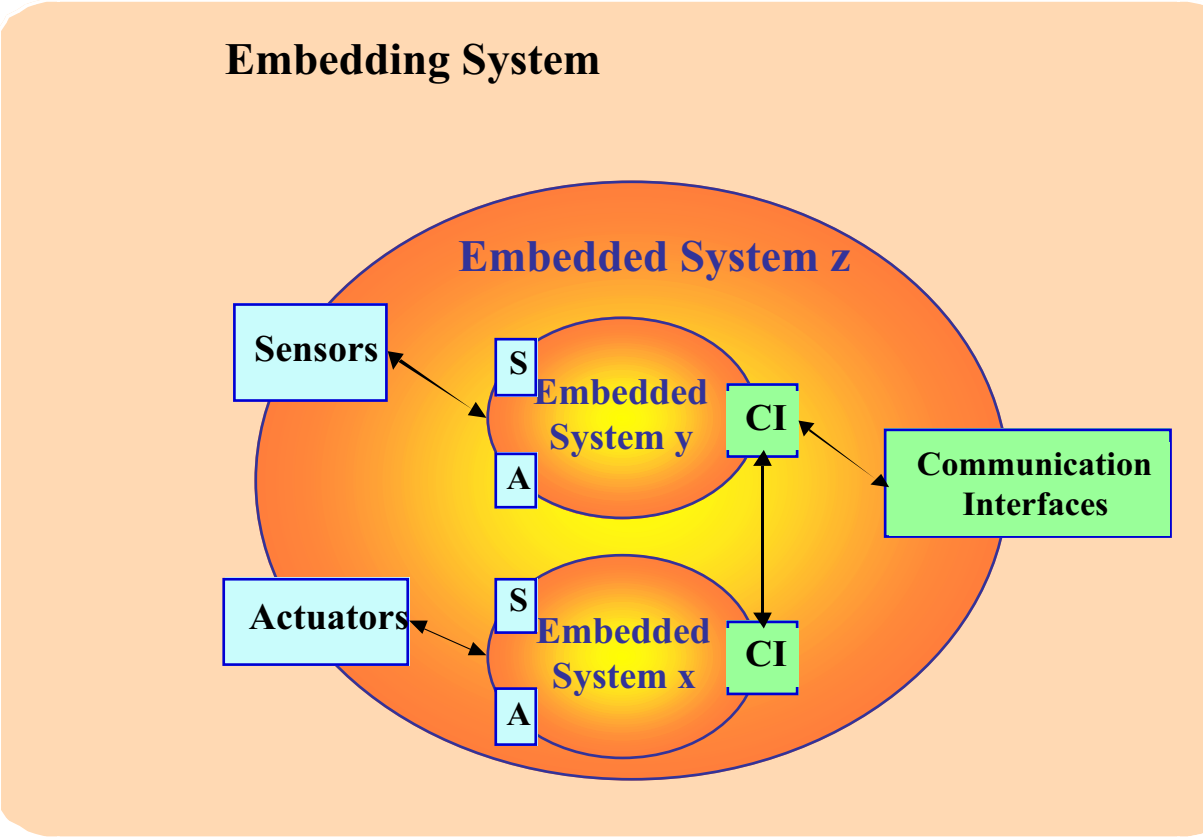


Figure 12: What belongs to an embedded system?

The picture below shows visually some of the positioning and major characteristics. It is not shown that an embedding system may contain more than one Embedded System. The focus in the picture is on the interfaces between the Embedded System and the embedding system and its environment. It gives a clear indication of our choice that all communication to the external world takes place via the embedding system. The direct interfacing between the processing functions of the Embedded System and the embedding system e.g. via shared memory access or message passing, is only suggested by an arrow and not further detailed. Direct interfacing between several (hierarchical) Embedded Systems of one embedding system is not shown. Not indicated is that an application domain is usually covered by a large variety of embedding systems. Also other functions and user interfaces that an embedding system itself might provide are not indicated. Despite all these missing elements this picture has been quite helpful in bringing clarity in some Core-Team discussions.

Three important themes play a major role in Embedded Systems:

1. Interaction (by hardware and software) with the external world of sensors, actuators and communication networks.
2. Processing of data with the performance and under the constraints imposed by the embedding system

3. The design of Embedded Systems. This last topic is the major subject of this domain paper for the Embedded Systems Roadmap.

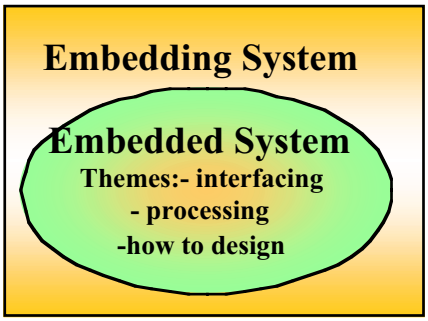


Figure 13: Embedded systems themes

4.2.2 What is special about the Design of Embedded Systems?

The design of Embedded Systems is special due to the following aspects:

- Many non-functional constraints
  - Strong influence on design objectives and architecture
  - Low cost (being invisible to user)
  - Low power (mobile, wearable)
  - EMI and EMC: electromagnetic interference and compatibility
  - Hard timing constraints (real time response, A/V)
  - Reliability, robustness and safety (restart impossible, autonomous)
  - SoC, Size, weight, …
- Specialisation and customisation of target platforms
  - Possible by detailed application know-how
  - Challenge is how to maintain some degree of flexibility
  - Related is the wish to increase the reuse of hw and sw components
- Distributed co-operating embedded systems
- Use of many disciplines and heterogeneity of applied technologies

4.2.3 A Design Flow for Embedded Systems Design

Design proceeds by top-down refinement, level by level, using bottom-up defined models and components, alternated with bottom-up composition/integration and verification. At each of the decomposition/refinement levels a mapping is made of the required functionality onto the target platform architecture. Subsequent performance analysis gives rise to potential modification of function or architecture choices. The following figure depicts this symbolically in a so-called Y-chart:

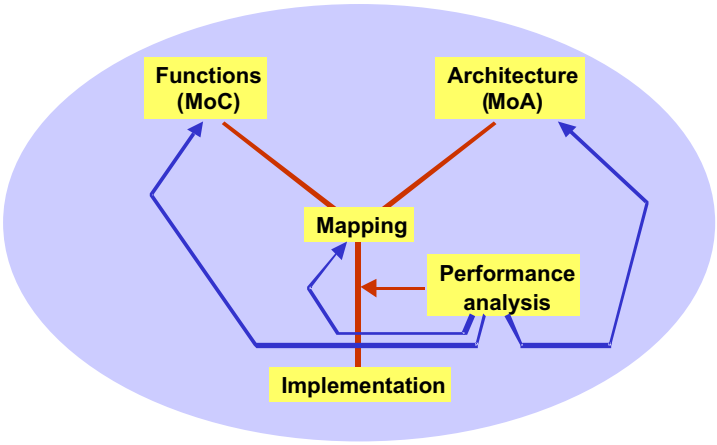


Figure 14: Y-chart

To discuss the progress of systems design with the purpose to derive target platform architecture for a specific domain, the following picture shows the sequence of design steps of the preferred method for such a case:

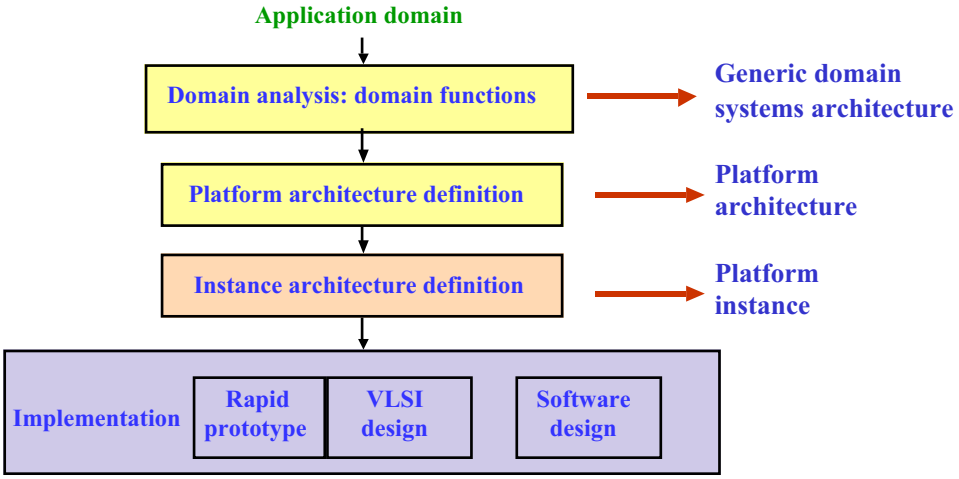


Figure 15: Sequence of design steps

A more detailed picture of an Embedded Systems design flow (the V-chart) is presented in the following figure. It covers, apart from the platform design for a specific application domain, also the design, test and system integration of an instance of the platform architecture, finally leading to a physical prototype of the intended Embedded System. As this concerns only the Embedded System, an integration flow has to be realised for the integration with the embedding system. It is to be understood that at each level a mapping process takes place from function onto architecture (or structure). The green feedback circles symbolise the iterative verification that takes place at each level to check the consistency of input and output specifications of each level. As any design starts by specifying the desired result and specification is both the input and output of any design activity, it is only indicated at the top level. but plays of course a major role in the whole design flow.

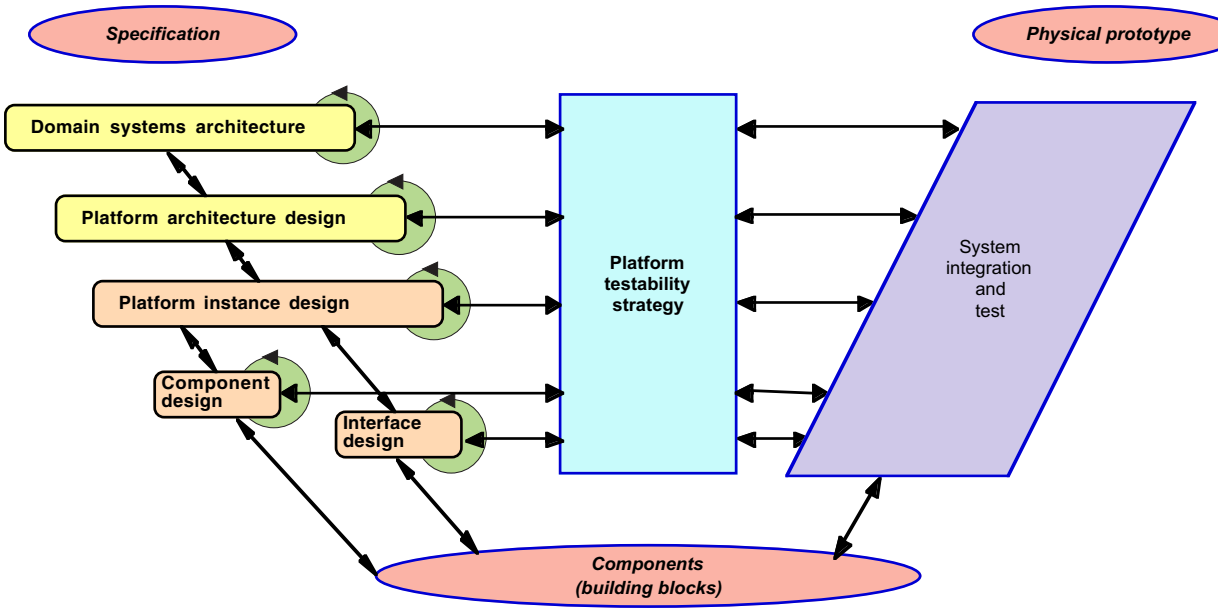


Figure 16: Embedded systems design flow

4.3 Trends relevant for Embedded Systems Design

4.3.1 General trends

For Embedded Systems design a number of general trends are relevant:

1. Increasing individualisation: this leads to more diversity in products and services, and therefore to the need for more flexibility in design, which in turn leads to an increasing software content
2. Increasing fashion sensitivity: this results in shorter product/service life cycle, and therewith in a necessarily shorter time-to-market leading to shorter available design time
3. Globalisation of products and services: multi-site design teams, need for standards
4. Increasing need for safety and security in transactions and communications leads to functionality extensions that must be designed-in.

4.3.2 Trends related to the evolution of Embedded Systems

The design of Embedded Systems is strongly influenced by the following evolutions:

1. Increasing product/service complexity: more functions, more intelligence, higher data rates, more storage: need for reuse
2. Increasing multi-disciplinary solutions and heterogeneity: more digitisation, more measurements, more sensors/actuators, more complex project management
3. Increasing communication needs: more networked (also the designers!), wireless, internet
4. Shorter life cycles: shorter time-to-market: decreasing available design time

5. More mobile/wearable devices: strong need for power reduction.

4.4 Vision on Embedded Systems Design

We are currently experiencing what could be called the ‘embedded system design crisis’ similar to what the software designers have experienced in the past few decades. The software community has been working hard to overcome this crisis by introducing concepts like components (and frameworks), component-based software design and scripting languages. Key issues are re-use (domain engineering - commonality and variability), interfacing, portability and rejecting complexity in favour of simplicity (for reliability), intentional programming.

A similar revolution is needed in embedded systems design. Apart from technological challenges (miniaturised sensors and actuators, low power,...), the innovation must come from software that is made available to the designer to support his design space exploration at higher levels of abstraction. This software should comply with current software standards and provide mathematically sound models, fast retargetable simulators and interpretation and visualisation tools that jointly allow for fast exploration, performance/cost evaluation and decision making. Re-use of software and hardware components and explicit modelling of generic interfaces must be advocated as much as possible. Mastering the complexity of embedded systems design will have to come from enforcing simplicity and imposing some design invariants to avoid state explosion in any possible dimension (complexity, cost, design time, etc.). Abstract design does not prevent the final system from being efficient: while the abstract designer approaches the details from the top, the bottom-level designer makes his IP components upwards available in terms of pre-defined models and parameters. Although ‘a correct by construction’ design methodology is difficult to achieve, we should strive at correctness by construction as much as possible.

Re-use: Re-use is a must. Although components can be seen as black boxes, they must have an explicit interface specification, and the interface must be generic enough that the component can be integrated in many structures. Related issues are portability and parameterisation of components.

4.4.1 Attractive design targets

1. Factor 30 design productivity increase
2. Bug free software design
3. Deadlock free system design.

4.5 Embedded Systems Designer Needs

A number of Embedded Systems Designer needs have been formulated to stimulate further discussion. The list is by no means exhaustive, but just serves to trigger the creation of a proper framework of methods and tools for Embedded Systems Design. The scenarios and rendezvous in the next chapters are a further first attempt to classify and catch the most important and highest priority items for Embedded Systems Designers. There it will be quite important to signal dependencies between the needs in different areas, and the linkages between the technologies that are going to be developed to satisfy the needs.

1. Higher design productivity: easier reuse of heterogeneous hw/sw modules, better formal techniques. Shorter time-to-market and differentiation (distinguish from the competitor) demand a higher design productivity. This goes throughout the entire design process. Reuse is part of this; we need formalisms to enable reuse of hardware and software components. Formal techniques are necessary to allow for verification: checking whether descriptions at various levels of abstraction conform to each other. The more extended use of IP will shift the emphasis within ES design towards the ability to 'knot things together' (to integrate). Again, communication is the starting point for this type of design.

2. Easier handling of flexibility: exponential growth of software: shift from hw to sw, reconfigurable computing. This is a key point in the design of embedded systems for the coming decades. It is a major part of the design space exploration. In the past (i.e. until now, 2001), the issue was to choose what to implement in hardware and what to implement in software. The exponential growth to more flexibility (software) will flatten, simply because of current and future power limitations. However, increasing mask costs for chip fabrication necessitates the possibility for 'subtle' changes in the functionality after fabrication. Reconfigurable hardware offers a trade-off between design flexibility and power efficiency, and will therefore comprise a major design paradigm in the future. With the emerging reconfigurable architectures there is even a third dimension: what to implement in hardware, software, and in reconfigurable parts of the architecture. At the tools this creates a new gap: tools for mapping parts to reconfigurable architectures as well as tools to do design space exploration in the above mentioned three dimensions.

3. Specification methodology that allows reuse of hw/sw modules at embedded system level. This point is related to point 1 above, but it is restricted to the highest levels: the levels of specification.

4. Design capturing at high system level with specification debugging facilities. This takes place at the higher levels of abstraction. Methods and tools are needed to play around at the level of specification, also before doing a hardware/software/reconfigurable partitioning where the same issues of specification recur. At higher levels of abstractions we also need to take the non-functional requirements into account as discussed at point 6 below.

5. Co-ordination environment for different computational models in embedded multi-processor systems. Once point 9 and 10 below (e.g. the concepts) have been found out, it is the next step to build tools that combine the various computational models and allow the designer to reason about the complete system without bothering about the specifics of the separate formalisms. This comes after points 9 and 10. 5. This is also related to the increasing emphasis on integration (point 1).

6. Design space exploration, also at high system level, taking non-functional constraints into account. In order to do that efficiently, with less pressure on the designers' intuition and experience, power and cost models are required for the various potential implementations. This is key for embedded systems, as they have the extra constraints (e.g. size, power dissipation) that are characteristic for

them. These constraints are known at the highest levels of the design, even before actual specification. We need a way to model these constraints in some way or another in the earliest specification phase so that we can deal with them right from the beginning. The embedded system designer's nightmare is to have the complete system implemented then finding out that it doesn't fit in the box it is supposed to fit in.

7. System level partitioning and allocation optimised for distributed multi-tasking embedded systems. This relates to tackling multiple embedded systems in a complete system. We will need the same set of tools we will have for designing embedded systems, but then a level up, where we combine multiple embedded systems into a new 'super' system (e.g. the embedding system).

8. Compiler techniques for minimum power. There are various ways to interpret this. If we look at compiling a high-level language (e.g. C) to machine operations running on an architecture, one could think of generating instructions that result in minimal power dissipation when running on an architecture. This is a relatively new area, and could be an interesting research topic. If we look at compiling a description of hardware to actual silicon, techniques like clock gating can be done to save power. Quite a lot of research on how to do this is already taking place, and not particular to embedded systems. If we look at compiler techniques for reconfigurable architectures there is a wide not-yet explored field of research, also related to design space exploration on the HW/SW/reconfigurable level. Example: a reconfigurable unit to do application-specific computation in a low-power fashion. The most significant power savings expected from future compiler technology is in reordering memory accesses thereby tuning the application to a given cache size. Extrapolating, one might expect future caches to yield more deterministic behaviour, probably controlled by the application software itself rather than the processor hardware.

9. Verification and debugging of embedded systems with heterogeneous computational models. At different levels of abstraction we have different formalisms with different semantics today. It will be important in the future to have a means of doing verification between the various levels of abstraction. To do this in a structured way, it is essential that the various (heterogeneous) models can 'talk to each other' by means of clearly defined interfaces etc. Research in necessary to highlight the differences in expressive power of the various formalisms, as well as the transfer of relevant information from one formalism to another.

10. Extended formal verification methodologies to include higher level embedded system properties relating to e.g. protocol analysis, bus arbitration and scheduling. This is a specialisation of point 9 to the higher levels of abstraction. It also takes into account the interfaces between the embedded system and the embedding system.

11. Accessibility and flexibility of tools. Apart from what we believe that a designer needs in the future, there is one thing that designers have always demanded: the possibility to interfere with what the tool is doing. The designer often has a pretty good idea what he wants, and the design effort is dominated by the ability to 'tell' the tool what the designer wants.

12. The ability to exploit application specific characteristics in silicon (or reconfigurable logic). Because of the huge power and cost savings associated with hardware acceleration together with the low profit margins in high-volume electronics and the increasing need for power minimisation, designers will continue to exploit application knowledge in silicon. This will be a show stopper for design productivity, unless tool support is offered. Libraries with highly optimised IP designs are only partly a solution to the problem. Although many DSP cores are available as IP, these DSPs only provide basic functionality. In the future we can expect these DSPs to be provided with reconfigurable logic to allow the customer to tune the DSP to his application. Because of the huge power and cost savings associated with application tuning, the customers' knowledge of the application and his ability to exploit that knowledge in silicon will probably comprise the most important competitive distinguishing feature (at the processor level).

4.6 Overview of Embedded System Design scenarios

The choice of areas for the scenarios is based on the Embedded Systems Design Flow, the V-chart. Each of the scenarios exists of three rendezvous. This number is more or less arbitrary, and may be changed in the course of the roadmapping process. It is also possible that several scenarios in parallel can or must be foreseen e.g. in the test and integration scenario where this will be a must to cover both areas properly. In other areas natural dichotomies may exist, or there may be different opinions on what the most probable scenario is. Besides, in several areas there will exist quite different needs that call for different scenarios of rendezvous of different technologies.

In the sequel all scenarios and the rendezvous of which they consist, need to be further detailed as to:

1. What is the current status with respect to methods and tools?
2. What is the rationale for the chosen scenario of evolution: What are the needs, what nature do they have (feature, productivity improver, show stopper)?
3. What technologies, both methods and tools, need to be developed for meeting the needs? How big a job is it? How probable is it that others will solve the problem?
4. What dependencies do exist between the scenarios and rendezvous?

The next page shows an overview of the scenarios generated in the different areas:

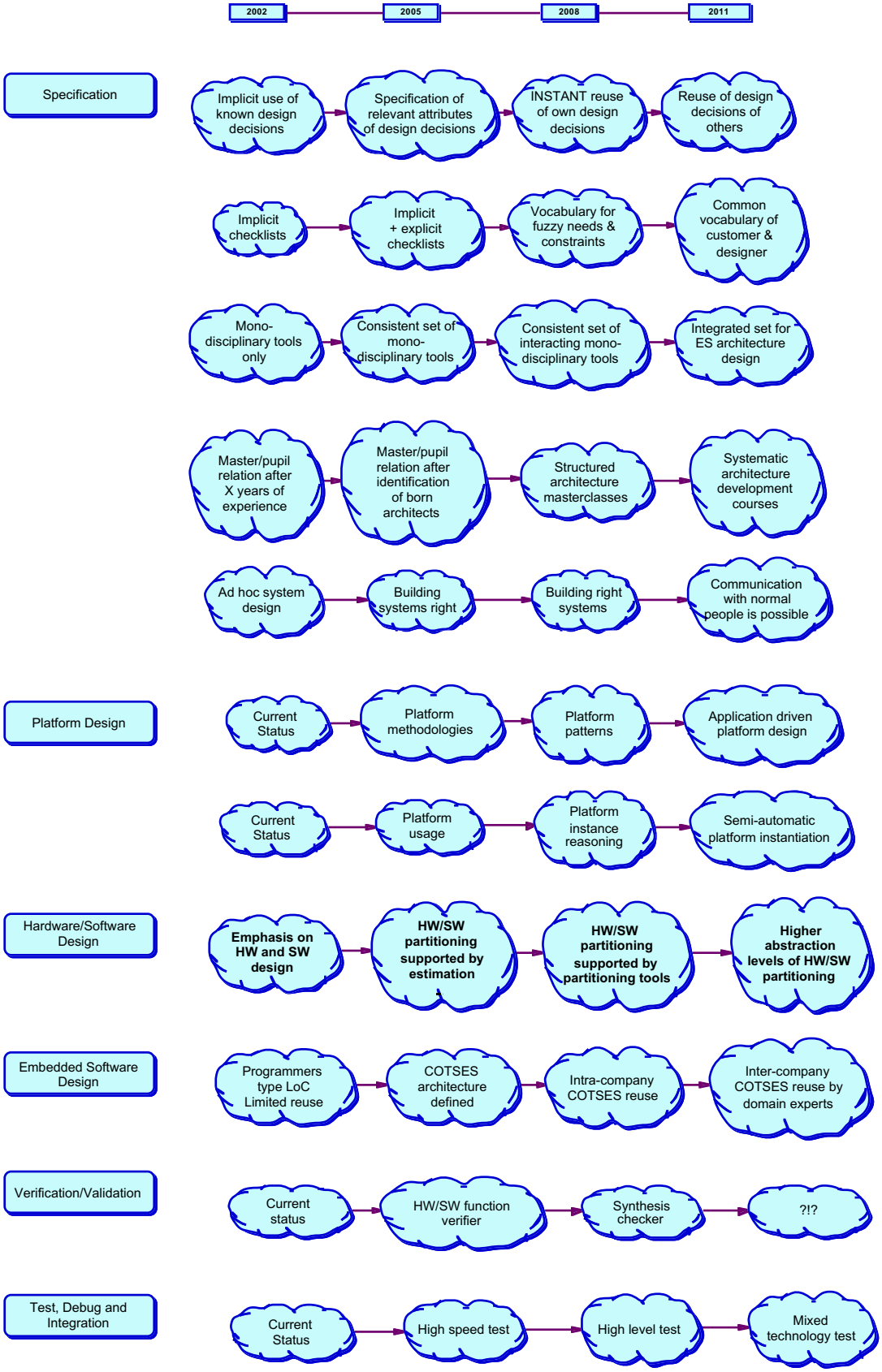


Figure 17: Overview of scenarios

4.7 The Idea to Executable Specification scenarios

4.7.1 Introduction

From idea to implementation.

What do we mean when we say that we want to turn an idea into an implementation? An idea is a conception of a certain application. An implementation gives concrete architectural form to the application. Turning an idea into an implementation comes to a translation of an application specification into an (software/hardware/reconfigurable) architecture description.

There is no unique implementation for a given idea or application, and there is no unique way to go from a given idea to any one of these possible implementations. This observation cannot but enforce some compelling design constraints in order to keep the design complexity manageable and the design cost affordable. To begin with, an application always belongs to some application domain, and it is wise and beneficial to focus on the domain rather than on each and every application in isolation. Thus the architecture will have to be an instance of a domain platform and the idea-to-implementation translation procedure will have to be applicable for all applications in the domain, if not for a set of related domains. Thus, the architecture description will have to be derived from the specification of the platform and, moreover, application and platform specifications must match in order to allow the translation procedure to be cost-effective, both in terms of time and reliability. The conclusion, then, must be that both specifications must be given in terms of models and that the translation from application to architecture must be in terms of mapping the application model into the architecture model.

The introduction of a platform that should serve as a common implementation basis for applications in a domain requires that this platform, and hence the applications, must be introduced at a high level of abstraction, which in turn implies that there are still many possible implementations for any given application in the domain. However, the total amount of possible implementations of a given application has now been greatly reduced, simply because no applications outside the domain can be considered and no architectures that are not instances of the platform can be taken into consideration. Thus, the models in terms of which the application and architecture specifications are given, as well as the translation or mapping of the former into the latter are all domain specific and start at a high level of abstraction. Of course, none of the possible implementations is abstract and, therefore, we can conceive of an abstraction pyramid, at the top of which resides the idea and whose basis is the space of all possible implementations. The set of feasible implementations, i.e., those that satisfy all constraints that are part of not only the application specification, but also the platform specification and the translation method, must then be reachable from the level at witch the specifications are given, by means of a sound design methodology that takes these specifications down the pyramid.

The above reasoning leads to the conclusion that the path from idea to implementation consists of two major parts. The first one is to take the idea to a specification, the second one is to take the specification to an implementation. The idea-

to-specification part goes from the top of the abstraction pyramid the level where specifications have got the form of models. The specification-to-implementation part is a stepwise refinement part in which decisions based on explorations turn the abstract into the concrete. Each and every transition from one level of the abstraction pyramid to the next one narrows down the design space to a region that is a tighter envelop of the feasible implementations. Models get refined and the mapping of application models into architecture models may change nature whenever the models do. We return now to the specification issue in the next section.

From idea to specification.

An idea is a conception of an application in a particular application domain. An idea may initially be vague, and the fist step to be taken before delving into the specification problem is to get the idea as sharp as possible. This will most probably include a few brainstorm sessions among experts, as well as the construction of some alternative scenario's on the back of an envelope. Once the experts believe that the application is feasible (implementable), they will decide on the choice of scenario to be taken further. Details are too far away to be visible though. We are, of course, not dealing with trivial applications. Trivial applications have trivial implementations. We are also not dealing with very specialised applications. Special applications have special implementations. The applications that are of interest are those that comes in versions, families and generations. They range from wafer steppers, to distributed pay road systems, to x^{th} generation radio telescopes and mobile communications systems, to personal well-being assistants, to intelligent micro, nano and pico robots, to autonomous MEMS co-operating through ad hoc networks, to down-to-earth automotive and multimedia type applications. Taking such applications to reality requires that we be more specific as to what exactly is to be that reality. That is, the idea must be worked out, and a system specification must be conceived and presented.

First, the application domain to which an application belongs must be identified. If no system platform for that domain is available, then such a platform must be defined/designed. This task is crucial and far from simple to accomplish. The platform determines the design space and must be capable to meet the design constraints. These constraints, however, come partly from the specification of the application and partly from the design methodology. It thus follows that specifying an application and defining a platform are part of the design methodology itself that should support these tasks.

Now, what is a specification? A specification of an application deals with the what of the application (the so-called behaviour) and not with the how (which is part of the architecture specification). It also gives certain constraints that must be met by any possible implementation. An application should never be under-specified nor over-specified, whether it is in executable form or not. Let me illustrate this point by means of two examples. The fist one illustrates what could be referred to as the idea is the specification archetype The second example is the counterpart of the first one illustrating the specification is the idea archetype.

Example 1. The idea is the specification.

It is well known that (mobile) wireless channels are unreliable. Moreover, they are the more so, the higher is the bit rate that is riding on the channels carrier wave. How to disentangle those two conflicting requirements or, devise an idea that is word an application. The idea is simple: split up the information data into low-bit rate streams and transmit these steams over multiple channels (using multiple carrier frequencies) so as to maximize the product of combined-channel reliability and information density. The idea is, in fact, so simple that simple calculations suffice to prove the feasibility, and a quick prototyping can pave the way to global markets. No more of a specification is needed, except for the constraint that carrier frequencies should be far enough apart that no cross talk can hamper the success of the idea.

Example 2. The specification is the idea.

Speech, when interpreted as an almost harmonic signal, is in many respect astounding. It is one of the rare physical signals that can be compressed to an extent that matches almost all mathematical artefacts. In less than two decades, compression factors of up to 30 have been achieved, and the end is most likely not in sight. Moreover, leading experts in the field have demonstrated that the complexity of the compression algorithms need not be excessively high, provided they are built on a deep understanding of fundamental, if subtle, properties of the speech signal. As a result, the development of algorithms that squeeze speech signals to the bone requires great care, and these algorithms will, then, serve as a specification for almost all applications that originate from almost all ideas that are versions of one main idea: have voice communication over networks, such as the internet, that are hostile to natural, uncompressed speech.

The ‘the-idea-is-the-specification’ example illustrates how systems can be easily under-specified. An idea is not and can never serve as a system specification, simply because ideas, by their very nature, need to be worked out before they can be implemented as applications. When taking an idea as a system specification, there will most likely be too much freedom left to the designer which may get lost because of lack of precision in the specification. The resulting system may turn out to perform poorly, and this result could probably have been foreseen if the idea would have been taken more carefully to a specification, i.e., if sufficiently many what-if questions would have been considered in the first place. Multi-tone systems (as specified in example 1) have been built and after that, the what-if questions that should have been addressed before, have let to better specifications and beautiful implementations of these (referring to the so-called orthogonal frequency division multiplexing alternative to the multi-tone multiplexing and to the corresponding efficient and robust fast Fourier transform based implementation).

The ‘the-specification-is-the-idea’ example illustrates how a system can be possibly over-specified; ‘possibly’ because – in contrast to the first example, in which the under-specification is unquestionable – in this example, only a potential over-specification is built-in. Whether this executable specification is an over-specification or not depends on which one of the many possible applications – which, as you may remember, came sort of after this particular specification – is

considered. Each one of these applications may come with additional constraints that the designer cannot fulfil because he may get stuck because of lack of freedom in the specification. He cannot pilfer that freedom from the behavioural specification, simply because that part of the specification is not familiar to him and seems to him to be untouchable. Even the developer of the algorithm may not have a clue as to whether more freedom can be given to the designer, because her algorithm is the result of a series of modifications to earlier compression algorithms, developed by her and her colleagues, with the sole objective of incorporating into it the sophistication which was found necessary to reach the ever more compelling compression ratio. It is likely, however, that this algorithm can indeed be turned into a specification that gives the designer more freedom for, if not, one may then have doubts about the suitability of the algorithm in the first place.

So, an application specification must provide all information that is necessary as well as sufficient for a designer to be able to get by.

Application specification together with platform specification may together be called system specification. Recall that both are part of the design methodology that, itself, is depending on the application domain. The implication is that both specifications will have to be given in terms of models. These models will most likely be structured in the sense that they are composed of computations and communications. That is naturally so for the system platform and must, then, also be the case for the application because both must match for the mapping step to be void of hidden artefacts. Thus, the original idea will have to be expressed in terms of such models by decomposing the highest level specification into computations and communications. Such a decomposition must not lead to an increase in complexity. This condition is more stringent than imposing a separation of concerns, it is requiring an orthogonalisation of concerns. There is a behaviour (application) and an organization (architecture), and there are computations (processes, processors) and communications (interactions of autonomous entities). Moreover, all specifications, whether original or derived, must be accompanied by validation metrics and assessments which have to be taken down the design pyramid, that is, which will be refined together with the models and the mappings. On any particular level of abstraction, including the highest one, no decisions should be taken if not necessary, and no complexity should be added if not necessary. And similarly for the constraints. In addition to that, the design methodology must not tolerate any intrusion on its own rules. Heuristics can only be accepted when based on sound arguments and when they do neither complicate nor obstruct the transparency of the next steps in the design procedure.

A design methodology as proposed above is still to be taken to maturity. There are still many issues to be addressed before we are that far. A methodology is a composition of methods and relations between methods. Implementation of models, methods and their relations requires good software practice. It is said that, in the past, the hardware design community has been more successful than the software design community. The time when hardware designers captured gates in logic diagrams is far behind us whilst software developers are still writing statement after statement. However, the fact that embedded systems design is heading towards crises is for a great deal due to the fact that designing embedded systems

has entered a phase in which software design (for tooling the design methods, for example) has become the dominant challenge. Good software practice is where the roadmap should take the embedded systems designers.

In many cases, an application is given in the form of an executable specification. This is almost always in terms of an imperative model of computation. This model matches the shared memory model of architecture, which is rapidly fading away in embedded systems land. Embedded systems are exploiting parallelism beyond instruction level parallelism and, therefore, specifications have to be given in terms of communicating tasks. Turning imperative models into network models is a very difficult task. The from-idea-to-specification part of the design methodology that is described above must be extended with a specification exploration part similar to the implementation exploration part. Defining an architecture platform for an application domain is a major breakthrough in embedded systems design, yet this is focusing too much on the architecture part of the system: It will be necessary to introduce, in addition, a specification platform and an accompanying methodology for the exploration of the application space defined by that platform. This is the only way in which the wide gap between application developers and system designers can be bridged.

There is a final aspect that has to be addressed. All parts that constitute the methodology described must be such that reusability, retargetability and reconfigurability are possible and common practice. This is not only true for the system platform, it is equally valid for the application specification and the methods used to explore, validate and design applications and architectures. Embedded circuit design and embedded software development should not be the task of the embedded system designer at the architectural level. An embedded system designer at this level typically does not know how to design circuits and does not know how to develop embedded software. This is the task of those who design circuit IP and develop software IP. However, for this to work, it is necessary that such IP blocks (software, hardware, tools) be designed in a way that the architectural level embedded system designer can integrate them into his design methodology and system, without having to go to extremely steep and long learning curves. He must get all specifications and validation methods that are relevant to his needs, at all levels of abstraction he needs. This concept is crucial and will only work if designers are willing to accept that it will take time before a policy of maximum reuse will be common usage. Universities should take the lead here by putting their toolboxes and building blocks in the public domain and encourage the embedded systems communities all over the world to make use of the offered objects as much as possible. No assessment can be more effective than an assessment published in the open, public domain. This is the only way in which one can guarantee that re-use is really beneficial. There is still a long way to go before embedded systems designers will have access to a global database containing tools, application and architecture components that can be integrated in design methodologies and designs. Only then will embedded system designers have come to terms with complexity management, time-to-market problems and cost

issues. An embedded system need not be optimal: It must meet the constraints, must have spare capacity to be capable of capturing future application evolutions, and must be reliable, safe and robust.

4.7.2 Scenario: From Idea to Executable Specification

Introduction

There you are, walking back to your office and thinking about the new products your company decided to develop. Only very vague requirements are known, and you, the just appointed Embedded Systems Architect, are filled with lots of questions and very few answers. Two years from now the factory will crank out the first version of the new product line like mad. And you have to figure out how to translate these vague requirements into a real commercial product.

The top specifications are clear: create a Gizmo, performing a number of functions. Of course it also must be networked, low cost and it is not allowed to produce any heat. And a lot of other 'common sense' requirements of course. You know what I mean. And by the way, please present next week a budget proposal for the development of the Gizmo.

Its functions require integrated mechanics, optics, electronics and software. But one level deeper you see hundreds of inputs and hundreds of outputs, which must be connected somehow. Too big to get a grip on.

How does top level design start?

Today, 2001 AD, you don't have much help. How do you become an Excellent Embedded Systems Architect in the first place? The answer is both simple and embarrassing: by learning it from another Excellent Architect. In the Netherlands we call it a 'master-journeyman' relation: you learn it by looking carefully at your master. She can't tell you what makes her a good Architect. She only can show you how to behave like one. It's called 'creativity', 'gut feeling', 'experience' or another immeasurable qualification. Neither words nor metrics exist to measure the quality of a decision of an Embedded Systems Architect. You even don't know if your problems have become smaller or bigger after the decision.

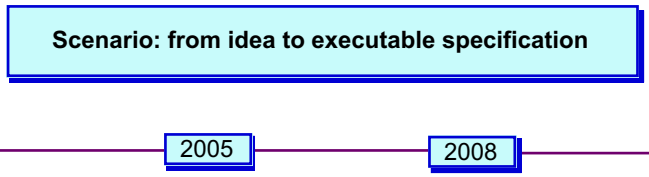
No tools are available to help you in the creative process. You can only compare your decisions with decisions taken which you remember from previous projects.

You need to talk to customers, people of factory, finance, service and people of the project itself with a mechanics, electronics, physics or computer science background. What is the common language all these people understand? It does not exist.

Today's Universities and Technical High schools don't deliver Excellent Embedded System Architects. No, you became an electrical engineer, or a physics engineer, or … when you have worked for a few years in a company, maybe you are spotted by an Architect as a person who has the right background to become an architect. When you are not spotted: bad luck. When you are spotted by a not-so-good architect, bad luck again. So today people, born as Excellent Embedded Systems Architects don't become one because of bad luck.

Specification environment

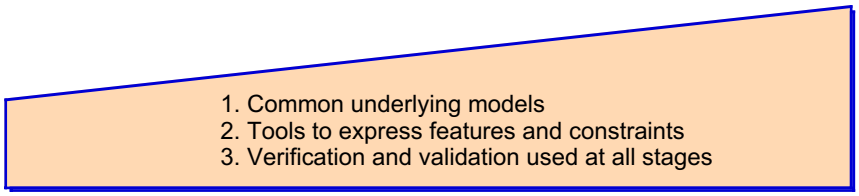
Some useful ideas from the software engineering community can be adopted here. Broadly speaking, embedded system specification can be either imprecise/ incomplete (in plain English) or in terms of an executable behaviour (commonly in an imperative language) augmented with a set of constraints. In the latter case, the imperative model of computation is most likely (at least partially) not the correct model to be used. Parsing from the imperative model of computation to the more appropriate model is something that is to be done. In case the specification is given in plain English, method and tools must be conceived and developed to iteratively derive a precise (if incomplete) specification. Any specification should automatically provide input to a verification/validation tool for the verification/ validation of all behaviour-equivalent models derived from it.



Designer Needs

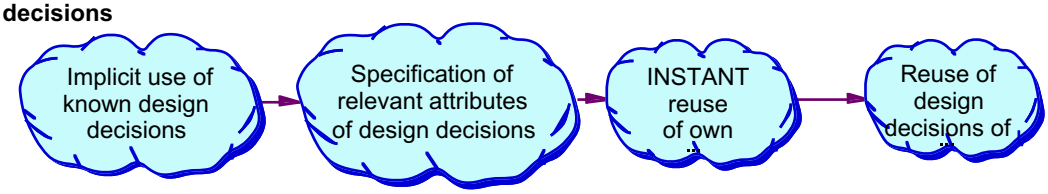
1. Keeping the overview: abstraction levels in specification: "When to leave out what"
2. Specifying all kinds of constraints (time. power, safety, fault tolerance, security
3. View of design options
4. Measure of quality of design decisions
5. Handling of complexity
6. Documenting decisions with their rationale
7. Rough estimate of development costs
8. Interaction with non-technological people and interaction between all involved disciplines

Vision:

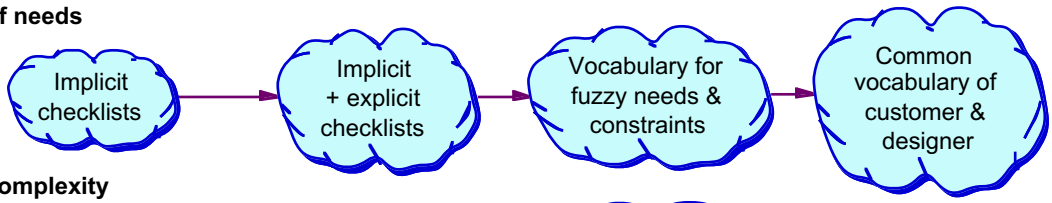


Scenarios of rendezvous:

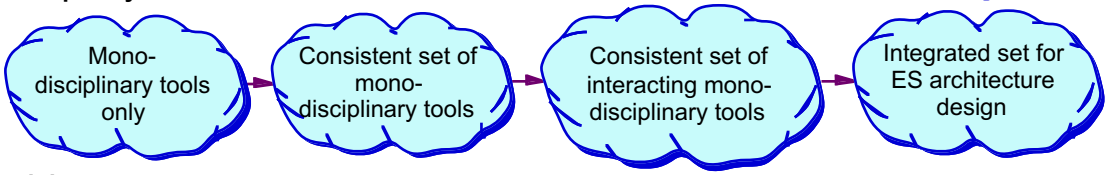
Reuse of decisions



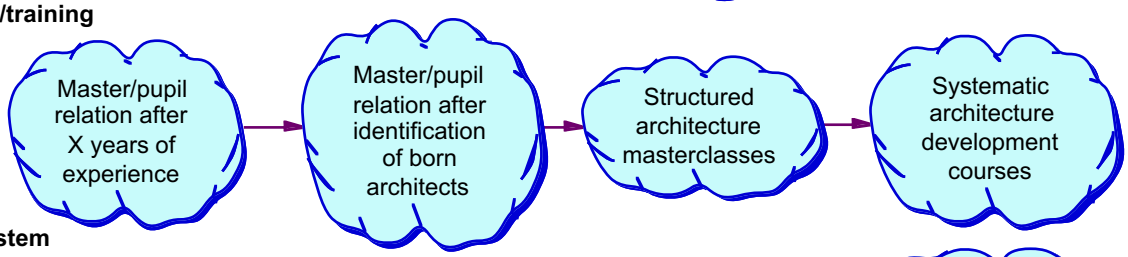
Exploration of needs



Handling of complexity



Education/training



Idea to system

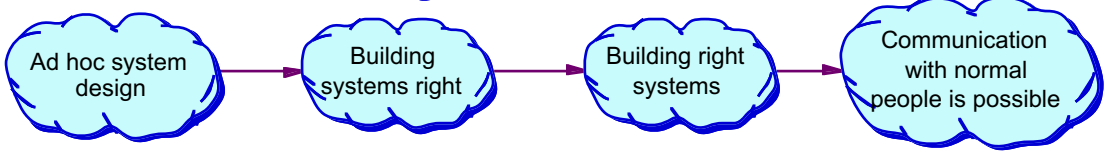


Figure 18: Scenario from idea to executable specification

Major functions			
	Verification and validation of specification Methods that deal with incomplete specs Compositional specification	Analysis of performance characteristics	Visualisation Presentation Interface to implementation tools
Technologies	Formal verification Models of computation Reuse Requirements engineering	Formal methods Performance metrics Bottleneck identification and description	Documentation tools!!!
Methods	Simulators Graphical programming tools Model checkers	Simulators Database of design options	Graphical input/output Precise interface description (2020)
Tools			
Gaps			
Other			

4.7.3 Remarks on scenarios of rendezvous for From Idea to Executable Specification

Scenario: Reuse of decisions

Research is needed to figure out which items of design decisions are needed for later reference by an Embedded Systems Architect.

Interview the top 10 architect/designers of today and find out how they work

A repository must be made where an Embedded Systems Architect can document his formal and informal design decisions to improve the creative process. Later these documented decisions can be used for exploiting analogies.

Also tooling is needed to compute the effects of a design decision. Is a design decision a step in the right direction or not? How to prove? One of the questions is what can be learned from other artists? How do they work?

A repository must be made where an Embedded Systems Architect also can access decisions taken by other Embedded Systems Architects in the same company, or even worldwide.

Scenario: Exploration of Needs (implicit/explicit)

Vocabulary, common for customer and designers, for description of fuzzy needs and constraints. Also words are needed to describe the quality of design decisions.

When we know the words, they can be used in the repository.

Scenario: Handling of Complexity

The goal is to find all relevant boundary conditions and make a design, which adheres to it.

A generic set of tools is available per discipline to help the Embedded Systems Architect with the top-level decisions. Tools do not use each other's data.

The toolbox contains all necessary tools, but without interaction.

An integrated toolbox is available for Embedded Systems Engineers, which supports them in taking the right design decisions at a number of levels of abstraction.

Scenario: Training

Means and methods are needed to spot natural-born Excellent Embedded Systems Architects as soon as possible. Tests must be developed. Masters must be appointed.

As soon as somebody qualifies to become an Embedded Systems Architect, he or she should be exposed to structured master classes, which are given by the best available Masters.

In this stage we expect that Architecture training is still a post experience training.

The art of System Design now becomes knowledge of System Design and knowledge transfer can be incorporated in the curriculum of Universities and Technical High Schools.

4.8 Scenarios for From Executable Specification to Implementation

4.8.1 Introduction

The design flow from idea to final product design can be divided into to parts, the first being the flow from idea to an executable specification and the second being the flow from executable specification to the final design. These two parts of the design flow have quite different characteristics.

The significance of these two parts of the design flow heavily depends on the application. For example, in case of a wide-area traffic control system, the moment at which the executable specification is obtained, only little improvements and optimisations are left. However in case of a signal processing algorithm that is to be implemented in an ASIC, emphasis is on the optimality of the implementation in terms of speed, throughput and cost, while the executable specification is easily derived from the mathematical description of the algorithm. Another example is the area of high-throughput applications with real-time constraints. Mapping of an algorithm on a fixed architecture then is far from simple and often requires much more effort than the definition of the algorithm.

Clearly, there exist no clear border between both parts of the design flow. In practice the executable specification might be on different levels of abstraction, more or less refined and in general in the beginning incomplete. Hence, during the second phase in the design flow there will be a strong interaction with the first phase.

An executable specification is supposed to be written in some executable specification language or programming language. A model that depends on the language used, links the executable specification to the desired reality. In general such an executable specification only specifies the behaviour, i.e. given an initial-

isation, the relation between the stream of input events and the stream of resulting output events. Hence, the specification must be annotated with a number of properties about timing, real-time constraints, chip area, power dissipation, etc.

In this scenario, we will focus on the design flow from executable specification to final design, the implementation, in which we assume that the design starts from an executable specification that reflects the desired (bit true) external[1] behaviour of the system to be designed, together with a number of properties about timing, real-time constraints, chip area, power dissipation, etc.

The flow from executable specification to the description of the final design again can be divided into two parts, viz. high-level synthesis and low-level synthesis. The first includes composition, decomposition, refinement, scheduling and resource allocation. Low-level synthesis includes, logic optimisation, retiming, placement and routing.

The technological possibilities are ever increasing; ever more complex embedded systems become feasible. But, design productivity it not keeping up with these developments. This so-called ‘embedded systems design crises’ can only be solved by more efficient design flows supported by more efficient tools. This will be on the account of less efficient implementations.

An interesting metaphor stems from the early days of computer programming. The first computers were programmed in assembly language. With the growing computing power ever more complex programs became feasible which resulted in high level programming languages, subroutine libraries for generally used functions and compilers. Even these languages turned out not to be sufficient. New programming paradigms, such as object orientation, were developed together with middleware concepts like com-objects. These new programming paradigms increased the software productivity considerably, but on the account of less efficient software designs in terms of computational overhead and storage overhead. Nonetheless, if efficiency is really crucial, all kinds of optimisation tools are available and some parts are still written in assembler.

A similar development is foreseen for embedded system design methodology. However, in contrast with software development, that maps behaviour onto just one architecture, embedded system behaviour can be mapped on a variety of architectures.

Clearly, different tools and design flows are needed for an implementation as ASIC, ASIP, DSP, VLIW or some reconfigurable architecture.

In order to improve design productivity and to cope with the technologies that will be offered in the near future, improvements are needed in the following areas:

- Design tools including compilers
- Languages and design representations

1.We differentiate between the internal and external behaviour of a system. The external behaviour reflects the desired behaviour on the ports of the system, i.e. the relation between the streams of events at the input port and the output ports. The internal behaviour describes the relation between all variables in the system description. An implementation is correct if its external behaviour equals the external behaviour of the specification.

- Design styles

This is clarified with the following statements:

Statements:

For each product group a design platform will be needed. Such a platform supports a particular design flow towards a particular architecture. It will be built from design tools and design languages that are also used in other platforms.

Design tools need to be concise and generally applicable. Dedicated or too specialized design tools tend to be hardly used and therefore will never become bug-free.

Design productivity is greatly improved when the simulation burden is reduced. Tools that are correct, i.e. preserve external behaviour, can obtain the latter.

Design languages or design representations with clear and unambiguous semantics must be standardized. Design tools should operate on these languages instead of on formats that are dedicated to these tools. A design language is characterized by the fact that it represents the design during many stages in the design flow.

Design productivity can be increased considerably by using IP blocks, but only if these IP blocks are correctly specified at the appropriate levels of abstraction. For example a VHDL description at gate level for a processor is not sufficient. A high level description at which the processor together with a program can be simulated must be provided too.

IP blocks must be standardized in a similar way as class libraries are now being standardized for object oriented programming languages.

Design tools that provide correctness-by-construction are more important than similar tools that provide more efficient implementations but are not bug-free.

System on a chip will become reality in the sense that:

- A chip will never be implemented on the bases of single design paradigm. (Area and complexity are becoming too large)
- Communication between the subsystems on the chip will be (quasi) asynchronous and based on protocols implemented by hardware. (Data transfer from one side of the chip to the other will take more than 10 clock cycles)
- Operating systems will become a part of the chip design and will partly be implemented in hardware. (Dynamic process scheduling)

Tools for design space exploration:

Tools that provide estimated information about the area, delay throughput, power dissipation in an early stage of the design flow are highly desirable. This requires fully automated and integrated design tools that automatically generate a design at an appropriate abstraction level in such a detail that these properties can be estimated with sufficient accuracy.

Executable specifications:

The main purpose of an executable specification is to validate the design in an early stage of the design process. This validation will be done by means simulation and possibly by the formal verification of particular properties. Simulation requires a simulation platform that is fast. Hence, it must be based on an efficient language such as C or a derivative of it.

For reasons of flexibility, such a simulation model must be based on a set of communicating processes. The model should be inherently free from deadlock, starvation and flooding, or it must be possible to automatically proof these properties.

Design description languages:

A suitable design description language is needed that

- unambiguously expresses behaviour,
- unambiguously expresses structure,
- is applicable at all stages of the design flow from executable specification to a description in terms of registers adder and gates
- is generally accepted and standardized.

  All design tools should operate on such a design description language.

Compilers:

Compilers for embedded systems will be used on a per-block basis. A system (on a chip) will be built from several blocks each having its own particular architecture and therefore requiring its own compiler.

In the embedded systems design flow, compilers can be divided into two classes:

- Compilers that are used for translating between different design representation formats. Such compilers or translators will always be needed, as a single design representation language that can be applied from an executable specification to the final design description, is utopian.
- Compilers that map a behavioural description onto a particular architecture. These compilers should be able to optimise on the basis of different criteria, such as area, speed, throughput and power dissipation.

Compilers that map the behaviour expressed in a design description language onto a particular architecture will become very important. The only way to improve the design productivity is design reuse and automation of the design process. Compilers used in the software world are translating a high-level programming language into an intermediate language, which thereafter is translated to the object code belonging to a particular architecture. In both parts some optimisation is performed. The latter, however, is limited because both the language and the architecture are of a general-purpose nature.

Compilers for embedded systems should be able to map on different architectures. So they should operate on a program and a description of a parameterised architecture and produce the object code and possibly also produce the parameters of the architecture.

Another approach is to design compiler generators, which start from an architecture and a language and automatically generate a compiler for that architecture, possible with some user intervention.

VLIW, (including Transport Triggered Architectures) and reconfigurable architectures are very promising for embedded system applications. Compilers for these architectures hardly exist or are still in its infancy.

The following remark might be considered as belonging to the scenario 'Idea-to-executable specification' as well.

Multi-paradigm design tools and multi-paradigm design representations:

In many cases sensors and mechanical subsystems are part of the embedded system. The behaviour and properties of these sensors and mechanical subsystems need to be described together with the digital part of the embedded system.

4.8.2 Designer needs and Vision

From the preceding we may extract the following designer needs:

1. Mapping directly behaviour on complex architectures with reasonable efficiency.
2. Less dependency on simulation for verifying the automated design steps.
3. Better design representations.
4. Improved specification languages.
5. Improved reusability of parts of designs.

Summarizing our vision leads to four main points, viz.:

1. Designs will become more complex.
2. Design cost per transistor will decrease on the account of less efficient designs (implementations).
3. Compilers will play an increasingly important role.
4. Compilation will include high-level synthesis.

These items can be linked up with many other; some of are to be considered as recommendations.

1. Design tools become concise and generally applicable at different platforms. They will allow correctness-by-construction. They will operate on standard design representation languages. Less attention is paid to optimising the last 20%.
2. Currently developed languages and styles for writing executable specification will become mature and standardized. Standardization and research is needed.
3. Next to languages for specification and hardware description, a standard design representation that can be used in a large part of the design process, will be needed.
4. Design reuse will become mature, standardized and the IP-blocks will be provided with standard interfaces, specification at different abstraction levels and a test sequences. Further development is needed.
5. Compilers for parameterisable architectures will become available.
6. Compiler generators that generate a compiler from the architecture and the language will become available.
7. Compilers that optimise toward different criteria will become available. Low-power will be more important than area.

8. Communication between the subsystems on the chip will be (quasi) asynchronous and based on (standardized) protocols implemented by hardware. Research is needed.
9. Operating systems will become a part of the chip design and will partly be implemented in hardware. (Dynamic process scheduling) How?
10. Tools for design space exploration will become available (Tools that deliver information about the area, delay throughput, power dissipation in an early stage of the design flow)
11. The designer community will more and more become aware of the need to formalize the design flow and to prove the correctness of the tools.

From the preceding remark the following research areas that need attention can be derived:

- Design tools
- Languages for executable specifications
- Design languages
- Design reuse
- Compilers for parameterised architectures
- Generators that derive compilers from architecture and language
- Compilers that optimise on different criteria
- Protocols on a chip
- Operating systems on a chip partly hardware implemented
- Tools for design space exploration
- Formalization of the design flow

4.8.3 Conclusions

In order to diminish the embedded systems design crises, all eleven areas mentioned above require research and development.

The development of tools for embedded system design depends on a few large tool providers and a large number of small ones. The cost of tool development is that high that it is impossible, even for large design companies, to develop a proprietary tool suite. So new tools will have to cooperate with the existing ones. This will strongly influence the developments in the field of embedded system design in an unpredictable way. This particularly holds for the design flow and the way of design representation. Nonetheless, research in these areas is of utmost importance.

In order to improve design productivity, all kind of compilers and compiler generators as described above are needed. However, currently only compilers for small application areas are available. Fortunately, developments in this area are less dependent on existing tools and design representations, although it must be possible to incorporate them in existing design flows. A particular compiler will only be used for a small part of the system.

Scenario: From executable specification to implementation

2002 — 2005 — 2008 — 2011

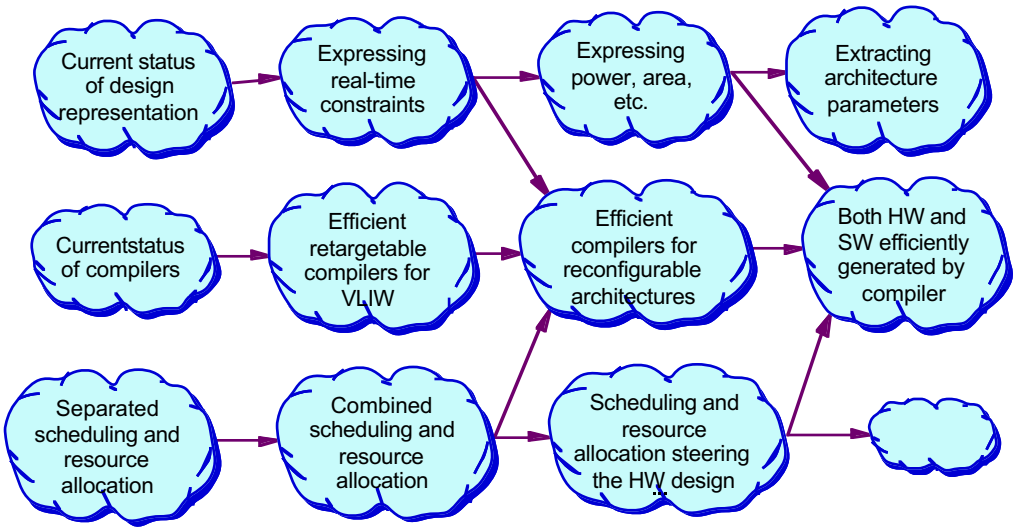
Designer Needs

1. Mapping directly behaviour on complex architectures with reasonable efficiency
2. Less dependency on simulation for verifying the automated design steps
3. Better design representations
4. Improved specification languages
5. Improved reusability of parts of designs

Vision:

1. Designs will become more complex
2. Design cost per transistor will decrease in exchange for faster, less efficient designs (implementations)
3. Compilers will play an increasingly important role
4. Compilation will include high-level synthesis

Scenarios of rendezvous:



Major functions				
Technologies				
Languages for executable specifications	Different proprietary models available and tested	Start of standardization	Standard(s) accepted	
Design languages		Including different models of time Real-time constraints	Including area and power constraints	Means for extracting architectural parameters

Retargetable compilers (for parameterised architectures)	Small class of architectures. Few parameters. Simple architectures.	Parameterised multi-data path multiprocessor.		Reconfigarable architectures
Generators that derive compilers from architecture and language	Start research. Some knowledge available		First research versions available to show the feasibility	First field trials
Compilers that optimise on different criteria	Only number of executions	Real time. Instruction count	Area. Power.	
Formalization of the design flow	Ongoing research, which should get more attention		Formalization of substantial parts of tools is feasible	
Methods	On-going research on formalisation of design flow		Formalisation of substantial part of tools is feasible	
Executable specification languages	Different proprietary models available and tested		Standards for models accepted	Standards in use
Design languages				
Design reuse				
Protocols on a chip				
OS on chip partly in HW implemented				
Tools	Results of only a few tools can be trusted		In the design flow only a few tools can not be trusted	Tools are becoming certified
Compilers for parameterised architectures				
Compiler generators from architecture and language				
Compilers that optimise on different criteria				
Tools for design space exploration				

Gaps	Correctness-by-construction Standard design representation languages Standardization of writing styles for executable specification Standard design representation Further development of IP-blocks Compilers for parameterisable architectures Compiler generators Compilers that optimise toward different criteria Research on on-chip communication mechanisms Critical OS components implemented in hardware Tools for design space exploration

Figure 19: Scenarios for Executable specification to implementation

4.9 The Platform Design scenarios

4.9.1 Introduction

Definition

Platform design is: based on a common architecture in which blocks can be easily adapted, such that in a limited time a ‘certified’ new platform instantiation can de developed, which will differentiate it self from other implementations. Platform design allows for the development of ‘a family of products’.

A platform instantiation contains various hardware and software as well as reconfigurable IP blocks, interconnect between these blocks as well as to the environment (i.e. the embedding system).

Note: what is the difference / relation between digital/analog and ‘platform design???

Why a platform?

Electronic products in the market will behave like fashion (Vision). Products have small derivations in taste, colour, etc. and functionality, but are designed on structurally the same basis.

Designs become more complex (vision); a single designer cannot have the complete overview at the detailed level anymore. Therefore (s)he wants to raise the level of abstraction of building blocks to be still able to create complex design in a shorter design-time. The ultimate goal is to do product creation by push-button platform instantiation (rendezvous in 2010+).

Shorter time-to-market, while still having the opportunity to create products that differentiate in the market place (company need).

4.9.2 Needs and Trends

Trends:

1. Electronics in the market will behave like fashion;
2. Time-to-market becomes shorter;
3. Sub-micron technologies allow for evermore-complex designs.

Needs:

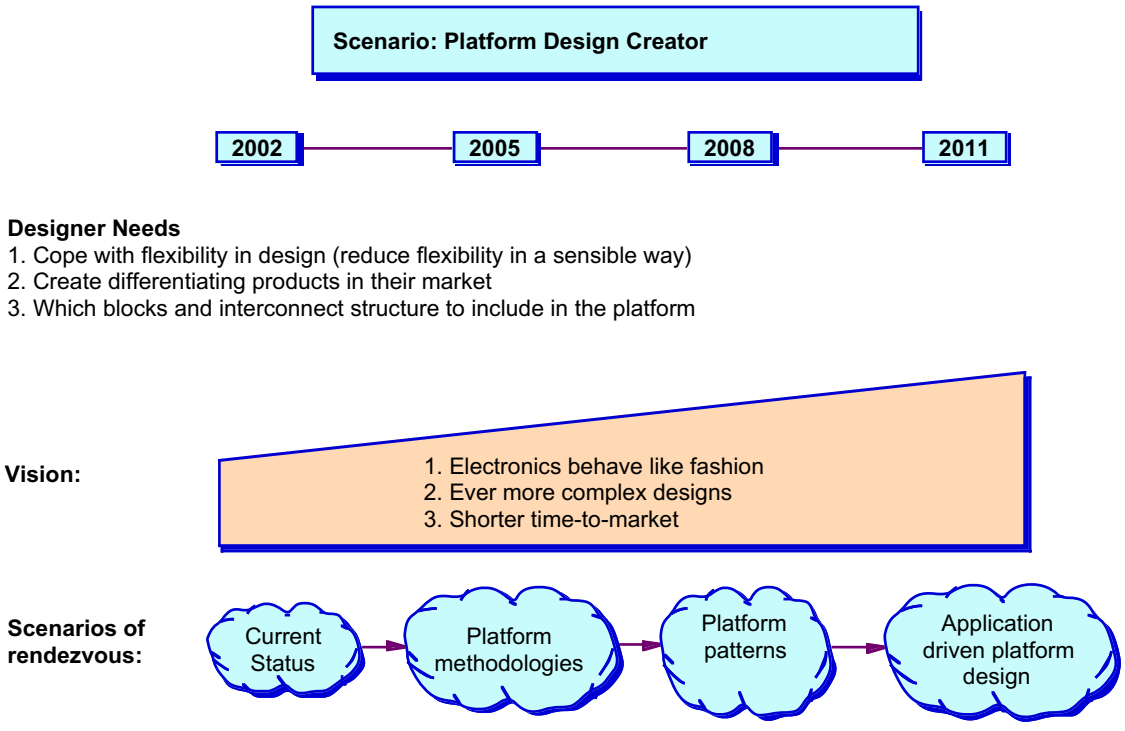
1. Create differentiating products in their market fast:

- Performance (speed, power, etc.) better than product of competitor;
- Features: more functionality than product of competitor;

2. Platform designer need: which basic architecture to select for the application domain of the platform. Which blocks and interconnect to allow in a platform:
3. Platform instantiator need: which blocks to include in a specific instance. Which parameters to select for each block. Also: plug-n-play instantiation.

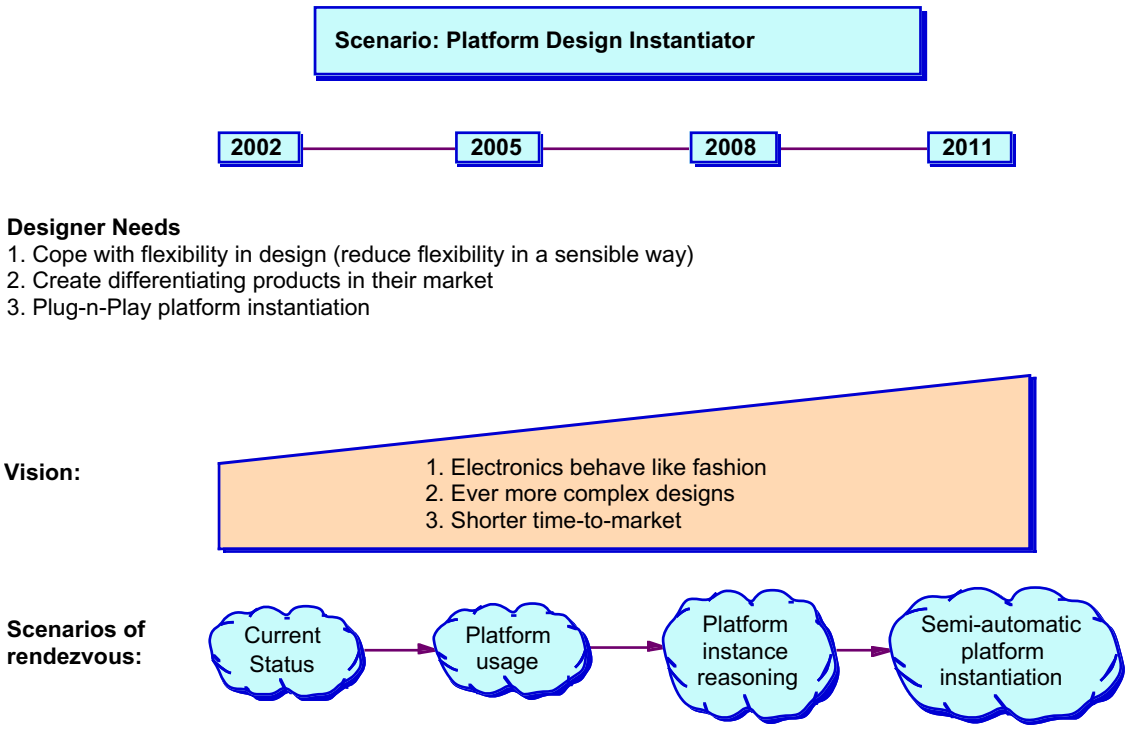
4.9.3 Scenarios for Platform Design

We make the distinction between two groups involved in platform design: the people who create a platform and the people who instantiate a platform. Therefore we have two scenarios: one for the 'Platform Design Creator' and one for the 'Platform Design Instantiator'.



Major functions				
	Getting to know platforms	Standardising blocks in platforms	Standardising platforms	Semi-automatic platform creation
Technologies				
Methods		Choosing the right level of abstraction Guidelines on the use of interfaces Coding rules Documentation standards Models of blocks and interfaces that are based on the same language Raise the level of abstraction of communication between the blocks	Standards on: - Components and interfaces - Coding rules - Documentation Project management keeps the instance(s) consistent with the platform	Automatic assistance in choosing parameters of platform
		Database of certified components	Database of certified platforms	
Tools				Tools produce a proposal of choices in the database of certified platforms

Figure 20: Scenario for platform design creator



Major functions				
	Getting to know platforms	Design trade-offs: manual design space exploration	Structured design space exploration	Automatic design space exploration
Technologies				
Methods		Uniform models of (non)-functional constraints Models for new styles of communication between blocks in the platform Error correction in communication Security issues in platform	Standards on communication interfaces between blocks	
Tools			Tools that reason about properties in an instance, at the instance level of abstraction Tools to check the consistency of an instance with the platform Tools for code rule checks Tools for (automatic) generation of documents	Tools doing part of the design space exploration automatically

Figure 21: Scenario for Platform design instantiator

4.9.4 Remarks on scenarios of rendezvous for Platform Design

The following text discusses for the two separate scenarios, the technology requirements.

Distinction between technology requirements for the platform designer creator and for the platform designer instantiator (user).

Platform Design Creator

Current status (2002): 'Initial discoveries'

Currently there are platforms available, for example the DVP (digital video platform) (Philips). This is an example of a product family. Also, on the processing core level, families of architectures are defined. These developments help us to discover the right levels of abstraction to define a platform on. Furthermore, they teach us that it's not only a matter of defining and using a platform, it is also important to keep instances consistent with the platform (i.e. project management).

Standard interfaces have to be designed by learning from experiments with the starting reuse of hardware IP blocks, software IP macro-blocks.

'Platform methodologies' (2005)

The initial developments give rise to in sights in the right levels of abstraction and detail that should be supported in a platform. This phase should result in proposals of sets of guidelines for the design of interfaces of blocks in a platform, coding rules, documentation standards, components etc. Standardization is needed for some of these aspects!

Design space exploration methods and tools are important to give the platform designer a means of reasoning: which blocks to include in the platform?

Models are needed for the boundaries of the various (hardware, software, reconfigurable) blocks in the platform such that their interfaces talk the same language!

For the interconnect (the communication structure between the blocks and to/from the embedding system), we need to raise the level of abstraction. We should not only talk about the communication implemented by buses, but by the communication protocol that runs on top of it (cf. moving one level up in the communication stack).

Further, standard hardware IP and software IP blocks should be placed in a database site of certified components. This database contains reference specifications for the IP blocks.

'Platform patterns' (2008)

Tools should be provided to automate the aspects that our now in standards due to the previous phase. The standardization on documentation, coding rules, components and their interfaces, etc. is now complete. Project management tools should be available to keep an eye on the consistency between the platform on the one hand and instances derived from the platform on the other hand. In terms of the certified database for IP components, we need a platform-layered database (i.e. certified platforms).

‘Application domain driven platform design’ (>2011)

Once we have established the previous phase, we can think of one level higher up in abstraction, by starting to think about methods that, given an application domain and a set of possibilities for platforms, assist in determining the boundaries and parameters needed for a specific platform. The application domains drive the choice of (IP) blocks in the platform and the decomposition with its interfaces. Given a set of constraints, tools search the database of certified platforms to come up with a proposal for the choice of possible platform usage.

Platform Design Instantiator

The key term in this scenario is design space exploration. An ideal design exploration tool is a tool that accepts as its input a desired performance/cost ratio, and returns the values of all the free parameters in the design platform plus some perturbation/sensitivity analysis. Although this ‘inverse problem’ approach may be possible in some cases, it is not in most cases. Therefore, the current approach is to view the problem as a ‘direct problem’: choose parameter values and measure the corresponding performance/cost ratio through simulation. Of course, there is a relation between simulation speed/cost and the level of abstraction (detail). To bridge the gap between the ‘direct’ and the ‘inverse’ exploration approaches, tools should be made available to interpret the performance numbers output by the simulator, to visualise them and help the designer by suggesting the changes he wants to make in his parameter space (of both the architecture and the behavioural specification). Parameters should be as orthogonal as possible (and design methodology constraints should be non-conflicting). Learning curves should altogether disappear.

Current status (2002): ‘Initial instantiation’

Designers have their first experience with designing instances from existing (often ‘ad-hoc’) platforms. Reuse of hardware and software IP blocks including their interfaces are done. These experiences are vital to communicate to the platform designers to find the right level of abstraction, as well as to think about keeping instances consistent with the platform, project management tools, etc. The basic idea of platforms is that they speed up the design. When this turns out not to be the case then the platform instance designers are the people who can pinpoint at issues that hamper to reach this goal.

‘Methods for platform usage’ (2005)

A big issue is the absence of models to model the non-functional constraints of embedded systems. They need to be modelled and quantified in a uniform model, which enables the platform user to reason about the consequences of choosing specific blocks to obtain an instance of the platform for a specific application.

In general, implementing a function in hardware offers a low-power solution, while implementing the same function in software offers flexibility in design. Therefore, the platform instantiator has a task in making a trade-off between hardware, software, and reconfigurable blocks when instantiating a platform towards an embedded system. Other trade-offs are for example making the design

memory-centric (in which the computational blocks use a single shared memory), or computation-centric (in which all computational blocks have their own memory). The former might be the better choice for control-oriented applications, the latter in data-intense applications like in video-streaming (MPEG).

We need models how to deal with new styles of communication (e.g. asynchronous buses, asynchronous communication between 'islands' of synchronous blocks), and their effect on global system performance and characteristics. For this, we need models that represent hardware and software (and reconfigurable) blocks that have the same communication on the interfaces, such that they understand each other. Also, we need models of the characteristics of communication, like error correction (low-level) and security issues (high-level, especially relevant for communication of data from and to the environment of the embedded system).

'Platform instance reasoning' (2008)

Using models to quantify functional as well as non-functional constraints the designer can do a structured design space exploration. Further, interfaces between blocks should be standardized such that the designer can manually put blocks together to obtain an instance, and then reason about the performance, power dissipation, etc. The knowledge gained by these 'manual' design space explorations (supported by tools) creates the necessary ingredients for the next phase.

Also, there should be tool support for aspects of platforms like documentation (generation), coding rules checks (and automatic semantics-preserving rewriting), consistency of interfaces between blocks etc. Also, automatic checks are needed which keep an eye on the consistency between the platform instance and the rules of the platform itself.

'(Semi-) Automatic platform instantiation' (>2011)

The goal is to have tools that perform (a part of) the design space exploration when instantiating a platform for a specific application automatically, taking the functional and non-functional constraints as well as platform data (which blocks may be used, which interfacing structures etc.) into account. Fully automatic platform instantiation is far in the future. Therefore only first steps to automatic design space exploration are feasible in the time frame of the embedded systems roadmap.

4.9.5 Recommendations

1. Standardization on communication (protocols) architectures and on the interfaces between blocks in the platform;
2. Common model of communication between the hardware, software, and reconfigurable blocks;
3. Database of certified components;
4. Models and tools to do structured design space exploration.

4.10 The Hardware/Software Design scenarios

4.10.1 Introduction

Definition

HW/SW design is: the process of partitioning the specification and deciding for each part, the type of implementation from a spectrum ranging from hardwired (dedicated HW) to General Purpose (Software) computing blocks. Note that this spectrum includes (re-)configurable computing (RC) hardware. Software comprises all programs running on programmable blocks.

This partitioning process is present at two levels: At the system level, the partitioning is in terms of tasks that are allocated to processors. These processors may range from Application-Specific ICs (ASICs) to GP processors. The type of processors and the communication protocols will for a large part be determined by the platform architecture. The other level of the partitioning process concerns 'flexible' processor cores in the sense that a default data path and/or instruction set can be augmented with functional units (ALU, etc.) or instructions tuned to the application (-domain). 'Partitioning' at this level should be interpreted as identifying frequently occurring patterns of computations (within the allocated task) that are suitable for hardware acceleration.

Hardware acceleration can be performed using either dedicated silicon or (re-)configurable logic. The trade-offs made during the partitioning process are affected by

- The constraints implied by previous design steps, e.g. choice of platform, communication protocols, size of FPGA HW, etc.
- Projected market share vs. cost of NRE, non-recurring expenses. This is the one-time cost (vs. e.g. silicon cost) independent of the production volume, and comprises mainly chip mask costs and design effort. The latter will increase with the detail of HW/SW design.
- The availability of design tools and designers with the required expertise.

HW-SW design, for the happy few?

Traditionally, HW design implies silicon fabrication, which is the domain of (large) companies. HW design takes a considerable effort and is justified from either the perspective of:

- Market: a sufficiently large market (High Volumes Electronics, HVE) to justify the cost of NRE
- Design Constraints: performance (e.g. video or network processing) or power consumption (e.g. mobile computing).

A potential evolution from this monopoly is marked by the development of (re-)configurable technology, e.g. a RISC processor core with a limited amount of FPGA for the 'computational kernels' of the code. In the current situation, there are already commercially available stand alone processors augmented with FPGA, and the corresponding tools to do modest HW-SW design at the processor level. This development allows small companies and universities to benefit from

HW-SW design. An interesting question in this respect is how these companies, without expertise or culture in HW design, will grow familiar with HW/SW design in the context of (re-) configurability.

4.10.2 Needs and Trends

Trends:

1. Time-to-market becomes shorter;
2. Reconfigurability allows more design trade-offs
3. Reconfigurability allows small companies to perform HW/SW design
4. Sub-micron technologies allow for evermore-complex designs.
5. In silicon fabrication technology, wire delay is growing dominant over computational delay

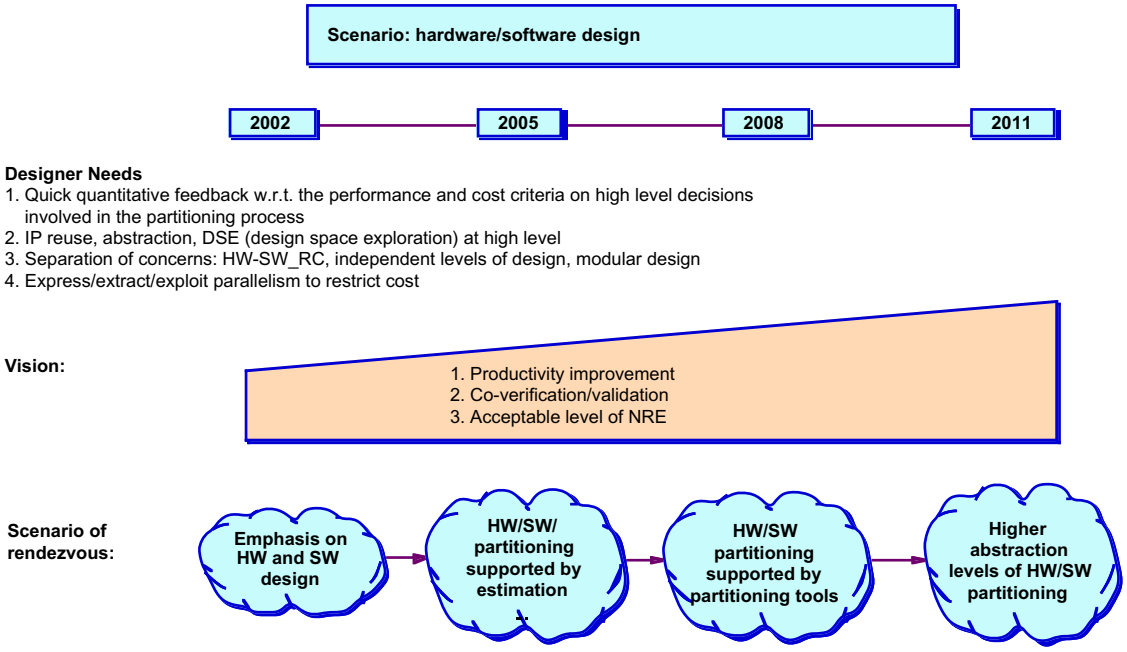
Needs:

1. Quick quantitative feedback w.r.t. the cost criteria on high-level decisions involved in the partitioning process.
2. IP reuse, abstraction, DSE at high level
3. Separation of concerns: HW-SW, independent levels of design, modular design
4. Express/extract/exploit parallelism to restrict cost.

4.10.3 Scenario: HW/SW Design

Terminology:

- IP blocks: processors, ASICs, memories
- Library components: ALUs, register files
- Cost criteria: all relevant criteria to be minimized, like power consumption and silicon area.
- Performance: Required performance translated from real-time constraints
- Parallelism: Tasks (task-level parallelism) or instructions (instruction-level parallelism) can/will/should execute simultaneously
- Embedded SW: software that can be embedded without additional effort. For DSPs this is machine code. For GP processors, it can be C-code provided it takes virtually no effort to generate machine code for that processor using a compiler



Major functions				
	Ad hoc HW/SW/RC design	Quick quantitative feedback on high-level decisions	Less expertise required for HW/ SW/RC design Less user interaction	'Automatic' partitioning
Technologies				
Methods re-use	IP block available, commercially and in-house	Interfaces between IP block standardised Small companies learn HW/SW/RC design	Reuse of IP blocks fully accepted and common practice	'Lego' methodology for composing embedded systems
Tools	Library components available HW synthesis techniques used for mapping on RC HW Companies invest in training designers to exploit tool support	Cost models of library components and IP blocks DSPs are designed for easy compilation	Tool support for HW/SW/RC partitioning and integration	Tools for modelling and executable specifications (Prototyping)
Tool development	Tools do not solve all the needs of the designer	More interaction between designers & tool builder Vendors offer design service		
	SW compilers & silicon compilers from different vendors	SW compilers & silicon compilers with a common design entry (language?) that expresses parallelism	SW compilers & silicon compilers in common framework	

Gaps	Standardised interfaces between IP blocks Cost/performance models for IP and library components Interdisciplinary communication Architecture and mapping tools are developed independently	Confidentiality of designs Portability of (high- & low level) code for DSPs		

Figure 22: Scenario for HW/SW design

4.10.4 Remarks on scenario of rendezvous for Hardware/Software Design

A number of themes are present in this scenario. Most notably, HW-SW design is a task where tool support and a reuse methodology are expected to play a dominant role. As a result, in this scenario there is an emphasis on the problems related to the development of these tools, and problems related to reusing HW/SW components.

Current status: 'Ad Hoc HW/SW design' (2002)

Low-level models (suitable for synthesis to silicon) and high-level models (suitable for e.g. simulation) are commercially available for IP blocks and library components. There is however a lack of cost-models of these (and RC!) components, which makes design space exploration a tedious job because cost figures are obtained by (detailed) implementation of the partitioning decisions. Another problem is the lack of standardization on the communication principles and on the IP interfaces. Therefore, designers have to build their own interfaces to the communication infrastructure for integrating and connecting the IP blocks, which often takes place in an Ad Hoc manner.

No tool support is yet commercially available for HW/SW partitioning, so designers rely on their experience to identify suitable targets for HW acceleration (directly in silicon or in RC hardware). After partitioning, HW and SW compilation are performed. The tools available for HW compilation require training and experience to use properly. High expectations w.r.t. ease of use, and underestimating the effort to master the tool, will lead to frustration and resistance to use the tool in the future. Also the physical distance between the HW designer and the tool designer is responsible for the limited interaction between them. This hampers both tool development and a smooth learning experience. SW compilation for general-purpose processors is very well supported with robust, reliable, push-the-button tools. For digital signal processors (DSPs), tools for SW compilation are not very robust and require a lot of user interaction to arrive at a satisfying solution. One reason for this is the fact that these processors have implicitly been developed for high performance/cost ratio for optimised assembly, and not for ease of SW compilation.

‘HW/SW design with limited tool support’ (2005)

IP vendors deliver performance and cost models with their IP blocks that enable ES designers to do design space exploration (using e.g. Excel as a database for the cost models) without the need for detailed implementation in order to get quantitative feedback. IP interfaces are standardized, which allows designers to integrate and connect the IP blocks in a systematic manner. HW, SW, RC compilation have a common design entry, so that the result of HW/SW partitioning can be easily transferred to the HW, SW, and RC compilation tools without extensive rewriting. Partly (re-) configurable processors and architectures allow small companies to learn and perform HW/SW design.

The physical distance between HW designer and tool vendor will probably remain because of the issue of confidentiality of the design. Instead, tool development and a smooth learning experience are enhanced either by design companies doing in-house tool development, or by tool vendors offering design services. DSPs will be designed that allow efficient or easy-to-use compilation.

‘HW/SW design with tool support’ (2008)

Tools are available to help partitioning the application. Because HW, SW, and RC compilation tools are embedded in a common framework, the tool is able to analyse (using performance/cost models) for each part of the application, the suitability of implementing that part with the available paradigms (ASIC, RC, DSP, GP) without too much user-interaction. The tool can suggest a partitioning and a suitable implementation paradigm for each part. As a result, designers without much experience and expertise can still perform HW/SW design. Reuse of IP blocks is common practice and well supported with standard communication infrastructures and tools that allow plug-n-play of IP blocks in the design.

4.10.5 Recommendations

1. Standardization on interfaces between IP blocks;
2. Develop performance and cost models of IP blocks and library components;
3. Train designers in the use of (higher level) tools, and create a tool-oriented culture in the design community; Decrease the distance between ES designer and tool designer;
4. Design architectures (processor, multiprocessor) from the perspective of mapping applications to the architecture.

4.11 The Embedded Software Design scenarios

4.11.1 Introduction

The specific user needs

The users of embedded software design are the domain experts, who have to build embedded systems. Domain experts are control engineers, physicists, optics engineers, electronics engineers, user interface designers, systems architects, etcetera

The users want to be able to build and modify the embedded software of the system themselves. In this way they expect to be faster to deliver the system on specs and in time. Also they want to be able to improve the system specifications in an evolutionary way of working.

The embedded software not only supports the end user requirements. To support the designers, the production engineers and the service engineers it also includes performance software, calibration software and diagnostics software.

Programmers producing embedded software are seen as overheads, causing undesired delays of the project. So programmers must do the work off the critical path of a project.

Users are building systems, where embedded software is only a part of the problem.

The generic design trends

Developing embedded software is growing to an unmanageable size. Something must be done to keep embedded software development teams small (team size < 100 designers).

New fashions in software engineering come and go in a few years, where families of embedded systems have economical lifetimes of ten to twenty years. (Re)Use of existing software (build in the previous millennium) is a must in embedded systems.

Component based development is a trend, but not at all common in the world of embedded systems. Too much environment is missing to make it work.

Formal methods and their tools are very difficult to implement, because they do talk the language of the software engineer, and not the language of the user.

The term ‘software architect’ is devaluating rapidly, while the importance of excellent software architecture is increasing. How to spot the real good architects in their younger years and how to put them on the fast lane to become a CSA (Chief Software Architect). No academic training is currently available.

Today programmers want to build new software in new products. In embedded systems most work is done in modifying and extending existing software. A change of mind is needed where re-using existing software is considered to be Cool. And building everything from scratch is considered as un-cool.

The architecture of the embedded software development environment is missing. As a result research on methods and tools is very scattered, because the framework where all these methods and tools are seamless integrated is missing.

Vision: Within 10 years embedded software is produced by domain experts.

Large chunks of software will be incorporated in the embedded system using the web by domain experts. By drag and drop the embedded system is put together.

Producers of embedded systems components are offering their solutions tot systems integrators. There is a business model where producers of reusable software are rewarded for their efforts. Building and supporting Common Off The Shelf Embedded Software (COTSES) components is not for free of course, but cheaper and faster than re-inventing the wheel.

A COTSES Architecture Team has defined and is maintaining the Architecture of COTSES. Standardization of development, interfacing, error recovery, message passing, documentation, testing, delivery and change control of COTSES is controlled.

A COTSES Quality Assurance Team is responsible for overall stability of the COTSES. One of the biggest dangers is unreliable and unstable COTSES components.

Methods and tools must be made where Users can select COTSES based on vague requirements, put them together and generate the glue software between the COTSES. Maybe something can be learned from ECAD tooling.

Scenario: Embedded Software Design

See below.

4.11.2 Remarks on scenarios of rendezvous for Embedded Software Design

Introducing Common Off The Shelf Embedded Software (COTSES) on a worldwide basis within a few years used by all companies for all domain areas is a few bridges too far. Therefore some intermediate steps are needed to be able to reach the goal.

1. The architecture of COTSES modules and their framework must be defined. Universities must use their international network and work together to define a working structure. The Linux development way of working may be a good starting point. The architecture must be finalized and change controlled in 2004.
2. The COTSES development environment needs research and must be developed and rolled out. This includes standardized configuration management, automatic test case generation, regression test tools, compilers, design tools, repository tools of COTSES, tools to find the best suited COTSES, etc.

When this is defined, COTSES can be developed within companies, with reuse within the same company (intra company reuse of COTSES). Problems with IP and payment of fees are not a problem in this case. Quality Assurance is an in-company problem: You are punished with your own bugs when you don't do it right. But it is possible to purchase and include third party software, using the same architecture. Companies can convert existing software to new architecture. Companies can decide which COTSES is company confidential and which may be used by third parties (when the fee is received).

Scenario: embedded software design

2002 — 2005 — 2008 — 2011

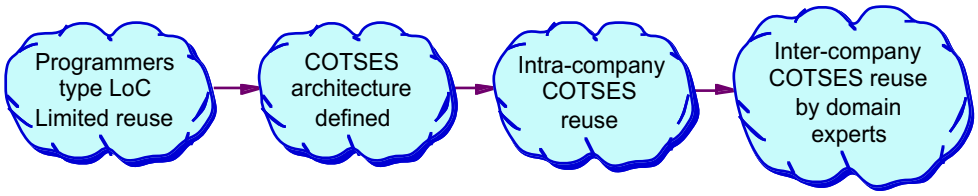
Designer Needs

1. Tool from Scratch to Modelling
2. Software Impact Analysis methods and tools for extensions of existing systems
3. Requirements & Tracability tools
4. How to incorporate large existing components in design
5. Accessable World Wide Library of usable components and quality assurance
6. Design Verification method and Tools
7. Graphical Tool Support è executable models
8. Early validation of design decisions, executable models
9. Generation of software based on models
10. Automatic Testgeneration from Design
11. Tools for automating testgeneration from design?

Vision:

1. Application Domain experts build their own Embedded Software
2. Existing Software will be reused in a controlled way
3. Less time spent in software- and system-integration

Scenario of rendezvous:



Major functions				
	Reusing embedded software in a controlled way Less programming, more assembling			
Technologies				
Methods		Executable models of software and timing	Executable models including computer hardware	Multi disciplinary executable models
	Programmers want to write their own new code		Existing software is reused in a controlled way	Users build their own systems from large blocks

Tools		Integrated design tools Design tools where you can import software blocks you want to reuse including a test environment	Partly multi disciplinary tools Common architecture (or 'bus') where components fit in (agreed framework for embedded software components) Reverse engineering tools Bad-weather behaviour modelling tools	Multi disciplinary tools (software plus appliance) SW/HW co-design Courseware
Gaps	Standardization on re-usability and interfacing Lack of standardization on error recovery, Lack of experience/learning Lack of continuity on gaps: new languages, modelling, tools, views IP to Dollars (IP2$$) business model for users and providers of components			
Other				

Figure 23: Scenario for embedded software design

Software Engineers are still building the embedded software, but reuse of standardized COTSES is the normal way of working. COTSES components fit together more or less. A reasonable amount of glue software is still needed to build systems.

This milestone should be reached in 2007.

To be able to use COTSES in inter-company and international environments some items need to be solved. When domain experts integrate COTSES modules, also extensive tooling is needed.

1. Quality control and change control of published COTSES must be solved.
2. The pricing, business model and IP security must be solved.
3. Methods and supporting tools must be available where COTSES producers can define, build, test, verify and validate their products.
4. Methods and supporting Tooling must be available where domain experts can select the COTSES they want to use, both intra- and inter- company, based on vague requirements. With help of software architects the Embedded Software architecture must be designed, which uses the defined COTSES. The remaining glue software must be generated from the requirements. The test scripts must be generated. The embedded software must be generated. Test of the embedded system can start.

   This milestone should be reached in 2010.

4.11.3 Recommendations:

1. Teach students in this way of thinking

4.12 The Verification/Validation scenarios

4.12.1 Introduction

Verification and validation include a wide range of techniques in the spectrum of formal verification to simulation and emulation. Formal verification attempts to check the complete state space a design can be in. Non-formal verification techniques check only a part of the design space. Though it is less accurate, the gain is speed in verification. Simulation and emulation are examples of non-formal verification.

In the future designs will become more complex. Formal verification techniques will become necessary to deal with the verification problem because simulation and emulation do not scale with the complexity of the design. Therefore this text will emphasize formal verification techniques.

4.12.2 General trends and user needs for validation & verification (V&V)

Introduction

It is very difficult to estimate the state-of –the-art of embedded system validation and verification 10 years from now. Both the world of embedded systems technology and that of validation and verification methods and tools are very dynamic and undergoing rapid changes that will have substantial influence on their future relationship.

In line with the philosophy of a roadmap we will work from the angle of what will be required to accommodate the necessary changes, instead of trying to foretell the actual future.

Moore's law and V&V

Moore's law affects V&V in two ways. On the one hand, the increasing power and memory of embedded systems leads to increased complexity of the embedded software in those systems. This leads to a substantial increase of the effort that is needed to assure the correctness and reliability of such systems. Moreover, traditional V&V techniques, mostly based on simulation and testing do not scale up well as they cannot handle the doubly exponential growth: the number of potential simulation/test scenarios grows exponentially in the state space of a system, which in turn increases exponentially over time as a consequence of Moore's law. This leads to excessively growing costs in terms of resources for system validation (currently 30-50% of the total development cost)

On the other hand, Moore's law influences the performance of V&V tools positively. The exponential growth in terms of processing power and available memory translates directly into a proportional growth of the analytical capability of such tools. In reality, the situation is even better: formal methods research into the algorithms and data structures used for the implementation of V&V tool functionalities has more than doubled the positive effect of Moore's law over the past decade.

Current industrial ES designers are by-and-large unaware of the latter developments.

V&V and ES designers

An important reason why ES designers are not following V&V developments very closely has to do with the fact that V&V methods and tools currently require substantial skills and knowledge that is unrelated to their domain expertise. Typically, working with such tools requires mathematical modelling skills, knowledge of algebraic and/or logical techniques, and familiarity with the idiosyncrasies of the tool implementations.

Current V&V methods and tools are inadequate for normal industrial usage.

Complexity and variety of ES

The complexity of ES can be enormous. In spite of the positive contribution of Moore's law to V&V tools (see 1.2), the analytical capacity of tools will always be significantly smaller than the complexity of ES at any given point in time. Typically, tools must therefore be applied to ES models that abstract away from as much irrelevant detail as possible, whilst still providing a basis for the analysis of interesting properties.

In addition, there is a tremendous variety of ES system, ranging from deterministic, sequential systems of limited complexity on smart cards to full-blown distributed, multiprocessor, networked systems involving complicated quality-of-service requirements.

Currently, there is no well-understood set of abstraction principles that can be used to efficiently produce such models for the various types of ES in combination with the kind of analysis that is required.

Mission and Vision

Mission

The main mission of the ES V&V community (both universities and companies) is keeping up with Moore's law and its consequences (e.g. Dunn's law):

> *How to improve the analytic power of V&V methods for ES systems by a factor four every 18 months, for the next 10 years.*

Interpreting this mission one should take into account that Moore's law itself does already contribute a factor 2 every 18 months. The mission therefore implies that additional improvements in methods, algorithms, data structures and implementation techniques also contribute a factor 2.

Vision:

On the user side of things we can suggest another milestone:

> *In 2010 SE designers are capable to carry out effective V&V of their designs using a wide array of (semi-)automated tools.*

This formulation carefully avoids the mentioning of "push-button" technology, as user interaction will always be required to produce convenient analytical models. The intended reading is that the SE designer will have a tool box consisting of many different tools whose correct use does not require expert knowledge of the

underlying theory and/or implementation. Still, working with such tools will and should affect the way in which ES are analysed and designed, and will require new skills on the part of their users (cf. business process redesign).

V&V technology needs:

V&V technical development:

1. Development of fundamental algorithms, data structures and implementation techniques to improve the performance of V&V tools
   - state space exploration algorithms
   - special purpose inference engines
   - efficient data structures
   - composition principles
   - abstraction principles
   - optimisation techniques
   - static analysis techniques
   - decomposition techniques
2. Development of tool functionalities that support ES design.
   - requirement specification
   - model construction & transformation
   - model simulation
   - model checking
   - consistency checkers
   - model driven test generators/executors
   - real-time, stochastic, performance analysis
   - theorem proving
3. Development of user-friendly V&V methodology.

- V&V configuration management
- application-oriented tool interfacing
- V&V scenario library
- V&V user guidelines

Measuring progress:

The space/time improvements that are achieved are best measured by adopting a number of benchmark applications for various application domains. Many such benchmarks have already been established by the scientific community, but these typically address only progress with respect to the fundamental techniques (i.e. issues mentioned under point 1). Efforts must be made to find good industrial benchmarks for the various application domains.

Goals

Both formal and non-formal verification techniques will have to be able to cope with ever-larger designs. Soft and hard real-time systems will become available and need to be verified. Also hybrid systems are foreseen in the future. Simulation, emulation, and formal techniques need to keep up with these developments.

But there is more to be done: formal verification needs to be integrated with the embedded system design flow. The embedded system designer should be able to perform formal verification without bothering about the underlying mathematics and techniques. Researchers from academia and industry need to sit together

to get this job done. Case studies of industry-size need to be considered for formal verification; tools need to interface to each other and eventually need to be integrated to form one seamless design flow.

4.12.3 Recommendations

1. Research on V&V methods and tools for ES can only be successful if there is a long term and stable commitment from both academia and industry to make the necessary investments.
2. It should be established what kind of embedded systems are most relevant in the context of these proposals, as different systems require different R&R approaches with different resulting time frames. Such knowledge can be used to further profile the V&V roadmap.
3. Based on 2 a list of industrial relevant benchmarks for ES V&V technology should be compiled. This list will be essential to monitor the progress of the V&V methods and tools.;
4. To develop V&V tools beyond the stage of academic prototypes requires implementation capacity that lies beyond the capabilities of academic institutions. Scenarios for tool technology transfer between academia and industry should therefore be anticipated and elaborated.
5. If industrial V&V technology is to cope with the exponentially increasing growth of the complexity of ES systems new V&V methods must be introduced. This entails extensive programs for technology transfer, education and the development of new design and implementation processes that will require substantial resources (time, money). Scenarios for such changes should be anticipated and elaborated.

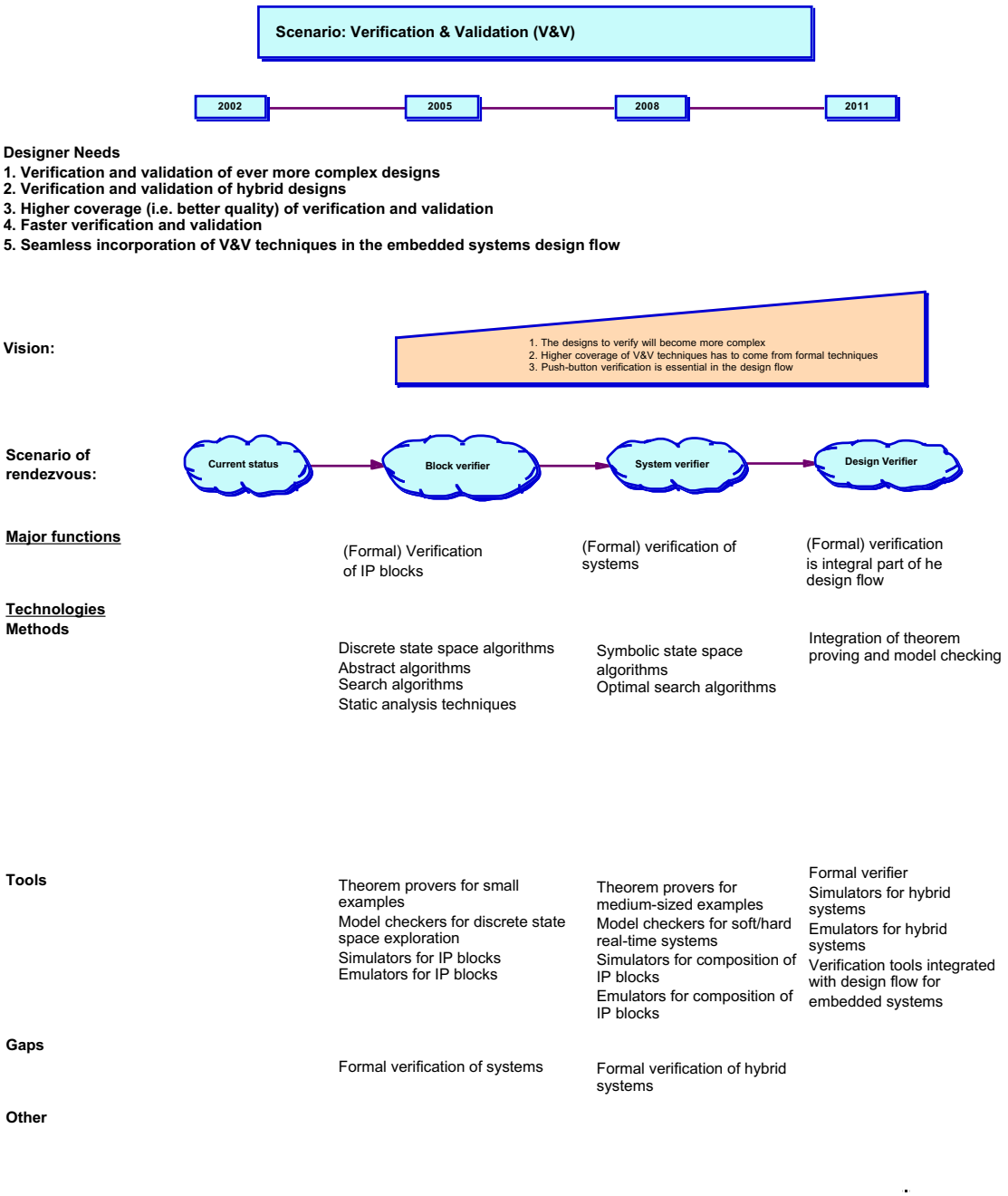


Figure 24: Scenario for verification/validation

4.13 The Test, Debug and Integration scenarios

4.13.1 Introduction

Testing concerns the quality assurance for the entire design, fabrication and application cycle. Every effort in pushing the limits of the micro-electronics technology has to create a coherent view on testing. The quality of the test of an embedding system is the expression of the expectation that the part will work in the specified manner in a partly specified environment and that malfunctions will have only a limited impact. We will largely focus on the System on Chip. This will be the lead theme in formulating the trends that can be expected in testing.

Such systems will be service oriented in the sense that the design only creates the potential for the product line, that the design will be augmented for a specific product from the line while lastly the product needs to be adaptable to the situation in which it is to be applied.

Testing will therefore not be a mere quality assurance at one point in time, but rather a part of the product design that will have influence over the entire lifetime. Integrated test is meant to reduce factory cost while raising product quality. This is mandatory as the embedding system can not be assumed to remain stable.

4.13.2 Needs and Trends

Testing used to be a single platform, single technology business. This restricted problem space allowed for in-depth modeling and analysis. There was board test, memory test, digital chip test and so on: each with their algorithm set-up and each prospering in splendid isolation. In the meantime the microelectronic scene has changed considerably. With decreasing lithographical dimension, more and more functionality is integrated on a single carrier, at the expense of more complicated timing physics. Today's board is to-morrow's chip. Small fabrication dimensions and high volume have a strong economic relation. But there is a limited amount of products that can be produced in high-volume in a single application. This observation has stimulated an increasing use of ways of obtaining flexibility, leading to e.g. programming facilities like hard-wired instruction ROM, flashable parameter EPROM and reconfigurable FPGA.

Where the plain vanilla ISA (Instruction Set Architecture) creates functional richness on a limited amount of control/data signals, the introduction of configuration registers and ultimately of configurable interconnect explodes the amount of logic paths to be tested. This added to the increased system complexity necessitates on-chip support. And as the chip becomes a system, this test support will follow the same development route as the system itself, ultimately leading to a configurable part that in turn must be tested.

In the end, the system will support both functional and test programming from a number of communicating blocks. As communication is the core of the offered functionality, separate attention is required to validate the internal timing from the electrical glitches to the abstract protocol.

Scenario: Test, debug and integration

2002 — 2005 — 2008 — 2011

Designer Needs

1. The chip designer needs more support for fault characterisation of flexi-custom architectures (test & debug)
2. The product designer needs better tooling for test-view development (debug & integration)
3. The application designer needs more support for diagnosis and reconfiguration (debug)

Vision:

1. Formal core validation is essential for polymorphous testing
2. Adaptivity takes the pressure from the final test by facilitating a distribution of concerns

Scenarios of rendezvous:

Current Status → Hierarchical test → Transparent combi test → Self-test

Methods		Moving interface and communication standards onto the chip	Cut-and-Fold notations Unified test techniques for HW/SW mixes	Extension of BILBO to Built-In functional Block Observations (BIFBO) Test pattern generation by generalisation
Tools		Support for characterisation of new fault models	Open development framework for test design & verification	Compaction of functional Profilation Verification of hierarchical, modular test view
Gaps	Hierarchical manupilation of existing test pattern sets	Mixed HW & SW test Support for hetegoneous system designs	Merging validation and testing Control of on-chip modular test & configure	

Figure 25: Scenario for test, debug and integration

4.13.3 Remarks on scenario of rendezvous for Test, debug & integration

Meddling complexity

Complexity of embedded parts has many aspects. Next to the bare number of transistors on a chip, it has also to do with the many technologies that interact through the part, the diversity of microelectronic concepts applied within the part and the judicious mixture of hardware and software programming, potentially all this in a networked service composition.

Today the influx of switching technology has melted the digital and analog domain into devices where both can be personalized from the same software program. Ass the system moves onto the chip, it becomes more reactive and the testing has to take the embedding system into account. Eventually the testing scope has to widen even further as the system is an integral part of a micro-system.

IP cores have originally taken the role of a half fabricate. Such parts come with test patterns that validate the part as a whole. This has sufficed for systems that merely added some functionality to the existing core. However, in a multi-core environment such parts will also be required to aid in supporting the test of other parts. Clearly such methods place demands on the granularity of the design, where the overhead is too large for small parts while larger parts are too tightly integrated to provide the required level of test support.

Where we see the levels of programming increase and a range of hardware/software interactions come into play, the test methodology needs to provide a more integral view. Today software and hardware test are overlapping in methodology and growing towards one another; for embedded systems this trend needs to be continued. The mutual test support of IP core need to become generic in this sense that a sound algorithmic basis is required.

Mixing various core in various technologies with a large amount of non-committed functionality will lead to heterogeneous architectures, which should become reflected in the test strategy. To a large degree this will become facilitated by on-chip circuitry such that the test program itself addresses such problems on a more abstract level.

Debug & test

The various levels of programming reflect a shift in the moment of final personalization from fabrication to application. As a consequence we will see a changing emphasis from in-line to on-line test. This is necessitated by the fact that an indepth test becomes even more impossible than it is to-day; it is facilitated by the fact that modifications remain possible till a later moment in the life of the system.

There will remain a constant desire for appropriate fault models. With each next step in process development, the dominant fault model may be different and should be adequately accommodated. From such fault models grow a wide range of fault instances that bring failure interaction between the wires but also between design parts.

The premise of off-line test when the part becomes a product is stability. The requirements can be loosened by allowing for redundancy and eventually replacement through the use of non-committed parts. The judicious application of such measures needs to be based on principles of modularity, as changes have to assume the prior success of fabrication test and have only a local effect.

At the application level, the user can hardly be bothered with applying and interpreting tests. A proper system of quality assurance remains indispensable, though the functionality is still open for adaptations. This can only be guaranteed when the embedding system takes a role and in turn this embedding system needs to have access over a well-defined mechanism such as a test bus. To limit this interdependence the test resources on the chip must be sharable and therefore part of an hierarchical ordering.

Modularity and hierarchy can be applied in an arbitrary mix. The result may have many forms and a careful standardization in terms of system architecture is required to handle such a polymorphic arrangement.

Integrated test

Where complexity meddling focuses on the test technology to facilitate the actual test and Debug & Test focuses on the test technique to use the potential of the Design & Test, a middle layer is required to glue such concepts together. This Integrated Test aims to provide a clear view on the overall product test that can be used over the lifetime of the product.

Access & Control lifts the issue of observability and controllability to the system level. Where the system becomes more and more integrated, the sequencing of parts to take care of the product as well as of the test behavior becomes a scheduling problem. The polymorphic architecture of heterogeneous parts together with the need sharing test resources in a supportive manner can not be pre-determined.

To keep the test problem manageable will then induce a need to bring standards on communication protocols and core tests on the chip. Reducing the complexity by using strict standards is supported from the presence of non-committed functionality to the degree that adaptive repair will play an important role.

Test drop-ins are a basic means to bring test facilities on the chip. In a sense, they are the IP cores of the test domain. With the increasing amount of non-committed functionality, the momentary field programming for test will also creep onto the chip. Mixtures of tests for the already installed functionality and for the still open functionality have to be part of a single test program.

The test program will not merely reflect the current situation but will also indicate how failures can be repaired by introducing new programs. This is especially mandatory for application tests as the end user can not be expected to come with the required level of Test & Diagnosis. Hence, the overall test program may be influenced interactively by the embedding system. In other words, the embedded system will have not only a Design view but also a Test view towards its environment.

4.13.4 Recommendations

The large diversity of hardware and software, that comes together in an embedded system in various shapes and quantities, needs to be addressed from a single unified view on testing. The testing problem will further be aggravated by the increasing role of redundant and reconfigurable parts. Increasingly we will need on-chip measures and dedicated test functions to allow for an acceptable fault diagnosis, isolation and repair over the life-time of the product. Introduction

Appendix 5. Important Embedded Systems aspects of a not only technological nature

5.1 On objectives

If one defines a roadmap for 10 years, and one wants also to look at the conditions that need to be fulfilled for its successful implementation, one has to broaden one's view considerably outside the pure technological arena. The objectives of the roadmap have to be put into a larger perspective. From a variety of viewpoints a number of observations can be made about conditions for success, but also about what aspects have to be brought into line with one another to come to a successful realisation of the roadmap.

E.g. from an economic perspective it may be observed that embedded systems represent a fragmented market, but at the same time they represent a market likely close to 100 x larger than the desk-top market and this market still leaves a lot of headroom for technology skills to prevail over marketing. The objective of the ESR could therefore be to use the embedded market as a technology vector to give Europe the leading world-wide position just like Europe took the lead in wireless telecommunications. This places the ESR in an environment which imposes that many more consequences are worked on than just technological ones.

E.g. to gain a European leadership also requires that the international dependencies are analysed and dealt with. Standards need to be international to be effective, although there are examples where the local scale is sufficient to allow local (and hence protected) standards. The problem of basic components and technology however is more severe. In normal (peaceful) conditions they should not be perceived as being problematic, but as many of the basic technologies are US owned (or patented), there is a certain dependency risk. Hence, while it is not efficient to seek complete independence and supremacy, Europe should at least have leadership in a sufficiently large domain to have a strong bargaining and negotiating position. This also implies that there is an economic dependency factor. If the economic and social environment does not provide the right incentives to develop and compete in this technology race, with also new players in Asia becoming powerful factors, the ESR will not give Europe the leading position, but a follower's position.

It will be useful to collect observations like the above and to investigate what initiatives need to be taken to create success in a long-term effort from several points of view. The following is a start of such an investigation and consists mainly of contributions from reviewers of he first internal version of the ESR.

The following points of views or aspects are considered below:

1. Concepts: how to best meet the challenges posed in the ESR?
2. Education: how to build the engineering work force that can execute the ESR?
3. Economics: how to create the infrastructure (in its social definition) that supports and musters such a goal?
4. Process: how to develop and/or acquire the necessary technology?

5.2 Conceptual approaches

This is the area where the ESR document focuses on in its approach to handle complexity and heterogeneity. The underlying theme is one of the needs to bring multiple domains (now only loosely linked or even separate because of underlying historical reasons) in a unifying ‘systems’ domain. The challenge here is that this means that conceptual bridges must be made from very high levels abstraction to very detailed implementation details, across multiple vertical domains while achieving the ambitious goal that embedded systems are developed very fast (live times shrink), more reliable (greater dependency), cheaper (widespread use) and less resource hungry (sustainable use). At the same time, the embedded devices must operate correctly in an autonomous as well as in an interconnected way as they become part of a bigger ‘meta-embedded world’.

The ESR roadmap is biased towards the SoC market and its way of approaching design problems. While this is an important technology vector on its own, this seems to forget that board level design and packaging it into a application product, will not go away but will also have their challenges.

A favourite paradigm for tackling complexity stems from CSP (Communicating Sequential Processes). CSP comes from the parallel processing domain as a formal, rather arcane mathematical language, but in the ESR context it serves as a framework to reason about systems that are composed of multiple sub system modules with well-defined interfaces. Once the ‘interface’ is well defined (which also means that the internal state-machine operates correctly and the interface definition is complete), it is no longer needed to know the inner workings to use the sub system module to build larger systems. This is the basis for a black-box approach to system design.

Let’s take an example. For the design engineer an embedded system might be a ‘box’ of which he knows how it works, but for the end-user, it often will be a ‘black box’ with input and output ‘connectors’. E.g. it can have sensors to measure certain entities, it can have knobs and handles, it can have a screen, mikes and/or speakers and it can operate on other devices. The inside can be anything (mechanical, hydraulic, electronic) as long as it does the job. These interfaces with the outside world define what the embedded system is supposed to be doing and also defines (or rather restricts) its operational envelop in terms of boundary conditions (e.g. temperature, weight, cost, energy consumption, timely behaviour,…). Good system engineering comes down to opening the black box and filling it up with well-defined sub system modules taken into account the boundary conditions. Hence, the job is reduced to define new black boxes until a level is reached where implementation issues make further reduction impractical or undesired. In practice, the main issues are:

1. -To select the sub system modules so that they satisfy the requirements and so that they can be us as ‘trusted components’
2. -To clearly and unambiguously define the interfaces.

Of course, this leads to a recursive process as this is the way to design trusted sub systems from trusted ‘basic’ components. At each stage, the designer must make trade-off decisions between what is supposed to be optimal and what is feasible. This can ultimately result in a very costly (time- and resource-wise) process

if the wrong trade-offs have been made. Hence the need for simulation. As simulation is done on 'computers' (another type of black boxes mainly designed to execute crude simulations of a real- world entity), the CSP concepts come to rescue. It is sufficient to have a set of programming 'tools/languages' that can accurately execute such a simulation of (concurrent) modules. Hence with a proper selection of 'programming/simulation' tools, one can have 'executable' specifications and simulations that can even be used for the final implementation (if the final implementation level of a given sub system component happens to be a computing device).

While this black-box decomposition methodology provides for a 'correct-by-design' approach, it also allows constructing systems in a bottom-up approach with built-in testability. Of course, designing the basic component can be a much more error-prone process, but once that achieved, it should be clear that the key to this methodology is to separate the functional content from the interface content/ protocol. This is exactly what CSP is all about. Just like the ESR outlines, this comes down to starting to work with well-defined (and standardised) interface protocols. This is a bit in contrast with the current practice in e.g. SoC design where the interfaces are even silicon technology dependent or defined in terms of the electrical signals. If extrapolated, this will come down to defining a standard electrical interface with a standardised 'communication' protocol. Note that e.g. the telecommunication sector has already adopted such an approach (e.g. TCP/ IP), but that at some point this approach will need to be made universal. The result should be a 'plug-and-play' connectivity with for most user level devices the capability to 'hot reconnect'. This in itself requires a higher- level protocol and state monitoring of the 'interface' device. Such approaches already exist. E.g. IEEE 1355 (pioneered from the INMOS transputer, itself a computing device architected to mimic the CSP model). ESA has adopted this as 'SpaceWire' (adding LVDS signalling) and is using it as the basis for on-board system architectures (OHMA: Open Heterogeneous Module Architecture). SpaceWire is used to connect computing processors between each other, with sensors, between boards and even to mass memory devices. Also the industry, driven by the needs of the telecommunications sector, has begun to adopt such architectures. The last one in the row is StarFabric (PCIMG 2.17) that resembles IEEE 1355. The CompactPCI industry is by the way a nice example of how things could evolve. This industry has adopted a wide range of standards specifications at various levels and has adopted a solid technology base to foster the re-use and integration of modules developed by third parties. PCIMG3.X illustrates also the move away from the bus and the adoption of a 'switched fabric' interconnect technology. Often what is found at the board level, is introduced some time later at the SoC level when the economics of the technology permit it.

Note that my view is not so optimistic as to the complementary need expressed in the ESR report. This is the view that methodologies need to be developed to enable re-use and integration of existing tools and components. While this is theoretically possible (like defined above, by developing interface adaptors or translators), this is often behest with problems and serious obstacles often at the semantic level. Often current tools and components have rather restricted or ill-

defined semantics (the embedded software industry is full of them, even if they are so-called standards like POSIX or ADA). As these were not developed with interfacing in mind, the inner semantics are often visible in their interfaces, hence making that often impossible to match with semantics defined for another tool or component. Developing a reliable and satisfactory system in such a way is often a bad compromise in the best case and not necessarily faster than a development from 'scratch'.

5.3 Educational

The biggest challenge however for the ESR lies in the educational domain. As such the ESR defines in fact an industrial process, not so much a technology. While various tools can be developed to support this industrial process, it is first of all to be executed by humans (who have skills and intelligence but also severe limitations in adopting new «paradigms»). The key to this might be to radically change the way «engineers» (but it applies equally well to many other classes of the workforce) are educated. Part of the problem stems from the history where «ex-cathedra» is seen as the most-efficient way to educate. The challenge is two-fold: one the one hand one must work to develop the diverse engineering disciplines into a set of «predictable» methodologies (read: industrial processes), while on the other hand experimentation and heuristics must allow the engineer to make the right trade-off decision. To put it into perspective: a construction engineer has tools to build bridges in a predictable way (the computer even does the calculations for him).

On the other hand we are still not at a point where we have a clear qualification for «software engineering», and certainly not for «system engineering». Think here what also Prof. Hugo De Man often puts forward. The reason is that software and system engineering are not fully understood disciplines. If one takes into account that the ESR puts emphasis on these domains, we are in trouble. Our view is that the engineering education can be greatly improved by re-introducing two main areas of attention.

The first one is that engineering is first of all a domain that is full of experimentation. First developing the crafts and skills (by experimentation) is more natural (see e.g. children learning to use a computer or speak their own language) than learning a skill by having explained the theoretical model. However this does not mean that the theoretical model is to be put aside. On the contrary, good engineering (and this applies to many other domains as well) is based on what one calls the «KISS» principle: Keep It Simple and Smart». This is for all those who engineer things (e.g. such as our law makers). There is an alternative «Keep It Simple and Stupid» that applies for the users. Designing a «thing» or «system» means that when well done the resulting solution should be elegant and simple, but this also means that the problem should be well understood. If the solution is complex, often the problem domain was not well understood. (again this applies to many other domains as well).

In order to develop these skills, engineers must go through a process whereby they must acquire a solid and broad background know-how (to discover analogies), the skills to analyse problems and to formulate requirements in an unam-

biguous way, to acquire craftsmanship and heuristic know-how to strengthen their intuition. Therefore heuristic know-how and «background» know-how must be complemented by an acquisition of the theoretical models that explain the why's and how's. Therefore formal methods of reasoning must be developed to acquire the skills to reduce a problem domain to its essential core. Such formalisation also helps to design-in reliability because it forces to think about the operating boundaries of the system. Given that we propose a CSP type approach, it should be clear that formal reasoning is also important because the CSP approach depends a lot on a formal and well defined interface behaviour between the sub system components. The reliance on simulation also necessitates this, as a simulation model is only as good as its specification. System engineering is the culmination of all these skills. It brings together different domain skills at the interface level. It requires background know-how on all the sub system modules in order to make the right trade-off decisions (including economical ones).

This is well illustrated by the design of a PWA. It covers many aspects: first of all, does it really serve a purpose (or is this just another experimental gadget?), How is the man-machine interface? How can we design it to let it do what it is supposed to do? How doe we keep it below a certain price cost? Is this even feasible? How do we solve the battery problem? How de we put it in an attractive shape? How do we make it reliable? etc. At this level, a good system engineer must be capable of traversing not only multiple domains (mechanical, electronic, software, chemical, economical) that cover the specification level down to the practical implementation details, he must also develop the skills to know what the impact of many of his trade-off decisions will be.

A quite nice example of multi-disciplinary education is the seminar for first year electronics students at KU Leuven entitled 'Ontwerpseminarie H838: Design of a People Mover'. Contact Hugo De Man for more information: deman@imec.be.

We would like to illustrate the inadequateness of the current university education, although there are of course many places (often 'polytechnic') where the courses remedy this to some extend. E.g. a computer scientist, educated in a mathematical mindset will often not even know how an 'int' looks like in the hardware, or certainly not know that this depends on the executing processing machine. On the other hand an electronics engineer, of which the majority ends up writing software, will never have been trained in the formal reasoning methods that computer scientists have been taught in. The result is that neither of them is prepared to tackle the challenge of the ESR. The reason for this 'chasm' is partly historical, partly the lack of flexibility in the government controlled education systems (with professors being nominated for live, having barely a budget to equip the labs and other things).

This brings me to the second important change that is needed: education and training must become a lifetime occupation. The exponential advance of science and technology poses severe dilemmas that impose on the one hand further specialisation, on the other hand information gathering from different domains to be able to understand the 'system'. This might lead to two classes of engineering: (system) design on the one hand and system implementation on the other hand.

Both require continuous re-training and re-education to be able to meet the needs of the ESR.

5.4 Economic

To achieve the ESR, which is a medium to long-term objective, one must put in place a complete supporting environment. The educational part of this is one of the domains with the longest-term impact. But the speed of change (lifetimes of one month) and the production volumes involved (widespread use) also indicate that this industrial process will be or is very capital and know-how intensive. The speed of change also entails rapid decisions and higher risk taking. Even if we have an engineering force that is prepared to tackle the system-engineering task, this must be tightly integrated in a business and social environment that is supportive and capable of following. The challenges are manifold:

1. -Economic: does the environment provide capital and infrastructure in an adequate way?
2. Social: does the (local) society accept this «rapid» change model?
3. -Human resources: does the (local) society provide the management skill sets (e.g. marketing, financial management, …). Is there a pool to draw from?

It should be clear that the ESR challenges are not just technological. It can only achieve leadership if the community supports it. Counter examples are e.g. Japan, who clearly has the technology skills, but where management and power positions are kept by an older generation that is risk and change averse. Another example is China who developed the engineering skills and is open for rapid changes, but where the society still has to develop the financial and business backbone and the processes to put these skills at work. In this context, the aging population in Europe could be a serious structural obstacle to change.

Using IP creates many economic challenges:

1. A sound business model, which challenges IP component builders to market, sell, deliver and support the right components where domain experts are waiting for.
2. An embedded software IP market, where IP components, consultancy, maintenance and support can be sold. And where consumers and producers meet to predict future needs.
3. Standardisation of API, architecture and external behaviour of a IP component
   a. Intra process, inter process and inter processor communication.
   b. Exception handling
   c. Debugging facilities
   d. Intra component verification and validation
   e. Inter component verification and validation (JTAG like)
   f. Interfacing with Real Time Operating system
   g. Hooks for hot-swappable software
4. Standardised IP component data sheet
5. Standardised IP component test data sheet
6. Standardised IP component publishing mechanism (like a webring with tools)

5.5 Process

The above outlined methodology can be applied even today, but an efficient implementation requires the availability and hence eventual development and refinement of tools, standards and basic technologies and their working together in well-structured processes.

Appendix 6. Roadmapping: objectives, process and concepts

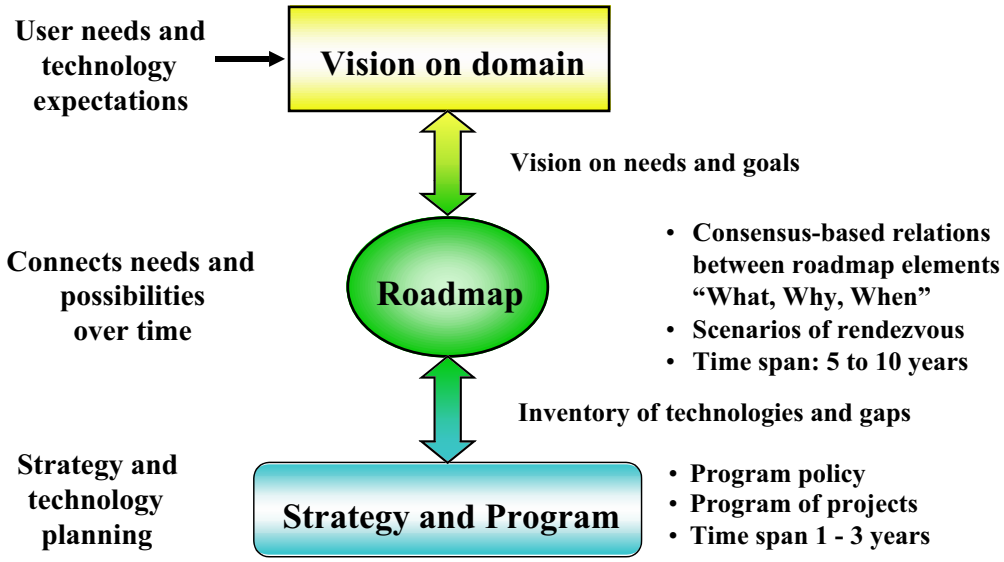
6.1 Introduction

The kind of technology roadmap that is presented in this document is a needs-driven inventory of technological possibilities of the embedded systems domain over time. This in contrast to classical product-technology and technology-push roadmaps.

As instrument for technology planning and co-ordination a technology roadmap for a domain has the objective to deliver:

1. A common vision on future needs and developments of a domain
2. Guidance for directing R&D of technologies in a domain
3. A rationale for collaboration activities
4. A strategy for long-term investments

A roadmap may also contribute to creating a common terminology to ease communication in a domain, and it may provide better ways to classify related work.



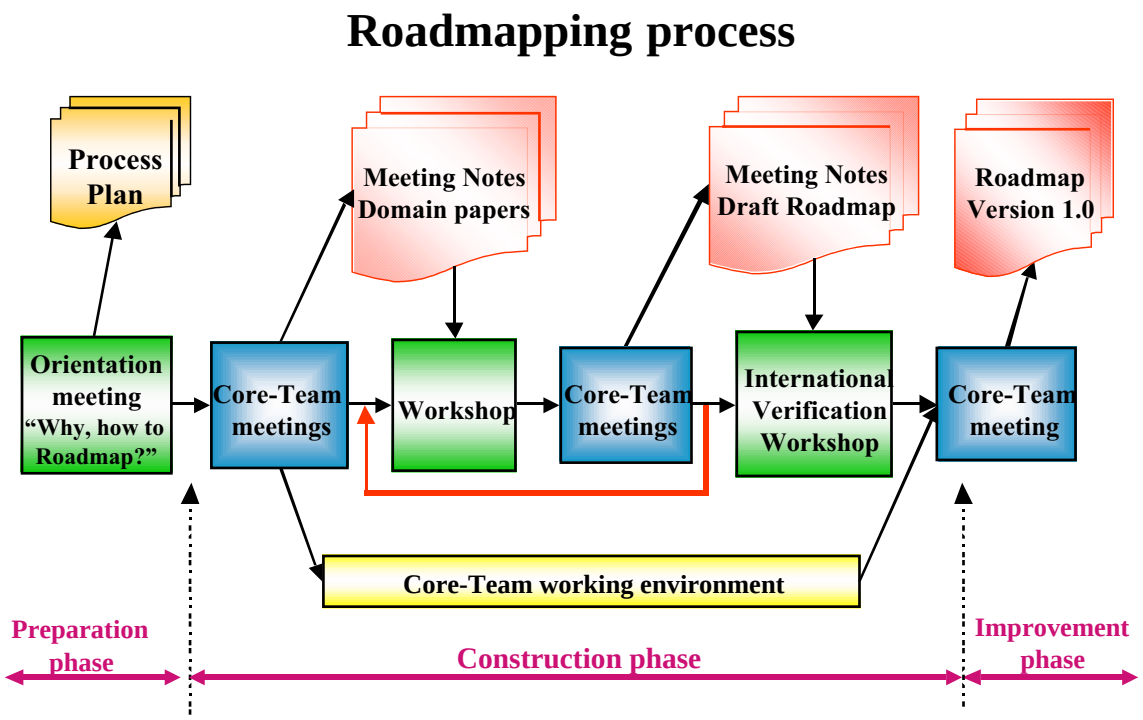
The process for obtaining a technology roadmap for a specific domain can therefore be positioned between vision development for such a domain and strategy definition for reaching defined goals in the domain by a research agenda or a programme of projects. A technology roadmap helps both in setting goals for technology development of a domain and in balancing off technology push arguments in priority setting against user or market needs.

It is by now well known that roadmaps do not originate easily, but require a carefully managed and facilitated process to come about, even now the methodology is reasonably well established. See e.g. the EDAA and ITEA roadmaps.

In the sequel the major concepts of roadmaps and the required roadmapping process are succinctly described.

6.2 The roadmapping process

The success of a roadmapping process depends to a large extent on a team of project leader and facilitator or a facilitating project leader who can guide the process and preferably has a general understanding of the domain. He will start by interviewing the major stockholders to establish the purpose of the roadmap. Thereafter he will outline the whole process, get an agreement with the stockholders on content and funding. Subsequently a Core-Team is formed, under responsibility of the principal stockholder, to perform the iterative search and learning process that making a roadmap is. Then execution follows with a sequence of CT meetings and workshops. Intermediate workshops are organised to add to the results and improve them where possible. It gives also an opportunity to obtain consensus with a larger group of interested persons and to have the results evaluated by independent experts.



6.3 Major concepts of roadmapping

A technology roadmap of the kind presented here looks quite far ahead: about 10 years. This poses the problem that not yet a market exists that can specify what is needed. It is just too far out for that. This necessitates special methods and corresponding processes to get sufficiently reliable information to base strategies on. Scenario writing is used in combination with reframing techniques to assure this.

Scenarios are written about driving applications in a specific domain. A domain is an area of interest in which products/services share certain characteristics. A driving application is a product/service that challenges technology capabilities in a domain to the utmost, even to the extent that it might imply the need for not-yet existing technologies.

The writing of scenarios is based on an exploration of user needs in that domain for the driving application. User needs are the expression, in non-technical terms, of the wishes of (target groups of) end-users that may motivate a search for fulfilment of these needs by product/service solutions in which technology may play an important role. To come to a solution of interest to the user a vision needs to be developed of how the fulfilment of user needs will evolve over time. A vision is here a description of the common view on the practical evolution of the major function characteristics of product/service solutions that fulfil (some of) the user needs.

The result of all this is written down in domain papers. Domain papers embody the vision in scenarios of rendezvous, in which a scenario is a sequence of events and a rendezvous is an event where technologies meet that are necessary for the emergence of a new generation of a product/service that fulfils a user need.

Characteristics of rendezvous:

1. At least two technologies are involved that need to converge in time
2. They can meet or miss
3. The outcome is uncertain: they can match or not
4. External events can influence outcome and success

Rendezvous are in a sense the equivalent of the milestones and deliverables of projects. The uncertainty involved made it however necessary to introduce this new concept.

The information in the domain papers is subsequently used in the roadmap. This requires that first agreement is obtained about the structure of the roadmap. Once agreed upon the domain paper information can be mapped into it. Next the correct ordering of technologies and the identification of technology gaps is performed. A series of interviews with experts is necessary to get a sufficiently detailed view on what technology gaps really mean and what should be done about them. Also an evaluation and checking with a group of international experts is necessary to complete a first version of the ESR. This leads to a final session of the Core-Team to conclude on its series of recommendations.