Three Proactive Themes in Computer Science

Conclusions from a FET Brainstorming Meeting

Brussels, 17/18 Januari 2000

Participants

- I. Antoniou (Brussels)
- J. Diaz (Barcelona)
- H. Ehrig (Berlin)
- G.F. Italiano (Rome)
- J-P. Jouannaud (Orsay)
- K. Mehlhorn (Saarbrücken)
- M. Nielsen (Aarhus)
- D. Sannella (Edinburgh)
- P. Spirakis (Patras)
- J. van Leeuwen (Utrecht, chair)
- P.J.E. Verissimo (Lisboa).

Present on behalf of the EC:

- S. Bensasson
- J-L. Fernandez-Villacanas-Martin
- L. Flores
- K. Glinos
- P. Karp
- P. Timmers
- C. Versino
- J. Wejchert.

Summary

This document contains the conclusions of a brainstorming meeting held to explore the need for new pro-active initiatives for long-term, innovative research in pure computer science in Europe. Three themes were identified that could be the basis for a pro-active initiative in the near future:

- **Mobile/Distributed Reactive Systems**
- **Guaranteed Software Systems**
- **Nature-Inspired Computation**.

1. Purpose of the Meeting

The area of information and communication technology (ICT) develops rapidly, and there is a great demand for know-how to support this development. At the same time there is a great need to look ahead and identify issues for research that could imply further breakthroughs in the years ahead. The brainstorming meeting was organized to try and identify a small number of themes in computer science which should be recommended for new pro-active initiatives within the *Future and Emerging Technologies* (FET) program.

The developments in ICT are generally rooted in the ideas, concepts and basic theories of computer science. The present developments in e.g. agent-based technology show the need for advanced knowledge in software engineering, in interactive algorithm design and in understanding the emergent behaviour of systems consisting of many independent, intelligent components. In order to facilitate similar developments in the future, the fundamental research for it has to start now.

Whereas the meeting was organized by IST/FET, the responsibility for the discussion and conclusions during the meeting was left entirely to the participating computer scientists. Three themes were identified that could be the basis for a pro-active initiative and that are recommended for immediate action within the FET-program:

- *Mobile/Distributed Reactive Systems* (informally called: *Global Computing*),
- *Guaranteed Software Systems*, and
- *Nature-Inspired Computation*.

This document summarizes some of the considerations that led to this recommendation.

2. Some Directions in Computer Science

The discussion emphasized the enormous development and (economic) impact of computer science as a science in the past ten years. In presenting its case for the urgent need for further research in information technology, the recent and widely accepted PITAC report[1] noted that the innovations depended on patient investment in fundamental and applied research.

In the discussion it was noted that research in computer science needs a much further integration of approaches than in the past: the study of formalisms, algorithms, and programming methods often goes (and should go) hand-in-hand nowadays. Given this observation, a considerable number of developments and issues for future research were noted by the participants, including:

- radically new principles for distributed computing and communication,
- 'sentient computing' (design of mobile, intelligent, and autonomous software components),
- coping with (the complexity of) interaction,
- programming abstractions for 'global computing',

[1] "Information Technology Research: Investing in Our Future", President's Information Technology Advisory Committee (PITAC), Report, February 1999.

- 'taming the web',
- aspects of 'Quality of Service' in new ICT-infrastructures,
- scalable techniques for the design and validation of complex, large, embedded systems,
- design of safe, verifiable, user-friendly software,
- new paradigms for integrated visual computing and system development,
- integrated programming methodologies,
- focus on proofs (of programs) and testing,
- need for verified/certified software/component libraries,
- shifting notions of feasibility in computing,
- scalable algorithms for very large networks and massive data-sets,
- alternative technology ('revolutionary') computing,
- 'physics of computing' and computational paradigms provided by nature,
- understanding heuristics, and
- new opportunities for discrete mathematics and probability theory (most algorithms nowadays use it).

Some characteristics of the suggested research directions are summarized in the following overview:

- New information infrastructures
  A: scalable, very large, ubiquitous, wireless, mobile.
  B: digitization, digital archiving, web retrieval, high-volume transfer, QoS
  C: dynamic, interactive, active agents.
  D: global computing.

- Advanced information systems
  E: adaptive, intelligent, autonomous agents, sentient computing.
  F: massive data, profiling, data-mining, discovery science.
  G: safe, secure, fast, intelligent retrieval of information.
  H: *e*-commerce, just-in-time enterprises.
- Software
  I: visual, embedded, interactive, intelligent, safe.
  J: software for radical technologies, integrated system development.
  K: component-based system specification and design, UML.
  L: proof assistants, integrated treatment of testing.
  M: libraries of verified algorithms, 'design science' (for libraries).
- Advanced algorithms
  N: algorithm experimentation, algorithm engineering.
  O: concrete algorithmic software-components, algorithm libraries.
  P: alternative, revolutionary, chaotic, (non-)equilibrium computing.
  Q: use of randomness, probability, distributions, heuristics.
  R: complexity theory in new perspective, new notions of feasible computing.

It was felt that the future applications should not be overstressed. The aim should be to have impact through theories and prototypes. A strong European position in information technology in the future requires a greater effort in research, and thus in research funding, aimed at theory and new areas at the interface of theory and practice.

3. Proposed Pro-active Initiatives

The discussion subsequently aimed to identify a (small) number of themes for which a pro-active initiative would be highly recommended. Pro-active themes, according to the Commission, ideally have the following characteristics:

- a vision of what one wants to achieve,
- a 'man on the moon' target,
- a radically different area,
- an appropriate granularity,
- a clear 'timeliness', and
- an identifiable community of researchers (viz. in Europe) that can contribute to it, preferably but not necessarily across the boundaries of existing (sub)disciplines.

Previous pro-active themes in the FET program included e.g. quantum computing and 'information eco-systems'. In the 2000-Call the pro-active themes will be 'the disappearing computer' (viz. the area of ubiquitous computing placed in a broad context) and 'neuroinformatics for 'living' artefacts'.

The brainstorming meeting of January 17/18 resulting in the following three themes, proposed as candidates for pro-active initiatives in the FET program:

- Theme 1: *Mobile/Distributed Reactive Systems*
- Theme 2: *Guaranteed Software Systems*
- Theme 3: *Nature-Inspired Computation.*

Detailed outlines of the themes are attached. Each theme describes a vision and key challenges that should be pursued.

The visions are ambitious enough that it is very unlikely that they will be fully achieved. The expectation is that there will be very considerable progress on understanding the problems and on a range of approaches to dealing with important subproblems. Certain avenues will turn out to be more fertile than others and promising future directions will emerge from a comparison and combination of the various efforts within a theme.

Because of the diversity of approaches required to achieve progress, it is unreasonable to expect that complementary results of different groups will be combinable without further work on integration. To foster such integration and achieve a whole that is greater than the sum of its parts, it is suggested that an effort should be made in the later stages to identify promising combinations (perhaps during workshops involving all of the projects, probably with the help of outside reviewers) and set up new projects to pursue them. A similar mechanism is tried in the 'the disappearing computer' initiative.

The best way to make good progress seems to be to attract the best people to work on the problem and let them follow their noses, rather than trying to predict in advance what will turn out to be important. In our view, imposing restrictions in an attempt to ensure anything more than this is likely to disqualify potentially important approaches, and is unlikely to really improve the situation. Pro-active initiatives are supposed to be high-risk, and this is part of the risk.

Utrecht, March 24, 2000.

Theme 1: Mobile/distributed reactive systems

Introduction

The theme concerns the study of systems of interacting entities[2] having the following three features:

- Entities are autonomous and activity is not centrally coordinated.
- No global information about the state of the computation is available. Information about the environment is uncertain. Any available information may be imprecise.
- The system is dynamic in the sense that: physical components may move; entities are mobile; the number of components, their connectivity, and the bandwidth of connections may change during computation. These changes may happen unpredictably, perhaps as a consequence of component or network failure.

One large class of examples would be systems composed of mobile entities distributed over the Internet. The term *global computation* has been used to refer to such systems, with application areas ranging from electronic commerce to distributed interactive games. *Active networks* are another example. A different kind of example might be a future road or air traffic management system where cars or airplanes communicate between themselves and with environment devices in order to make efficient use of the available road or air space, or a factory populated by mobile robots that perform their duties while coordinating with each other and avoiding obstacles.

Vision

To provide a sound basis which allows us to:

- analyse and understand such systems,
- reason about their behaviour and performance,
- design them (better: contribute to their design),
- control them, insofar as this is possible, and
- understand their limits.

The aim of work on design and architecture should be to produce systems which are dependable, flexible, secure, robust and efficient.

Research community

A large subset of the Computer Science community could contribute to research in this area. There is relevant work on: semantics, theory of computation, formal methods; dependability, real-time, security, safety-critical systems; programming languages, programming methodology, software engineering; architecture of distributed systems, self-organizing systems; networking, operating systems; performance analysis, simulation and modelling; planning and scheduling, algorithms and complexity; coding and information theory. Contributions could

[2]We use the neutral word "entity" rather than the word "agent", which has an existing technical meaning that is too specific.

also come from outside Computer Science. One example: inspiration might be obtained from the behaviour of social insects.

Challenges

Foundational work is required which provides a solid scientific and architectural framework, and effective engineering principles, for building such systems. Research that is mainly on applications should be discouraged.

The following issues are relevant among many others. Experience suggests that projects that address more than a few of these simultaneously are likely to be over-ambitious: it is better to gain a deep understanding of a few fundamental issues than to study a wide range of issues at a relatively superficial level. Simplifying assumptions are permitted – for instance concerning the issues not addressed – provided that some class of interesting and possibly useful systems is included, or there is some reason to expect that studying the given subclass of systems will yield useful insights.

- *Openness:* Sometimes we can make certain assumptions about the nature of entities, perhaps because they have all been built using a given programming language. In an open system, such assumptions cannot be made.
- *Scalability* It is desirable, but not essential, that approaches that deal with small systems should scale smoothly to deal with very large systems.
- *Model of computation:* What is a suitable model of computation? What are the programming abstractions that "package" these models so as to enable ordinary programmers to construct reliable and robust applications?
- *Programming:* Well-engineered programming languages that provide direct but flexible support for building such systems need to be designed and implemented, and then questions arise in the development, analysis and transformation/optimization of programs in such languages. What features would a type system for such a language include?
- *System development:* What is an appropriate logic for specifying and reasoning about such concepts? How do you understand a computation when you have only a partial view of it and your collaborators also have only (different) partial views? How does one test such a system? How does one organize distributed development of software? Which architectures are safe?
- *Adaptability:* How can an entity adapt to the situation it finds itself in, given its lack of knowledge of the state of the computation and the configuration of the environment?
- *Emergent behaviour:* Methods used to study the emergent behaviour of large dynamic systems may well be applicable.
- *Security barriers:* Networks are partitioned into administrative domains by firewalls and other security barriers. Movement of entities through security barriers should be possible under appropriate circumstances, without undermining the security of facilities.
- *Nature of entities:* Interesting classes of systems may be obtained if we make assumptions about the computational power of entities.
- *Nature of communication/interaction:* Different assumptions could be made about the nature and/or quality of communication. For instance: one-to-one versus broadcast or

multicast; reliable versus unreliable message delivery; timed versus time-free interactions; various message delivery disciplines, from totally ordered to unordered.

- *Distributed computing:* Issues and algorithms from distributed computing (agreement, failure detectors, self stabilization, clock synchrony, establishment of communication, etc.) are relevant but need to be reconsidered in this new context.
- *Performance:* What is an appropriate measure of performance?

Theme 2: Guaranteed Software Systems

Introduction

In our vision, the concept of "guaranteed software system" goes far beyond the more traditional notion of "verified software system". It is of course already difficult to provide suitable techniques for verification of functional and non-functional behaviour for algorithms and small software systems, but it is widely accepted that this should and can be done with the current technology. On the other hand it is often argued that verification of large software systems is not possible at all. Vice versa it is often claimed that software systems should be built up from small enough software components such that verification is possible for these components and that there are techniques how to combine components in a consistent way such that correctness of the software system follows from correctness of its components. We agree that this is a meaningful perspective which can be achieved as a medium term research goal. Hence, only the components have to be verified. This is a reason why we want to build up libraries of (at least) verified software components.

Correctness via verification or proof extraction -based on model checking or proof checking techniques- is surely not enough to achieve predictably reliable and secure systems. Verified algorithms may be implemented on a chip, in which case they need be tested. We advocate for the automatic generation of test sets from specifications or from their proof. The third important aspect of predictably reliable and secure systems is to predict performance, including time and space consumption, in the sense of complexity theory. Last but not least, reliability of software systems requires in addition to all the formal properties we have mentioned above also an intuitive understanding of the functionality of the system. This can be supported, in particular, by visual modelling techniques for different views of the system. At least in object-oriented design and system development the Universal Modelling Language UML has become a quasi-standard already.

Finally in order to achieve predictably reliable and secure systems it is important to guarantee maintainability and evolution capabilities. The purpose is to be able to adapt the software system to small changes of the environment, and also to migrate the system according to technological requirements like the change of execution platforms. These aspects of software redesign are especially important for information and communication infrastructures and it is certainly a long-term task to develop the kind of mathematical techniques that will allow to support maintenance, redesign and reengineering of guaranteed software components.

Today the commercial software market offers already "standard components off the shelf". Although these standard software components are sold in huge quantities no guarantee is given at all in the sense discussed above. This is one of the main reasons why todays software systems as well as communication and software infrastructures are still not reliable and crashes of software systems are considered as an unavoidable fact of life. This situation could change drastically if in the future the software industry would be able to build and sell guaranteed software systems from guaranteed components as discussed above.

Summarizing our notion of guarantee for software systems includes correctness, verification, certification, intuitive understanding, maintainability and evolution capabilities. Moreover, the main concept to achieve guaranteed software systems is that of components and compositionality which allows to built up guaranteed systems from guaranteed components.

Vision

The first vision of theme 2 is to support (at least separately) all the aspects for predictably reliable and secure systems, mentioned in the introduction, by suitable mathematical techniques. This is a task of the different communities listed below, where some basic results have been obtained already for some of these aspects (but not for all).

The second vision of theme 2 is to develop mathematical techniques for all the different aspects which can be integrated into a single coherent formal system. However, the mathematical techniques to guarantee each aspect is heavily dependent of the corresponding research community. This will require bringing together all these different research communities to develop compatible mathematical techniques which allow to guarantee all the aspects for predictably reliable and secure systems mentioned above.

The main vision of theme 2, however, is that after a decade of European research in the area of guaranteed software systems the "standard components off the shelf" offered by European commercial software houses are guaranteed software components in the sense discussed above. Moreover, there should be a composition technology which allows to construct a large part of commercial software systems from guaranteed components. This would lead to a significant difference in quality between conventional and guaranteed software systems where in the long run the guarantee becomes a must for all commercial software products by law.

Objectives:

- create and develop theories, languages and tools that support specification, analysis, certification (of functional and non-functional behavior), intuitive understanding and evolution of software components and their aggregation into predictably reliable and secure systems.
- build libraries of guaranteed software components, where the guarantee includes intuitive understanding and certification of functional behavior, resource consumption, performance analysis, as well as main-tenability and evolution capabilities.

Communities:

- specification
- verification
- software development
- testing
- algorithms
- algorithm engineering
- performance prediction
- UML

Challenges:

- bridge the gap between intuitive understanding and precise semantics of software com-

ponents (visual techniques for specification, animation, and integration of components and formal techniques for visual methods).

- accurate prediction schemes of resource consumption of software components and their composition for predicting systems behavior (performance analysis, resource consumption, memory hierarchy)
- certification of software components (testing, program checking, verification, model checking, program extraction from proofs or specifications, automatic program generation)
- mathematical notations for guaranteed software components including specification, timing behavior, testing, proof, development process, and intuitive understanding. Languages and tools based on such notations.
- collecting guaranteed software components into libraries.
- aggregation of guaranteed software components into predictably reliable and secure systems.
- mathematical techniques for maintenance and evolution of software components and software libraries.
- predictable reliability/security for evolutionary systems built of guaranteed components.

Theme 3: Nature-Inspired Computation

Introduction

Work on nature-inspired computational paradigms has been motivated by the recognition that natural systems compute in an entirely different way than (conventional) computers. Natural systems are highly complex, nonlinear, parallel and they have the capability to perform certain computations much faster than the fastest digital computers in existence today. This fact is expected to be helpful in the solution of hard computational systems in all branches of science and engineering.

In recent years new computational methods inspired by nature for data analysis and simulation of complex processes have been explored: Artificial Neural Networks, Cellular Automata, Genetic Algorithms, Probabilistic and Fuzzy Processing, Biomolecular Computing, Quantum Computing. The results from these domains indicate considerable possibilities for the solution of problems in areas like pattern recognition, optimization, searching, simulation (of complex systems and processes).

Vision

Make productive the emergent effective computation of complex natural systems in order to solve hard computational problems.

Contents/Methods

- biomolecular computation (based on bio-chips),
- Genetic algorithms and evolutionary strategies,
- Chaos-based computation,
- Neuro-computation,
- Agent-based computation,
- Quantum computation,
- Computations exploiting stochasticity and randomness.

Challenges

- Devise/understand heuristics,
- Tame hard problems like pattern recognition, global optimization, searching, simulation of complex system behaviour (pattern formation and control, internet, organisms, organizations, management, finance, games, population dynamics, ecology, urban/rural planning, natural resource management),
- Understand and evaluate the proposed methods in order to decide their applicability and use in practice,
- Reach the frontiers of feasible computation.

Communities

Computer scientists, mathematicians, engineers, biologists, physicists.