
Grand Challenges

in *Software Development and Certification PowerTools*
and *Global Computing Foundations and Systems*

Manuel Hermenegildo

CS Department, Technical University of Madrid, Spain
(and CS and ECE Departments, University of New Mexico, USA)

ISTAG Member

ISTAG WG on Grand Challenges

Brussels, January 29/30, 2004



IST Challenges

Infrastructure (HW) challenges...

- Ubiquitous, high-speed, lowest cost connectivity, Miniaturization of sensors and computing (Moore's law does not just happen), Human-machine interfaces (displays, speech, bionic devices, brain interfaces, ...), etc.

... and challenges when developing applications on it:

- intelligent behaviour, speech, information mining, augmented reality, life log, telepresence, virtual organizations, telework, collaborative work, health, e-commerce, etc., etc.
- Will focus on the latter, mainly on how we develop such applications.

Software Development is a Key, Pervasive Element

- Many of the challenges pointed out by previous speakers include fundamentally *challenges in SW*:
 - SW architecture / methodology (Sandewall/Weigel/Reinhart)
 - Application interoperability / integration (Weigel/Reinhart)
 - Application ease of use (Sandewall/Weigel/Reinhart)
 - Safety/security (Weigel/Reinhart)
 - ...
- SW is clearly a pervasive technology cutting across all others: a real **enabler**.
- But, we still use quite low-level means to develop it. And, specially, to find and really eliminate bugs.

A Grand Challenge: “The SW PowerTools Kit”

Develop the technology (“PowerTools Kit”) that will allow us to:

- write more *powerful, easy to use, intelligent, pervasive* software
 - that is *reliable* (e.g., bug free and *certified* so)
 - in a *shorter time and with lower cost*.
-
- In Europe 59% of SW developed internally, 16% on outside contractors, and only 24% packaged.
 - Growth of individual software productivity significantly behind growth of needs and performance/cost ratio of hardware.
 - The cost of development of *correct* SW (writing code, debugging, certifying) is directly at the core of European *productivity*, and *competitiveness*, fundamental Lisbon objectives.

A Grand Challenge: “The SW PowerTools Kit” (Contd.)

- Fundamental aspects:
 - Writing code (languages, environments, compilers, ...).
 - Finding bugs in programs –serious hindrance to productivity.
 - Program certification: *required* (at different levels, depending on application) in open, interoperable, mobile code, secure, component-based, etc., environments.
 - Still *hard to automate*.
 - But enormous potential competitive advantage for EU industry at a time of much competition.
- Area with very high European know-how and traditional advantage. Major advances achieved in past few years.
(But competitors cutting quickly into this advantage!)

SW Development Challenges

- Examples of at least two related challenges from the UK exercise:
 - *The debugging, verifying compiler:*
finds bugs and certifies a program bug free.
 - *Fundamental theory and programming tools for global overlay computing:*
allows the understanding and development of Aml-like applications and the platforms they run on.

Global (Overlay) Computing

- Global Computer:
 - Programmable computational infrastructure,
 - distributed on a worldwide scale, and
 - available globally.(e.g., the Internet)
- (Global) overlay computer:
 - Abstraction that can be implemented on top of a global computer.

Global (Overlay) Computing: Examples

- Examples:
 - The Web, P2P file sharing schemes, IPSEC, programming languages for distributed computing, systems of trusted mobile agents, search engines, distributed networks, ...
- ... and, very notably:
 - The GRID (current and future)
(great potential for standardization!)
 - The Aml (ambient intelligence) vision
clearly requires overlay computing technology.
- Several overlay abstractions can be “stacked.”

Do we know enough to build them?

- Up to a certain level —yes:
Some of the necessary results have already been produced by previous computer science and engineering research:
 - The powerful operating systems providing all services.
 - The computer languages in which it is all written (+ the associated compilation and run-time support technology).
 - The program development environments.
 - The communication protocols that make the Internet possible.
 - The algorithms (including, e.g., security).
 - The theory that allows certifying some of these as correct.
 - The hardware (processors, memories, interfaces, optical communication devices, etc.) on which it all runs.
- Made current progress (Web, current GRID, P2P, etc.) possible.

Do we know enough to build them? (Contd.)

... but *many open questions* remain.

- We still do not know how to solve, in a satisfactory way, issues like:
 - Scalability and modularization
 - Program development
(specially complex, concurrent/distributed programs)
 - Validation, certification, debugging
 - Task identification and partitioning
 - Distribution of data and tasks
 - Security
 - Resource management
 - Discovery
- Could be addressed by a Grand Challenge in

Global Computing Foundations and Systems
--

We Need Fundamental Advances in Computer Science

- The development of
 - the Ambient Intelligence vision,
 - other global overlay computers,
 - and most of the other Grand Challenges,

requires *long term, foundational advances* (i.e., *getting to the core of issues and separating the essential from the incidental*)
in computer science areas such as:

security frameworks,	resource usage,
methods and infrastructures for trust,	abstraction mechanisms,
models of interaction and cooperation,	components and modularity,
programming languages and concepts,	validation and verification,
algorithmics and software principles,	computational paradigms,
program analysis technology	semantics, logic, etc.

Examples: Verification / Proof Carrying Code

- Nobody would put in a car a wheel that does not come with specifications and a certificate from the manufacturer.
- Same when accepting code from others in the GRID, downloading code, distributed program development, etc.
 - Imagine that tasks to be run come with certificates that they meet certain properties (specifications) regarding, e.g., resource usage, correctness, safety, completeness, ...
 - Certificate may come from “the word of a human” but best is to
 - generate the proof automatically
 - get the proof as part of the certificate

Examples: Verification / Proof Carrying Code (Contd.)

- We need to develop:
 - Languages for describing such specifications.
 - Programming environments that allow writing these properties and check them.
 - Execution environments that take this into account.

→ we need to be able to reason about and prove automatically properties of programs meant for global overlay computers.
- The state of the art is such that (with techniques such as, e.g., *abstract interpretation*, model checking, etc.) it is starting to be possible to do this verification automatically for relevant properties and in many cases.
- Needs further development + application in future systems.

Examples: High-Level, Concurrent Languages

- Example: In what language to describe complex applications with distributed processing?
 - Current languages (e.g., Java): incorporate some of the progress made in language technology (objects, abstract machines, dynamic memory management and garbage collection, ...)
 - But are very primitive in other respects (e.g., very primitive concurrency and coordination primitives, no higher-order, no built-in problem-solving facilities, no constraint solving, limited typing, no verifying compiler, ...)
- We need new *language concepts* which can express well task partitioning, distribution, coordination, resource management, ...
- Advance by prototyping integrated next-generation languages and also coordination languages to use on top of existing languages.
- They will percolate to everyday languages as in the past (e.g., Java).

Examples: Automatic Resource Management

- The technology currently exists to, given a program procedure,
 - infer upper and lower bounds on its consumption of memory and CPU, size of output, etc.
 - as a function of input data sizes and types.
- This can be used to automatically (e.g., by the compiler):
 - Compute the time that the data will be in transit and calculate the communication/computing/cost tradeoff automatically.
 - Decide whether to run a task locally or at a remote computer (and which one).
 - Decide to join several tasks (with no deadlock).
 - Decide where to place data, when to migrate it, and how much.

(ideal for integration with “economic models” of scheduling)

Examples: Distributed Automatic Memory Management

- Substantial progress in several areas is need to achieve higher-level abstraction of concurrency, network transparency, resource awareness, etc.
- Example: distributed memory management (in part, the equivalent in a distributed environment to garbage collection in sequential languages).
 - Automatic distribution and caching of data
 - Distributed cache coherence protocols
 - Distributed garbage collection
- Very related to P2P.
- Some exciting new developments in this area also.

Examples: Automatic Parallelization / Task Partitioning

- Scheduling of complete jobs for parallelism (“parameter scheduling”) can only go so far.
- Many real-life distributed commercial applications (e.g., Aml applications) will need to distribute much smaller tasks –pieces of computations, pieces of the data, etc.
- This is too hard to do for humans.



- We need to develop more powerful compilers that do this automatically.
- The technology (pointer aliasing analyses, granularity analysis, complexity analysis, etc.) is finally starting to be available to do this for realistic programs.
- *This (and the following) seem vital technologies for future GRIDs.*

Open Source: An Opportunity for a Stronger EU SW Industry

- We simply cannot give up on having a stronger SW and SW development tool industry.
- Real window of opportunity exists via Open Source movement.
- An area of EU reputation and strength.
- More a policy / regulatory issue than a research issue:
 - Stopping *SW patents* needed to have a real chance.
 - *Legislation* to outrule use of proprietary standards in public procurement specifications.
 - *Recommendations* for public procurement: use to promote research rather than the opposite (more usual now).
 - Business models (support, development, etc.).
- (See ISTAG WG4 recommendations.)

Summary

- Grand Challenge Proposals:
 - *Software Development and Certification “PowerTools”*
 - *Global Computing Foundations and Systems*
- Policy/Regulatory Issues Proposals:
 - Regulate non-discrimination/encouragement of Open Source.
 - The SW Patents problem.
 - Encouraging research and innovation via procurement.
- We need *long term, foundational* advances in a number of Computer Science areas.

Grand Challenges

in *Software Development and Certification PowerTools*
and *Global Computing Foundations and Systems*

Manuel Hermenegildo

CS Department, Technical University of Madrid, Spain
(and CS and ECE Departments, University of New Mexico, USA)

ISTAG Member

ISTAG WG on Grand Challenges

Brussels, January 29/30, 2004

