

One-Day Workshop on “Teaching Resilient Computing”

Teaching Resilient Computing in Antwerp: Report, Considerations, and Vision

Vincenzo De Florio



<http://www.pats.ua.ac.be>

- Reporting on the teaching activities in resilient computing at the [University of Antwerp](#)
- Stating what are the key messages that we believe should be part of those activities
- Stating our “vision” to a possible Master level degree on Resilient Computing

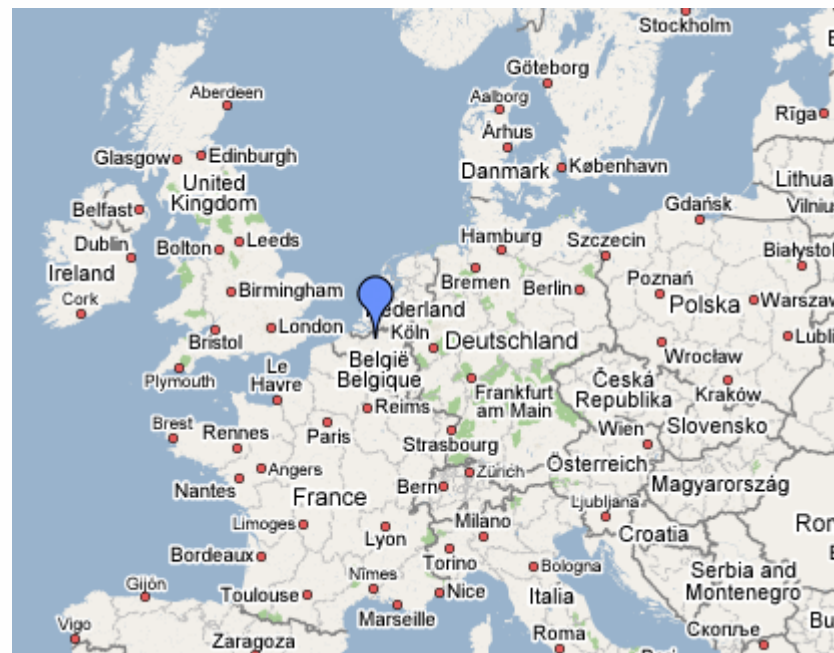


1. Teaching Activities

Teaching Resilient Computing at the UA



- UA, University of Antwerp, Belgium
- Quite young university
 - 2003, merge of three smaller universities
 - roots go back to 1852
- Approximately 10.000 students
 - Third largest in Flanders





- Over 1.000 of these students – exchange students not included – are from foreign countries
- Seven Faculties, including Sciences
- Bachelor and master programmes, including Computer Science
- ...which *does not include* any programme on dependability engineering
 - Dependability education is scattered among several courses (computer security is the only course focusing explicitly on one of its attributes)



- Since 2 years I'm responsible for a mandatory course
 - [Seminarie Informatica](#), last year M.Sc. CS

Performance Analysis of Telecommunication Systems
Research Group

Seminarie Informatica

By V. De Florio

The seminarie informatica course consists of 10 seminars on hot topics of computer science. The theme of the current cycle of seminars is "Fault-tolerant Systems: The Software Viewpoint".

1. 25 October 2006, 14:00 to 15:30, room T105, Groenenborgercampus.
[Vincenzo De Florio, UA/PATS: Fault-tolerant Systems: The Software Viewpoint. 06/12/2006.](#)



- Main role I envisage: providing students with a view to different approaches to the **design** and **use** of resilient services in several research units in Belgian universities and some enterprises
- The first of these seminars presents the workplan, provides basic information and reports on the local research activities
- More than this, it begins laying the foundation to a number of “key messages” in resilient computing to be reinforced and explained in the lectures to follow



- Last year: Adaptive-and-dependable services
 - “Services that are prepared to continue the distribution of a fixed, agreed-upon quality of service despite:
 - Internal changes (e.g., resource depletion, the manifestation of a hidden design fault...).
 - Changes in the location of the client software (e.g., the robot embedding the service is moving elsewhere).
 - Changes in the characteristics of the environment (e.g., temperature is increasing because of a fire);
- we call this *change tolerance*” [presentation 1]



Current cycle: [Fault-tolerant Systems: The Software Viewpoint](#)

- V De Florio, UA: Fault-tolerant Systems: The Software Viewpoint
- G Deconink, K.U.Leuven: A fault-tolerant info'structure for energy applications
- Peter Van Roy, Catholic University of Louvain: Self Management and the Future of Software Design.
- Marc Leeman, **Barco Security and Monitoring Division**: Control for High Available, Mission Critical Networked Visualisation Systems.
- Nico Janssens, K.U.Leuven: Dynamic Software Reconfigurations in Programmable Networks.
- Raphael Collet, UCL: Asynchronous failure handling with Mozart.
- Danny Weyns, K.U.Leuven: Architecture-Centric Approaches for Software Engineering with Situated Multiagent Systems.
- Yves Younan, K.U.Leuven: Countermeasures for security vulnerabilities in C and C++ programs
- Harry Vermeulen, **KBC Bank**, Migration to Service-oriented Architectures.
- Bruno Volckaert, Univ. Ghent, GRID-based fault-tolerance architectures



- In both series, we began with the missing pieces
 - Classical works by Prof. Laprie on dependability, its basic concepts, attributes and terminology
 - Three objectives:
 1. Enhancing the awareness
 2. Providing key messages, fighting common misconceptions
 3. Substantiate the theory with practice



- Then we focus on a particular aspect
 - **Application-level fault-tolerance**
- The other seminars provide
 - Other approaches
 - Other ideas
 - Other solutions



- Passing the exam means understanding the concepts and being able to reason with them
 - Differences, common points
 - Reports and lessons learned from use cases

2. Providing key messages

- Excerpts from the first lecture
- In red, the messages I try to convey



- Human society more and more *expects* and *relies on* good quality of complex services supplied by computers

Here I try to explain that the, the more the performance increases, the more the ease-of-use is enhanced, the more we tend to rely on computer services. Behind the lines what I say is, « Is this safe? »

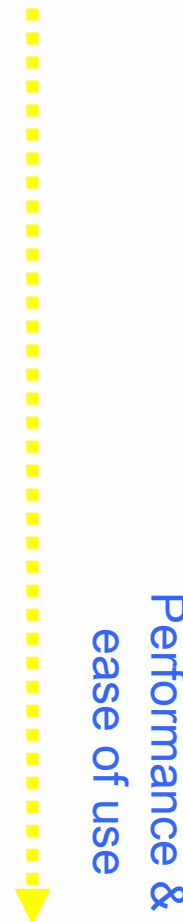
- **Computer services are everywhere – do you trust them? Why?**
 - **Religious answer**
 - ⇒ **“Things will go OK”**
 - **Societal answer**
 - ⇒ **“I hope things are OK” – statistics, fraction**
 - **Engineer’s (ethical) answer**
 - ⇒ **“I want a quantitative guarantee” – numerator**

Messages are: (1) If a computer service is crucial, I want evidence that it won’t fail and (2) the engineer must (should) not see the fraction, but the numerator – and design accordingly

Excerpts from first lecture



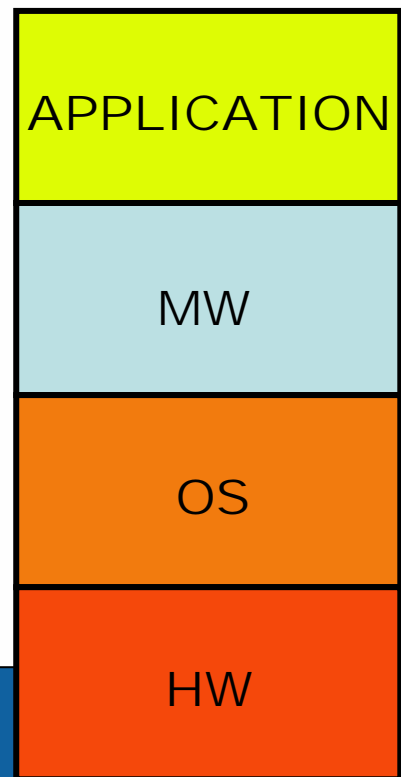
- Consequences of a failure in the '40s:
(Computers as fast solvers of numerical problems)
 - Errors in computations, long downtimes
- Consequences of a failures nowadays:
(Computers controlling nuclear plants, airborne equipment, healthcare...)
 - ❑ Incalculable penalty (catastrophes)



The more we improve things, the more badly things can go wrong

- Traditional solution:
Hardware Fault Tolerance
- This is an important ingredient, but not the only one needed today!
 - Complexity is also in the SW layers
 1. Hierarchies of complex abstract machines

SW



M1: Software FT is fundamental



- Complexity is also in SW layers (cont.'ed)
 2. Software is often networked and distributed
 3. Relationships among software components are often complex
 4. Object model \Rightarrow Easier SW reuse \Rightarrow Hidden + explicit Complexity

M2: Software is extremely complex \Rightarrow error prone

M3: Software reuse is good, but it introduces hazards

**Object orientation, service orientation are compositional tools
What about the quality of what you are composing out of 'em?**



- In conclusion: “No amount of verification, validation and testing can eliminate *all* faults in an application and give *complete* confidence in the availability and data consistency of applications”

M4: « xW faults result in x failures » is WRONG!

⇒ Fault tolerance in SW is key

! SW failures can have the same extent in consequences of failures in HW

Ariane 5 !

Excerpts from first lecture



The lighter the color,
the more general purpose
the (virtual) machine

The lighter the color,
the more complex
*the problem of
expressing fault tolerance*

M1': we can't exclude any layer (E2E arg)...
and M5: ALFT is no piece of cake



“No amount of verification, validation and testing can eliminate *all* faults in an application and give *complete* confidence in the availability and data consistency of applications” [Huang&Kintala, '95]

“The only alternative and effective means for increasing software reliability is that of *incorporating* in the application software provisions for SFT” [Randell, 1975!]

**M6: « OK, SW is important too, so let's get rid of all bugs and problem's solved »:
NO WAY!**



- The Application software has to manage
 - Functional aspects
 - Fault tolerance (FT) aspectsat the same time / in the same space

M5': Ouch!



- Hazard : code intrusion
 - FT provisions are specified side by side with the service
 - Conflicting design concerns
 - Overall design complexity gets increased
 - ⇒ Larger development and maintenance costs & times
 - ⇒ Larger probability of introducing software bugs

M7: Complexity is a threat



- Separation of design concerns (**SDC**)
 - In what follows we call an “ALFT” a means to express fault tolerance in the application software
 - A criterion to compare ALFT’s is by their degree of **SDC**

We introduce a first criterion to judge / compare / reason about ALFTs

M7: SDC reduces complexity



- Hazard : porting code \neq porting service
 - FT code assumes fault model = $f(\mathbf{e})$
 1. If \mathbf{e} changes, or
 2. If the code is moved to another environment \mathbf{e}'
 - **M8: Porting a service is different from porting a code.**
 - A code is a physical entity, stored as voltage values in a set of memory cells, which is supposed to drive the production of a service. A program is independent of its environment, whereas a service is.
- the QoS may degrade

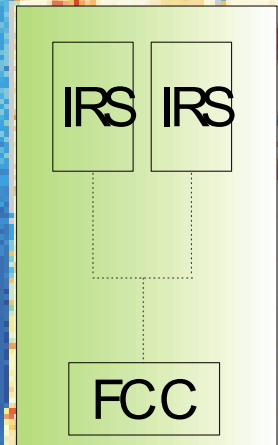
Excerpts from first lecture



- Hazard: porting code \neq porting service
- An interesting case: Ariane 5 501
 - Ariane 4 missions software re-used in Ariane 5
 - The early part of the trajectory of Ariane 5 differed from that of Ariane 4 and resulted in quite higher horizontal velocity values

This could be a case study for the exam

...370
Million
Euros
in the
sink





Problem: service portability

- Porting FT comes not for free
- “*Hardwired*” fault model = static environment
- More difficult to adapt / test / maintain
- More prone to Ariane 5 - effects

M9: “ What is the most often overlooked risk in sw engineering?
That the environment will do something the designer never
anticipated ” [J. Horning]



- Adaptability (**AD**)
 - Does the ALFT provide means to adapt, dynamically, to new environmental conditions?
 - A criterion to compare 2 ALFT's is by their degree of **AD**

M9': Designing with **AD** in mind is better



- Redundancy has a cost, and non negligible risks are
 - Overshooting, i.e., over-dimensioning a fault-tolerance provision with respect to the actual threat being experienced.
 - Undershooting, namely underestimating the threat in view of an economy of resources.
- Consequently, modern (mobile) services ask for fault-tolerant provisions whose fault model take into account the dynamicity of the environment (**AD** services)



Problem: adding complexity can decrease the dependability

- The ALFT (the means to express FT) must be based on a simple strategy
- It must be syntactically adequate to host several mechanisms

M7: Complexity is a threat

Counterpositive M7: A simple, neat, clean design is very positive

(Never forgetting Einstein's quote

«Make everything as simple as possible, but not simpler» ;-)



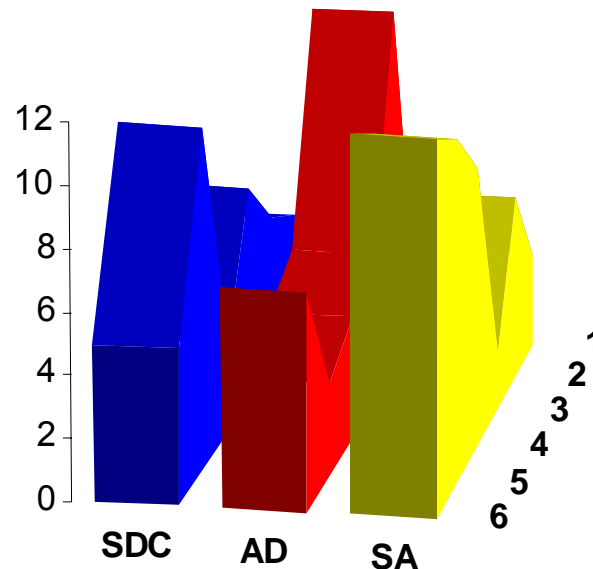
- Hazard:
 - “Languages shape the way we think ...” [Warf]
 - “If all you have is a hammer, everything looks like a nail” [usr/share/fortune]
 - !! ...but – is it really a nail?
- Syntactical Adequacy (SA)
 - Does the ALFT provide simple means to host many FT solutions?

M10: SA has an impact on complexity

Summary: A decalog of messages, a triple of properties



- Separation of design concerns (**SDC**)
- Adaptability (**AD**)
- Syntactical Adequacy (**SA**)
≡ A “base” of attributes we can use to compare ALFT’s with one another





3. Vision



- Decalog and triple have been used to design these series of seminars
 - As much as possible, speakers provide stories that confirm and replay the messages in different contexts
 - Industrial & academic
 - Domain specific
 - Students need to grasp this structure and highlight relations in order to pass



- I believe a master course on resilient computing should expand this « model »
 - A core of fundamental courses should provide the basic know-how
 - Other courses should focus on deepening the Attributes, Impairments and Means
 - Also fault-tolerance specifications, fault models, failure semantics, system assumptions

- But some other courses could be designed « around » the 10 messages and 3 properties
 - « Adaptive fault-tolerance » (how to design provisions that are attuned to the environment)
 - Varying the environment, or moving the service, the fault tolerance provision does not become stale
 - « Application-level fault-tolerance »
 - Designing FT software through approaches such as
 - Single-version FT
 - Multiple-version FT
 - Object model
 - Linda Model
 - FT Languages
 - Recovery metaprogram
- and so forth

- Other ancillary courses should also be offered
 - Language design, compilers and translators
 - Computer architectures
 - E.g., memory technologies (and failure assumptions)
 - Single event effects (single-event latchup, single-event upset, single-event functional interrupt) vs. single-bit errors, ...
 - Impact of architectural resources on the fault model
 - MMUs?
 - Etc
 - Computer networks
 - Especially mobile services design
 - Dynamic resources, unstable environments...

- Finally, we must never forget that "It is the supreme art of the teacher to awaken joy in creative expression and knowledge"
- Courses should be designed and given in a creative way and trying to comply to the new characteristics of younger pupils
 - New way of learning, new way of teaching
 - Virtual reality could be used as an educational tool

Thank you for your attention

Questions?