Workshop “Challenges and Directions for Dependable Computing”

41st Meeting of IFIP WG 10.4 — Saint John, US VI — January 4-8, 2002

— — —

01. From Fault Injection Experiments to Dependability Benchmarking
    Jean Arlat (LAAS-CNRS, Toulouse, France)
02. Building a Hierarchical Defense: An Immune System Paradigm for Fault-Tolerant System Design
    Algirdas Avižienis (University of California at Los Angeles, USA)
03. Dependable Computer Networks
    Dimiter R. Avresky (Northeastern University, Boston, MA, USA)
04. Interactions and Tradeoffs between Software Quality Attributes
    Mario Barbacci and Charles B. Weinstock (SEI-CMU, Pittsburgh, PA, USA)
05. Production Process of Dependable Systems / Human Factors / Emerging Applications / Middleware Platforms
    Andrea Bondavalli (University of Florence, Italy), Felicita Di Giandomenico (IEI-CNR, Pisa, Italy) and Luca Simoncini (CNUCE-CNR and University of Pisa, Italy)
06. Creating Robust Software Interfaces: Fast, Cheap, Good — Now You Can Get All Three
    John DeVale and Philip Koopman (Carnegie Mellon University, Pittsburgh, PA, USA)
07. Pervasive Dependability: Moving Dependable Computing Towards Mainstream Systems
    Cristof Fetzer and Richard D. Schlichting (AT&T Labs–Research, Florham Park, NJ, USA)
08. Dependability Challenges in Model-Centered Software Development
    David P. Gluch (Embry-Riddle Aeronautical University, Daytona Beach, FL, USA)
09. Report Correlation for Improved Intrusion Detection and Fault Diagnostics
    Walter L. Heimerdinger (Honeywell Technology Center, Minneapolis, MN, USA)
10. Future Automotive Scenarios and R&D Challenges
    Günter Heiner (DaimlerChrysler AG, Berlin, Germany)
11. Dependability Challenges in Pushed-based Systems
    Yennun Huang (PreCache Inc., Bridgewater, NJ, USA)
12. Dependability Challenges in the Field of Embedded Systems
    Hermann Kopetz (Vienna University of Technology, Austria)
13. Dependability: Information Assurance Research Agenda
    Jaynarayan H. Lala (Defense Advanced Research Projects Agency, Arlington, VA, USA)
14. Research Directions in Trustworthy {Trusted, Dependable} Computing
    Carl E. Landwehr (National Science Foundation, Arlington, VA, USA)
15. (Some Remarks About) Dependability of Large Networked Systems
    Jean-Claude Laprie (LAAS-CNRS, Toulouse, France)
16. Can we Avoid Faults in Requirements Capture Process? / Which Computational Models for Dependable and Real-time Mobile Computing, Autonomous Complex Systems?
    Gérard Le Lann (INRIA, Rocquencourt, France)
17. Some Reflections
    Brian Randell (The University of Newcastle, England)
18. Designing and Assessing Adaptive Dependable Distributed Systems: Putting the Model in the Loop
    William H. Sanders (The University of Illinois at Urbana-Champaign, USA)
19. The European Dependability Initiative in the RTD Framework Programme 6th: An Opportunity for Global Collaboration
    Andrea Servida (European Commission DG INFSO C-4, Brussels, Belgium)
20. High End Commercial Computer Fault Tolerance: Trends and Directions
    Lisa Spainhower (IBM Corp., Poughkeepsie, NY, USA)
21. Metrics for the Internet Age: Quality of Experience and Quality of Business
    Aad van Moorsel (Hewlett-Packard Laboratories, Palo Alto, CA, USA)

From Fault Injection Experiments to Dependability Benchmarking

Jean Arlat
LAAS-CNRS, Toulouse, France

Background

Fault injection has long being recognized as a pragmatic way to assess the behavior of computer systems in presence of faults. It provides a useful complement to other dependability assessment techniques ranging from measurement to axiomatic methods, especially when used in combination with them.

Fault injection is meant to speed up the occurrence of errors and failures by the development of controlled experiments in which the target system is deliberately exposed to artificial faults. It allows for understanding the effects of real faults and thus of the related faulty behavior of the target system. The contribution of fault injection to dependability assessment encompasses both fault forecasting and fault removal [1]. In particular, fault injection can be seen as a means for testing fault tolerance mechanisms with respects to special inputs that are meant to cope with: the faults.

For the past 30 years, many valuable efforts have been reported that address the use of fault injection to contribute to the validation of fault-tolerant systems, sometimes in cooperation with other techniques such as analytical modeling and formal verification. In relation to analytical modeling, the estimation of the distribution of the coverage achieved by the fault tolerance mechanisms is most of the time the principal objective of the fault injection experiments.

Numerous techniques have been investigated ranging from i) simulation-based techniques at various levels of representation of the target system (technological, logical, RTL, PMS, etc.), ii) hardware techniques (e.g., pin-level injection, heavy-ion radiation, EMI, power supply alteration, etc.), and iii) software-implemented techniques that support the bit-flip model in memory elements [2].

While the first technique considers a simulation model of the target system, and thus can be applied in the early phase of the development process, the last two apply on the real system or at least a hardware-software prototype of the target system, and thus incorporate actually all (or most) implementation details that characterize the target system. Hybrid techniques, in particular using scan-chain devices incorporated in modern ICs have also emerged. Many supporting tools have been developed to facilitate and automate the conduct of fault injection experiments based on these various techniques [3].

Besides its contribution for supporting the evaluation of fault-tolerant systems, fault injection also proved very much useful for characterizing the behavior of computerized systems and components in presence of faults. Recently, particular emphasis has been put on software executives and operating systems, but the studies carried out span the full range of levels related to information processing: from the physical structure of IC devices to the Internet infrastructure. In addition, three major potential benefits brought by fault injection can be identified that concern its contribution to fault removal: i) fault injection experiments can be tailored as special tests to help reveal fault tolerance deficiency faults, ii) the so-called mutation testing technique, featuring simple modifications of a software program, can be used both to help elaborate a test for that program or to assess several software testing techniques, iii) fault dictionaries can be derived from fault injection experiments to support fault diagnosis and maintenance activities.

Building up on the significant advances made by the research efforts and on the actual benefits procured, fault injection made his way to industry, where it is actually part of the development process of many providers, integrators or specifiers of dependable computer systems (e.g., Ansaldo Segnalamento Ferroviario, Astrium, Compaq/Tandem, DaimlerChrysler, Ericsson SAAB Space, ESA, Honeywell, IBM, Intel, NASA, Siemens, Sun, Volvo, just to name some). This definitely confirms the value of the approach.

Some Challenges for Dependability Benchmarking

In spite of several pioneering efforts made during the 1990's (e.g., see [4-8]), there is still a significant gap between the level of recognition attached to robustness benchmarks and fault injection-based dependability benchmarking, on one hand and the wide offer and broad agreement that characterize performance benchmarks, on the other hand. Much effort is clearly needed before similar standing can be achieved.

Based on the working definition proposed by the IFIP WG 10.4 SIGDeB[1], a dependability benchmark can be viewed as: "A way to assess measures related to the behavior of a computer system in the presence of faults,

[1] See http://www.dependability.org/wg10.4/SIGDeB.

supporting the evaluation of dependability attributes or the characterization of the systems into well-defined dependability classes."

Three major properties have to be fulfilled by a dependability benchmark [9]: i) *usefulness*, for the benchmark users in supporting a relevant characterization and/or quantification of the dependability features and attributes that are of interest to them and easy to interpret, ii) *fairness*, in the sense that it should support meaningful comparisons of various target systems, iii) *acceptance* by the computer industry and/or the user community as an agreed reference that is to install and run on various platforms.

First, it is important to point out that a *dependability benchmark* cannot simply be achieved as mere combination of performance benchmarks and experimental dependability assessment techniques, such as fault injection. There are several specific concerns that need to be accounted for. In particular, several dimensions have to be considered depending on: the target system, the application domain, the life cycle of the target system, the usage of the benchmark, the measures of interest, etc. However, as is the case for performance benchmarks, it is expected that a dependability benchmark will heavily rely on experiments and measurements carried out on a target system.

Analogously to was introduced for fault injection-based evaluation [1], the experimental dimension of a dependability benchmarking can be characterized by an *input* domain and an *output* domain. The input domain corresponds to activity of the target system (*workload* set) and the set of injected faults (*faultload* set). In particular, the workload defines the activation profile for the injected faults. The output domain corresponds to a set of observations (*measurements* set) that are collected to characterize the target system behavior in the presence of faults. A set of specific or comprehensive *dependability measures* can then be derived from the analysis and processing of the *workload, faultload and measurements* sets. In practice, it is often the case that the selection of the set of *dependability measures* of interest has a significant impact on the determination of the other sets.

Although these sets have each a significant impact on the three properties identified above for a dependability benchmark, it is definitely the determination of the *faulload* set that poses the most significant problem, in this context.

Faultload Characterization

One important question is to figure out whether a limited set of techniques could be identified that are sufficient to generate the relevant *faultload* sets according to the dimensions considered for the benchmark or if rather many techniques are necessary. Indeed, although it is necessary to match the fault injection techniques with respect to the faults they are intended to simulate, the benefit of having to support only a limited number of techniques would be definitely beneficial from a portability viewpoint.

Moreover, it is worth pointing out that what is important is not to establish an equivalence in the fault domain, but rather in the error domain. Indeed, similar errors can be induced by different types of faults (e.g., a bit-flip in a register or memory cell can be provoked by an heavy-ion or as the result of a glitch provoked by a software fault).

Accordingly, two main related issues and questions have to be considered:

A) Fault representativeness (of a fault injection technique): To what extent the errors induced are similar to those provoked by real faults or by a representative fault model?

B) Fault equivalence (of fault injection techniques): To what extent distinct fault injection techniques do lead to similar consequences (errors and failures)?

So far, the investigations carried out concerning the comparison of i) some specific fault injection technique with respect to real faults (e.g., see [10-12]) and ii) several injection techniques (e.g., see [13-17]) have shown mixed results. Some were found to be quite equivalent, while others were identified as rather complementary.

In particular, in [11], it was found that about 80% of the mutations considered led to errors similar to real software faults. On the other hand, the study reported in [15] revealed that: i) the compile-time form of the software-implemented fault injection technique used to inject faults in the *code segment* of the executed application was found to sensitize the error detection mechanisms included into the MARS target system in a similar way than the physical fault injection techniques considered (pin-forcing, heavy-ion radiation, EMI), ii) faults injected into the *data segment* led to significantly distinct behaviors.

Accordingly, further investigations are clearly needed so as to better identify the technology that is both *necessary* and *sufficient* to generate the faultload to be included into a dependability benchmark. Several important issues have to be accounted for in this effort:

1) Although strict *fault representativeness* can be difficult to address directly because it relates to real faults that are difficult to trace, relevant insights can still be obtained by considering a (recognized) fault model as the reference in place of real faults. This form of analysis is thus very much related to the notion of *fault equivalence*, as expressed by question B) above.
2) Several relevant (ordered) levels of a computer system can be identified where faults can occur and errors can be observed (technological, logical, RTL, control/data flow, kernel, operating system, middleware, application). This form of hierarchy can be related to the one depicted in [18]. Concerning the faults, these levels may correspond to levels where real faults are considered and (artificial) faults can be injected. Concerning errors, the fault tolerance mechanisms (especially, the error detection mechanisms) provide convenient built-in monitors.
3) For characterizing the behavior of a computer system in presence of faults (from the point of view of either fault representativeness or fault equivalence), it is not necessary that the injected faults be "close" to the target faults (reference), it is sufficient that they induce similar behaviors. What really matters is that the respective error propagation paths converge before the level where the behaviors are monitored.
4) Two important parameters can be defined on these various levels:
   - the *distance* $\boldsymbol{d_r}$ that separates the reference faults from the level where faults are injected;
   - the *distance* $\boldsymbol{d_o}$ that separates the level where the faults are injected from the levels their effects are observed to be compared.

   The shorter $\boldsymbol{d_r}$ and the longer $\boldsymbol{d_o}$ and more it is likely that the injected faults will exhibit the same behavior as the targeted faults.
5) In the context of dependability benchmarking, it is mandatory that the application of the same benchmark on two (or more) target systems leads to a fair and meaningful assessment. However, in practice, it may well be the case that the presence of a specific fault tolerance mechanism (FTM) on one target system (and not on the other one(s)) will alter the error propagation paths. This has a significant impact on the fairness property whenever the FTM is implemented at level located in between the level of the targeted faults and the level where the faults are injected and thus intercept the error propagation paths. Indeed, assuming a perfect 100% coverage for the FTM, then the fairness (with respect to the targeted faults) of the benchmark using the faultload characterized by the injected faults would be zero. This could be simply expressed by introducing another distance parameter: the *distance* $\boldsymbol{d_m}$ separating the level where the faults are injected from the level where the FTM is acting. The kind of problem previously mentioned would thus be characterized by negative value for $\boldsymbol{dm}$.
6) From a dependability benchmarking point of view, it might not always be possible nor cost-effective to have access to the actual structure of the target system to identify *a priori* the faultload that would comply with the *fairness* property. Accordingly, an alternative could be to favor a standard fault injection technique that is less than perfect, but that is easy to implement and that covers a large faultload scope, and then to establish a dialogue with the target system provider in order to derive a fair interpretation or post processing of the benchmark measurements. This form of *a posteriori* processing can be related to the approach presented in [19]. Besides the sample faultload used in the fault injection experiments conducted was not representative of the probability distribution of occurrence of the real faults. Still, meaningful results could be derived by compensating the bias in the sample by an *a posteriori* weighted processing of the results.

Some Directions

In order to tackle this problem, a specific effort is currently being conducted within the framework of the IST-2000-25425 DBench project. As an attempt to elaborate on the conceptual framework sketched in the previous section, a comprehensive set of controlled experiments encompassing several levels of application of faults, as well as observations are being conducted. The target systems include hardware controllers for embedded applications, off-the-shelf operating systems and database applications. The fault injection techniques encompass VHDL simulation, pin-level fault injection, software-implemented fault injection and software mutation.

However, it seems necessary i) to seek for a more general consensus in particular with the industry sector (including computer and component providers, integrators, and end-users) and ii) to conduct a wider scale

controlled research effort on this topic. While the IFIP WG 10.4 SIGDeB provides a suitable forum for making progress with respect to the first item, a proactive research incentive is required to support the second item. In particular, it is worth pointing that new challenges are raised by the consideration of the impact of accidental failures on the communication infrastructures widely used to process information, not only from the point of view of availability, but also from the point of view of the potential threats caused to the protections aimed at ensuring security.

Acknowledgment

The views expressed here have maturated during the years I devoted to fault injection research and more lately within the framework of the project IST-2000-25425 (DBench), as well as the IFIP WG 10.4 SIGDeB. They also benefited from interactions with several colleagues and in particular from recent discussions with Yves Crouzet, Jean-Charles Fabre, Karama Kanoun, Philip Koopman, Jean-Claude Laprie, Henrique Madeira, Lisa Spainhower and Don Wilson. Nevertheless, the blame for any inaccuracy should be put on me.

References

[1] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.-C. Fabre, J.-C. Laprie, E. Martins and D. Powell, "Fault Injection for Dependability Validation — A Methodology and Some Applications", *IEEE Trans. on Software Engineering*, vol. 16, no. 2, pp. 166-182, February 1990.

[2] J. V. Carreira, D. Costa and J. G. Silva, "Fault Injection Spot-checks Computer System Dependability", *IEEE Spectrum*, vol. 36, 50-55, August 1999.

[3] M.-C. Hsueh, T. K. Tsai and R. K. Iyer, "Fault Injection Techniques and Tools", *Computer*, vol. 30, no. 4, pp. 75-82, April 1997.

[4] T. K. Tsai, R. K. Iyer and D. Jewitt, "An Approach Towards Benchmarking of Fault-Tolerant Commercial Systems", in *Proc. FTCS-26*, Sendai, Japan, 1996, pp. 314-323 (IEEE CS Press).

[5] A. Mukherjee and D. P. Siewiorek, "Measuring Software Dependability by Robustness Benchmarking", *IEEE Transactions of Software Engineering*, vol. 23, no. 6, pp. 366-323, June 1997.

[6] P. Koopman and J. DeVale, "Comparing the Robustness of POSIX Operating Systems", in *Proc. FTCS-29*, Madison, WI, USA, 1999, pp. 30-37 (IEEE CS Press).

[7] J. E. Forrester and B. P. Miller, "An Empirical Study of the Robustness of Windows NT Applications Using Random Testing", in *Proc. 4th USENIX Windows System Symp.*, Seattle, WA, USA, 2000 (USENIX Association).

[8] A. Brown and D. A. Patterson, "Towards Availability Benchmarks: A Cases Study of Software RAID Systems", in *Proc. 2000 USENIX Annual Technical Conf.*, San Diego, CA, USA, 2000 (USENIX Association).

[9] H. Madeira *et al.*, "Conceptual Framework - Preliminary Dependability Benchmark Framework", *DBench Project, IST 2000-25425, Deliverable CF2*, September 2001.

[10] R. Chillarege and N. S. Bowen, "Understanding Large System Failures — A Fault Injection Experiment", in *Proc. FTCS-19*, Chicago, IL, USA, 1989, pp. 356-363 (IEEE CS Press).

[11] M. Daran and P. Thévenod-Fosse, "Software Error Analysis: A Real Case Study Involving Real Faults and Mutations", in *Proc. ISSTA'96*, San Diego, CA, USA, 1996, pp. 158-171 (ACM Press).

[12] H. Madeira, D. Costa and M. Vieira, "On the Emulation of Software Faults by Software Fault Injection", in *Proc. DSN-2000*, New York, NY, USA, 2000, pp. 417-426 (IEEE CS Press).

[13] G. S. Choi and R. K. Iyer, "FOCUS: An Experimental Environment for Fault Sensitivity Analysis", *IEEE Trans. on Computers*, vol. 41, no. 12, pp. 1515-1526, December 1992.

[14] C. R. Yount and D. P. Siewiorek, "A Methodology for the Rapid Injection of Transient Hardware Errors", *IEEE Trans. on Computers*, vol. 45, no. 8, pp. 881-891, August 1996.

[15] E. Fuchs, "Validating the Fail-Silence of the MARS Architecture", in *Dependable Computing for Critical Applications (Proc. DCCA-6, Grainau, Germany*, M. Dal Cin, C. Meadows and W. H. Sanders, Eds.), pp. 225-247, Los Vaqueros, CA, USA: IEEE CS Press, 1998.

[16] P. Folkesson, S. Svensson and J. Karlsson, "A Comparison of Simulation Based and Scan Chain Implemented Fault Injection", in *Proc. FTCS-28*, Munich, Germany, 1998, pp. 284-293 (IEEE CS Press).

[17] D. T. Stott, G. Ries, M.-C. Hsueh and R. K. Iyer, "Dependability Analysis of a High-Speed Network Using Software-Implemented Fault Injection and Simulated Fault Injection", *IEEE Trans. on Computers*, vol. 47, no. 1, pp. 108-119, January 1998.

[18] Z. Kalbarczyk, G. Ries, M. S. Lee, Y.Xiao, J. Patel and R. K. Iyer, "Hierarchical Approach to Accurate Fault Modeling for System Evaluation", in *Proc. IPDS'98*, Durham, NC, USA, 1998, pp. 249-258 (IEEE CS Press).

[19] D. Powell, E. Martins, J. Arlat and Y. Crouzet, "Estimators for Fault Tolerance Coverage Evaluation", *IEEE Trans. on Computers*, vol. 44, no. 2, pp. 261-274, February 1995.

Building a Hierarchical Defense: An Immune System Paradigm for Fault-Tolerant System Design

Algirdas Avižienis
University of California at Los Angeles, USA

There is a growing demand for very dependable systems, especially in telecommunications and embedded systems. There are over 100 companies marketing such systems in the USA alone. Looking at the entire commercial market, we see three major approaches to building very dependable systems:

1. The "clustering" of complete systems;
2. Hot standby duplexing of critical subsystems;
3. The "intelligent platform management" for one chassis.

All three approaches do not sufficiently exploit fault tolerance support in hardware: in (1), the cluster is under software control; in (2), critical hardware and software modules are not fully protected themselves,e.g., the Ziatech architecture; in (3), the IPM hardware is itself not fault-tolerant. Furthermore, none of the three approaches offers support for design fault tolerance by means of design diversity.

The thesis of this talk is that an orderly evolution of very dependable systems absolutely needs a hardware-based infrastructure that supports, but does not require support from defenses at higher levels, such as clustering software. Such an infrastructure is in some ways very analogous to the autonomous, cognition-independent immune system of the human body.

The argument will be supported by an illustration of one possible approach to building an all-hardware fault tolerance infrastructure that supports design diversity and is compatible with various existing and proposed higher-level fault tolerance mechanisms.

Dependable Computer Networks

D. R. Avresky
Network Computing Lab., ECE Department
Northeastern University
Dana Research Bldg.
Boston, MA 02115
avresky@ece.neu.edu

Within several years, it will be common to have private networked computing systems, which will encompass tens of thousands of computing devices. Some of these networks (military, banking, e-business, transportations) will run mission-critical applications. The challenge is to build computer networks, based on COTS, that are inexpensive, accessible, scalable and dependable. Private virtual networks are created by composing a complex set of hardware and software components that are heterogeneous and subject to continuous upgrade, replacement, and scaling their numbers. These systems are too complex to model formally, and reducing the failure rate of individual components may not substantially reduce the rate of overall system failures due to unexpected interactions between components. One may consider that failures are inevitable. Having this in mind, we have to consider detection and recovery-oriented computing, a technique for achieving high availability that focuses on detecting and recovering from failures rather than preventing them entirely.

Dependability of service is of paramount concern for industrial and military applications when information is required. In addition to being continuously available, these systems must be free from data corruption. Absolute data integrity must be ensured through full self-checking and fault isolation. As the number of components increases, so does the probability of component failure. Therefore, a fault-tolerant technology is required in networked environments. Thus, one of the objectives of the research is to increase the dependability of scalable networked systems.

Interactions and Tradeoffs between Software Quality Attributes

Mario Barbacci and Charles B. Weinstock
SEI-CMU, Pittsburgh, PA, USA

We propose to discuss interactions and tradeoffs between different software quality attributes that should be identified as early as possible, preferably while the architecture is being defined.

An architectural description is the earliest representation of a system that can be analyzed for desired quality attributes, such as performance, availability, modifiability, interoperability, and security. The architecture also provides insights as to how these attributes interact, forming a basis for making tradeoffs between these attributes.

The Software Engineering Institute has developed a method for evaluating the architecture of a software-intensive system against a number of critical quality attributes, such as availability, performance, security, interoperability, and modifiability. The evaluation is based on test cases that capture questions and concerns elicited from various stakeholders associated with the system. The process of eliciting questions allows stakeholders to communicate directly, thereby exposing assumptions that may not have surfaced during requirements capture.

The purpose of the proposed talk is not to present the method ("Quality Attribute Workshops" http://www.sei.cmu.edu/publications/documents/01.reports/01tr010.html) but rather to discuss linkages between quality attributes that present opportunities for tradeoffs. In our experience conducting architecture evaluations, it is often the case that stakeholders (e.g., users, maintainers, developers) interpret terms like "security" or "dependability" in different ways, depending on their roles and experience.

The taxonomies produced by communities of experts on different quality attributes tend to be attribute centric. Thus, a dependability taxonomy might include "security" as a concern, somewhere down the tree, while a security taxonomy would have performance or dependability as sub-concerns of security.

What is missing in the taxonomies is the acknowledgment that no single attribute dominates the field, that the importance or relevance of the attribute depends on the context or environment in which the system operates. While the taxonomies are necessary, their usefulness would be greatly enhanced by explicit identification of tradeoffs between attributes. These could take the form of identifying techniques that affect more that one quality attribute, the extent to which the choice of technique constitutes a sensitivity point for an attribute, the extent to which a method improves multiple attributes (a win-win-win... situation!) or how much we lose of some attribute by using a technique that benefits some other quality attribute.

For example, "usability" in enhanced if users are given friendly advise when they make a mistake BUT not when the users make mistakes of certain types. Thus, the users do not get hints when they log-in with the wrong password -- this reduces usability but enhances security. This is a tradeoff between these two attributes.
The objective of the proposed presentation is to identify opportunities for collaboration between experts in different domains, to identify the extent to which techniques used to achieve or control some quality attributes have an impact on other attributes, and if necessary, to design appropriate experiments to validate tradeoffs hypothesis.

Position paper for the IFIP WG 10.4 Workshop, St. John, USA

"Challenges and Directions for Dependable Computing"

A. Bondavalli *, F. Di Giandomenico **, L. Simoncini***

* University of Florence, Italy
** IEI-CNR, Pisa, Italy
*** CNUCE-CNR and University of Pisa, Italy

Dependable systems provide some of the most critical, and intellectually challenging, problems and opportunities facing computer scientists, and indeed the whole computer industry. Over the past ten years or so, significant advances have been achieved in the topic of dependable computing systems. Today, we understand a lot more about many of the fundamental techniques involved in achieving dependability. But much further research is required when considering the application of these concepts to different scale systems, maybe distributed, their cost-effective integration into the system design process, the problems of validating a planned design with respect to some given required balance of various system properties and the human factors involved in their operation. A broad approach is required, encompassing theoretical studies, careful consideration of possible alternatives and their likely consequences, and (in some cases quite extensive) design and implementation activities.

Among the many open research problems that deserve further investigation, we focus here on the following ones:

1. **Production process of dependable systems:** Research in this area is needed to understand better how systems should be designed and implemented. Going from requirements to specification, to design decisions, to prototype implementation up to implementation, requires a continuous interaction between the decisions taken in each step with the validation and verification of each step, taking into account the possible problems arising in design integration and consolidation. This aspect becomes even more important when considering that the major recent trend in the area of system development is on building systems out of existing components (e.g., legacy systems, COTS components). Environments enabling system development through the composition of software components, offering methods and tools supporting the design, analysis, construction and deployment of such systems are therefore of primary importance. Integration, ability to compose and re-use appear to be very challenging issues in the validation (of both design and implementation) of complex systems, in particular dependable ones for controlling critical applications. A special effort towards the emergence of a new discipline - System Engineering – is required, which will encompass and integrate the current design discipline.
2. **Human factors:** It is now recognized that in complex systems the dependability of the system as a whole is heavily influenced by the dependability of man-machine interaction. Consequently, there's an urgent need to support system designers and software engineers with a deep understanding of human resources and capabilities so that they are able to identify and anticipate potential *usability* problems during the early

phases of design. In the area of safety critical systems, where formal description techniques are being successfully used for the specification and verification of dependable systems in addition/alternative to simulation tools, it is important to extend such formal approach to explicitly consider human factors within the design and assessment processes. In fact, with the user as a key "system" component, the design strategies also need to reflect/accommodate "human in the loop" as a design pre-requisite. Humans can interject more unpredictably via combinations of complex activities than any conventional fault model can ever envision. So the question becomes whether one needs to adopt a defensive design strategy that constrains what the user can do to perturb the operations or should one design around all foreseeable situations? In fact, one of the next major developments in computing will be the widespread embedding of computation, information storage, sensing and communication capabilities within everyday objects and appliances. Interacting with such systems will involve multiple media and the coupling of distributed physical and digital artifacts, supporting a continuous flow of information. This shift towards more *continuous interaction* between user and system is a direct consequence of two aspects of a new generation of interactive systems: *ubiquity* and *invisibility*. Since interaction devices will be spread in the surrounding environment, the distinction between *real world objects* and *digital artifacts* will become immaterial. Users will behave naturally as human beings without adopting a simplified behaviour that characterizes a state of *technological awareness*. Consequently, there's a need for the systems to adapt to users, to be aware of their operating context, and to be able to take autonomous decisions to some extent. Within this framework, human dependency on the correct behaviour of systems in many (if not all) aspects of everyday life has a growing impact.

3. **Emerging applications:** The pervasive spread of computer systems in most human activities and the technological advances have determined an increase of critical applications calling for dependable services, including new emerging ones, with great demand for *working* and *affordable dependability*. For such applications, such as e-commerce, financial/banking systems, telecommunication, embedded systems, the emphasis is not exclusively on pursuing top-level dependability requirements, but solutions have to be defined which accommodate a number of (more or less) equally important user requirements in a satisfactory manner. Scalability, heterogeneity, flexibility, distribution, timeliness are among the most challenging issues of dependability connected with new business and everyday life application scenarios. Assurance of a guaranteed level of *QoS* is the research objective in such contexts, where QoS encompasses many aspects such as traditionally-related dependability attributes, performance-related indicators, and other measures representative of user perceived quality of service. The connotation of the term *safety critical* system results also extended from usual environments, such as flight or nuclear plant control systems, to denote a larger class of systems that are becoming *critical* for their impact on individual's every-day life. In particular the very large spread of the same type of embedded systems which are operated by a very large number of non-trained users introduces a more general concept of safety criticality, which consists of possible catastrophic failures induced by a large number of individually non-catastrophic failures. For such systems the concepts of *usability* and *man-machine interface* are interconnected and will be a leading research problem.

4. **Middleware platforms:** The growing large-scale character of critical systems makes the correctness of such systems to be more and more dependent on the dependability of the infrastructures management. Current research on *middleware platforms* relates to enabling interoperation among the various hosts composing the system including resource-constrained ones, such as wireless personal digital assistants, whose usage should soon outrun the one of PCs, which further requires to account for the diversity of the underlying interconnection networks. Introduction of dependability and timing constraints into *middleware platforms* is still an open problem.

Creating robust software interfaces: fast, cheap, good Now you can get all three

John DeVale & Philip Koopman Carnegie Mellon University

Extended Abstract

Although our society increasingly relies on computing systems for smooth, efficient operation; computer „errors‰ that interrupt our lives are commonplace. Better error and exception handling seems to be correlated with more reliable software systems. Unfortunately, robust handling of exceptional conditions is a rarity in modern software systems, and there are no signs that the situation is improving. In this talk we shall examine some central issues surrounding the reasons why software systems are, in general, not robust, and present methods of resolving each issue.

While it is commonly held that building robust code is impractical, we present methods of addressing common robustness failures in a simple, generic fashion. Even though a solution to creating generic hardening wrappers had previously proven elusive, new techniques have been found that provide a scalable strategy based on extending ideas used in the object-oriented testing strategy of the Ballista testing harness. We have developed uncomplicated checking mechanisms that can be used to detect and handle exceptional conditions before they can affect process or system state (preemptive detection). This gives a software system the information it needs to gracefully recover from an exceptional condition without the need for task restarts.

The perception that computing systems can be either robust or fast (but not both) is a myth perpetuated by not only a dearth of quantitative data, but also an abundance of conventional wisdom whose truth is rooted in an era before modern superscalar processors. The advanced microarchitectural features of such processors are the key to building and understanding systems that are both fast and robust. We present a quantitative analysis of the performance cost associated with making a software system highly robust. The generic robustness hardening techniques used have been found to make software interfaces robust for less than a 5% performance overhead (often less than 1% overhead) in reasonable usage contexts.

Previous studies have indicated that most programmers have an incomplete understanding of how to build software systems with robust exception handling, or even the importance of good design with respect to handling errors and exceptional conditions (e.g., Maxion's 1998 DSN paper). Those studies, while large in scope and thorough in analysis, contain data from students who in general have little professional programming experience. This work presents data collected from professional programming teams at IBM that measured their expected exception handling performance against their achieved performance. The data provides an indication that despite industry experience or specifications mandating robustness, some teams could not predict the robustness response of their software, and did not build robust systems. Furthermore, many of the problems missed were "easy" ones. This, combined with similar observations in previous experiments, suggests that something other than simply telling programmers to write robust code is needed to assure that software is indeed written to be robust.

Thus the Ballista project has finally reached the point where we feel we have answers to the basic technical questions we were investigating. We have learned how to test an API in a scalable manner; we can wrap modules to achieve 0% robustness failures per Ballista robustness testing results; we can create robust software with minimal performance impact; and we can use Ballista to assess not only the robustness of software, but also the effectiveness of programmers at attaining their desired level of robustness. Unfortunately, we have also discovered that simply knowing about these techniques is not enough -- in the absence of improved methodologies and a quality assurance/testing tool, writing robust code seems to be a difficult task for even some experience programmers.

We can offer some speculation on important challenges in robustness and dependable system topics based on what we learned during the course of the Ballista project. Key challenges for research and education include:

- Creating effective processes for enabling teams to create robust software interfaces in practice. An emphasis needs to be placed on attaining repeatable dependability results for software even though programmers can have poor understanding of their own ability to attain robust operation. While many of the key technology tools and mechanisms are now in place to accomplish this, significant efforts are needed to design and validate approaches that will actually work in industrial project environments.
- Methods to assess and reduce software aging problems, including both creating software with reduced aging vulnerability and putting software rejuvenation on a solid metrics-based foundation. (We have reason to suspect that this has been a major source of robustness vulnerability in the Windows NT code base in particular.)
- Incorporating robustness concepts (and even just testing concepts) into undergraduate programming courses so that programmers develop a reasonable sense of what is required to write robust code. (If we don't teach our students what robustness and dependability even mean until they reach graduate school, we can hardly expect everyday software written by everyday programmers to be dependable.)

Pervasive Dependability: Moving Dependable Computing Towards Mainstream Systems

Christof FETZER, Rick SCHLICHTING
AT&T Labs–Research, 180 Park Ave, Florham Park, NJ 07932, USA

Traditionally, dependable computing has been targeted towards safety- and mission-critical systems. These systems are typically closed systems that are well maintained and tightly monitored. Dependability is achieved using various kinds of redundancy during all phases of operation. We argue that dependable computing will need to become pervasive as the scale and reach of computing systems and embedded devices inevitably expands. As it does, the tradeoffs will change and lead to new scenarios, new opportunities, and new challenges. For instance, dependability may be needed in some low margin systems just to make these economically viable. Our talk concentrates on one example in this area, dependable home networks.

Maintaining networks that connect computers and smaller devices within the home will be a very low margin business. Market research indicates that home owners are typically only willing to pay a few dollars per month to keep their network up and running. Too many service calls made by home owners in response to problems can compromise the commercial viabiliity of such a business. Hence, dependable home networks that are inexpensive to install and to maintain are needed. In particular, one has to minimize the manual labor needed to set up and maintain such systems. This raises several new challenges for the dependable computing research community.

- **Automatic Reconfiguration** A dependable home network must use inexpensive off-the-shelf hardware with as little redundancy as possible. Instead of using space redundancy to mask failures, the system has to be able to reconfigure itself automatically to recover from failures.
- **Automatic Diagnosis** To facilitate automatic reconfiguration, the system has to be able to diagnose the root cause of a failure. While automatic diagnosis and reconfiguration are not always possible, the system has to support easy remote manual diagnosis in case the failure cannot be resolved automatically.
- **Automatic Installation** Every home will have a different configuration. This not only increases the difficulty of having automatic diagnosis and reconfiguration, it also increases the difficulty and cost of setting up a system. Since there is only a very low monthly margin, the system must not incur a high setup cost.
- **Automatic Adaptation** Home networks will have different applications and usage patterns. The network has to be able to adapt to these different requirements automatically.

Our talk will conclude with a brief description of how we attempt to address some of these challenges.

Dependability Challenges in Model-Centered Software Development

David P. Gluch

Department of Computing and Mathematics
Embry-Riddle Aeronautical University
Daytona Beach, Florida

Visiting Scientist
Software Engineering Institute
Pittsburgh, Pennsylvania

This presentation discusses some of the challenges associated with a model-centered software development paradigm. It also outlines some preliminary engineering approaches that may address these challenges. The approaches are adapted from model based verification practices [Gluch 98] that can be used in a traditional software development approach.

Model-centered development involves the automated generation of source code directly from models and often includes the immediate and (almost) exclusive representation of requirements as models. Translating requirements into models is the first step in the process. It is a manual engineering step that involves substantial domain knowledge. Other than code this set of models (and associated textual material) are the only artifacts of the design. Code is generated automatically from them. The model-centered paradigm is shown in figure 1. Throughout the development effort models are increasing detailed and are analyzed until a complete, consistent, and correct set exits. These are used to create the product's sources coded using automated code generation.

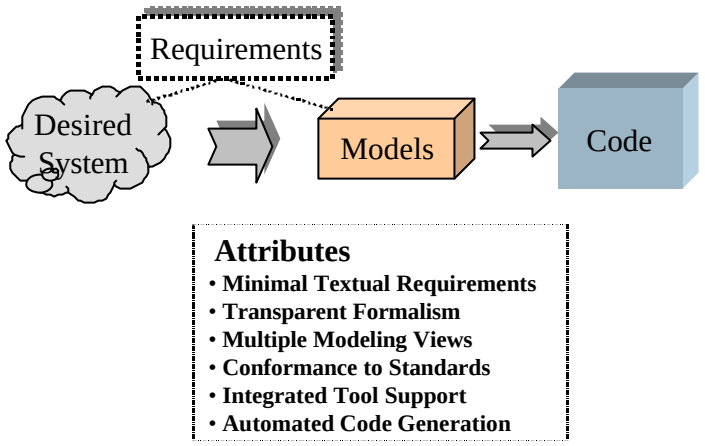

Figure 1. Model Centered Development

The dependability challenges in this paradigm are to verify and validate the models and to validate the product (final executable code) of the development effort. The artifacts to be verified and validated are part of a nexus of models and model components rather than a sequence of increasingly refined representations extending from requirements to design to executable code. At the heart of ensuring dependability is the analysis of the models and their constituent parts that comprise the nexus. This is especially valuable if confidence in the code generating system has been confirmed (i.e. the source code generated correctly and completely represents the model).

Robust analyses, especially for mission and safety critical systems, will require formal models and knowledgeable software engineers to effectively and efficiently conduct the analysis. The challenges include a need for additional research to identify enhanced analysis techniques, for development efforts to define engineering practices that are centered on model analysis techniques, the production of commercial systems that support those practices, and the education of software engineers in practices and techniques.

An approach for a model-centered paradigm that is based upon traditional approaches is the formulation of a distinct suite of verification and validation (V&V) strategies, processes, and support artifacts. At the heart of the approach are model analysis techniques. This parallel (concurrent) effort is shown in figure 2. The solid arrows show dependencies between the artifacts and the evolution of the development effort. The dotted line indicates that the test cases are part of the set of verification and validation artifacts.

In Figure 2 the verification and validation artifacts and associated activities are portrayed as concurrent. Concurrency does not imply that V&V is a separate activity. The autonomy of the V&V activities may vary. They may be highly integrated within the project through to being conducted by an independent organization (IV&V).

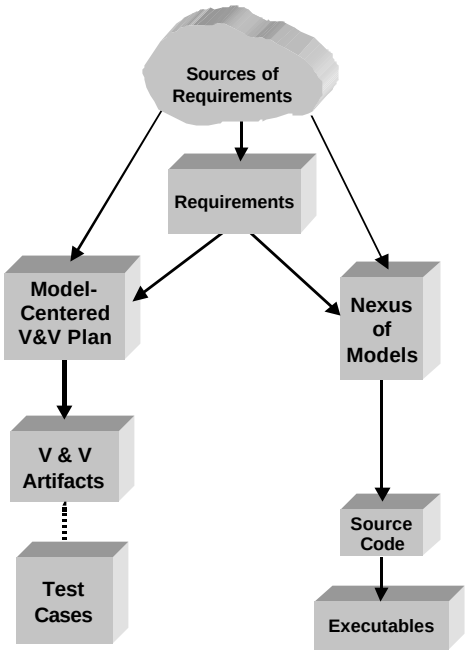

Figure 2. Concurrent V&V in a Model Centered Paradigm

The nexus of Models (NoM), shown in Figure 2, is the complete interconnected collection of the constituent parts of all of the models required for the design and implementation of a system. The nexus consists of highly interdependent entities.

In the development effort software tools capture the model elements and textual description units that comprise the nexus. These tools support some well defined (formal or quasi-formal) modeling paradigm (e.g. UML). Given current technology, all of the models cannot generally be captured in a single integrated tool. There may be some in Rhapsody, some in TimeSys or other too that is more than a simple configuration management system (i.e., there is some reasonably well-defined syntax, semantics, and rules for consistent manipulation of elements of the model).

Report Correlation for Improved Intrusion Detection and Fault Diagnostics

Walter L. Heimerdinger
Honeywell Technology Center, USA

The steady march of technology continues to increase the complexity and interdependence of critical systems. This increases the exposure of such systems to both accidental and intentional failures. Intentional or malicious failures are of special concern because they usually exploit a design fault in a particular feature and the spread of features expands the opportunities for exploits.

Efforts to detect abnormal situations in complex systems such as oil refineries have moved from attempts to develop a monolithic estimator to the development a correlator that processes reports from a variety of localized diagnostic modules.

This approach can be used for intrusion detection in complex networks. Detecting malicious intrusions in a complex system is an increasingly difficult task. Intrusion Detection Systems (IDSs) are being developed and deployed which react to artifacts of known exploits, but they are ineffective in providing intrusion awareness for a number of reasons. First, is the problem of scope. An IDS that is narrowly focussed tends to react after an exploit has been successful and often misses exploits that vary from the expected. A broadly-tuned IDS tends to generate a flood of reports, many of them false alarms, which tend to overwhelm a security analyst. Second, is the problem of placement. An IDS must be located where artifacts of exploits can easily be observed. This means that multiple IDSs must be deployed throughout a system. Finally, context is a problem for most IDSs. An individual IDS is unaware of the locations and functions of the various parts of a complex network and is unable to track the constant change that takes place in such a network.

Recent research has applied the approach developed for abnormal situation detection to intrusion detection. Each individual IDS acts as a sensor that provides information about a specific portion or aspect of a network. Reports from these sensors are correlated in the context of a comprehensive *Intrusion Reference Model* of the network, its resources, services, vulnerabilities, and objectives. This provides the context for an *Evidence Aggregator* to associate hypotheses with IDS reports and to estimate the plausibility and severity of each hypothesis. A number of challenges have to be overcome to make this work, including the normalization of reports from dissimilar IDSs, the maintenance of a current reference model, and the merging of reports from sensors that have entirely different scale factors.

Future Automotive Scenarios and R&D Challenges

Position paper
for the IFIP WG 10.4 Workshop on
"Challenges and Directions in Dependable Computing"
St. John, USA, Jan. 2002

Günter Heiner, DaimlerChrysler AG

Considering economical facts and technological trends it is evident that the automotive industry is a major driver for the economy of the industrial countries and that the automotive industry itself is increasingly driven by information and communication technology:

- Taking Europe as an example, in the last five years European automotive industry has grown 40%, which is double the growth rate of the total manufacturing sector.
- Today software and electronics account for about 25% of the production costs of a car, in the year 2010 we will expect more than 35%. Today more than 80% of the functions in a car are already driven by embedded software, and the rate continues to increase.

On the other hand, every year more than 40000 people die in traffic accidents on European roads. We estimate that over the next 15-20 years every second accident can be prevented, assuming the vehicles are equipped with driver assistance systems.
Therefore, the DaimlerChrysler research program "Accident-free Driving" was launched, which focuses on preventive and active safety strategies. Both approaches benefit from the progresses of electronics and software which facilitate better sensing and information monitoring (preventive safety) and improve vehicle dynamics and eventually automated driving (active safety). Information technology will be used to support the driver and even to take over many of his tasks:

- Perceive:
  sense the vehicle, sense the environment, sense the driver, communicate;
- Think:
  classify objects, detect obstacles, interpret scenes, decide;
- Act:
  inform and warn the driver, activate safety functions, intervene automatically.

Our vision of "Accident-free Driving" requires further research in enabling technologies, in particular in safety-critical real-time systems, dependable open communication systems, embedded control, rigorous design and validation methods, and certification strategies. Specific dependability research issues are for example middleware concepts for fault-tolerant systems, composing dependable systems, and validation of advanced driver assistance systems.

Dependability challenges in pushed-based systems

Yennun Huang
VP of Engineering
PreCache Inc.

Most network applications today use a pull-based client/server model. In this model, a server provides information or some other kind of service, and clients explicitly request specific information or services and synchronous "pull" it from the server. Clients often request and consume large amounts of information over long periods of time, and they periodically check with the server to see if new information has been produced. This pull style of information and service delivery is intuitive and straightforward, but has a number of drawbacks:

- **Complicates Interoperability**
  When clients communicate directly with servers, interoperability has to be achieved with specialized logic resident on the clients and/or servers. This tight coupling of producers and consumers is counterproductive to interoperability, which is best achieved when an abstraction layer is used to connect producers and consumers**.**

- **Poor Fit to Distributed Computing Communications**
  Distributed computing requires loose coupling of producers to consumers, yet the pull model is most effective where there is a very tight coupling of produces and consumers

- **Limited Scalability**
  As the number of clients increase, so do the number of network connections and associated resources that the server must maintain. Eventually the server becomes overloaded. Pull-based models are often used to implement push-based solutions in an attempt to improve scalability, but these attempts rely on a centralize message broker, which defeats a significant percentage of the potential scaling benefits of push.

- **Poor Fit of Applications to Multicast**
  Many clients often wish to receive the same information of a type or format that does not lend itself to traditional multicast, such as mass e-mail distributions. As a result, servers must discretely send duplicated information to numerous clients.

- **Request-Based Architecture**
  Clients must repeatedly check the server to determine if new information is available. This process is wasteful for both the clients and the server, since many of the client checks will be unsuccessful, meaning that clients and server will engage in many unnecessary and costly interactions.

- **Network Model Not Matched to Modern Applications**
  In many situations, the tight cooperation between client and server, which is required by the pull model, is not well suited to application requirements. This tight coupling is simply an artifact of a network based on an application model that is decades old.

An alternative to the pull style of communications is the push model, which was commercially pioneered by TIBCO (Nasdaq: TIBX) in the late 1980's. The push model, as implemented by publish/subscribe middleware, employs the concept of producers and consumers as a means to simplify the communications underlying distributed computing.

In this model, a server pushes or publishes messages (events), and the publish/subscribe middleware delivers them to interested subscribers. Publishers are often decoupled from subscribers, which is key to the scalability of the technology. This very loose coupling of publishers and subscribers allows application developers to implement highly flexible business systems. Producer and consumer applications don't have to know about each other's existence, location, or state. The benefits are numerous. Dynamic system reconfiguration is possible. One can add new clients or services without interruption, and participating application components are completely shielded from any implementation considerations involving other parts of the system. This allows for an efficient integration of business systems across different Lines Of Business (LOB) or even across enterprises. In fact, publish/subscribe is often considered the most effective solution to EAI (intra-enterprise) and supply chain management (inter-enterprise). Producers are freed of the need to manage consumer-specific connections and consumers become freed of the need to periodically check with the producer to see if new information is available. Whether there are one or a hundred or a million consumers, the load on the producer remains constant, and the efficiency of the consumers is increased.

However, the push-based model has many dependability challenges. Unless these dependability challenges are resolved, the applicability of push-based systems is severely limited. In this talk, I will describe the dependability challenges of push-based systems and possible solutions for these challenges.

Dependability Challenges in the Field of Embedded Systems

Hermann Kopetz
Vienna University of Technology, Austria

The dependability characteristics that must be met by a particular embedded computer system are determined by the requirements of the embedding system, i.e., the given application.

In the last few years, the technological advances and the associated cost/perfomrance improvements of semiconductor devices and communication technologies have opened a new application areas for embedded computer systes, such as:

- the replacement of mechanical and hydraulic control system by distributed embedded computer systems, (e.g., "fly by wire", or "drive by wire");
- the integration of complex computer systems in all kinds of mobile devices

The challenges for the embedded system designer is to meet the dependability requirements (safety, security, reliability, maintainability) within the given constraints (timeliness, low-power consumption, mass production) and the cost level of competing technologies.

In particular, the following challenges have been identified

- The provision of distributed architectures that tolerate arbitrary node failures at competitive costs.
- The validation and certification of ultrahigh dependability.
- The provision of the proper level of security in mass market systems that are maintained by "non-trustable" institutions.
- Security in normadic systems connected by wireless protocols.
- Meeting the above dependability requirements under the constraints of mass production.

DEPENDABILITY:
Information Assurance Research Agenda

Jaynarayan H. Lala
Defense Advanced Research Projects Agency
Arlington, Virginia, USA

Motivation: Information systems in the US, and much of the world, are under constant threats of cyber attacks. Every couple of months, there is a media attention-grabbing cyber event, the latest examples being Code Red and Nimda. Code Red I and II reported corrupted nearly a million servers over several weeks and caused over $2Billion in damage. What is imaginable is far, far worse than this everyday reality. Code Red type worm can be easily "weaponized" to infect millions of servers and other computers, in much shorter time periods, and carry a much more malicious payload such as the one carried by Nimda virus. US Secretary of Defense said recently "The surprises we will encounter a decade from now will very likely be different from the one that struck us on Sept. 11. To deal with those future surprises, we must move rapidly now to improve our ability to protect U.S. information systems….." [1]

State-of-Practice: Systems do not continue operating through most attacks unless isolated, custom-built using trusted computing components, and protected with access control. Data can be corrupted, information ex-filtrated, user services interrupted and user capabilities impaired during an attack . Currently available technologies cannot keep intruders at bay. Most attacks are not even detected until after damage is done. Systems are disconnected from networks and/or shut down to cope with attacks. Owners, operators and users do not know how well systems will cope with a cyber attack. Following an attack, tedious and manual data and code reconstruction must take place and vulnerabilities identified and patched. Systems are unavailable to mission commander during the manual restoration process. Following manual restoration, systems are still vulnerable to unidentified weaknesses, known-but-unpatched vulnerabilities, and misconfigurations. System administrators do not have the time and/or expertise to keep up with all the patches.

State-of-Art: Fundamental concepts to construct intrusion-tolerant architectures are being explored under the OASIS (Organically Assured and Survivable Information Systems) program. It should be feasible to design systems that can maintain data integrity, confidentiality, and un-interrupted user services for a limited period during an attack using the following principles:

- Avoid single points of failure
- Design for graceful degradation
- Exploit diversity to increase the attacker's work factor
- Disperse and obscure sensitive data
- Make systems self-monitoring
- Make systems dynamic and unpredictable
- Deceive attackers

Future Research Directions: However, much remains to be done to protect our information systems from current cyber threats and future novel attacks. Following are some of the goals that researchers should strive to achieve.

Intrusion Tolerant Architectures (Networked Applications and Embedded Systems)

- Integration of defense-in-depth layers to achieve intrusion-tolerance (avoidance, prevention, detection/diagnosis, isolation, recovery, reconfiguration, response)
- Create self-healing systems that can operate through cyber attacks and provide continued, correct, and timely services to users.
- Adapt security posture to changing threat conditions and adjust performance and functionality.
- Always know how much reserve capability and attack margin are available.

Self-Healing Systems

- Restore system capabilities to full functionality following an event
- Autonomously reassess success and failure of all actions before, during and after an event
- Autonomously incorporate lessons learned into all system aspects including architecture, operational procedures, and user interfaces

Theory of Information Assurance

- Development of taxonomy of vulnerabilities and attacks
- Methods to characterize cyber threats
- Assessment methodologies to characterize cyber-survivability, (assurance attributes, survival time, functionality, etc.) vs. attack space/vulnerability coverage vs. cost
- Techniques to optimize information assurance attributes (integrity, availability and confidentiality) at minimum cost

[1] Donald H. Rumsfeld, “Beyond This War on Terrorism”, Washington Post Op-Ed Column,Thursday, November 1, 2001.

Research Directions in Trustworthy {Trusted, Dependable} Computing

Carl E. Landwehr
National Science Foundation
Arlington, Virginia, USA

Vision

A society in which:

- People can justifiably rely on computer-based systems to perform critical functions. These systems include not only critical national-scale infrastructures, such as the power grid, gas lines, water systems, air traffic control systems, etc., but also more localized systems in aircraft, automobiles, and even in home appliances that perform safety critical functions. In such an environment, people can live without undue worry (i.e., securely).

- People can justifiably rely on systems processing sensitive information about them to conform to public policy. Increasing volumes of information, much of it sensitive in some respect to some party flows on our financial networks, health networks, and even our library systems, not to mention our conventional communication systems and our networked systems of personal and corporate computers. Confidence that these systems conform to public policy (and that the public policy is understood) will permit people to make informed and rational citizens about their own information.

- In both cases, the justification needs to take account of malicious attacks as well as accidental faults.

Present State

Today's computing and communications infrastructure does many things well, but suffers from a number of flaws and weaknesses that make it less than dependable, particularly in the face of attacks. These shortcomings include (1) latent flaws in widely distributed software, (2) decreasing diversity of software components, (3) poor technical means for managing security infrastructure, (4) inadequate technical controls for needed collaboration policies, (5) lack of convenient, scalable, strong authentication, and (6) inadequate security mechanisms for new technologies. Further, the infrastructure lacks effective means for detecting when these flaws and weaknesses are exploited, and for responding when such exploitations are detected.

Today's methods and tools for design, development, analysis, and evaluation of systems that can satisfy stated security requirements, are inadequate, including methods for designing secure systems and for designing effective human interfaces to security mechanisms in our systems. While past research has yielded some methods for designing and implementing security mechanisms and secure systems, they are frequently unused because they cost extra money and take extra time.

Goals

The National Science Foundation seeks to create and sustain the science and technology needed to discover, develop, and deploy methods and tools for the construction and analysis of trustworthy systems that can stand up to the pressures of the marketplace. Instead of railing at the developers and consumers, we need to provide them with methods and tools that lead to trustworthy systems yet are demonstrably cost effective. Getting these methods into the marketplace also means having a technically trained workforce that can produce them and that knows how to exploit them.

Research Directions

The Trusted Computing Program aims to support research in a broad range of related areas including component technologies – both methods for producing trustworthy components and identification of useful elements from which trustworthy systems can be constructed. Composition and decomposition methods that can help us understand the properties of systems we have and help us synthesize systems with properties we want need to be developed.

Systems today are developed and enhanced incrementally, yet incremental changes often upset overall system behavior. Techniques that can help maintain system trustworthiness as the system changes are needed.

Many of today's systems, from video recorders to personal computers, baffle or mislead their intended users, and user bafflement can lead to misuse of the system with potential dangerous results. We need better methods for improving human understanding of critical system behavior and control. How can system trustworthiness be visualized, particularly for operators of critical systems, including geographically distributed systems?

To build more trustworthy systems, we must be able to model, analyze and predict the critical properties of systems and components. For example, to provide trustworthy systems that can succeed in the marketplace, we need system engineering and evaluation tools that support explicit evaluation of tradeoffs among security design alternatives and permit prediction of security behavior of large-scale systems.

Conclusion

NSF's Trusted Computing program is just getting started. The research areas outline above are intentionally broad. We've had an excellent response to the initial program announcement; as these proposals are evaluated over the coming months, the program will come into sharper focus, but the vision will remain on developing a society in which trustworthy computing is the norm rather than the exception.

(Some remarks about) Dependability of large networked systems

Jean-Claude Laprie — LAAS-CNRS

1) Findings

Comparing wired telephone or computer systems of the 90's, and cellular phones or web-based services, availability has dropped from 4 or 5 nines to 2 or 1 nine. Are causes due to a) ignorance of good practices, to velocity in deployment or to b) complexity (e.g., of server farms) and hacking?

2) Causes of failures

At first sight, they are ranked as follows:
1) Operations, hacking
2) Physical and natural environment
3) Hardware and software
However, root cause analysis of operations and hacking often points at the design of hardware and software.

3) Some remedies

a) *Co-design of human acitivity and system behavior.* All too often, and in spite of accumulated evidence by cognitive ergonomists, the human-system interaction comes last. Co-design ranges from tasks allocation to reciprocal monitoring and recovery.
b) *Diversity.* The decreasing natural robustness of both hardware and software make systems based on uniform solutions fragile. Hence a need for revisiting diversity at all levels of a system.

4) Some gaps

a) *Composability.* Composability of dependability properties is still perfomed on a largely ad-hoc basis, and very few mathematically formal results have been established, thus limiting transferability and durability.
b) *Metrology.* This a prerequisite for substantiating current and future research (current gaps? expected future threats? identified from the exploitation of measurement trends), that needs a widely accepted basis for data collection and measurements of accidental and malicious events.

IFIP WG 10.4 Workshop on Challenges and Directions for Dependable Computing

Jan. 2002

Contribution proposed by G. Le Lann [Gerard.Le_Lann@inria.fr]

Among the many open issues and challenges raised with dependable computing, I would like to offer to consider two classes of problems and possible directions.

One class is in connection with the "requirements capture" processes followed in projects concerned with computer-based systems. The issue examined is how to avoid faulty processes, which are known ò lead to project setbacks and/or system failures. This class encompasses issues of an interdisciplinary nature.

Another class is related to the following issue, raised in computer science & engineering: Which computational models for designing applications based upon mobile computing, for designing autonomous complex systems, bound to meet dependability and real-time requirements? The examination of respective model coverage, as well as the concept of "model binding", permit to identify potentially promising directions.

1. Can we avoid faults in requirements capture processes?

A number of studies and reports about computer-based systems/projects conclude that, too often, project setbacks and/or operational system failures are due to faulty requirements capture processes. Faults made during the requirements capture phase, which is the very first phase in a project, are very difficult to catch in subsequent phases. Such faults inevitably lead to difficulties, which translate into (1) project setbacks, e.g. contractual deadlines set for deliverables being missed, and/or testing phases being failed, and/or excessive costs – due to the need to redo some of the work tested "incorrect", and/or project cancellations, (2) failures of operational systems.

Two recent examples in the space domain are the maiden flight of European satellite launcher Ariane 5 (1996), and the US Mars Climate Orbiter mission (1998). Causes of failures will be briefly presented.

When comparing the costs involved with failures due to such faults, and the reasons why such faults go through current processes, one is led to conclude that it is of utmost importance to improve current practice, so as to increase "projects dependability", as well as systems dependability.

Such faults are human made. It is well known that communication in natural language almost inevitably leads to ambiguity and incompleteness. Unfortunately, any real project starts with human communication.

Over the years, many attempts have been made at improving the quality of the requirements capture processes. Gathering experts from various areas – e.g., psychology, ergonomics, in addition to computer scientists and engineers – has been tried (see, e.g., ACM SIG CHI). For

a number of years now, we have been witnessing a strong move in favor of more formal approaches to requirements specifications. Professional organizations, such as INCOSE, or professional communities, such as the IEEE ECBS community, have working groups devoted to these issues. Nevertheless, failures and budget overruns still occur too frequently.

We will report on recent experience gathered with real projects where a non formal method based upon proof obligations has been used, with special focus on three space projects, one of them funded by the European Space Agency, the other two funded by the French Space Agency. Reasons why proof obligations help in eliminating faults while conducting a requirements capture phase will be detailed.

2. Which computational models for dependable and real-time mobile computing, autonomous complex systems?

Soon, real-time applications based upon mobile processors will be emerging. Emerging also are autonomous complex real-time systems, such as, e.g., those systems which are designed to redefine their missions on-line (autonomous re-planning during remote space missions, earth-based systems operating in hostile environments).

These relatively new paradigms raise challenging issues, such as the refinement of conventional failure models in the case of wireless group communication/cooperation, or how to perform on-line V&V in the case of autonomous space systems (see, e.g., the NASA RIACS Workshop, Dec. 2000).

Features that are common to these new applications/systems are highly varying communication latencies, and/or a significant degree of uncertainty associated with any prediction regarding such latencies. In other words, a design conducted and proved correct considering some time-based or synchronous computational model has a poor coverage.

Many techniques, constructs, algorithms, solutions, to problems in fault-tolerant computing rest on timing assumptions, i.e. on some form of (postulated) synchrony. Conversely, there are impossibility results established for the pure asynchronous computational model (no timing assumptions at all). Which is unfortunate, given that the coverage issue does not arise at all with this model.

Hence the question: How "close" can we get to the pure asynchronous model, and still be able to design highly dependable and real-time mobile/autonomous applications/systems?

Another question arises: If high dependability can be achieved with asynchronous designs, is there any severe penalty incurred in terms of efficiency? In other words, can such designs be as efficient as synchronous designs?

We will report on recent work, which shows that asynchronous designs should be favored. The issue of how to prove timeliness when designing in an asynchronous computational model will be addressed. (Paper accepted for publication in 2002 in an IEEE Transaction).

Of course, if necessary due to time constraints, only one of these two proposed topics may be presented.

Challenges and Directions for Dependable Computing:

Some Reflections

Brian Randell

30 Oct 2001

1. Introduction

An analysis of the challenges and directions for dependable computing, to be of any utility, must take into account the likely future, and must address not just technical questions concerning what will be the most interesting and timely topics, but also socio-technical questions, concerning the impact (or lack thereof) that successful work on such topics might have.

2. Three Laws

As Dan Quayle once said, predictions, especially of the future, are very difficult. However, I believe that in this task we are aided by not just one, but three "natural" laws, the names of which form the pleasant alliteration: "Moore, Metcalfe and Murphy". The "3M" Laws will, am sure, all remain in force for at least a number of years to come – and they provide a very convenient hook on which to hang the rest of this little account.

Moore's original law [4] stated that:

> "The number of transistors per chip will double every eighteen months".

This was in 1964, and at the time Gordon Moore (co-founder a few years later of Intel) suggested that the law would remain valid until 1975. Though some years later the trend slowed a little, to a "mere" doubling every two years, his law still seems to hold true – and is apparently self-fulfilling. Indeed, as each year passes, it would appear that the scientists and industrialists most directly concerned vote for a

further year's extension of the law, by confirming their belief that it will continue to hold for at least the next ten years or so. Moreover, the general idea of the law seems to apply to the other crucial hardware technologies involved in computing, namely digital communications, and magnetic storage devices.

It is this amazing and continuing rate of technological development that has fuelled such an enormous growth in the amount and variety of computing, through the impact it has had on computer performance, and on power and space consumption, and hence on both costs and capabilities.

Metcalfe's Law is perhaps not quite as well known as Moore's law. It was coined by Robert Metcalfe, one of the inventors of the Ethernet. His law can be stated as:

> "The usefulness of a network varies as the square of the number of users."

An alternative version has been given by Tidwell [8]:

> "The power of the network increases exponentially by the number of computers connected to it. Therefore, every computer added to the network both uses it as a resource while adding resources in a spiral of increasing value and choice."

As others have remarked, it is the combination of Moore's and Metcalfe's laws that lies behind the frantic growth of the Internet. Indeed, I would suggest that it is not too much of an over-simplification to say that Moore's Law made this growth possible, Metcalfe's Law made it happen.

The third law is, I believe, of much earlier date than Moore's and Metcalfe's laws – to the point that it is not all clear who first coined it, or whether his or her name actually was Murphy. It has a number of forms and variants. However the most common is:

> "If anything can go wrong, it will."

One of the most frequent alternative versions, or addenda, is;

> "If there is a possibility of several things going wrong, the one that will cause the most damage will be the one to go wrong."

The above wordings are just the first, and best, of many given in the Murphy's Law Web-site[1]. One could perhaps describe Murphy's Law as

[1] http://www.fileoday.com/murphy/murphy-laws.html

being, like Moore's Law, merely an experimental law – however it seems regrettably safe to assume that there is no chance of it ever being repealed.

The whole subject of dependability is both bolstered and bedeviled by Murphy's law. It motivates the need for system fault tolerance, but unfortunately also applies in principle, and often in practice, as much to any fault tolerance mechanisms as it does to the systems they are intended to protect. But it is certainly the case that the dependability community is in effect much more conscious of, and constructively concerned about, Murphy's Law than many of the researchers in other branches of information technology.

The merit of bracketing Murphy with Moore and Metcalfe is that it acts as an antidote to the hubris that all to often surrounds discussion of the technological futures that are more or less directly a consequence of the first two laws. For example, at a recent IST conference, a panel of very senior industrialists enthusiastically discussed the "The Ambient Network", a newly-coined term that that they used to describe the situation they were predicting would arise of there being in effect a single system of billions of "networked information devices" – of everything connected to everything. In the whole debate the only major challenge (i.e. problem) standing in the way of creating The Ambient Network that was mentioned, by any of the very eminent speakers, was "usability" - there was not a single mention of "mis-usability" or of "dependability", leave alone of the possibility of large scale failure, e.g. of a Europe-wide set of blue screens, caused either by accidental or malicious faults!

In fact, although it is now commonplace to posit a world in which "everything is connected to everything" many of these connections will in fact be transient, due for example to the use of mobile devices. Indeed many of the connections will be unexpected and unwanted – and hence a major source of undependability. Thus it is certainly not the case that the whole population of inter-connected computers and computer-like devices will be components of one defined system. Rather, there will be a huge number of separately designed systems (ideally all carefully pre-specified and designed), but also systems-of-systems - both designed, and accidental (though even such latter may have people becoming dependent on them). And many of the dependability problems that arise will be caused by uncontrolled interactions between these systems, not necessarily via direct electronic links. Indeed, in what follows, I will use the term system not in the narrow sense of computer system, consisting of just hardware and software, but in the sense of computer-based system. (By this term I mean a system that comprises both computers and the humans who directly use, control, and are affected by these computers.)

However, unforeseen indirect effects, and social effects of the above trends such as public attitudes, and the activities of governments and (certain) companies are much less easy to forecast.

3. The Wider World

We need to consider the wider environment in which computers and computer networks will exist, and whether/how it might differ from today's world – a world in which public attitudes to dependability in general are confused and confusing. For example, we are in a world in which it has been acceptable to allow an automotive industry to develop whose products are implicated in the deaths of 40,000 people annually. Perhaps because such deaths typically occur in ones and twos, rather than in hundreds, and because drivers assume they are individually in direct control of the situation and hence their destiny, individual road accidents cause much less concern and attract much less publicity and official attention than plane crashes, even ones in which few if any people are killed. Which of these widely differing views is the more likely to be representative of attitudes to the crashes of computers, in years to come? (Similarly wide variations exist in public attitudes to system dependability, even when lives are not at risk, e.g. concerning the value to be attached to privacy. And it is evident that the dependability levels that are currently tolerated in desktop computers would not be acceptable in television sets – a situation which may or may not change enough to affect the desktop computer market.)

Government pressures often have detrimental effects regarding system dependability, indeed in many kinds of systems. For example, so-called "efficiency savings" can lead to systems that are far more fragile than their predecessors – simple queuing theory explains what is likely to occur when hospitals, for example, are forced to run too close to maximum capacity even in situations of normal load. And automation, though it can reduce the frequency of minor failures, often does so at the cost of occasional much more costly failures – railway control and signaling, at least in the UK, are a case in point.

One further comment regarding government pressures. The impact on dependability, and in particular security, research of recent government actions particularly in the United States is much more difficult to predict. The previous balance that was held between individuals' rights to privacy, and the state's ability to monitor and control data communications has shifted abruptly. What effects this will have on government and personal behaviour, and on various aspects of the design and operation of networked computer systems, remains to be seen. It has been argued, by the Robust Open Source software movement, that development of a dependable international computer and networking infrastructure is being impeded by government and

commercial policies – and that the solution lies in the hands of dedicated volunteers working on open source implementations of the necessary system components. It this still the case, assuming it ever was?

All this is at a time when it is commonplace to remark that individuals, organizations, governments, indeed society as a whole, are becoming ever more dependent on computers and computer-based systems. In many cases such dependence is proving satisfactory, or at least acceptable. However, given our closeness to the technologies involved, we cannot ignore the potential dangers of allowing such dependence to exceed the levels of dependability that can be achieved in practice, or the difficulties of building complex systems of predictable dependability.

Allied to this is the fact that we are having to live in and try to cope with a world in which, to quote Dirk Gently, Douglas Adam's Holistic Detective: "everything is deeply inter-twingled" – but as computers and computer networks play an ever greater and an ever less visible role, the extent and speed of system interconnection is greatly increased, and much more difficult to deal with. Luckily, there is now a growing recognition by governments, if not yet the general public, of the dangers facing our various global infrastructures, such as finance, power, telecommunications, etc., as these become more closely interconnected through data networking. But the pressures for ever-greater interconnection continue, and dangers increase.

For example, plans for electronic highways, in which computers rather than drivers are responsible for maintaining separation, might have a very beneficial effect on the rate of relatively minor accidents. But how catastrophic will be their possible major failure modes? And what effect might such failures have on public perceptions of the risks of road transport, and indeed of public attitudes to the information technology industry?

Commercial pressures, especially in unregulated industries, rarely assign dependability a very high priority. The personal computer industry pays far more attention to issues of functionality than security, for example. Hence the proliferation of numerous unnecessarily insecure software technologies in recent years, and the consequential greatly-enhanced impact of viruses and worms.

The commercial software world, left to its own devices, will continue to obey the economic law which in effect states that in industries with very high development costs and virtually zero production costs, it is very difficult for new entrants to dislodge, or even co-exist alongside, a prominent market leader. In such situations competition becomes a much less effective force for promoting improvement to system

functionality, performance and dependability than might otherwise be the case.

Finally, another characteristic of the current commercial software, indeed the whole personal computer, world that is of particular relevance to dependability is the increasing dominance of (ad hoc) standards. Uniformity, for example of programming and user interfaces, presumably has a beneficial effect on the rate of at least certain types of accidental fault, e.g. by application programmers and by users. However, this lack of diversity contributes greatly to the impact of malicious faults such as those created by virus writers. Mono-culturalism seems to be as dangerous in the software and hardware world as it is elsewhere.

4. And So?

Against such a background, it is surely insufficient to concentrate on interesting technical dependability challenges, and simply on defining and prioritizing possible future computer science dependability research projects. (In any case, this usually results merely in a list that bears a strong resemblance to the author's current projects and proposals.) Rather, it is I suggest also worth trying to discuss how any such programme of future research should:

(i) be chosen and conducted so as to maximize the chances of effective subsequent industrial and public take-up

(ii) be situated within its overall research context, e.g. within the IST programme as a whole

(iii) consider what else the research community can do to use its expertise to achieve a wider social benefit.

In subsequent sections, I address each of these points briefly, though first I will attempt the obligatory discussion of a technical agenda.

4.1. A Technical Agenda

A useful recent summary of the present state of the art, and of various authoritative views of future requirements for dependable computing systems, especially large networked server systems, is given by [5]. A welcome common theme is the need to take a wider view of such computing systems, allowing more fully for the problems caused by and to their administrators and maintenance personnel. Networked servers typically aim at achieving essentially continuous operation round the clock, despite such failure-prone activities as upgrades and maintenance, and the frequent installation of new hardware and software, as the service expands. There is thus a need to relieve the burdens on

administrators, especially in dealing with failure situations, by achieving much more automated and faster means of recovery after system failure, or "human-aware recovery tools" as Patterson and Brown term them.

In fact, the approach the authors advocate, "rewind, repair and redo", involves traditional transaction-processing reliance on backward error recovery. The possibility that the root cause of a problem might be external and undetectable by input validation, or that the world outside the system might not be capable of simply backing up after being supplied with erroneous results, is ignored. (These are the more general problems first addressed by Davies [1] in his pioneering work on "spheres of control", work that has strongly influenced our ideas on CA actions [9], and which I would argue is a more appropriate basis for planning recovery in general computer-based systems.)

Furthermore, recent world events surely motivate increased concern for the problems of making all types of computer system more resilient in the face not just of accidental faults, but also of deliberate attacks, though in the computer world such attacks are as likely to be made by insiders as by external agents. The provision of means, especially automated means, of tolerating attacks is far from straightforward, even just for communications systems, even though for years these have been designed with this as one of their objectives. For example, there are widely differing opinions as to the relative success with which the conventional telephone system and the Internet largely withstood, in their very different ways, the destruction of the World Trade Center.

In now remarking that a concern for the possibility of attacks should be applied not just to closed systems, but to systems-of-systems, and rapidly evolving systems-of-systems at that, I am in danger of being accused of simply and unimaginatively suggesting that two existing IST projects with which I am associated, MAFTIA (Malicious- and Accidental-Fault Tolerance for Internet Applications) and DSoS (Dependable Systems of Systems), should be succeeded by a project that combines their technical aims. In fact I'd like to claim that this is not lack of imagination, but rather an indication of the farsightedness of the original MAFTIA and DSoS proposals!

Another priority I'd like to advocate is that of investigations of the problems of making appropriate allocations of responsibility (concerning functionality, error detection, and fault tolerance) between the computer and human elements of a computer-based system so as to achieve effective overall dependability. This, I am afraid, can also be criticized as simply an obvious extension of another large existing project led by Newcastle, namely DIRC, the EPSRC-sponsored Dependability Inter-Disciplinary Research Collaboration. However,

ambitious though DIRC is, there is plenty of room for more projects that address various aspects of the socio-technical problems of achieving and guaranteeing whatever is needed in the way of balanced usability, dependability, cost, performance, etc., in various organizational contexts.

In line with earlier remarks, there is evident need for more research aimed at alleviating the dependability problems of systems that are made up of mobile, and often inaccessible, components. (A nice such problem occurred here recently, involving a laptop that was being stored temporarily in a secure cupboard while efforts were being made to safeguard the campus network and systems from attacks caused by the Code Red virus. When the laptop was reconnected, the safeguards which should have prevented it from becoming infected and then attacking the network itself failed - for complicated but understandable reasons - to get installed, and the laptop itself became an attacker, inside the campus firewalls.)

Research is also needed on the dependability of systems whose boundaries, and specifications, are ill-defined. Much of the natural and the human world manages to cope with failures in such systems – a study of how this is achieved might lead to a better understanding of how to achieve this in systems that incorporate computers. Properties akin to self-stabilization [2] are perhaps what should be sought and investigated. Similarly, many large systems are continually evolving, as new functionalities and facilities are added – it is a major problem to retain the flexibility and scalability that such systems need to have while at the same time ensuring that high levels of dependability are maintained.

All the above research areas need to be buttressed by research into improved means of evaluating, and ideally predicting, the various different facets of system dependability, including what is probably the most difficult one to evaluate, namely security. Such evaluation involves gaining much greater understanding of such slippery concepts as ”system complexity”, hopefully matching the sort of progress that has been made in recent years in understanding the concept of diversity.

The stress that I am placing on avoiding narrow concentration on the problems of “mere” computer systems, but instead including concern for the human element, means that the search for applicable techniques and research insights need not be limited to the computer science literature. (As a parenthetical comment, let me mention that, regrettably, computer “scientists” have for some time all too often regarded it as unnecessary to seek out and read about any research that is more than ten years old – now the tendency seems to be to limit one’s

attention to reports that are on the web, and findable in less than a minute or so using Google.)

Rather, the literature of economics, general systems theory and cybernetics, is also likely to be of use. (See, for example the survey by Heylighen [3].) One directly relevant example, suggested by my colleague John Dobson, is the concept of "requisite variety", which comes from general system theory, and which can be used to understand some of the problems of interconnecting systems in such a way as to ensure that that errors signalled by any of the systems can be responded to suitably by the other systems.

4.2. Recommendations Affecting Take-up

The reflections I have under this heading are obvious in the extreme. Any dependability research project which is aiming to produce techniques or mechanisms that it hopes others will take up needs to try to ensure that such take up will be as easy, and as quick to deliver evident benefit, as possible. In general it easier to attract attention to a demonstratable mechanism, whether it be a system component, or a software tool that aids some aspect of the task of designing dependable systems, than to a technique that has to be taught and learnt.

System components that can readily be integrated into existing systems have obvious advantages. This is one of the advantages of transparent mechanisms, such as the now almost forgotten Newcastle Connection, which could be added to each of a set of Unix systems to join them together into what appeared to be a single Unix system without needing to change any existing system or application code. The modern equivalent of the technique used is "reflection" used as a means of imperceptibly adding various sorts of dependability characteristics to object-oriented application programs.

Dijkstra, at the 1969 NATO Software Engineering Conference [7], said:

> "But there are a few boundary conditions which apparently have to be satisfied. I will list them for you:
>
> 1. We may not change our thinking habits
>
> 2. We may not change our programming tools
>
> 3. We may not change our hardware
>
> 4. We may not change our tasks.
>
> 5. We may not change the organizational set-up in which the work has to be done.

> Now under these immutable boundary conditions, we have to try to improve matters."

At the conference his remark was greeted by applause, in which I joined – my view now is that the art is to see how to achieve a worthwhile effect while obeying as many of these boundary conditions as possible.

As regards achieving technology transfer of a technique, this I believe is most readily achieved by transferring personnel skilled in that technique. Alternatively it typically involves extensive effort in the preparation and delivery of teaching materials. An important milepost is passed when someone other than the researcher can then undertake further such efforts successfully.

4.3. The Research Context

When the European Dependability Initiative was being formulated, as a contribution to the planning of the IST Programme in FW-5, it was strongly argued that it was important to ensure that adequate attention was paid to dependability issues in *all* relevant research projects, especially those in particular computer application areas. The aim was two-fold: to ensure that state-of-the-art dependability techniques were employed wherever needed, and that any novel and challenging dependability requirements were identified, and brought to the attention of dependability researchers.

The mechanism we suggested was simple. It was that there should be a budget, under the control of the dependability directorate, quite separate from its budget for dependability research projects, that could be bid for by other research directorate. Such bids would be for use to augment deserving research projects in their areas, so as to ensure that these projects incorporated suitable dependability-related activities in their plans, and interacted as needed with specialized dependability projects.

Regrettably, this suggestion was not accepted, and to my mind a very useful opportunity was lost.

4.4. The Wider Scene

There are obvious comments that can be made about the importance of people with expertise in system dependability (i) taking an active part in efforts aimed at enhancing public understanding of science, and (ii) attempting to bring suitable influence to bear on relevant government and commercial policy-forming activities, and to ensure that wherever possible the best current technical, and socio-technical, practices are employed.

However, I will otherwise largely confine my remarks under this heading to the importance of the idea of "closing the loop", in order to ensure that system designers, and relevant authorities, are well-motivated to treat dependability issues with the attention they deserve.

One of the neatest examples of loop-closing is Edward de Bono's suggestion that, in the interests of the environment, factories should take their water intake from a river downstream of any outflow from them to the river. Closer to home, I understand that in the R.A.F. it is, or at least some years ago was, the rule that when a plane is handed back to a pilot after maintenance, the pilot has the right to insist that the maintenance engineer who signed the certificate that the plane is now again airworthy comes on the first flight.

Years ago I myself experienced the benefits (to the compiler I and a colleague producing) of working in the same large office as the people who had to use this software. Of course it is better still when a software developer is an active user personally of the software he or she is developing, and so is forced to be directly aware of its weaknesses. This is probably one of the reasons behind the success of (some of the) open source software projects. However, there is an evident danger – it is equally important that developers should also be aware of what is being achieved elsewhere. Thus I suggest that the developers of operating system A, as well as being active users of system A, should also have reason to keep aware of developments in system B and C. (The letters A. B, and C can be replaced by "Windows", "MacOS", and Linux, for example, *à choix*..)

Unfortunately, such loop-closing is not always appropriate – the developers of a nuclear reactor's safety shut-down software are not going to be its users, though of course I suppose they could be forced to live near the reactor! But in all seriousness, it does seem to me that one of the most major issues is not that of research into new ways of building more dependable computer systems, valuable though such research can be. Rather it is that of trying to ensure that all in relevant positions of influence pay appropriate attention to dependability aspects of significant existing and proposed computer-based systems.

A final (obvious) point to be made is that dependability is, unfortunately, not usually taken seriously until there is a failure. At one end of the scale, few individuals take advice regarding back-ups seriously until they first lose some of their precious files. On a national or global scale, such a change of attitude seems to require an incident seriously adversely affecting either a very large number of people or a smaller number of sufficiently important people. But computer systems vary so greatly concerning the seriousness of their possible failures, from situations in which failures might be almost viewed as beneficial,

due to the improvements they prompt, to ones in which failure would be so catastrophic that it must be avoided at all costs. Petroski's analysis of the effects of civil engineering failures on the progress of civil engineering [6] is perhaps one of source guidance on these issues.

5. Concluding Remarks

By way of concluding remarks I simply re-iterate that in considering future dependability research initiatives, I think that it is vital (i) to take full account of (all three of Moore's. Metcalfe's and Murphy's Laws, and (ii) to take an adequately broad view of system dependability, and of system research, and the environments in which it is carried out and for which its results are intended.

6. Acknowledgements

These reflections owe much to interactions over the years with my various colleagues, and in particular to recent discussions with Cliff Jones, John Dobson and Tony Lawrie, but any inadequacies are mine alone.

7. References

1 C.T. Davies, "Data processing spheres of control," *IBM Systems Journal*, vol. 17, no. 2, pp.179-198, 1978.

2 E.W. Dijkstra, "Self-Stabilizing Systems in Spite of Distributed Control," *Comm. ACM 17*, vol. 17, no. 11, pp.643-644, 1974.

3 F. Heylighen and C. Joslyn. "Cybernetics and Second-Order Cybernetics," in *Encyclopedia of Physical Science and Technology (3rd ed.), ed. R. A. Meyers*, New York, Academic Press, 2001.

4 G.E. Moore, "Cramming More Components Onto Integrated Circuits," *Electronics*, vol. 38, no. 8,1965. (http://www.intel.com/research/silicon/moorespaper.pdf)

5 D. Patterson and A. Brown. "Recovery-Oriented Computing," in *Ninth International Workshop on High Performance Transaction Systems Workshop (HPTS-9)*, Asilomar Conference Center, Pacific Grove, California, 2001. (http://www.research.microsoft.com/~jamesrh/hpts2001/presentations/Recovery-Oriented%20Computing.ppt)

6 H. Petroski. *To Engineer Is Human: The Role of Failure in Successful Design*, New York, St. Martin's Press, 1985.

7 B. Randell and J.N. Buxton, (Ed.). *Software Engineering Techniques: Report on a Conference Sponored by the NATO Science Committee, Rome, italy, 27-31 Oct. 1969*, Brussels, NATO Science Committee, 1970, 164 p. (http://www.cs.ncl.ac.uk/people/brian.randell/home.formal/NATO/nato1969.PDF)

8 J. Tidwell. “The Third Place,” in *Second International Conference on Virtual Communities,* Bath, 1999. (http://www.infonortics.com/vc/1999/tidwell/tsld001.htm)

9 J. Xu, B. Randell, A. Romanovsky, R.J. Stroud, A.F. Zorzo, E. Canver and F.v. Henke. “Rigorous Development of a Safety-Critical System Based on Coordinated Atomic Actions,” in *Proc. 29th Int. Symp. Fault-Tolerant Computing (FTCS-29),* Madison, IEEE Computer Society Press, 1999. (http://www.cs.ncl.ac.uk/research/trs/papers/662.pdf)

DESIGNING AND ASSESSING ADAPTIVE DEPENDABLE DISTRIBUTED SYSTEMS: PUTTING THE MODEL IN THE LOOP

William H. Sanders
Coordinated Science Laboratory and
Department of Electrical and Computer Engineering
The University of Illinois at Urbana-Champaign

To a large extent, the system modeling and dependable distributed system communities have evolved separately. Much progress has been made in each community, but (in my humble opinion) with very different ways of conducting research. In the modeling community, the work has been quite formal, with much concentration on developing mathematical techniques for the solution of large systems. In the dependable distributed systems community, there as been both formal and informal work, but the majority of practical dependable system middleware has been developed in a very experimental way, which stresses the importance of prototyping and demonstration, but lacks formal assessment of the degree of dependability that has been achieved. The formal and theoretical nature of system modeling research has resulted in a large gap between what is known in theory, and what can be applied by a practitioner. The informal nature of the dependable distributed system community has led to practical use of many of the developed techniques, but questions remain about the goodness of proposed designs, and algorithms designed to respond to changing system requirements and conditions (via adaptation) are largely ad hoc. In this presentation, I will argue that it is time bring these communities together in two ways.

First, it is time to develop modeling methods and tools that can be applied to practical dependable distributed systems. Techniques currently exist that can be used by experts to meaningfully assess the dependability of large-scale systems, but they require much expertise and creativity to apply. Before they can be applied in a widespread way, work is required on model representation techniques, model composition techniques, and model abstraction techniques that can focus the effort of model solution on system details that are important with respect to the measure being considered. Practical model representation techniques may need to be domain specific, and help from the distributed system design community will be necessary to help define appropriate domain-specific formalisms. Composition techniques need to be developed that are both natural to the dependable distributed systems community, and preserve properties that aid in the solution of a composed model. Abstraction techniques are needed to preserve parts of a system design that important, with respect to a particular dependability measure, while removing unimportant details. In short, techniques are needed that can turn the current “art” of system modeling into a “science.” If a science of system modeling can be achieved, techniques that currently require an expert to apply could be interfaced to existing software and hardware design tools, so that they can become an integral part of the design process.

Second, the system modeling and dependable distributed system communities should come together to build systems that use online models to adapt as necessary in order to provide specified performance and dependability properties to applications. The idea of the use of a system model in control theory is longstanding, but has not typically been used in distributed software systems, perhaps due to their complex, non-continuous, nature, or the perceived cost of on-line model solution. Recent results show that the cost of on-line solution need not be prohibitive, and that models can be used in an on-line fashion profitably to guide adaptation at many time scales in dependable distributed systems. The goal in this case is not to build models that yield accurate measures in an absolute sense, but build those that can be solved quickly, and used to make decisions on how to adapt. Much work is needed to understand data needs to be collected to serve as input to on-line models, how models can be constructed that can be solved quickly, yet make appropriate decisions, and how models can be constructed in a hierarchical fashion, such that decisions in various parts of a system can be made in a consistent way, without costly global state knowledge. If successful, this work can result in systems that can act in a largely autonomous fashion, healing themselves so that they can continue to deliver requested services when unintentional and malicious faults occur.

The European Dependability Initiative in the RTD Framework Programme 6th: An Opportunity for Global Collaboration

Andrea Servida
Head of Sector
European Commission,
DG INFSO C-4,
Rue de la Loi 200,
B-1049 Brussels, Belgium
E-mail: andrea.servida@cec.eu.int

ABSTRACT

Last February, the European Commission proposed the multiannual Framework Programme 2002-2006 (in short FP6) for research, technological development and demonstration activities aimed at contributing towards the realisation of the European Research Area policy. The main principles, research priorities and innovative aspects of FP6 would be presented to highlight how the European Initiative on Dependability has a unique opportunity to further growth and develop its scope and ambitions.

The characteristics of the new implementing instrument called "Integrated Project" provide an ideal and coherent framework to promote dependability in Information Society by stimulating innovative, multidisciplinary and global RTD to tackle the scale issues of dependability connected with new business and everyday life application scenarios. Important aspects of these scale issues would be those associated with the increasing volatility and growing heterogeneity of products, applications, services, systems and processes in the digital environment.

The state of play and the preliminary results of the work to consolidate, mobilise and stimulate the European dependability constituency on pursuing this opportunity would be addressed.

High End Commercial Computer Fault Tolerance: Trends and Directions

Lisa Spainhower, IBM Corp.

I have collected and evaluated both machine-generated and human-recorded data on the source of computer system failures. Stepping back from the individual instances permits observation of some general lessons:

- Without sufficient system management discipline, even the best technology cannot deliver high availability.
- The best system management discipline cannot overcome technology shortcomings.

Conventional wisdom holds that software is so overwhelmingly the root cause of unplanned downtime that it is unwise to invest in robust hardware; actual data indicates that this isn't accurate. What is true is that the common level of hardware fault tolerance has increased a great deal in the past 5 years. I will also describe the practical fault tolerant methodologies that have been implemented and discuss their effectiveness.

As well as the change within the past five years, it is instructive to review the predictions for where the industry would be with respect to fault tolerance and high availability in 2001 which were made in the mid 1990s. High availability clusters built on open systems standards, for example, were expected to proliferate beyond what has actually occurred and the difficulty in implementing them has been a surprise. A discussion of lessons learned, particularly those which were surprises, is a key component of the presentation.

Next it's time to look ahead while considering what is known about the nature of computer system failure, effective fault tolerance, and what has not turned out as expected. I will discuss extensible technologies and areas where new innovation is needed. Hopefully, this can spur academic research as well as industry technology.

Metrics for the Internet Age: Quality of Experience and Quality of Business

Aad van Moorsel,
E-Services Software Research Department
Hewlett-Packard Laboratories
Palo Alto, California, USA
aad@hpl.hp.com

Abstract

We discuss quantitative metrics for evaluating Internet services: what should we quantify, monitor and analyze in order to characterize, evaluate and manage services offered over the Internet? We argue that the focus must be on quality-of-experience and, what we call, quality-of-business metrics. These QoE and QoBiz metrics quantify the user experience and the business return, respectively, and are increasingly important and tractable in the Internet age. We introduce a QoBiz evaluation framework, responding to the emergence of three types of services that impact quantitative evaluation: business to consumer services, business-to-business services, and service utility through service providers. The resulting framework opens avenues for systematic modeling and analysis methodology for evaluating Internet services, not unlike the advances made over the past two decades in the area of performability evaluation.

Quantitative Evaluation in the Internet Age: What is Different?

Who cares if your web-shopping site has an availability of 'three nines'? 99.9% availability corresponds to a little over eight hours of total down time in a year, but most customers do not care, nor do line-of-business managers. The customer only cares if the site is up when she wants to shop. Moreover, if the site is down for a few seconds during a shopping session, she will only care if it makes any difference for the execution of her ongoing tasks. The business manager only cares if she loses business because of down time, and for how much money this could have been prevented.

From the above example, it follows that system metrics such as availability are not sufficient to evaluate an Internet service. Instead, user experience metrics and business metrics are necessary. Surely, this is not a new realization—at any point in computing system history, there have been implicit considerations about user and business requirements when evaluating systems and services. For instance, decisions about upgrading computer equipment routinely consider user needs and costs. However, the problem has always been that the relationship between system quality, user experience, and business cost is hard to make concrete. QoS, QoE and QoBiz analysis has therefore never been done in systematic and integrated fashion.

In the Internet age this must change. The relation between QoS, QoE and QoBiz has become very apparent, and more easily to discern. In the web-shopping scenario, the user experience directly influences how much shopping the customer does, and how much

money the business makes as a consequence. Results are available that relate response times with user behavior and business loss. In the Internet age, not only are QoBiz metrics increasingly important, they are also more directly measurable and therefore better tractable.

The crucial difference in the Internet age is that people, processes and institutions play roles that are closely integrated with the system (that is, the Internet) itself. They form ecosystems in which business is conducted through technological means (business-to-consumer as well as business-to-business). In addition, technology services are being provided and consumed in similar fashion as electricity and water (the service utility model). In particular, we identify the following trends as drivers for QoBiz management:

- Integration of the user in Internet ecosystems (B2C)
- Integration of enterprise business processes in web-service ecosystems (B2B)
- Emergence of the service utility model (service providers/xSPs)

We discuss the metrics that are suitable to evaluate the ecosystems that arise from these three trends. Intuitively, B2C naturally leads to QoE analysis, and B2B requires QoBiz analysis. We introduce a more abstract notion of QoE and QoBiz, which allows us to construct a general framework in which every participant in an Internet ecosystem deals with QoS, QoE and QoBiz concerns at its own level of abstraction. That is useful for structured evaluation of service providers, which may have various participants interacting in a joined ecosystem (ISP, ASP, MSP, CSP, HSP, etc.). Eventually, we hope that the proposed framework is a useful steppingstone towards generic methodology and software support for QoBiz evaluation of Internet services.