School of Computing Science,
University of Newcastle upon Tyne

Dependable Pervasive Systems

Cliff Jones and Brian Randell

DEPENDABLE PERVASIVE SYSTEMS

Cliff Jones, Brian Randell

School of Computing Science, University of Newcastle upon Tyne

3 March 2004

EXECUTIVE SUMMARY

1. The UK is becoming ever more dependent on large networked computer systems yet the dependability of such systems is by no means always satisfactory. Techniques and tools available today make it *possible* to produce complex computer systems that work adequately dependably. However, there is a huge "deployment gap", with many organisations attempting to produce and use complex systems and in particular software (which is where the complexity of such systems mainly, and appropriately, resides) using technical and management methods which are far from "best practice".

2. Present trends indicate that huge networked computer systems are likely to become pervasive, as information technology is embedded into virtually everything, and to be required to function essentially continuously. Even today's "best practice" will not suffice for such systems. This document therefore concentrates on the problems of dependably producing large complex distributed systems to match their specifications within time and budget constraints, and the problems of actually achieving adequate operational dependability from such systems when they are deployed.

3. Such systems will often be constructed out of multiple pre-existing systems and need to be highly adaptable. Most will embody human beings as, in effect, system "components." The successful design and deployment of such systems is a major challenge, and calls for socio-technical as well as technical dependability expertise. Interdisciplinary approaches are essential.

4. Dependability (also termed "trustworthiness") is the ability to avoid failures that are more frequent or more severe, and outage durations that are longer, than is acceptable - the causes of such failures are termed faults. The four basic dependability technologies are (i) fault prevention (to avoid the occurrence or introduction of faults), (ii) fault removal (through validation and verification) and (iii) fault tolerance (so that failures do not necessarily occur even if faults remain), the effective combination of which is crucial, together with (iv) fault forecasting (the means of assessing progress towards achieving adequate dependability).

5. A variety of fault prevention and fault removal techniques are in use, in some cases as part of a formal (mathematically-based) design method – however, there is a need to make such methods and their tools easier to use. Fault tolerance is very effectively used for hardware faults, and in some arenas for software faults. Fault forecasting (i.e. system evaluation, involving estimation of the number and likely consequences of any remaining faults) can be quite effective, but has limitations with regard to large systems and extremely high dependability targets.

6. The problem of deliberate attacks on networked computer systems, and via them on other major infrastructures, by amateur hackers or well-resourced terrorist groups is already serious and seems certain to grow as systems become ever more pervasive. Detecting the onset of such attacks is insufficient – means are also needed for maintaining satisfactory service despite such attacks.

7. Much further research is required on all four dependability technologies to cope with future systems, but needs to be aimed at making system dependability into a "commodity" that UK industry can value and from which it can profit. Indeed the target is for those involved in creating future complex networked systems to find it technically feasible to offer *warrantable* software and systems. This can be facilitated by means of a system development approach, which (i) enables the likely impact on system dependability of all design and deployment decisions and activities to be assessed throughout the system life cycle, and (ii) caters for system adaptation, and the realities of huge, rapidly evolving, pervasive systems.

8. The research, to be successful, must be interdisciplinary. It must bring together various technical disciplines that presently concentrate unduly on particular types of systems, dependability attributes, types of fault, and means for achieving dependability, and must span socio-technical issues, through the involvement of *inter alia* sociologists, psychologists and economists.

DEPENDABLE PERVASIVE SYSTEMS

TABLE OF CONTENTS

1. STATE OF THE ART

Virtually all aspects of society in the UK and other developed countries are now dependent, to a greater or lesser degree, on computer systems and networks. For example, a US National Research Council Report [Schneider 1999] states:

> "The nation's security and economy rely on infrastructures for communication, finance, energy distribution, and transportation—all increasingly dependent on networked information systems. When these networked information systems perform badly or do not work at all, they put life, liberty, and property at risk. Interrupting service can threaten lives and property; destroying information or changing it improperly can disrupt the work of governments and corporations; and disclosing secrets can embarrass people or hurt organizations. The widespread interconnection of networked information systems allows outages and disruptions to spread from one system to others; it enables attacks to be waged anonymously and from a safe distance; and it compounds the difficulty of understanding and controlling these systems."

One from the US National Academy of Science [Estrin 2001] states:

> "Information technology (IT) is on the verge of another revolution. Driven by the increasing capabilities and ever declining costs of computing and communications devices, IT is being embedded into a growing range of physical devices linked together through networks and will become ever more pervasive as the component technologies become smaller, faster, and cheaper. These changes are sometimes obvious – in pagers and Internet-enabled cell phones, for example – but often IT is

> buried inside larger (or smaller) systems in ways that are not easily visible to end users."

It is easy to bemoan the fact that computer systems are less dependable than one might want but it is essential to understand that large networked computer systems are among the most complex things that the human race has created. The huge advances in the short history of computing must not be forgotten. Progress on hardware (as shown by the continued validity of Moore's "Law"[1]) has delivered platforms on which many major software applications have been successfully constructed. No human (design) activity can be expected to progress at the same rate; this accounts for many of the dependability complaints, but the techniques and tools available today have made it possible to produce software and systems with impressive functionality that do in many cases eventually work adequately dependably, though their development all too often incurs major schedule and cost over-runs. However, there is a huge "deployment gap" with many organisations using technical and management methods which are far from "best practice", both with regard to the effectiveness of system development, and the quality of the resulting systems.

The current situation looks set to become more precarious: present trends and predictions indicate that huge, even globally-distributed, networked computer systems, perhaps involving everything from super-computers and large server "farms" to myriads of small mobile computers and tiny embedded devices, are likely to become highly pervasive. In many cases such systems will be required, by governments, large industrial organizations, and indeed society at large, to function highly dependably and essentially continuously. This at any rate would be the consequence should various current industrial and government plans –see for example [European Commission 2002]– come to fruition. Such plans have been formulated despite the fact that the envisaged systems are likely to be far more complex than today's typical large systems. Hence questions as to how, and at what cost, and indeed whether, such systems can be developed so as to have acceptable functionality and dependability, and thus gain the trust of the people affected by them, are matters of great importance.

The present document analyzes this situation, concentrating mainly on issues related to software. This section summarizes the current state-of-the-art concerning the overall subject of system dependability, a term which encompasses such characteristics as reliability, security, safety, availability, etc. (According to needs and circumstances, some appropriate balance of such characteristics will be needed for a given system to be regarded as "*trustworthy*", i.e. dependable, by its users and owners.) A further section describes some general research goals that we believe are among those that will be fundamental to ensuring that the next decade will have computer systems whose dependability is commensurate with the requirements that are implied by present policies and trends.

The present situation is perhaps best illustrated by the following sample facts and statistics regarding the costs of undependability:

- The average cost per hour of computer system downtime across thirty domains such as banking, manufacturing, retail, health insurances, securities, reservations, etc., has recently been estimated at nearly $950,000 by the US Association of Contingency Planners [2]

- The French Insurer's Association has estimated that the yearly cost of computer failures is 1.5 –1.8B Euro (10 – 12B Francs), of which slightly more than half is due to deliberately-induced faults (e.g. by hackers and corrupt insiders) [Laprie 1999]

- The Standish Group's "Chaos Chronicles" report for 2003 analyzed over 13,000 IT projects, and estimated that nearly 70% either failed completely or were "challenged" (i.e. although completed and operational, exceeded their budget and time estimates, and had less functionality than originally specified). This led to their estimate that in 2002 the US "wasted $55 billion in cancelled and over-run IT projects, compared with a total IT spend of $255 billion." (Quoted in the July 2003 report on "Government IT Projects" [Pearce 2003].)

- This July 2003 report states that in the UK: "Over the past five years, high profile IT difficulties have affected the Child Support Agency, Passport Office, Criminal Records

---

[1] In 1965 Gordon Moore, of Intel, noted that the number of transistors per integrated circuit was doubling every year! This situation lasted until the 1970s, when the doubling slowed down to every 18 months, but this still dramatic rate of progress has continued ever since, and shows every sign of going on for some years to come. Similar improvements are occurring in storage and network technologies, presaging continued spectacular improvements in system costs and performance.

[2] http://www.acp-wa-state.org/Downtime_Costs.doc

> Bureau, Inland Revenue, National Air Traffic Services and the Department of Work and Pensions, among others."

- The AMSD Overall Dependability Roadmap [AMSD Roadmap 2003] cites a recent UK survey of 1027 projects which reported that only 130 (12.7%) succeeded. (Many of these projects were maintenance or data-conversion projects – of the 500+ development projects in the sample surveyed, only 3 (0.6%) succeeded!)

This situation would undoubtedly be considerably improved if all project developments were carried out using the levels of technical and managerial expertise that some of the best developments exhibit. However, merely encouraging the take-up of today's best practice, though it would greatly help the current generation of systems, will not suffice for the decade to come, given the changes these coming years will undoubtedly bring.

These changes will occur in part because of the plans referred to above – but also because of the continuing headlong pace of technological development. (To get some feel for what the next ten years will bring, one need merely think back to the situation of ten years ago, when a typical PC ran at 50Mherz, and had perhaps 4Mbytes of memory and a 100Mbyte disk. Regarding wide area data communications, for most network users a 9600 bps telephone modem was the norm – and a new and little known scheme called the World Wide Web was only just starting to attract attention in parts of the research community.)

We therefore concentrate in this document on current and especially future dependability issues related to large complex distributed systems, both the problems of dependably constructing such systems (i.e. of producing systems to match their functional and dependability specifications) within time and budget constraints, and the problems of actually achieving adequate operational dependability from such systems when they are deployed.

Such systems are rarely constructed *de novo*, but rather built up from earlier systems, indeed often actually constructed out of multiple pre-existing systems – hence the term "*systems-of-systems*" (SoSs). Moreover, there is almost always a requirement for systems to be capable of being *adapted* so as to match changes in requirements –such as changed functionality, and changes in system environments– and there is likely to be an increasing need for systems that can themselves *evolve*, for example in order to serve a dynamic environment constituted by large numbers of mobile devices moving into and out of range.

Most of the systems of concern embody human beings as, in effect, system "components" (though sometimes such humans become actively involved only when things go wrong with the computers and their communications) – such systems we term "*computer-based systems*" (CBSs). The various types of "component" have rather different failure characteristics: (i) hardware suffers mainly from aging and wear, although logical design errors do sometimes persist in deployed hardware – see for example [Avizienis and He 1999] and [Van Campenhout, Mudge et al. 2000], (ii) it is mainly software that suffers from residual design and implementation errors, since this is –quite appropriately– where most of the logical complexity of a typical system resides, and (iii) human beings (users, operators, maintenance personnel, and outsiders) can at times, through ignorance, incompetence or malevolence, cause untold damage. (Malevolent activities can range from acts of petty vandalism by amateur hackers, through the efforts of expert and highly devious virus creators, to well-resourced and highly sophisticated attacks by terrorists and enemy states.) The successful design and deployment of a major CBS –and indeed of any system that interacts directly with humans– thus calls for socio-technical as well as technical expertise, a point we will return to in Section 3 below.

The task of developing a complex computer system involves human beings, possibly in large numbers, aided by extensive computer facilities that provide means of recording, and analyzing specifications, compiling and evaluating detailed designs, etc. These facilities and their users in fact constitute what we term here a "*system design system*" (SDS) [3]. (This is a special class of CBS; in fact all three categories –CBS, SDS and SoS– overlap.) As indicated earlier, in this document we concern ourselves with the dependability of the system-building process (which will be crucial to the timely production of acceptable systems), i.e. of such SDSs, as well as that of the resulting complex SoSs and CBSs.

---

[3] The task undertaken by an SDS is of course both a management and a combined technical/socio-technical challenge – this report concentrates on the latter challenge (which when handled inadequately can greatly exacerbate the management challenge).

(However, issues regarding the provision of the networking infrastructure that such systems will be built and will depend on fall outside the scope of this document.)

1.1 Concepts and Definitions

The topic of dependability, or trustworthiness, cannot be tackled adequately without a careful identification of the basic concepts involved and their definitions.[4] (The choice of actual terms is less important; it is the clarification of basic concepts that matters.)

> The **dependability** (or, equivalently, **trustworthiness**) of a computing system can be defined as the "the ability of a system to avoid failures that are more frequent or more severe, and outage durations that are longer, than is acceptable".

We need to distinguish between three basic concepts, termed here *failure*, *error* and *fault*:

> A given system, operating in some particular environment, may fail in the sense that some other system makes, or could in principle have made, a *judgement* that the activity or inactivity of the given system constitutes **failure**.

The judgemental system may be an automated system, a human being, the relevant judicial authority or whatever. (It may or may not have a fully detailed system specification to guide it – though evidently one would expect such a document to be available to guide the system construction task.) Different judgmental systems might, of course, come to different decisions regarding whether and how the given system has failed or might fail in the future – in differing circumstances and from different stake-holders' viewpoints, the crucial issue might concern the accuracy of the results that are produced, the continuity or timeliness of service, the success with which sensitive information is kept private, etc.

Moreover, such a judgmental system might itself fail –in the eyes of some other judgmental system– a possibility that is well understood by the legal system, with its hierarchy of courts. So, we have a (recursive) notion of "failure", which clearly is a relative rather than an absolute notion. *So then is the concept of dependability.*

From the notion of failure, we move on to discuss and distinguish between what we term here errors and faults:

> An **error** is that part of the system state that is *liable to lead to subsequent failure*: an error affecting the service that is being provided by a system is an indication that a failure occurs or has occurred. The *adjudged or hypothesised cause* of an error is a **fault**.

Note that an error may be judged to have multiple causes, and does not necessarily lead to a failure – for example error recovery might be attempted successfully and failure averted.

Thus a failure occurs when an error 'passes through' the system-user interface and affects the service delivered by the system – a system of course being composed of components which are themselves systems. Hence the manifestation of failures, faults and errors follows a "fundamental chain":

> . . . → failure → fault → error → failure → fault → . . ."

Such chains can exist within systems. System failures might be due to faults that exist because of, for example, the failure of a component to meet its specification. However, these chains also exist between systems. For example, a failure of an SDS might result in the wrong choice being made of component in the system that is being designed, resulting in a fault in this latter system that leads eventually to its failure. Also, a system might fail because of the inputs that it receives from some other system with which it is interacting, though ideally it will be capable of checking its inputs thoroughly enough to minimise such failures.

It is essential to have separate terms for the three essentially different concepts named here *"fault", "error"* and *"failure"* – since otherwise one cannot deal properly with the complexities (and realities) of failure-prone components, being assembled together in possibly incorrect ways, so resulting in systems, which in reality will still be somewhat failure-prone. Moreover, with the sort of system we are concerning ourselves with here, there are likely to be uncertainties and arguments about system

---

[4] Here we provide only a very brief outline of these concepts – fuller accounts can be found in [Laprie 1991] and [Randell 2000]

boundaries, the very complexity of the system (and its specification, if it has one) can be a major problem, judgements as to possible causes or consequences of failure can be subtle and disputable, and any provisions for preventing faults from causing failures are themselves almost certainly fallible.

The above definitions, in clarifying the distinction in particular between fault and failure, lead to the identification of the four basic means of obtaining and establishing high dependability, i.e., minimising the frequency and seriousness of failures, namely:

- *fault prevention* (prevention of the occurrence or introduction of faults, in particular via the use of rigorous design methods),

- *fault tolerance* (the means of delivery of correct service in the presence of faults),

- *fault removal* (i.e. verification and validation, aimed at reducing the number or severity of faults) and

- *fault forecasting* (system evaluation, via estimation of the present number, the future incidence, and the likely consequences of faults).

In the absence of (successful) fault tolerance, failures may occur and will presumably need to be tolerated somehow by the encompassing (possibly socio-technical) system - however this notion of "failure tolerance" is just fault tolerance at the next system level.

One of the most important lessons from work over many years on system dependability is the fact that fault prevention, removal and tolerance should not be regarded as alternatives, but rather as complementary technologies, all of which have a role to play, and whose effective combination is crucial – with fault forecasting being used to evaluate the degree of success that is being achieved. Moreover, and this is not so well-understood, (i) all four of these dependability technologies are as relevant to the system design. implementation and deployment process, i.e. to SDSs, as they are to the systems that are being produced, and (ii) in many situations effective use of the strategies calls for the exercise of socio-technical as well as technical expertise. These two lessons guide the structuring and content of what follows.

1.2 Current Systems, and Current Abilities

The AMSD Overall Dependability Roadmap [AMSD Roadmap 2003] estimates that for large and complex computer systems, namely those involving 1M – 100M lines of code, current development techniques, i.e. SDSs, can at best produce systems that achieve a level of reliability in the range 10 to 100 failures/ year.

In fact, in some ways the system dependability situation has been getting worse rather than improving in recent years. Quoting the AMSD Roadmap, "the availability that was typically achievable by (wired) telecommunication services, and computer systems in the 90s was 99.999% – 99.9%. Now cellular phone services, and web-based services, typically achieve an availability of only 99% – 90%". (One can speculate that this is in part because of both greater overall system complexity, but also the move away from a highly regulated economic environment made up of major, often nationally owned, companies that were required by their governments to meet stringent reliability and availability standards.)

It is extremely difficult to obtain any objective, leave alone quantitative, assessments of the present state of the art with respect to system security. However, the AMSD Roadmap states that "the increasing number of breaches that is indicated by industry and government surveys and statistics on cyber-crime is already impacting on user trust and confidence and hence take-up. For example, the EU project Statistical Indicators for Benchmarking the Information Society (SIBIS) indicates that perceived online risks are deterring large numbers of consumers from making online transactions".

Schneier's book "Secrets and Lies: Digital Security in a Networked World" [Schneier 2000] starts with an alarmingly long list of security breaches reported during just the first week of March 2000, and goes on to say:

> "Digital security tends to rely wholly on prevention: cryptography, firewalls and so forth. There's generally no detection and there's almost never any response or auditing. A prevention-only strategy only works of the prevention mechanisms are perfect; otherwise someone will figure how to get around them. Most of the attacks and vulnerabilities listed . . . were the result of bypassing the prevention mechanisms."

In fact both Schneier and also Anderson [Anderson 1994] report that the great majority of reported security problems are due to the undependability of the underlying technical infrastructure and, especially, the socio-technical environment, rather than deficiencies of the cryptographic and other security mechanisms employed.

Moving on to consider SoSs – this concept is not a new one, but has been given much impetus in recent years by various industry-led standardization efforts. Currently, the major emphasis is on the "web services" approach, which facilitates the use of web-servers by other computers, not just by human users of web browsers. By such means major pre-existing computing services can be taken advantage of in constructing new SoSs [Narayanan and McIlraith 2002]. (An example that is commonly used to explain this approach is that of constructing a general travel agent enquiry service out of a set of separate pre-existing enquiry services operated by various hotel companies and airlines.)

The web services approach to constructing SoSs takes advantage of XML and allied technologies for defining and structuring the formats of data messages sent to and received from web-servers. As a result, a large number of industry teams are now working on establishing special XML-based standards, each intended for use within a particular application sector. However, defining the exact meaning of such messages (as opposed to just their formats), and dealing with inter-sector incompatibilities, remain difficult, and hence a likely source of future SoS failures, as is any inability on the part of the designers to establish the exact specification of the services offered by a pre-existing system that has to be incorporated into their SoS.

Perhaps the biggest problems at present concern (i) SoSs whose component systems belong to separate, even mutually-suspicious organisations, perhaps operating in different legal jurisdictions, (ii) SoSs that need to be created quickly but have only a temporary existence, in support of what the military term "dynamic coalitions", or their commercial counterparts, and (iii) "accidental" SoSs (ones that have come into existence through almost haphazard system interconnections that have subsequently proved so useful as to create a system upon which large numbers of users become dependent),

Regarding general CBSs, the whole subject area is in an early state, with comparatively little socio-technical expertise being available to guide system development, which thus tends to be treated as a solely technical challenge. However, with specialised CBSs in the arena of safety-critical control, for example, things are somewhat better. There has been study in this arena of how best to partition the overall task between humans and computers so as to reduce the probability of failures due to misunderstandings across the human-machine interface, and to make best use of the greatly differing abilities that humans and computers have with respect to (i) following complicated detailed sequences of instructions, and (ii) recognising that a situation is both novel and potentially dangerous. But elsewhere there have been a number of major failures, due to the little-understood realities of CBSs, which have even led to complete project cancellations. (For example, it is all too easy to specify and implement a would-be highly secure CBS whose technical security features are completely nullified by the workarounds that users find themselves having to adopt if they are going to get their work done.)

There are already a number of major government and industry initiatives, especially in the US, aimed at improving the present situation. These include projects by IBM on "Autonomic Computing" [5], Microsoft on "Trustworthy Computing" [6], DARPA's OASIS Program [7], and NSF's "Trusted Computing" Program. [8] (Much further detail can be found via the URLs given in the footnotes.)

1.3 Current Usage of the Dependability Technologies

A system typically can fail in several –possibly many– different ways, and these failures can be of widely varying severity as far as the system's environment is concerned. Thus it is important to design systems so as at least to minimise the frequency of serious failures, and where possible to set up or to modify system environments so as to avoid undue dependency on the more demanding functions expected of a system. Indeed, minimising dependency can be as important as maximising dependability. Unfortunately these issues are not always well understood.

---

[5] http://www-3.ibm.com/autonomic/index.shtml

[6] http://www.microsoft.com/security/whitepapers/secure_platform.asp

[7] http://www.darpa.mil/ipto/programs/oasis/goals.htm

[8] http://www.cise.nsf.gov/fndg/pubs/display2.cfm?pub_id=5370

These issues are equally important within systems, where they take the form of architectural design strategies concerning (i) the positioning of system interfaces, especially between humans and computers, (ii) the structuring of system activity so as to limit error propagation (e.g. via techniques that can be seen as generalisations of the concept of a data base transaction), (iii) the assessment of what dependence can safely placed where (i.e. which components can be "trusted", in some appropriate sense), a crucial aspect of the task of "risk management".

Successful resolution of such issues provides an excellent starting point for making good use of the four dependability technologies, which we now deal with in succession.

1.3.1 Fault Prevention

The task of fault prevention centres on extremely careful, well-documented and well-managed processes for specifying a system's required functionality (and so-called "non-functionalities", in particular dependability) and then making the myriad decisions, from architectural to detailed coding choices, that constitute the full development process – a process which will of necessity be tentative and hence iterative, and which may continue well after initial versions of a system are first fielded. This process is aided particularly in the case of software by the appropriate use of well-supported modularization and information-hiding techniques, rigorous design conventions, etc., while techniques such as shielding and radiation-hardening can prevent the introduction of operational physical faults into hardware systems.

With software, the production task that is needed in order to generate multiple instances of a software component is essentially finished once all the design and implementation decisions have been taken (and confirmed), given the trivial ease with which one can generate accurate copies. Production of an actual hardware system however involves extremely challenging manufacturing processes that also can introduce faults if great care is not taken.

Additional complications arise when what is being designed is some sort of generic system, from which a version of the system intended for a particular environment and set of circumstances has to be produced by some "configuration process", indeed an activity which may need to be repeated in order to adapt a system to changing needs. However, this is in reality just a continuation of the design and implementation process, and hence equally an arena in which fault prevention has a major role to play, though with the additional difficulty of needing to confirm that the system's basic functionality actually admits of being configured and re-configured as required.

Europe in general and the UK in particular have made major contributions to *formal development methods* for systems (especially software) [9]. Used sensibly, such formal methods can produce systems such as the French RER train control software that are (known to be) highly reliable [Guiho and Hennebert 1990]. Furthermore, as for example Praxis' use of formal methods on a CAA air traffic control system has shown, the resulting software can be of significantly higher quality, but can also be produced more quickly and at lower cost than is typical of the much less rigorous techniques that are employed in many other parts of the software industry. (Several other European software and systems companies have reported similar findings.)

None of this should really be surprising: any significant engineering task requires documentation and justification and the mark of an engineering discipline is the use of mathematically based methods. What is surprising is that such methods are not more widely applied in software design, at least for relatively modest systems and system components. Lessons which have to be learnt by formal methods researchers include the fact that that there is a cost-benefit trade-off in the mathematical sophistication expected of users, and that pictorial representations can both have firm semantic foundations and be readily accepted. From the users side, it is true that as increasing percentages of practitioners have sound engineering training (and possibly formal engineering professional status) this will result in a culture which will more readily accept new (formally based) notations, especially if these are well supported and integrated with conventional software engineering tools [Rushby 2000].

This is not however to claim that current formal methods and their tools are readily applicable to large and complex systems in their entirety. Indeed a review commissioned by the German Bundesamt für Sicherheit in der Informationstechnik (BSI), is reported in the AMSD Overall Dependability Roadmap as having given the following rough estimates of their present limits:

---

[9] Surveys of this area are provided by, for example, [Clarke and Wing 1996] and [van Lamsweerde 2000].

- Equivalence checking of 1 Million gate ASICS

- Model checking of 1000 latches at a time, using techniques for modularising and looking at components, e.g. there are claims of verifying large (10^{20}) state spaces

- Software verification from design to code of ~80K lines of code

- Simple formally verified compilers for special purpose languages

- Static analysis of >150K lines of code

- Specification and modelling of >30,000 lines of specification

This of course does not mean that completely formal methods have no part to play in the development of large systems – rather that it is likely to be confined for some time to come to small highly-critical areas, and to the higher architectural levels of systems. A major issue here for wider use and applicability is tool support.

A separate, equally important issue, given the very frequent requirement to adapt existing systems, is that there is a need to find means of adaptation that at least preserve existing dependability levels, and which make good use of the existing rigorous design documentation and arguments. The present situation in which, for example, a security certification is typically valid for only a single hardware and software configuration, and in which formal validation typically has to start again from scratch after any significant change is made, will have to be improved greatly if formal methods are to contribute to making adaptation a dependable and cost-effective process.

1.3.2 Fault Removal

Fault removal is employed both during system development and system deployment – in the former case it can form an integral part of a formal engineering approach. A first level of such fault removal relates to faults of a general nature, such as deadlocks or buffer overflows; the more challenging level relates to system-specific faults, i.e. ones that are related to the particular requirements placed on a system.

Though terminology varies, it is typical to distinguish between the task of "*validating*" a system specification (i.e. establishing that it is consistent with the requirements that are placed on the system), and that of "*verifying*" that a system adheres to its specification. Validation and verification can take place at all stages of development and deployment, though the aim is to perform both at an early a stage as possible. One can distinguish between static verification, which involves inspecting or analyzing the system without actually exercising it, and dynamic verification, which involves exercising the system on symbolic or actual inputs. (This latter approach, system testing or "debugging", is in fact often the primary means by which confidence in a system is achieved.) Validation and verification can be fully integrated into a formal development process and, if well supported by tools, can greatly encourage the industrial take up of such processes.

Tools cannot replace the need for thought but they can increase the effectiveness of the design intuitions and inspections. An interesting case study is the ready acceptance of *model checking* tools. Some model-checking systems can be used to detect critical timing problems such as "deadlocks"; other systems check that logical (often temporal) assertions are true of executions of a system. At least in their initial forms, model checking tools worked on executable code of a system. Their attraction is that one can deploy them on the final product but this is also their weakness: they do not help construct a correct implementation they only help locate errors. A badly architected system will never be made dependable by work at the final stages of development.

Another example of tools that help locate errors is the class of "extended static checkers" (ESC). Hoare [Hoare 2002] reports significant progress with the adoption of *assertions* within Microsoft where ESC tools support their use. To gain the full benefit of such formal approaches they must however be used early in the development process where the key architectural decisions are made –and recorded as abstractions of the detailed code– which affect (related) issues such as usability, and security, rather than just as an adjunct to detailed implementation.

For many years much if not most fault removal has been done through various forms of conventional system testing. There is a large and active academic and industrial test community, whose activities range from the development and use of sophisticated tool-supported techniques for selecting and automatically processing test cases, down to simply making "alpha" and "beta" versions of software

available to large, perhaps very large, numbers of customers, so as to receive bug reports from them. Clearly this latter approach is of little relevance to one-off systems.

Ideally, the result of using fault prevention and removal techniques, especially highly formal ones, is a "*correct*" system, i.e. one that is completely free of design faults by the time it is deployed. In practice, even with relatively small software systems, intended for safety-critical applications and developed using highly-prescribed well-tooled techniques, the fault density (faults per thousand lines of code) is rarely less than one – though significantly better than this can be achieved by taking extreme care, and it has been claimed that some Space Shuttle software had less than 0.01 faults per thousand lines of code [AMSD Roadmap 2003]. But such statistics relate to systems that are far less complex than the ones that concern us in this document, systems whose deployment must surely also involve extensive reliance on fault removal and/or fault tolerance.

1.3.3 Fault Tolerance

The subject of fault tolerance originated in the earliest days of computing, and was aimed at reducing the frequency and/or seriousness of system failures due to operational hardware faults. The provision of means of tolerating operational hardware and environmental faults, whether at a detailed level, for example via the use of error detecting and correcting codes, or at an architectural level via the use of replicated major components, indeed replicated complete computers, is now comparatively well-understood, and well-supported by design methods and tools. One interesting current example of the successful use of such fault tolerance, on a well-known system with a huge global user base, is that made by Google, which uses massively replicated processor and storage facilities in order to minimise system down time and unavailability of any of its huge web archive [Ghemawat, Gobioff et al. 2003].

Tolerance of (software) design faults is not so well developed, but is achieved in some areas. These include database systems (whose facilities for logging and rerunning transactions have been shown to cope with a significant fraction of any residual software bugs), and safety-critical systems (such as the Boeing 777 flight control system, which employs redundant, diversely designed, software modules [Yeh 1999]), but also –to a degree at least– in a number of well-known standard desk-top applications (by means of run time assertions, restart facilities, etc.).

Fault tolerance in SoSs is typically provided using additional software components, in particular in the form of "wrappers", i.e. software that surrounds an existing system, and –for example– augments its means of error detection, and tries to cope with any disparities between the service it provides, and that expected by the rest of the SoS [Anderson, Feng et al. 2003].

CBSs can suffer from any of the problems that can afflict more confined computer systems but are also vulnerable to many others because of the involvement of humans. A computer system can perfectly match its specification but the overall system can fail if unrealistic expectations are put on operators (e.g. a response to many alarms in an extremely short time). A general problem with systems such as those that help operators control nuclear power plants is to ensure that users can gain a clear picture of the "state" of the physical system they are supposed to be monitoring. On the other hand, the human "components" of a CBS may have critical roles to play in assuring that failures of the computer systems within the CBS are detected and recovered from.

Regarding the use of fault tolerance during the system design process, many of the tasks involved in developing a system gain such dependability as they possess by exploiting redundancy and diversity. For example, checking the consistency of a design with a specification, providing they have been produced "independently", whether this checking is undertaken manually or automatically, can assist greatly in locating errors in each. However, a specification that has been generated automatically from a design (or a design that has been synthesized from a specification) contributes little to the task of finding and removing faults from either the specification or the design, though the chances of their being consistent with each other will presumably be enhanced by such automation - whereas redundant diverse specifications have been shown to be very effective [Anderson 1999]. Activities such as model-checking and static analysis can indeed all be viewed as contributing to the dependability, via fault tolerance, of the SDS within which they are employed.

In such wider systems, the problems of fault tolerance, indeed the problems of dependability in general, are socio-technical problems – this is especially the case when it is necessary to take into account the possibility of faults arising from malicious behaviour, whether by insiders or by external attackers. However, the present state of the art with respect to the dependability of complex CBSs is at early stage of development.

The idea of employing fault tolerance to enhance system security is not new [Dobson and Randell 1986], but is still a subject of research, e.g. by the recent EU-IST MAFTIA project [Powell, Adelsbach et al. 2001] [10] and elements of the DARPA OASIS Program, mentioned earlier. The aim of such research is to find means, either by fault masking or by error detection and recovery, of continuing to provide a useful level of secure service despite the fact that attackers are exploiting some vulnerability that has been discovered and not yet remedied. (Fault masking might for example involve the use of multiple diverse redundant sub-systems, so that service could continue as long as only a minority had been compromised by the attacker; intrusion detection is a form of error detection – though at present in practice any subsequent recovery is still largely manual.) But the discipline of choosing the goals for would-be secure systems in such a way that their (human and organizational) environments will be able to tolerate the systems' inevitable occasional failures is already of current practical importance, though rarely followed [Schneier 2000].

1.3.4 Fault Forecasting

Quoting from an Appendix to the AMSD Overall Dependability Roadmap [Littlewood, Bloomfield et al. 2003]:

> "Underpinning almost all work on dependability evaluation must be a realisation of the 'inevitability of uncertainty' – in the processes we use to build systems, in the behaviour of the systems themselves. We cannot predict with certainty how dependable a system will be, even if we use the best process (e.g. as mandated in a standard); we cannot predict with certainty the failure behaviour of a system. Dependability evaluation is thus an evaluation in the face of uncertainty and will generally involve probability . . . An exception to this observation concerns those circumstances where formal verification of certain dependability properties is possible – e.g. proof that a certain class of failure is impossible under certain prescribed circumstances. However . . . the possibility of fallibility of the proof process needs to be addressed . . ."

The quantitative estimation of operational hardware dependability, at least for relatively small assemblages of hardware components, even in sophisticated fault-tolerant architectures, is a well-established field, well supplied with powerful tools, based for example on Stochastic Petri Nets (e.g. Möbius [11] and SURF-2 [12]). The accuracy of the results obtained naturally depends on the accuracy of the component dependability statistics used, and the correctness of any assumptions made, e.g. that faults will occur essentially independently.

The basic difficulty underlying all forms of evaluation of large computer systems, in particular that of estimating their dependability, is that of their discrete nature and complexity. One dare not assume that their logical design is fault-free, and thus conventional engineering concepts such as "strength", "safety factor", etc., are not available, and enumeration of all relevant possibilities, though in theory required, is in fact totally infeasible.

Statistical experiments aimed at estimating the number of design faults remaining in a complex software component, leave alone the frequencies with which any such fault might be activated, need extremely large samples and are crucially dependent on the faithfulness with the statistical sample mirrors (with respect to the effects of the unknown design faults!) the characteristics of real operation. The present situation is that though –with considerable effort– software components (of relatively modest size and complexity, such as are typically used for safety-critical control applications) can be built that in operation show every sign of achieving extremely high dependability, say 10^{-9} probability of failure on demand, it is rarely feasible to justify claims beyond say 10^{-3} or at most 10^{-4} by such methods.

Quantitative evaluation of the dependability of the various activities carried out within an SDS is particularly difficult, and rarely attempted – it is, for example, far easier to assume (unjustifiably) that once the design of some component has been "proved" correct, or model-checking has been completed,

---

[10] http://www.newcastle.research.ec.org/maftia/

[11] http://www.crhc.uiuc.edu/PERFORM/mobius-software.html

[12] http://www.laas.fr/surf/

that these processes were themselves carried out faultlessly, rather than come up with some credible numerical estimate of the probability of this being the case.

In practice, one has to settle for a "dependability case" [13] which brings together all the available evidence about a system and the processes that were involved in its construction and validation – the unsolved problem is that of knowing how to combine this evidence, in order to come up with an overall estimation that is itself dependable. Thus, perhaps not surprisingly, one current approach is to try to make the system evaluation process itself fault-tolerant, by constructing the dependability case out of a "multi-legged" argument [Bloomfield and Littlewood 2003], and providing some believable evidence that there is sufficient diversity and independence between the various "legs" of the argument that, although each on its own may be insufficiently strong, their combination will be persuasive.

At bottom, the fundamental problem is that of identifying all the assumptions that have been made (e.g. that particular faults will never occur together, that an alleged proof is in fact correct, that a specification is complete and fully-appropriate, that the compilation process did not introduce any errors, that two processes really are highly diverse, etc.) and then of justifying these assumptions.

1.4 Drivers

Moore's Law, as discussed earlier, facilitates the prediction of future hardware capabilities and costs, i.e. of the direct impacts of technological developments. But the indirect impacts, e.g., of the new markets that will be created, the new functions that it will be found feasible and attractive to automate, are much more difficult to predict, and likely to be at least as great in practice. The creation of these new functions, perhaps in the form of entirely new applications or types of device, will however be greatly influenced by certain factors which though not unique to the IT industry, are unusually significant.

In particular, much of the software industry, at least those parts of the industry that are concerned with commodity software, is inherently likely to result in the creation of so-called "natural monopolies" because, other than for one-off or specialised software, costs are greatly dominated by those of development, rather than manufacturing. Thus in such situations it is difficult to compete with any company that gains a dominant position in a particular market. (The economics of this situation are however, at least in part of the software world, complicated by the emergence of the "open source" movement, which some have argued will have a beneficial influence regarding system dependability, especially security.) There is thus a particular pressure to be "first to market", usually at the expense of dependability. Add to this the fact that outside particular closely-regulated market areas, for example those concerned with specialised safety-critical or security-critical systems, much software can currently be sold with little regard for issues of product liability, and it is clear why to date ordinary market forces are often not very effective in ensuring that dependability concerns are properly valued. [14]

Instead, government policies, and perhaps social and media pressures, may prove at least as significant. (One can readily trace several recent US industrial initiatives, such as those mentioned in Section 1.4, at least in part to public post-September 11 pressure placed directly on the companies by the US Government.) In addition, factors such as the relative weight given to the protection of personal privacy versus law enforcement considerations, and how issues of cross-border legal jurisdiction develop, may also greatly influence how system dependability issues are treated in the future.

Another non-technical influence on what levels of system dependability will be achieved is the current fashion for "efficiency gains". This often results in over optimisation: running very close to capacity margins makes load fluctuations much more likely causes of system failure, and arguments for the provision of extensive protective redundancy can fall on deaf ears. This particular point applies to all sorts of systems, not just computer systems of course, but with computer systems it is exacerbated because of the difficulty in predicting and quantifying dependability gains.

---

[13] The notion of a "dependability" case is an obvious generalisation of the now well-accepted idea of a "safety case."

[14] For example, it would appear that the present problems of wholesale virus propagation largely stem from the fact that Microsoft's email systems ignore (in favour of functionality and "user convenience") the restrictive rules in RFC 1341 (http://www.faqs.org/rfcs/rfc1341.html), the 1992 MIME standard for email attachments, which were specifically aimed at reducing "the danger of transmitting rogue programs through the mail."

Finally, an even more direct driver is the seemingly ever-increasing level of malicious and fraudulent behaviour aimed at computing systems – to say nothing of possible serious terrorist attacks. The future impact of planned and possible further law enforcement measures, both on offenders, and on system providers, is extremely difficult to forsee.

1.5 Pervasiveness

As computers become ever smaller and cheaper, it is becoming feasible to integrate them into ever more things, and ever more aspects of life and commerce. At the same time there are now many ways of interconnecting computers, over various distances, and effective means of reaching agreement on protocol standards, largely based on IP.

Thus the quotation, attributed to Lamport [15]:

> "A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable"

was but a harbinger of things to come. One can now, for example, imagine situations in which computers you didn't know you were wearing or carrying are involved in large-scale dependability problems that are adversely affecting huge numbers of people you've never heard of. Indeed, there are predictions of "everything being connected to everything", and of "disappearing computers", which have led to evident interest in the many new commercial opportunities, and –in some quarters– grave concerns about possible loss of privacy.

More generally, there is the question of how effectively the environment which is pervaded by computers can tolerate any serious overall system failures – in other words of the extent to which this environment has become dependent on these pervasive systems. The telephone system can be regarded as representing an early at least somewhat pervasive system. Isolated and temporary failures of small parts of this system, e.g. effecting individual subscribers, were relatively common, and acceptable. However, incidents such as the one in which almost the whole of the US east coast telephone system was brought down for many hours, as a result of a cascading set of faults (arising during a system upgrade) provides a dramatic foretaste of what might happen in future [Kemp 1990].

Pervasive computing implies dynamic groups of "computation facilities" that are in communication. Unlike the problem of "network partitioning", this means (for example) that each individual node has no concept of say "half of the nodes being reachable". Thus there is a real challenge of knowing which nodes are trying to do what. Add to this the fact that there will be trusted and untrusted partners and you get (playground like) pictures of a node realising that it is surrounded by untrusted nodes. Obviously this whole picture is unbelievably dynamic. This raises challenges of what is the "purpose" of a node and where this purpose comes from? All of these practical problems need theory underneath to really pin them down.

2 FUTURE DIRECTIONS

Technology, and in particular R&D, forecasts often attempt to 'predict' breakthroughs, as well as incremental technological developments, and ignore many types of possible (socio)-technical failures and their consequences. However, we feel confident in predicting that there will be no magic bullet to solve the problems of system dependability. Furthermore, we believe that major improvements in the general dependability of the sorts of large scale system that are now being created can be achieved simply via full employment of best practice, and industrialisation of current and already-planned future research. But even widespread use of the best methods available today will not suffice for the systems of the future. Thus the body of this section identifies a number of specific long-term challenges for researchers.

2.1 Some Specific Long Term Research Targets

2.1.1 Dependability-Explicit Systems and SDSs
Current approaches to the design of dependable complex computer systems are in some ways analogous to the early methods for designing real-time systems, i.e. systems whose operations have to conform to stringent timing constraints. These early methods were essentially iterative, in that only by

---

[15] http://research.microsoft.com/users/lamport/pubs/distributed-system.txt

completing a version of the system could it be found whether possibly extensive redesign was needed in order to ensure that timing constraints would be met. (Imagine that an aircraft design team had to build a first example of their new plane before they could find whether it met overall weight constraints!)

By mentioning these analogies, a worthwhile, but highly challenging long term goal for dependability research immediately suggests itself – that of incorporating effective concern for the achievement of any necessary dependability targets into the full system life cycle. This involves (i) identifying explicit dependability requirements before a system is specified and designed, (ii) gathering dependability information and estimates concerning system components, and of the estimated effect on system dependability of any use of the various development and adaptation tools and techniques, and (iii) exploiting this information throughout the development process, and during system deployment, in order to provide dependability-related guidance to all design and adaptation decisions.

The notion therefore is to annotate all design information files (specifications, algorithms, code, etc.) with dependability-related meta-information, and to use and update this meta-information as the files themselves are processed (by validation & verification tools, etc.), using meta-level operations associated with these tools which carry forward the dependability estimation process. Such meta-information is likely to be composite, and in part qualitative – and the task of determining how it should be processed will be far from straightforward, compared even to the processing of timing information during the design of a real-time system. Thus this research aim is as challenging as it is we believe worthwhile.

The record of the meta-level operations performed during system development on the dependability information will provide a trace of the dependability-related decisions and actions taken during design, so resulting in the construction of a "dependability case" that could be used to support the design work, provide traceable evidence regarding the dependability of the system, or support a formal safety or security case, prior to system deployment. Any subsequent system adaptation will be equally well informed regarding the impact of any proposed subsequent changes to the system.

A separate but related goal should be to create, and provide tool support for, a process model (i.e. a set of rules governing the way in which the overall development process is carried out) which ensures that all four dependability technologies (fault prevention, removal, tolerance and forecasting) are properly employed throughout system development – i.e. that the SDS is itself well-designed to be highly dependable [Kaâniche, Laprie et al. 2002]. This would be done by (i) considering the activities related to the four dependability technologies as explicitly forming four distinct fundamental processes, within a comprehensive development model, (ii) providing a tool environment which guides and constrains the interactions among these processes, and with all the other processes (e.g. requirements definition, specification, implementation, etc.) which make up the overall development process.

Achievement of these two goals taken together would turn the process of developing complex computer systems that meet given dependability requirements into a one that, via its explicit concern for dependability, is much more akin to a true engineering process than it is at present.

2.1.2 Cost-Effective Formal Methods

The use of formal methods tends to be confined to safety critical applications. We have already argued above that this need not be the case. But it is clear that any development process (SDS) which creates a poor system and then tries to improve the quality either by testing or by formally based approaches is doomed to discard much work: scrap-and-rework is the productivity drain of large system design. (Somehow the temptation to believe that things can be put right later is greater with software than elsewhere in engineering because there is no physical artefact going into the bin.)

The crucial step towards cost-effective use of formal methods is to use them throughout the development process. (They can be of particular value in developing robust fault tolerance mechanisms.) There is a significant challenge here for tool builders in that support is needed that keeps track of the entire development process and can support changes to requirements (in the sense that the impact of changes is clear and that unaffected parts of a system do not need to be revalidated).

Another key tools challenge is for common interfaces so that tools themselves can co-exist. There are also here some harder research problems (which are being addressed) of finding ways to use design abstractions to cut down model checking and testing. The argument that testing cannot prove that a complex system is correct is well known but does not argue against any use of testing. In fact, running a single test case establishes a result about a class of executions; the missing argument is about the size

of that class. What is really required is means of combining arguments from testing and from formal reasoning ([Bernot, Gaudel et al. 1991], [Marre 1999]). Earlier research on "symbolic execution", "abstract interpretation", and "partial evaluation" is all of relevance to this aim, but there is a clear and urgent need for more research on a theory of testing.

It is clear that ensuring the effective use of formal methods, and of the tools that support them, involves socio-technical as well as technical research.

In the area of concurrent and real-time systems there is a research challenge to find adequately compositional approaches. This requires a full awareness of *interference*, which is what makes compositionality difficult to achieve with concurrency. It is also clear that a full study of mobility is required to provide an underpinning of the sort of ubiquitous computing that is envisaged above.

Furthermore, most interesting security issues are non-compositional and this poses additional questions for research, alongside those caused by the problems of information flow analysis and control in a world of open evolving networks of mobile as well as fixed devices. It is all too obvious that any system that is not closed to outside signals will attract the unwelcome attention of malicious attacks. Even the WWW is plagued with worms, viruses, etc. The sort of systems described above which are built from dynamic coalitions will need to be much better architected than the WWW if malicious attacks are not to render them unusable.

2.1.3 Architecture Theory

Systems architecture concerns the way in which a system is constructed out of appropriately-specified components, and the means of component interconnection to be employed. A well chosen system architecture will result in a cost-effective and high-quality product. Architectural-level reasoning about dependability has the merit of counteracting the situation in which system dependability concerns currently tend to be left until the later stages of system design, and to result in solutions that are hidden within the detailed system design and implementation. There is now an initial body of work on architecting dependable systems, and on tool support for specifying, analysing and constructing systems from existing components, building in part on the notion of architecture description languages [Shaw and Garlan 1996] as a contribution to ways of classifying systems designs. This now encompasses research on, for example, the dependability of SoSs based on web services [Tartanoglu, Issarny et al. 2003], and on providing means of tolerating intrusions [Veríssimo, Neves et al. 2003]. It must also be recognised that the architecture of a system might have to reflect the need to *assess* the system dependability as well as to maximise that dependability.

It is imperative that such research makes further progress because of the importance of the overall architecture in the understandability and fault-containment properties of a system, whether the resulting dependability mechanisms are integral across an entire architecture, or contained within particular architectural components. However, though the structure even of the technical part of a system should be designed carefully from the earliest stages of design, this is certainly not an argument for pure top-down design. Indeed, one of the key issues with architecture concerns the need for it to evolve as requirements change, not just to facilitate detailed changes that leave the architecture intact [Jones, Periorellis et al. 2003].

2.1.4 Adaptability

There is a danger of misinterpreting the previous three sections as implying that the problems of trustworthy pervasive systems are those of how to create systems *de novo* that can be trusted to meet a pre-specified set of new dependability requirements. In some cases this will indeed be what is needed – though incremental procurement may well often be the best way to obtain such systems, so that system adaptability will be an important issue even in such circumstances. However, the more common problem, especially as far as very large and complex systems are concerned, is that the new system will have to be built starting from a situation in which there already exist various systems that are being depended upon, more-or-less successfully, by various sets of users whose needs will continue to have to be met during any changeover period. (One can draw an analogy to the road construction world – traffic has to continue to flow somehow, even while major road construction projects are being undertaken, and wherever existing facilities still have a useful role to play they have to be preserved and remain usable if at all possible.)

The problems of (i) making use of pre-existing designs, and more specifically, dependability cases (in particular existing validation and verification data), in producing a new variant of an existing system, and (ii) those of dependably changing over to the use of the new system, possibly without significant

interruption of service, are thus of great significance. Ideally, the required adaptations will be confined to the internal activity of particular system components. In practice, there will often be need to adapt exist interfaces, and to do so dependably – a much more difficult problem [Jones, Periorellis et al. 2003]. However, the extreme case of adaptability is the system that evolves automatically in response to changes in its environment. Such a property will become of increasing importance in the dynamic world of pervasive systems involving large numbers of mobile component systems – and maintaining dependability during such evolution will be a challenging research goal.

Realistically, adaptation and evolution activities will at times fail, so a further challenge is to integrate system change and exception handling facilities, so as to facilitate recovery following such failures, both internal to the system and, when necessary, involving the human activities into which the system is integrated.

A particular class of evolutionary activity concerns fault removal. Fault removal can be undertaken either before a system is deployed, or when faults are identified during system operation. It has typically been regarded as an at least partly manual activity. However, there are now research programmes aimed at automating the identification and even repair of faults, indeed in operational systems – i.e. of building what are sometimes called "self-healing" or "autonomic" systems. The extent to which these aims are practical remains to be determined.

2.1.5 Building the Right (Computer-Based) System

Even if it were the case that we routinely built programs that perfectly satisfy their specifications,. the difficulty often remains of getting the specification itself right. Risk management strategies constitute one approach to this issue – these involve assessments (albeit using rather coarse scales) of the likelihood and severity of the consequences of system failure, and of the effectiveness of means by which such failures might be tolerated, and so provide decision criteria that can be applied to many dependability requirements. This approach forms the basis for many security standards.

Similarly, there are major research challenges to face in order to construct dependable CBSs. Notions of, for example, "trust" and "responsibility" which inform the day-to-day operation of an environment such as a hospital or a government department are elusive and must be fully understood and recorded if a supportive system is to be specified. Furthermore, one must be able to reason about the activities of parties who might wish to disrupt a new system. The EPSRC Interdisciplinary Research Collaboration on the Dependability of Computer-Based Systems (DIRC) [16] is active in this research area but much more needs to be done.

Today, most computer systems are deeply embedded within groups of humans. What the Moore's law reduction in hardware cost and size has given is ubiquity and closeness to the users. Nearly all professionals, and a huge percentage of manual workers, could not do their job without computers. But many of these computer systems are ill thought out and difficult to use. In many cases, the system is designed with scant attention to –or understanding of– the needs and capabilities of the user. For example, in the highly professional contexts that DIRC has been investigating (e.g. the NHS or NATS) there are crucial relationships of trust and responsibility that must be fully understood if a system is ever to be deployed successfully.

There has been a considerable amount of work (dating back to Enid Mumford's early "participative design" approach [Mumford and Henshall 1979] and including the more careful applications of UML "use cases") on understanding the needs of users but there is still a considerable research gap in finding ways of describing notions such as trust and responsibility. These fundamental notions are of particular importance when planning a system which will change the way people work: one cannot just infer the specification from current practices but nor should one ignore the key functions and relationships of the people whose work will be affected by a new system.

There appears to be an inevitable feedback loop that the more dependable computing systems become, the more *dependence* is placed on them, i.e., the more they will be trusted. (Incidentally, the more dependable a major system is, the more any eventual failure is an emergency situation – and trust, which typically will have been built up gradually, will be lost abruptly.) Similarly, when for whatever reason –perhaps a subconscious guesstimate of the likely frequency or cost to them personally of failures, a run of bad publicity, etc.– a person or whole population is unwilling to place significant trust in a system, the use they are willing to make if it, given any choice in the matter, may well be far from

---

[16] http:// www.dirc.org.uk

sufficient to justify the cost incurred in constructing the system. There is thus a need for inter-disciplinary work on how (appropriate) levels of trust can be built up, nurtured, and when necessary re-gained. Of evident close relevance to this is the whole subject of risk assessment and management – research is needed on methods for user-oriented risk assessments that address regulatory, legal and liability issues, as well as direct system requirements. A closely-related point is that consideration of the wider system context is typically essential in order to determine an appropriate system specification – to identify what demands the environment will place on the system, and what constraints might need to be exercised, and strategies followed, outside the system – in order to achieve a satisfactory level of overall dependability. For example, excessive demand for a system that is a shared resource may not be controllable by the system itself, but instead have to be controlled outside the system. If this demand originates with people, the wider context together with the computer system can be regarded as a CBS whose overall dependability is at stake, again implying the need for a socio-technical approach to the problem.

2.2 The Overall Aims

The research agenda summarised in the previous section aims not just to identify a set of priority technical dependability research topics. Rather –and more importantly– it tries to ensure that the research which is carried out will be *effective*. By this we mean that it will relate well to the realities of government and industry as well as to evident technological trends, and to the realities of systems in which human beings are frequently as important a part of system dependability problems (and their solutions) as are technical matters concerning hardware, software and communications.

We do not suggest that the task of developing and deploying trustworthy pervasive systems can be one that becomes entirely guided by formal and quantitative considerations, even when supported by tools that cope effectively with requirements for system adaptability. However, we do feel that it is possible that enough progress can be made so that conventional business practices and market forces start to play a dominant and effective role in ensuring that governments, industry and society at large are provided with systems, of the required size and functionality, whose dependability matches the dependence that has to be placed on them. Therefore, rather than provide a set of specific and rather arbitrary quantitative goals for various dependability-related research targets, we suggest that it is more appropriate to have as an overall goal:

> the generation of dependability techniques and tools which, when supported by appropriate policy and training initiatives, enable the UK software and systems industry involved in creating complex networked systems to offer *warranted* software and services, even for pervasive CBSs and SoSs.

Complementary goals are then those of trying to ensure that:

> dependability research is pursued in such a manner as to take full cognizance of the consequences of the human involvement, both as (accidental and malevolent) sources of dependability problems, and of dependability solutions, in CBSs and SDSs

and that:

> in such research, whether aimed at the problems of developing or of deploying complex pervasive systems, fault prevention, fault removal and fault tolerance are regarded as complementary strategies, whose effective combination is crucial, and fault forecasting is used to evaluate the degree of success that is being achieved.

3 OPPORTUNITIES FOR CROSS-DISCIPLINARITY

Quoting again from the AMSD Overall Dependability Roadmap:

> "There is a need for interdisciplinarity among the various technical communities whose work - whether they realize it or not - is highly related to overall system dependability. Many of the technical disciplines focus very narrowly on particular types/levels of systems, dependability attributes, types of fault, and means for achieving dependability. For example, only in recent years has a (small part) of the security community started acknowledging the importance of fault tolerance. In addition to this technical-interdisciplinarity there is a need for multi-disciplinarity in the sense of socio-technical interdisciplinarity."

The need for such cross-disciplinary research and activity in the area of achieving dependable systems is already clear today when one considers major applications such as those envisaged in medical records, electronically mediated voting or defence systems involving military personnel working in complex coalitions, for example. Furthermore, our national infrastructure is at risk if we fail to take an interdisciplinary view of the capabilities of potential attackers.

Serious though these issues are today, we face even larger challenges in the immediate future. We are yet again faced with the impact of the observation that, as dependability increases, society increases its dependence on the technology. Furthermore, the continued progress along the line of Moore's Law is making it possible to deploy huge numbers of computing devices which will themselves form coalitions. But these virtuous coalitions will face attack from (large numbers of) computation units launched by malicious forces.

We need interdisciplinary teams to design systems, and to study their development processes. We need to find ways to facilitate effective collaboration within such teams. There is a need to harness at least sociologists, psychologists and economists, for example by building on the efforts undertaken some years ago by the ESRC's "Programme on Information and communication Technologies" (PICT) [Dutton and Peltu 1996], to promote such inter-disciplinarity, and now the work being pursued explicitly in the dependability arena by the current EPSRC Interdisciplinary Research Collaboration on the Dependability of Computer-Based Systems (DIRC), referred to earlier.

REFERENCES

[AMSD Roadmap 2003] AMSD Roadmap. *A Dependability Roadmap for the Information Society in Europe,* Accompanying Measure on System Dependability (AMSD) IST-2001-37553 - Work-package 1: Overall Dependability Road-mapping. Deliverable D1.1., 2003. http://www.am-sd.org

[Anderson 1994] R. Anderson, "Why Cryptosystems Fail," *Communications of the ACM*, vol. 37, no. 11, pp.32-40, 1994. [An extended version - http://www.cl.cam.ac.uk/users/rja14/wcf.html (1999)]

[Anderson 1999] R. Anderson. "How to Cheat at the Lottery (or, Massively Parallel Requirements Engineering)," in *Proc. Computer Security Applications Conference (ACSAC'99),* Phoenix, AZ, IEEE Computer Society Press, 1999.

[Anderson, Feng et al. 2003] T. Anderson, M. Feng et al. "Protective Wrapper Development: A Case Study," in *Proc. of COTS-Based Software Systems Conference (ICCBSS 2003), Ottawa, Canada, February 2003, LNCS 2580,* pp. 1-15, 2003.

[Avizienis and He 1999] A. Avizienis and Y. He. "Microprocessor Entomology: A Taxonomy of Design Faults in COTS Microprocessors," in *Proc. Dependable Computing for Critical Applications (DCCA '99),* pp. 3-23, San Jose, CA, 1999. ISBN 0-7695-0284-9

[Bernot, Gaudel et al. 1991] G. Bernot, M.-C. Gaudel et al, "Software Testing Based on Formal Specifications: A Theory and a Tool," *Software Engineering Journal*, vol. 6, no. 6, pp.387-405, 1991.

[Bloomfield and Littlewood 2003] R.E. Bloomfield and B. Littlewood. "Multi-Legged Arguments: The Impact of Diversity upon Confidence in Dependability Arguments," in *Proc. Dependable Systems and Networks,* pp. 25-34, San Francisco CA, IEEE Computer Society Press, 2003. ISBN 0-7695-1952-0

[Clarke and Wing 1996] E.M. Clarke and J.M. Wing, "Formal Methods: State of the Art and Future Directions," *ACM Computing Surveys*, vol. 28, no. 4, pp.626-643, 1996.

[Dobson and Randell 1986] J.E. Dobson and B. Randell. "Building Reliable Secure Systems out of Unreliable Insecure Components," in *Proc. Conf. on Security and Privacy,* Oakland, IEEE Computer Society Press, 1986. http://www.cs.ncl.ac.uk/research/pubs/inproceedings/papers/355.pdf

[Dutton and Peltu 1996] W.H. Dutton and M. Peltu, (Ed.). *Information and Communication Technologies: Visions and Realities,* Oxford, Oxford University Press, 1996, 484 p. ISBN: 0198774591

[Estrin 2001] D.L. Estrin, (Ed.). *Embedded, Everywhere: A Research Agenda for Networked Systems of Embedded Computers,* Washington D.C., Computer Science and Technology Board, National Academy of Science, 2001, 236 p. ISBN: 0-309-07568-8 http://www7.nationalacademies.org/cstb/pub_embedded.html

[European Commission 2002] European Commission. *e-Europe 2005 Action Plan: An Information Society for All,* COM(2002) 263 final, European Commission, 2002. http://europa.eu.int/information_society/eeurope/news_library/documents/eeurope2005/eeurope2005_en.pdf

[Ghemawat, Gobioff et al. 2003] S. Ghemawat, H. Gobioff et al. "The Google File System," in *Symposium on Operating System Principles (SOSP-203),* ACM, 2003. http://www.cs.rochester.edu/sosp2003/papers/p125-ghemawat.pdf

[Guiho and Hennebert 1990] G. Guiho and C. Hennebert. "SACEM Software Validation," in *Proc. 12th International Conference on Software Engineering,* pp. 186-191, IEEE Computer Society Press, 1990.

[Hoare 2002] C.A.R. Hoare. "Assertions in Programming: From Scientific Theory to Engineering Practice (Keynote Speech)," in *Proc Soft-Ware 2002: Computing in an Imperfect World, First*

International Conference, Soft-Ware 2002 (Lecture Notes in Computer Science 2311), pp. 350-351, Belfast, Northern Ireland, Springer, 2002. ISBN 3-540-43481-X

[Jones, Periorellis et al. 2003] C. Jones, P. Periorellis et al. "Structured Handling of On-Line Interface Upgrades in Integrating Dependable Systems of Systems," in *Proc. Scientific Engineering for Distributed Java Applications International Workshop (FIDJI 2002),* pp. 73-86, Luxembourg, Springer Verlag, 2003.

[Kaâniche, Laprie et al. 2002] M. Kaâniche, J.-C. Laprie et al, "A Framework for Dependability Engineering of Critical Computing Systems," *Safety Science*, vol. 40, no. 9, pp.731-752, 2002.

[Kemp 1990] D.H. Kemp, "Technical Background on AT&T's Network Slowdown, January 15, 1990," *The Risks Digest*, vol. 9, no. 63,1990. http://catless.ncl.ac.uk/Risks/9.63.html#subj3

[Laprie 1991] J.C. Laprie, (Ed.). *Dependability: Basic Concepts and Associated Terminology,* Dependable Computing and Fault-Tolerant Systems. Springer-Verlag, 1991. [(Contributors: T. Anderson, A. Avizienis, W.C. Carter, A. Costes, F. Cristian, Y. Koga, H. Kopetz, J.H. Lala, J.C. Laprie, J.F. Meyer, B. Randell, A.S. Robinson, L. Simoncini, U. Voges.)]

[Laprie 1999] J.C. Laprie. "Dependability of Software-Based Critical Systems," in *Dependable Network Computing, ed. D. R. Avresky*, Kluwer Academic Publishers, 1999.

[Littlewood, Bloomfield et al. 2003] B. Littlewood, R. Bloomfield et al. "Trends in System Evaluation (Vol. 3, Appendix F)," in *A Dependability Roadmap for the Information Society in Europe,* pp. 90-93, Accompanying Measure on System Dependability (AMSD) IST-2001-37553, 2003. http://www.am-sd.org

[Marre 1999] B. Marre. "Symbolic Techniques for Test Data Selection from Formal Specifications," in *Formal Methods 99,* Toulouse, Springer Verlag, 1999.

[Mumford and Henshall 1979] E. Mumford and D. Henshall. *A Participative Approach to Computer Systems Design: A case study of the introduction of a new computer system,* London, Associated Business Press, 1979, 191 p.

[Narayanan and McIlraith 2002] S. Narayanan and S.A. McIlraith. "Simulation, Verification and Automated Composition of Web services," in *Proc of WWW'2002,* 2002.

[Pearce 2003] S. Pearce. *Government IT Projects,* Report 200, Parliamentary Office of Science and Technology, 7 Millbank, London, 2003.

[Powell, Adelsbach et al. 2001] D. Powell, A. Adelsbach et al. "MAFTIA (Malicious- and Accidental-Fault Tolerance for Internet Applications)," in *Supplement of the 2001 Int. Conf. on Dependable Systems and Networks,* pp. D32-D35, Göteborg, Sweden, IEEE Computer Society Press, 2001.

[Randell 2000] B. Randell, "Facing up to Faults (Turing Memorial Lecture)," *Computer Journal*, vol. 43, no. 2, pp.95-106, 2000. http://www.cs.ncl.ac.uk/research/pubs/articles/papers/245.pdf

[Rushby 2000] J. Rushby. "Disappearing Formal Methods," in *Proc. High-Assurance Systems Engineering Symposium (HASE-5),* pp. 95-96, Albuquerque New Mexico, ACM, 2000.

[Schneider 1999] F.B. Schneider, (Ed.). *Trust in Cyberspace: Report of the Committee on Information Systems Trustworthiness, Computer Science and Telecommunications Board, Commission on Physical Sciences, Mathematics, and Applications, National Research Council,* Washington, D.C., National Academy Press, 1999, 332 p. [Full text available at: http://www.nap.edu/readingroom/books/trust/]

[Schneier 2000] B. Schneier. *Secrets and Lies: Digital Security in a Networked World,* John Wiley & Sons, 2000, 412 p.

[Shaw and Garlan 1996] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*, Prentice Hall, 1996, 242 p. ISBN 0-13-182957-2

[Tartanoglu, Issarny et al. 2003] F. Tartanoglu, V. Issarny et al. "Dependability in the Web Services Architecture," in *Architecting Dependable Systems, LNCS-2677, ed. R. d. Lemos, C. Gacek and A. Romanovsky*, pp. 89-108, Springer, 2003. http://www-rocq.inria.fr/arles/doc/doc.html

[Van Campenhout, Mudge et al. 2000] D. Van Campenhout, T. Mudge et al, "Collection and Analysis of Microprocessor Design Errors," *IEEE Design &Test*, vol. 17, no. 4, pp.51-60, 2000.

[van Lamsweerde 2000] A. van Lamsweerde. "Formal Specification: A Roadmap," in *Proc. Conf. on The Future of Software Engineering,* pp. 147 - 159, Limerick, Ireland, ACM Press, 2000. ISBN:1-58113-253-0

[Veríssimo, Neves et al. 2003] P. Veríssimo, N.F. Neves et al. "Intrusion-Tolerant Architectures: Concepts and Design," in *Architecting Dependable Systems, LNCS-2677, ed. R. d. Lemos, C. Gacek and A. Romanovsky*, pp. 4-36, Springer, 2003. http://www-rocq.inria.fr/arles/doc/doc.html

[Yeh 1999] Y.C.B. Yeh. "Design Considerations in Boeing 777 Fly-By-Wire Computers," in *Proc. 3rd IEEE International High-Assurance Systems Engineering Symposium,* pp. 64, Washington DC, 1999.