DRAFT 04/04/03

Theories for Ubiquitous Processes and Data

Platform for a 15-year Grand Challenge

OVERVIEW This paper is written as background for a proposed Grand Challenge in Computing Research, entitled **Science for Ubiquitous Computing**. The paper has a specific purpose: to trace the evolution of theoretical models for distributed and mobile computing, and the mutual influence between these models on the one hand and software design and development on the other. It also discusses many concepts that are necessary for understanding global and ubiquitous computing. Some predictions are given of what may be achieved towards the Grand Challenge within the short term.

We aim in this paper to establish the case that a theoretical approach is essential. We do not here explore ways in which research community should organise itself to meet the Challenge; but it is clear to us that it can only be met by a close interplay between theoretical work and practical experimentation.

more on this phase

1 Models for distributed computing

Robin Milner

In this section we trace some theoretical ideas that have become well-established in computer science over the past fifty years. These ideas begin with stand-alone computation, and develop into a concepts for concurrent and interactive computation – which also embraces communication. As we proceed towards ubiquitous computing more and more concepts become relevant, and demand precise definition. At the end of the section we list several of these concepts; we then explore them individually in the succeeding sections.

1.1 The separation of models from software

Models of computation preceded programming. A Turing machine [152] (called a 'paper machine' by Turing) modelled the way a human performs well-defined calculational procedures; automata theory arose from a study by McCulloch and Pitts of neuronal activity; Church's lambda calculus modelled the evaluation of a mathematical function represented by an applicative expression. Programming arises via Turing's observation that a computing model, even a paper machine, can be *universal*; loading a program is no more or less than the presentation of the *description* of a specific paper machine to a universal one, so that the latter can 'enact' the former. Thus stored-program computers, such as Wilkes's EDSAC, are engineering realisations of the conception of a universal computing machine.

Despite these beginnings, programming has outstripped computational models. Software technology has been expanded and adjusted to meet each advance in hardware technology, more recently including network technology. Loss of tight connection between models and programming has been inevitable. On the one hand, the market for software has been so demanding that no software company could afford the delay caused by basing their software production process upon scientific analysis; on the other hand, scientific models of computing have not kept up with opportunities afforded by the enormous increase in computing power.

The resulting separation between software practice and science has had ugly consequences. We are faced with mountains of inscrutable legacy software. More insidiously, the habit of this separation is deeply ingrained in the software engineering process, which has even come to be treated as a challenge to management rather than to science. Superficially, it may appear hard to recover from these consequences, and from the separation that caused them. But our Grand Challenge is founded on the argument that the separation need not end in divorce, and indeed must not do so if we are to retain informed control over our computing artefacts.

Here we trace some of the positive – though sporadic – interplay between models and software over the past four decades, revealing a trend that justifies the Grand Challenge we are putting forward: nothing less than to reunite software and models so that, from having deployed divergent formalisms, they come to deal in one and the same repertoire of ideas and modes of expression.

We do not find a steady pattern in the past interplay between models and software. But we can detect three rather distinct phases (overlapping in time): the strongly developed models for sequential, stand-alone computation; the onset of new models for interactive computing; and the ramification of concepts essential to understanding ubiquitous computing phenomena. We now trace these individually.

1.2 The science base for stand-alone computing

The first prominent feature is the steady innovative flow of models informing the meaning and structure of programs and data. The list is long, and we only select a few high spots as illustration. From the 1950s, Automata theory, originating as a model of brain activity [McCulloch-Pitts], together with a formal language hierarchy, has helped understand the syntax of programming. In the same decade the lambda calculus[Church], dating from the 1940s, fed into McCarthy's LISP [114], and later predicate calculus fed into logic programming [Colmerauer-Kowalski]; these are two rare instances of a programming language founded closely on a logical formalism. Each of them, in response to the demand for efficient implementation, then burst the bounds of its parent formalism; this accurately illustrates the tensions between, on the one hand, science seeking tractable concepts, and on the other hand engineering practice in response to application demands.

+ data bases

Turning to data, the relational database model [52] in the 1960s led to an upsurge in ways to organise and manipulate large masses of data, and also to a tradition of special programming languages to access such databases [Buneman]. The mutual enrichment of concepts of data and programming is a prominent feature of our Grand Challenge. Meanwhile, in the same decade, a search was re-mounted for a mathematical foundation for programming [Strachey, Landin], and new languages inspired by it (one of them, CPL, led eventually to the modern language C, though lost some principled structure on the way). This in turn inspired the mathematical concept of *domains* [Scott,Plotkin], providing an information-based theory of computational structures that continues to develop and enlighten later languages.

2002 ?

In the same period, but reaching prominence later, there was increasing concern to model programs not only abstractly, but in terms of what laws they obey while running. This took more at least two forms, both based upon logic; *program logics* [Floyd, Dijkstra][90] with greater emphasis on analysing how particular programs behave, and *operational semantics* [134, 120] [Kahn] with greater emphasis on defining the meaning of a whole language as a guide to implementers. Running alongside this development was an increasing awareness of *data types* (and higher types as well) [Hoare2, Reynolds, MacQueen] as a tool both for defining languages and for analysing program behaviour. Simultaneously, computer-based reasoning tools [Good, Gordon][144] were designed to assist this analysis.

All of these studies are now seen as an inevitable and central development of the science of computing. They are not just about programming; they constitute the first detailed models of discrete dynamical systems, of which computer programs are a leading ecample. Why is their impact on engineering practice delayed and incomplete? A sufficient answer is the force of the market; the demand for languages and systems has been too urgent to allow scientifically informed design.

parametric polymorphism is a success story

1.3 Models of concurrency and interaction

The models discussed above are largely concerned with what happens in a single sequential process, such as the execution of (say) a program written in Fortran. But increasingly it became important to model the way in which such sequential systems, running simultaneously, might interact. Surprisingly, although neuronal activity in the brain is massively concurrent, the classical theory of automata that it prompted did not model this interaction adequately. The well-known Crone-Rhodes [Crone-Rhodes] decomposition theorem showed that any finite automaton can be decomposed into a number of cooperating ones, but they cooperate only in pipelining fashion.

Among early models of truly interactive behaviour, Petri nets [130] – invented in the 1960s – are a conceptual landmark. A simple Petri net models the interaction between two automata by requiring them to *synchronise* certain of their transitions. Further, Petri used the nets to model not only interacting computers, but real everyday concurrent processes such as office systems. It became clear that computer science would contribute modelling concepts across a broad spectrum of human and mechanical activity. In the late 1970s came compositional process calculi such as CSP, CCS and ACP [91, 117, 25], with semantic models originally inspired by the sequential models already discussed. Around the same time came new logics, such as temporal logic [Pnueli], specifically concerned with interactive behaviour. These models contribute not only to computer science but also to mathematics and logic.

BPEL

Matching the sequential development, computer-assisted tools emerged to analyse the behaviour of concurrent systems. A rich tool-kit grew up for Petri nets, exploiting their topographical nature. An essential development was the arrival of model-checking [49]; it permits a fully automated analysis of whether a finite state system satisfies a certain property expressed (say) in temporal logic.

A missing ingredient in the earlier models of interaction was *mobility*. The interacting components of a complex system often have a fixed topography, but equally often they do not; indeed, the very effect of a component's interaction can be that it gains new neighbours for further interaction. This lack was addressed around 1990; new models, exemplified by the π-calculus [119], introduced the necessary extra theory. A large new domain of modelling became accessible, including systems interacting over the internet. Experimental programming languages and analytical tools were extended to match [PICT, MobilityWB].

More

As in the case of sequential computing, models of interaction have influenced widely-used programming languages, notably Ada and Java, but only incompletely. This is partly due to the habituated separation of software from science; it is also due to need, realised perhaps mostly in the last decade, for the many new or imperfectly understood concepts that must underpin both the design and the modelling seriously distributed – now called ubiquitous – computation. To these we now turn.

1.4 Conceptual ramification

All computational models are concerned both with *what* is computed and with *how* it is computed. In concurrent systems, the emphasis shifts towards the latter; theories of concurrency describe the degree of parallelism in a computation. Further, in distributed systems, we become concerned with *where* things happen, i.e. localities; in mobile computing, with *movement* among those localities; and so on.

Before extending this list, we note that *locality* and *movement* for concurrent processes have long been studied. In a comprehensive survey, Castellani [42] distinguishes between *abstract* localities, those introduced (somewhat like types) to assist the understanding of processes, and *concrete* localities that can model their actual movement. *Mobile ambients*, introduced by Cardelli and Gordon [39], are an important step forward in the latter respect; they model movement between *regions* (which may be physical or administrative), complementing the mobile *connectivity* modelled by the π-calculus. These two forms of movement are generalised in graphical reactive systems [73, 118], where arbitrary reconfigurations of may occur.

Localities and mobility well illustrate the shift from *specifying a program*, i.e. saying what it should do, to *modelling a phenomenon*, whether natural or artificial. The challenge to understand ubiquitous computing arises because the variety of relevant phenomena is large, and not fully explored. Nonetheless, many such phenomena have been identified. Here are several of them:

Why

Security The management of privacy, both of data and of communication;

Boundaries, Resources, Trust The discipline of access to resources (including space, time, hardware and services) and the criteria for authorising such access;

Distributed data multiply located databases, semi-structured data, provenance of data;

Game semantics Understanding agent-environment interaction as a strategy;

Hybrid systems The modelling of continuous variables (including space, time) in a system that is part computational, part real;

Stochastics Probabilistic modelling: quantifying uncertainty, evaluating performance;

Model-checking automatic verification of a system against a logical specification, via state-space search.

(The list cannot be exhaustive.) In the following sections we treat these concepts individually. For each one we indicate its relevance to ubiquitous computing and outline what has been achieved hitherto. We also discuss what may be achieved within a few years; we regard these as steps that are both predictable and necessary to meet the Grand Challenge.

2 Security

Andrew Gordon

At least a decade ago, Marc Weiser of Xerox PARC predicted that sometime before the beginning of this century we would enter a third wave of computing, *Ubiquitous Computing* [158], in which each person has many computers at their disposal, at work, at play, and in the home, with form factors receding from the foreground into the background. This would reverse the situation during the first wave, the mainframe era, and take the number of the world's computers well beyond the second wave, the era of the one-to-one relationship between person and desktop PC.

Today, in fact, we still use mainframes, though these days we tend to call them websites. We are grateful that the ones on the public internet mostly allow us to login anonymously (well, as anonymous as an IP address) though sometimes they don't and then we have a hard time tracking all those unguessable passwords we have to think up. The ones on private intranets provide work-related services like tracking customer relationships, arranging travel and having it authorised, and so on; we need work-related credentials to login to these services, so they are inconvenient to access from anywhere but work, and it is next to impossible to access services on other intranet, such as those of a business partner.

Today, we still use PCs, but they no longer sit merely on every office desktop, but sit in the study at home and beside the TV, with an old model in the guest room, and they are around us in other form factors like laptops and tablets. Dialup connections seem so Last Century; broadband is finally taking off, and institutional and home wireless networks are a huge market. We no longer care much about viruses spreading via boot sectors of floppies. Instead, we worry about attacks over the internet, via macros in emails or via buffer over-run attacks on protocol stacks, and so on.

Additionally, many of us use an array of other kinds of computerized gadgets, such as mobile phones, digital cameras, PDAs, weather sensors, and so on, that increasingly interoperate with each other and with PCs via various kinds of wireless network. What used to be a burden mainly for professional IT staff, the task of managing networks—installing patches, configuring firewalls, making backups—is rapidly becoming a burden for everyone, since all of us are running networks.

In summary, ubiquitous computing has yet to emerge quite as Weiser predicted, but clearly computers have tended to be smaller, more interconnected, and more numerous. Mainframes and PCs have not gone away. Meanwhile, the security landscape has shifted; some old threats (e.g., viruses on floppy discs) are receding, new threats have arisen (e.g., macros in email), and future threats loom (e.g., unprotected home wireless networks).

2.1 State of the art

In parallel with the trend towards ubiquity, informal principles of security engineering are emerging from more than thirty years of experience and are being codified in books and articles [2, 19, 145]. To risk oversimplification, security engineering should be based on analysing the threats—physical, human, legal, and technological—faced by a system, weighing up the costs of prevention, detection, and recovery, and hence adopting explicit stances—policies—towards each threat. It has become a truism of computer security that we need to consider security at the level of systems, rather than focusing just on particular mechanisms. We need to articulate clearcut policies, and find suitable underlying mechanisms, and continue to monitor their effectiveness after deployment.

Given this background—the gradual influx of ubiquitous computing, and the gradual distillation of security experience into informal engineering principles—the purpose of this note is to identify some areas where theoretical computer science has made and can make useful contributions. Inevitably, this note omits many important problems and important theoretical work, but I hope it will nonetheless be a useful starting point for discussing challenging future directions.

2.2 Buffer Overruns

Buffer overruns are a dreadful source of attacks on networked PCs; currently, a large proportion of software patches concerns buffer overruns. A buffer overrun occurs when input data is not properly validated, and hence overwrites memory beyond the input buffer. Hence, it can be possible for an attacker to construct an input that deposits runnable code in a victim computer's memory, in such a way as to gain control of the machine.

The programming language community has long advocated memory safe languages, like ML or Java, as a solution. There has been some progress in their adoption, but hardly an overwhelming wave. Nonetheless, there is going to be an enormous legacy of critical computing infrastructure (operating systems, network stacks, browsers) written in unsafe languages—largely C—for the forseeable future.

There are two competing responses. One is to build tools to find defects such as buffer overruns in existing C codebases. Microsoft uses Prefix [133] to search for defects in Windows. Engler's group at Stanford uses metacompilation [80] to find defects in the Linux kernel. Wagner and others at Berkeley have built tools for finding various code defects [156, 147] in C. Necula's CCured [126] uses a combination of static analysis and runtime checks to find memory bugs (such as buffer overruns) in ANSI C programs.

The alternative response is to design safe languages that are so close to C that large existing codebases may be ported with only minor modifications. Vault [64] from Microsoft Research and Cyclone [99] from AT&T Research and Cornell are examples.

2.3 Security Protocols

The BAN logic for cryptographic security protocols has been hugely influential [34]; it is one of the most cited articles in computer science. BAN was not the first formalism for security protocols, guarantees very little, and indeed never had a completely satisfactory semantics. Nonetheless, the paper stimulated a wide range of theoreticians to consider a formal problem properly credited to Dolev and Yao [65]: given that the opponent can monitor and replay network traffic, can encrypt and decrypt messages if it knows the key material, but cannot simply guess unknown keys, can compliant principals still enjoy guarantees such as message integrity, authenticity, and confidentiality?

By the mid-90s, several solutions had emerged, along with a trade-off: finite-state techniques such as model-checking [110] could find defects automatically, but could not guarantee the absence of Dolev-Yao attacks; symbolic techniques such as theorem-proving [129] or bisimulation proofs [1] could show the absence of such attacks, but were rather labour intensive. The race was on to eliminate this trade-off and by now we may say that the Dolev-Yao problem is largely solved; certainly, there is a range of tools [148, 53, 28] that with little or no human intervention can either verify or detect defects in suites of abstract protocol descriptions, such as the Clark-Jacob collection [46].

Still, in spite of this substantial progress, significant research questions remain. Properties such as privacy protection and resistance to denial-of-service attacks have risen in prominence, and need better theoretical support. Verification of the actual implementation code of protocols—rather than abstract descriptions—has rarely been achieved; verification of smartcard implementations, for example, should soon be within reach. Further connections between the formal logics applied to the Dolev-Yao problem, and the probability- and complexity-theoretic techniques emerging from the cryptography community could usefully be made [3].

2.4 Mobile Code and Mobile Agents

In the mid-90s, there was a lot of theoretical research on mobile code and mobile computation [41, 155, 85], inspired in part by the excitement around Java applets. Much effort was devoted to technologies such as bytecode verification [108] and proof-carrying code [125] that can protect a host from untrusted mobile code. There was pre- and post-Java work on mobile agents, envisaged as roaming the network, gathering information on behalf of the user. But there was no very compelling technology for protecting mobile code from untrusted hosts, that might rob an agent of its secrets.

Microsoft recently announced an initiative, originally codenamed "Palladium" [55, 20], to build hardware-based security features into PCs. These features could be exploited to provide safe havens for mobile agents, and so may give a new lease of life to the idea. Whether or not mobile agents make a come back, formal analyses of Palladium could help establish the trustworthiness of the platform, and may suggest improved APIs.

2.5 Network Management

How can organisations share their infrastructures, so that an employee A of company B can access a resource C belonging to another company D? Much basic theory has been developed over the last five or more years [104, 66, 143]. Still, little has been deployed, and the next step is implementation on top of technologies for e-science (Grid computing [124]), e-business (web services [95]), and the e-home (uPNP [67]). There is scope for theoretical work as the basis for tools such as firewalls, intrusion detection, and configuration analysis; a fine examplar is Guttman's theory of firewall configurations [79].

3 Boundaries, resources and trust

Vladimiro Sassone

Ubiquitous computing (UC) concerns (embedded) devices with limited capabilities moving between unknown, untrusted networks, adapting their behaviour in response to changes to their computational environment, using shared network resources according to given policies, and maintaining specific key services, and safety and robustness properties. All this requires specific safety provisions, some of which have been addressed in theoretical work. We review here some of the most relevant contributions.

3.1 State of the art

In this section we consider research centered around *boundaries*, *resources* and *trust* and based on work on *behavioural* models and *execution* structures, two interwoven streams which feed each other reciprocally.

Boundaries Traditional approaches to resource access security rely on operating system mechanisms which offer a fixed set of basic safety properties with little flexibility. A more satisfactory practice offers typing as a fundamental vehicle for the analysis and enforcement of access control. The development of type systems for process calculi originated with [131], and types have been used since to ensure type-consistent data exchanges on communication channels, and as prescriptions to control access to such channels and locations as, e.g., in [146, 101, 86].

1996
Pierce
Sangiorgi

The work on typing systems for ambient-based calculi has developed static analysis to control diverse behavioural properties. These include type systems that trace agent behaviour to gain control over ambient mobility, access and boundary crossing [38, 37, 116]. Properties of boundaries and their control have recently been formalised via spatial logics, such as the *ambient logic* [40], whose connectives and modalities speak of processes' spatial structure as well as behaviour.

Pict [132], the first programming language based on the primitives of the π-calculus, underlined the implementation difficulties involved with synchronous exchanges and nondeterministic composition, and contributed to a shift of focus towards asynchronous calculi [69, 86]. *JoCaml* [54] is a distributed implementation of the distributed join-calculus [70]; only purely local synchronisation mechanisms are present in such calculus, and this very limitation of synchronisation primitives stresses the distinction between 'local' and 'remote.' Implementing ambient mobility raises considerable difficulties, due to the presence of multiple, unsynchronised threads inside ambients. In [71] a distributed implementation of ambient migration is achieved via ingenious protocols of (asynchronous) synchronisation messages; *SAM* [153], a distributed abstract machine for SA, takes advantage of the type system of [107] to simplify the mechanisms for the execution of well-typed terms.

Resources Spanning from CPU cycles, to memory blocks, from networking services to database access, the notion of resource is truly pervasive in UC, where third-party resource usage is intrinsic to the paradigm. Relevant theoretical work includes [86, 127] which use systems of capabilities to control resource usage, and [159, 141, 84] where behavioural and dynamic types account for policies varying over time.

Space control has recently been the focus of important research. Crary *et al.* [57] use a typed intermediate language to control safe deallocation of memory regions. Hofmann [92] introduces a notion of resource type, which can be thought of as an abstract unit of space, and uses a linear type system to guarantee linear space consumption. Crary and Weirich [58] guarantee quantitative bounds on time usage using a dependently typed assembly language; [96] puts forward the first general formulation of resource usage analysis. Concerning ambient-based calculi, [76] develops a calculus of mobile resources which can be moved across locations provided suitable space is available at the target locations; [151] and [44] use static techniques respectively to control resource usage and to analyse the behaviour finite control processes. Regarding specification languages, the logic of *bunched implications*, BI [128], has been applied to notions of resource, most notably program pointers [142].

Typed assembly languages [121] are execution models embodying methods of explicit resource accounting, based on sophisticated dependent type systems. Current research is investigating provision of resource usage guarantees for mobile code; this gave rise to a virtual machine, *Grail* [111], together with a cost model for memory consumption and a bytecode logic [109] based on BI.

Trust Management Systems for trust management constitute essential parts of actual access and resource control systems. They define languages to express policies of authorisation to access resources, together with evaluation engines to grant or deny requests accordingly. Typical instances are described in [30, 51]. Most of the formal models in the literature are based on logics, as e.g. [140, 35], which can express notions such as '*belief*' and '*authority*' and can, therefore, be used to formulate trust policies.

A mathematical formalisation has been proposed in [157]. A principal's policy is modelled there as a function of the other principals' trust levels (delegation), and the actual trust levels it expresses can be obtained from the least fixed point of the collection of the individual policies. Current research aims at modelling *trust evolution* [36].

Existing execution models for trust, *trust engines*, provide essentially authentication services based on trusted key servers, protocols and system for public-key infrastructures. Among these, *PolicyMaker* [30], *KeyNote* [29], *Referee* [45].

3.2 Expected advances over the next three years.

Transient resource usage, whereby migrating programs access resources owned by others, lies at the very heart of UC. In a near future scenario, devices will have severely limited capabilities, and programmers will need to secure their applications' needs relying on services on remote execution environments.

Boundaries are very special entities in this respect: crossing them marks a transition to new *resources*, local to the new enclosing environment. Knowledge of one's own surroundings will be very limited in the global network. Agents and hosts will have to learn about each other dynamically, and this will have to rely on *trust mechanisms*:

trust will be the infrastructure underpinning decision taking in UC. Future research themes will then include:

- negotiation, monitoring and protection of *resource bounds*, whereby owners and users dynamically agree on transient resource allocations;
- *exploration* by migratory code of the largely unknown and untrusted neighbourhoods they inhabit and consequent *adaptive* behaviour.

These will rely on a notion of trust informed also by the principals' past histories. Below are some relevant open issues that we expect to be successfully tackled in the next 3/5 years.

Boundaries As it is not conceivable to endow processes with a static acquaintance with the global network, we need to formulate protocols to *acquire* and *manage* such knowledge and equip process with adequate such mechanisms. A form of *access negotiation* will be needed as a basic infrastructure for agent-based computational environments. Types for access control need a general mechanism to make an agent's type depend on the different contexts where it resides, so as to allow different types of exchanges at different locations, while preserving the safety of such interactions. Behavioural types can then be used to express and enforce policies depending on forms of process 'behaviours' or 'modes.' Foundational work on systems of capabilities is needed to express actions typical of UC (e.g. copy, move, upload), as well as primitives for their management (e.g. duplicate, forward, and revoke).

As for abstract machines, we need to focus on mechanisms for *communication* across boundaries and access *negotiation*, and on ensuring *failure resilience*.

Resources Upon crossing boundaries, migrating agents need to secure resources from the target environment. Protocols will be needed for host and agent to exchange information and requests, and engage in complex *negotiations* of *resource bounds*. This calls for foundational calculi which capture the essence of resource and their control, and underpin techniques for *usage analysis* and *quantitative* resource bounds. Spatial logics will serve to develop logic formalisms able to express properties of resource-aware computation, which will evolve to fully-fledged logics of resources suitable for UC applications.

The work on computational mechanisms for negotiation of access will need to be progressively lifted to design abstract machines accounting for the features of negotiation and guarantees of *resource bounds*, and implementing the relative protocols.

Trust Management None of the existing trust models covers dynamic trust evolution and re-evaluation satisfactorily. Still, real migrating agents evolve, interact, learn about each other and, therefore, change opinion. We need to develop models of trust accounting for its formation, evolution, and propagation. These must designed in accordance with the highly dynamic prerogatives of UC entities, take into account the dynamic formation of spontaneous networks between them, and be realistic about the speed of propagation of trust information. In particular, knowing the 'global' *web of* *trust* at any particular instant is beyond the ability of any single process in our framework. We will need techniques to approximate such knowledge using *local* exchanges and information, and protocols of *asynchronous* remote communications.

Work is needed to develop the basic computational mechanisms entailed by such vision, and in particular a set of primitives for distributed *trust evaluation* engines and for distributed *delegation* and *propagation* of trust levels between principals. It will be important to design suitable *languages* to express trust policies and their dynamic upgrading in response to events.

3.3 Relevance to Ubiquity Theory

UC's inherent difficulties originate in the extreme dynamic reconfigurability of the computational infrastructure. They include the lack of reliable mechanisms for coordination and trust between agents unknown to each other, the migration of untrusted agents to untrusted sites, the corresponding access control policies and their enforcement, the protection and safe management of resources, the privacy and confidentiality of data.

This requires the acquisition of a wide-spectrum conceptual understanding of foundational issues and computational models relying on principles built upon novel semantic models. The integration of all such aspects is fundamental for success. Theory plays a pivotal role in unifying notions and yielding flexible, adaptable mechanisms. Reciprocally, feedback from implementations is needed for the theory to be robust: theoretical and experimental investigations ought not to be kept apart at such an early stage in the development of a new computational infrastructure.

3.4 A list of the leading groups worldwide

Aarhus, DEN; Bologna, ITA; CMU, USA; Cambridge, UK; Cornell, USA; Edinburgh, UK; Florence, ITA; INRIA (Rocquencourt, Sophia), FRA; London, UK; Lisbon, POR Microsoft Cambridge, UK; Munich, GER; Princeton, USA; Santa Clara, USA; Sussex, UK; Tokyo, JAP.

4 Distributed data

This is some background material on distributed databases and ubiquitous computing. It provides a very brief survey of relevant background in databases and a selection of recent work that is, in the author's opinion, likely to be relevant to the development of databases in ubiquitous computing.

The topic of databases was emerged from the interplay of two requirements. The first was to have a simple *abstraction* for large quantities of structured data; the second was to make the manipulation of this data both robust and efficient. The relational model [52] was invented for the first of these. It proposed a simple tabular data model and an algebra of operations on tables. What was remarkable about this algebra is that its equational theory served as a basis for query optimisation and that queries could be efficiently implemented through indexing and join techniques. This accounted for the success of relational databases; the details are described in good database textbooks [139].

Even though databases are often designed in an attempt to contain all the information relevant to some "enterprise", they can never achieve this goal. In practice, the resources needed to answer a query are often to be found in a variety of databases distributed across a network. The need to query distributed data sources gave rise to one of the early, practical, examples of mobile code [77, 98, 161]. In distributed query optimisation one may decide to decompose a query and move parts of it to another site in order to reduce network traffic. The problems of this form of data integration were compounded by the emergence of new data models and the need to deal with a variety of data formats. For these it was necessary to find appropriate query languages in which one could generalise the optmisation systems developed for relational databases [50, 31].

The last ten years have seen a steady progression towards higher degrees of distribution and increasing mobility of data. An interesting case study is the field of bioinformatics in which there are some 500 public databases and many times that number in commercial use. Although the computing substrate in bioinformatics is far from ubiquitous, the trends towards ubiquity are evident.

- Only an handful of these databases are source data. The others are constructed by a process of filtering, tranforming, cleaning and annotating data in other databases. These databases have added value because of the effort that that has gone into constructing them.

- The structure of the databases evolves to capture new forms of scientific knowledge.

- Many of the databases make use of purpose-built data formats. Our query languages and optimisation techniques must be adapted to cope with these.

continuous spatio-temporal

- An increasingly important activity is monitoring other data sources for new information. Some important discoveries have been made by monitoring the "stream" of genetic sequence data.

[reference?]

In addition to challenging conventional database technology [61], bioinformatics raises new issues that have largely been ignored by database research. One is *provenance*. Given that fragments of data are beign copied between databases on an unprecedented scale, how do we record where these fragments have come from? Understanding and recording where data comes from and the route by which it arrived in a given database is crucial to data quality. Moreover, provenance is essential for almost any form of data exploration. The revolution caused by the proliferation of scientific databases on the Web has also damaged scientific research by making it difficult or impossible to cite, attribute and verify one's data sources. Part of provenance is *archiving*. Scientists expect to cite a data source and expect that the source will be preserved for others to verify. Yet databases change, they do not record their history, and citations become invalid. Archiving techniques are needed [32] in order to preserve past states of the data in an accessible form.

A related issue is *annotation*. Scientists are starting to communicate by a process of overlaying their annotations on existing data and making those annotations visible to others [149]. This brings up new challenges for database research: How do we describe the attachment of annotations to data? How do we carry annotations through queries? How do we organize our databases to receive annotations, and how do we attach annotations to most "legacy" databases that do not store annotations.

Although bioinformatics is an interesting step towards ubiquitous data, it is only a first step. Most bioinformatics application depend on conventional "pull" technology. The databases are largely unmovable, and queries are moved to the data. Also, the connection with mobility – in the programming language sense – though real is simple. Distributed query optimisation seldom involves more than a handful of databases, and most techniques are static: the optimisation strategy does not change during the evaluation of a query.

The emergence of XML as a universal data exchange format is having a profound impact on how we think of distributed data. In theory, XML only solves a low-level data serialisation problem – one that was arguably solved 50 years ago by Lisp. In practice it has had two important consequences. First we have had to revisit all the work that has been developed for databases – the query languages, type systems, the storage and optimisation techniques – and rework them for this form of semistructured data [4]. The second is that XML, being serial in nature, is suited to "push" technology. Rather than thinking of the database as static and the queries being mobile, one can consider the data being mobile and "streamed" past the queries that are static and filtering out information of interest. This dual version of data processing has been explored in recent work on data streams [43, 78].

To speculate on how databases will be used in ubiquitous computing, consider a distributed health-care system. There are huge problems associated with keeping patient records. One increasingly popular proposal is that each individual should "own" their medical record and give out all or part of the record to trusted parties on demand. Now assume that a medical record is not just a sequence of tests and treatments, but contains vast amounts of sensor data and, say, one's genetic sequence. Now suppose that a researcher wants to correlate the occurence of a cardiovascular condition with some genetic structure. The task is not to integrate a few databases, but literally millions of them. The task requires a mixture of push and pull technology. Assuming that the exchange formats are agreed, there is still a heterogeneous integration and optimisation problem, because some medical records will be held in conventional databases while others will be available from individuals via web services. Worse, the medical record itself may be distributed and parts of it may only sporadically be connected to a network. There has been some speculation on how databases will evolve to meet this challenge [72]. It is likely that we will also require models of mobility that have been developed in programming languages.

5 Game Semantics

Samson Abramsky

Game semantics has developed, from various antecendents in logic (especially the work of Lorenzen and his school, and Andreas Blass), and to a lesser extent combinatorial game theory and category theory (Conway, Joyal), into a signficant approach in the semantics of both programming languages and logics. Beyond the study of specific languages, it is an emerging *theory of interaction in general*. As such, it stands comparison with the theory of concurrent processes — which also forms one of the intellectual and technical ancestors of game semantics.

Like process theory, game semantics allows the modelling of systems of interacting agents or processes. The main *differences* are as follows:

- The distinction between System and Environment — the two protagonists in the 'game' — is made explicit in Game Semantics. This apparently small step has profound repercussions in the way the theory is structured.

- Game Semantics naturally organizes itself into structured categories (in which the objects are games and the morphisms are strategies), and hence into models of type theories, logics, λ-calculi and high-level programming languages, in a way in which process calculi do not (despite the efforts of the present author and others).

ha!
more successful than action semantics?

5.1 Achievements

Game Semantics has achieved striking successes in giving precise models for a wide range of logics, type theories, and λ-calculus based programming languages. In particular, it has proved possible to capture exactly which interactive processes are definable using certain computational features, such as: purely (sequential) functional means, local state, reference types, control operators, exceptions, nondeterminism, probabilities, ... and various combinations of these, in terms of *which structural constraints are imposed on strategies*. Thus for example, being purely functional involves only local information being available in each branch of a computation; relaxing this constraint characterizes *stateful* programs. Again, obeying a well-formed call-return discipline is captured by indentifying moves (basic actions) as *questions* or *answers* and imposing a 'bracketing constraint'. Relaxing this constraint characterizes those processes which can make use of non-local control features. In this way, an understanding of the *space of programming languages* begins to emerge; and it is becoming increasingly routine to capture subtle combinations of computational features with appropriate choices of forms of game semantic structure.

These foundational developments are increasingly leading to applications of various kinds. In particular:

- A significant current development is towards *Algorithmic game semantics*: taking advantage of the concrete nature of games and strategies to capture the semantics in machine representable form (e.g. representing strategies by automata) as a basis for program analysis and software model-checking. The inherent compositionality of game semantics – the ability to give direct meaning to incomplete program fragments - is one of the main attractions of this approach.

- Game semantics also offers a natural perspective for analyzing information flow and security properties, and for specifying and refining reactive program modules or components.

On the foundational lelvel, there are also significant developments towards a better understanding of the axiomatic basis for game semantics, and its connections with domain theory. These developments are important for improving the mathematical tractability, robustness and generality of the theory.

5.2 Expected advances in the near term

We anticipate the following advances over the next three or four years:

- Thus far, most of the applications of game semantics have been to *sequential* languages. It seems clear that concurrent languages can be treated with relatively minor modifications. (Non-determinism has already been handled with reasonable success).

- Object-oriented languages will similarly be treated. In fact, recent work by Jeffrey and Rathke, although not explicitly game semantical, is of a clearly related flavour. C1

- A 'nominal' version of Game semantics, incorporating the same kind of capabilities for scoped name creation as the π-calculus and related 'nominal' calculi, is being developed by the present author. This will certainly interact with both the previous directions.

- On the foundational side, the recent work by Jim Laird on sequoidal categories, and on locally boolean domains and bistable functions, promises some exciting progress on making game semantics more algebraic and mathematically tractable, and leading to a deeper understanding of the 'landscape' of interactive models, and how they relate to more traditional, extensional models.

- Another current development is towards physics-based models of computation, and in particular reversible and quantum computation. Game semantics is one of the promising ingredients towards a 'high-level' approach to quantum computation.

- Finally, we mention the theme of combining qualitative and quantitative approaches to information (to be discussed more extensively elsewhere). Again, it seems that game semantics is likely to play a role in these developments, e.g. in the study of information flow.

5.3 Relevance to Ubiquity

Why is Game Semantics relevant to Ubiquity?

- Firstly, a key part of the challenge of mastering ubiquitous computing is to thoroughly embrace the standpoint that every piece of hardware or software is a *component*, embedded in a larger system – and moreover this larger system is never known in advance, and is in fact constantly changing. What we need then is to understand components in terms of their interactions with their environment across some boundary. It is exactly this kind of understanding which Game sematics offers, in a highly structured form.
- Behavioural properties of a wide variety of forms, rather than traditional notions of 'extensional' correscness, will increasingly be a primary concern in the analysis of ubiquitous systems. Again, game semantics is well-adapted to the precise formulation and compositional verification of such properties.
- Finally, it is likely that agent-based views of computation, with explicit considerations of rational decision-making and maximization of utilities, which is already a significant research area in view of Internet applications, will be important in the context of ubiquitous computing. A rapprochement betwen Game Semantics and Game Theory, already the subject of some ongoing research, would be a powerful tool in a theory of ubiquity.

5.4 Leading groups worldwide

- The U.K.
  - Oxford (Abramsky, Ong)
  - Cambridge (Hyland)
  - Sussex (McCusker, Laird)

  Other interested researchers: Lazic (Warwick), Reddy (Birmingham), Malacaria, Honda (Queen Mary London), Schalk (Manchester), ...
- France
  - Paris (Curien, Melliès, Danos, Laurent, Baillot)
  - Marseille (Girard)
- Italy
  - Udine (Honsell, Lenisa, di Gianantonio)
- Canada
  - Montreal/Calgary (Seeley, Cockett, Panangaden)

6 Hybrid systems

Marta Kwiatkowska

A *discrete* (boolean) system is one in which state changes occur in a discrete fashion, between states that can be coded as boolean vectors. Sensor inputs, on the other hand, exhibit *continuous* dynamics, typically described in terms of differential equations in a multi-dimensional real-valued space. *Hybrid systems* are characterized by a combination of discrete and continuous components: they take the form of a discrete controller embedded in an analogue environment, where the analogue part of the system may concern variations of the ambient temperature of the system, the volume of coolant in a chemical process, or, more simply, the passage of time (between system events).

6.1 State of the art

[NOTE FROM RM: We'd like a bit more here on hybrid models, e.g. Henzinger, with references. They should balance the references concerned with hybrid model-checking in the next paragraph]

Formal models for hybrid systems typically take the form of a *finite-state automaton* (representing the discrete component) equipped with a finite set of real-valued variables (representing the continuous component), though we are aware of a first attempt to define a process calculus for such systems (the Φ-calculus). The values of the variables may influence the transitions among the states of the automaton (for example, by enabling particular transitions for choice), and, conversely, the transitions between such states and the passage of time when control remains within the states can affect the values of the variables (for example, a transition may reset the values of some continuous variables to a particular value, and differential equations describe how the variables change over time when control resides in a given state). A subclass of hybrid automata called the *timed automata*, which admits real-valued clocks that increase at the same rate as time as the only continuous variables, has been applied successfully to model and verify real-time systems.

The presence of real-valued variables in formalisms for hybrid systems means that the underlying semantic model of such formalisms is infinite-state. In contrast, model checking is applied traditionally to finite-state systems. A breakthrough result concerning the development of automatic verification methods for real-time systems was made by Alur and Dill [18], who presented a technique for obtaining a faithful finite-state representation of timed automata. Unfortunately, the decidability of basic sub-tasks of verification such as reachability are undecidable for many classes of hybrid automata (for example, the reachability problem for timed automata equipped with one clock which can be stopped in some states and restarted in others is, in general, undecidable), although some decidability results have been developed [?, 87, ?]. Typically, model checking tools for timed automata, such as UPPAAL and KRONOS, implement decidable algorithms, whereas semi-decidable model checking algorithms are implemented in model checkers of hybrid automata, such as the tool HYTECH. The development and implementation of efficient algorithms for timed automata [?, 105, 160] has made possible the verification of several (moderately-sized) industrial case studies.

6.2 Expected advances

[NOTE FROM RM: We should also have some expected advances in the models, not just in the model-checking.]

The successful development of timed automaton model checking tools such as UPPAAL will be consolidated in two respects: on one hand, there will be further advances in the development and implementation of efficient algorithms for timed automata verification; on the other hand, methods for the verification of extensions of the basic timed automata model – for example, with parameters, probabilities, and costs/rewards – will be implemented and applied to real-life systems. In the context of timed automata, and also the more general hybrid automata, the development of high-level system description languages, using appropriate concepts of hierarchical and parallel composition, will continue to be developed, along with associated modular verification techniques. Finally, the challenge of obtaining appropriate representations of state sets generated by complex continuous dynamics of hybrid automata will be addressed, developing ideas implemented thus far in tools such as HYTECH, CHECKMATE and d/dt.

6.3 Leading groups

- UPPAAL team of the universities of Uppsala and Aalborg (Larsen, Yi)
- VERIMAG (Sifakis, KRONOS)
- University of California at Berkeley (Henzinger)
- University of Pennsylvania (Alur)

7 Stochastics

Marta Kwiatkowska

[NOTE FROM RM: We should include something a little more detailed on stochastic process calculi, at present only mentioned in passing in paragraph 2.]

Many systems exhibit *stochastic dynamics*, either in the form of discrete probabilistic behaviour that is the outcome of coin tossing, or as described by continuous probability distributions (the probability of the system moving to a given state within a specified time is given by a probability density function). *Probabilistic models*, typically some variants of discrete or continuous *Markov processes*, are those whose transition relation is probabilistic (i.e. with probability 1/3 the next state is left, and with probability 2/3 it is right). Probabilistic modelling is used to represent and quantify uncertainty; as a symmetry breaker in distributed co-ordination problems; to model unreliable or unpredictable behavior; and to predict system behaviour based on the calculation of performance characteristics.

7.1 State of the art

Performance evaluation, historically the earliest, began in the early 1910s with A.K. Erlang's stochastic capacity planning for telephone exchanges, and developed into queuing theory much studied throughout the 20th century. The now established field of performance evaluation [150] aims to develop formalisms and tools for modelling systems and analysing their performance measures, as a means to support the process of design and engineering. The analysis involves building a probabilistic model of the system being considered, typically a continuous time Markov chain (CTMC), but often more general probability distributions are needed [74]. Such models can be derived from high-level descriptions in stochastic process calculi [89] or Petri nets [113]. The model serves as a basis for analytical, simulation-based or numerical calculations which result in steady-state or transient probabilities and the associated performance measures (resource utilisation, average call waiting time, etc). The focus is on *quantitative* characteristics, including measurement and testing, and covers a broad spectrum of issues.

Probabilistic model checking is an extension of model checking techniques to probabilistic systems, first introduced in [83] and further developed in [154, 56]. As in conventional model checking, a model of the probabilistic system, usually in the form of a discrete or continuous time Markov chain (DTMC/CTMC) or a Markov decision process (MDP), is built and then subjected to algorithmic analysis in order to establish whether it satisfies a given specification. The model checking procedure combines traversal of the underlying transition graph with numerical solutions of linear optimisation problems (for Markov decision process models) [26, 24] and linear equation systems and linear differential equation systems (for DTMC/CTMC models) [81, 21, 23].

Model checking of non-probabilistic systems has developed very quickly from first algorithms into implementations of industrially relevant software tools. In contrast, model checking of probabilistic systems has not followed the same path: although the algorithms and proof systems have been known since the mid-1980's, little implementation work has been done up until recently, culminating in the development of tools PRISM[1] (for DTMCs/CTMCs and MDPs) and ETMCC[2] (for CTMCs), with the help of which a number of systems have been studies (IEEE 1394 FireWire, Crowds anonymity protocol, etc). As in the case of conventional model checking, state space explosion is a particular difficulty, and has been tackled through an adaptation of *symbolic*, BDD-based, methods [22, 88] and parallel, distributed, disk-based techniques [100, 63]. Compositionality in the form of assume-guarantee reasoning has proved particularly difficult [62] and is not yet satisfactorily resolved. The direction being pursued is to seek out new algorithms, modelling and specification languages, industrially relevant case studies, as well as tool building and experimentation.

7.2 Expected advances

The challenges for ubiquitous systems are: spatial mobility, where we expect progress based on the stochastic pi-calculus [136] and the Φ-calculus or hybrid automata; and scalability, where a broader range of methods, ranging from sampling-based, through analytical and numerical solutions will be needed. Along with the very large systems, infinite-state systems also pose significant difficulties, which will require proof techniques. We anticipate that industrial-strength probabilistic model checkers will be developed and applied to a number of relevant case studies.

7.3 Leading groups

- University of Birmingham
- Twente (Katoen, Haverkoort)
- Bonn (Baier)
- Erlangen (Siegle)
- University of California at Santa Cruz (de Alfaro)
- UNSW/Maquarrie (Carroll Morgan and Annebelle McIver)
- Rice U (Vardi)
- U Toronto (Craig Boutilier, planning)

[1]`http:www.cs.bham.ac.uk/~dxp/prism/`

[2]`http://www7.informatik.uni-erlangen.de/etmcc/`

8 Model Checking

Marta Kwiatkowska

The concept of automated software verification is attributed to Floyd, author of the influential 1967 paper on the verifying compiler, but it was already apparent to Turing in 1950. The crux of Floyd's proposal was to annotate programs with assertions and involve a theorem prover to establish program correctness with respect to specification. *Assertion-style reasoning*, a methodology for manually deriving program correctness proofs, was developed over the following years, notably by Hoare and Dijkstra. The inherent complexity of real software and its features (concurrency, for example, posed a particular difficulty for the proof rules, as can be seen from the work of Owicki & Gries and Cliff Jones), combined with the relative inaccessibility of theorem provers for this task, have delayed the acceptance of the methodology in practice.

8.1 State of the art

A first breakthrough for automatic verification came with the advent of *model checking*, an alternative, model-based, software verification approach already implicit in Pnueli's seminal paper on Linear Time Temporal Logic (LTL) [135]. Model checking consists of *model building* (automatic derivation of a state-transition system from a model description in some appropriate language, possibly concurrent) and algorithmically checking by exhaustive state-space search if the executions of the model satisfy a given *property*, typically stated as a temporal logic formula; otherwise, error diagnostics are returned in the form of an execution up to property violation. Model checking was developed in 1981 independently by Clarke & Emerson [49] (for logical circuits and the logic CTL, Computation Tree Logic) and Quielle & Sifakis [137] (for protocols modelled in CCS and modal mu-calculus specifications). Some model checkers validate *equivalence* or *refinement* between two systems. Equivalence checking is often employed in hardware verification, to ensure that the modifications introduced in the circuit have not affected its functionality. Refinement checking is used to establish a relation between a specification and implementation.

Since model checking is an algorithmic procedure, the model representation must be *finite* to enable automatic analysis. In its simplest form, model checking reduces to a traversal of the underlying state-transition graph of the model, and hence is guided by fundamental research in the pertinent theories of automata, graph, formal language and logic. However, a distinctive feature of model checking is its emphasis on practice. In tandem with research effort invested in studying decidable classes of model checking problems and their complexity, as well as expressiveness of property specification and rigorous design of modelling languages, there is work on designing efficient data structures and engineering of software tools. This, together with the *fully automatic, push-button* technology nature of the model checking process, has contributed to wide acceptance of model checking in industry (Intel, IBM, Motorola, Siemens, Lucent/ Cadence). Its advance was aided by the ability to detect previously unknown errors, such as the Futurebus protocol (CMU SMV tool), Needham-Schroeder authentication protocol (CSP/FDR tool) and more recently the AODV (An Ad-hoc On demand Distance Vector routing protocol) (SPIN tool).

The 1980s saw major effort directed towards *finite-state* model checking, largely towards combating the *state-space explosion* problem, namely when the model size can become prohibitively large, which is characterized by the exponential growth of the number of states of the model with respect to the number of concurrent components[3]. *Symbolic* model checking, a heuristic approach based on Binary-Decision Diagrams (BDDs) introduced by Bryant and adapted to temporal logic verification in [33], contributed much to the success of model checking in practice. Symbolic model checking uses BDDs as a compact data structure to represent the state-transition graph that performs well if regularity and sharing occurs in the model. The primitive operations manipulate sets of states, and traversal is performed via fixed point calculation (in powerset lattice). First implemented in the SMV tool [115], symbolic model checking has been the single most important development that resulted in industrial exploitation of automatic verification, particularly in the context of circuit design. More recently, SAT solvers, their underpinning software technology having substantially advanced recently, were also employed in *bounded* model checking [27], a verification method applicable up to a fixed execution depth, in some cases resulting in real-time improvement over symbolic model checking. Unfortunately, these heuristic methods can be unpredictable and rarely scale up well to industrial size systems.

Finiteness of the model is a significant limitation of the model checking approach. Not only is it cumbersome (for example, a new check has to be performed for every instantiation of parameters), but also unrealistic since systems can be *infinite-state*. Infinite-state model checking has been a major concern throughout the 1990s. The sources of infinity vary, and may arise from *unbounded* data types and control structures, *parametric* specifications, or *real-valued* data such as time. Clearly, infinite-state models may give rise to undecidable verification problems, that can only be tackled with a theorem proving approach, but for certain problems representing the model in full detail is often unnecessary – an abstract version will suffice. Key techniques that have accounted for much progress in the last decade, and which we anticipate to yield further advances, are *abstraction*, *compositional reasoning* techniques and *automated proof support*.

Predicate abstraction [48] identifies all states that satisfy the same set of predicates, thus reducing the size of the model (which can be infinite) to a finite quotient. The success or failure of model checking on the abstract model determines the corresponding property on the concrete system, with possible further abstraction/refinement steps guided by counterexamples [47]. Several data type reduction techniques exist, including symmetry reduction [97] and data independence [106]. Unfortunately, such methods are rarely fully automatic, often requiring much insight into the modelling problem.

Compositional reasoning, such as *assume-guarantee* reasoning, allow to decompose the system and the verification problem into feasible tasks. Originally introduced by Misra and Chandy (1981) for manual verification, assume-guarantee was adapted to

[3]For example, a systems composed of 100 concurrent components each having at most 10 states can have up to 10^{100} states

model checking in [?] (tool MOCHA) and [?] (tool Cadence SMV) and its effectiveness demonstrated on large examples. Hierarchical verification combined with refinement checking can improve scalability further.

There has been much progress made recently regarding verification of systems with unbounded structures, resulting in decidability and complexity results for *push-down systems* [68] and their implementation (tool MOPED). Unfortunately, verification of many infinite-state systems is only possible via automated *proof support*. A suitable combination of deductive verification based on theorem proving with algorithmic model checking clearly has potential, and has already been explored in e.g. [138], but its success in practice will be determined by how much low-level user intervention is needed for the task at hand. An alternative is an enhancement of the power of model checking with automatic support for proof rules (Cadence SMV tool). The verification of inductive properties is broken up into model-checkable cases, yielding automatic and efficient verification of large, possibly infinite-state systems, albeit for a restricted class of systems and properties compared to a theorem proving approach.

The availability of automated proof support within a model checker brings us a step closer to Floyd's vision of 'the verifying compiler', but unfortunately does not address the biggest challenge of software verification – the gap between the model and *real software*, be it embedded (which may contain clock and other sensor inputs, electronic coin tossing, stochastic features, etc) or programs written high-level programming languages such as C and Java (which may contain pointers, pointer arithmetic, dynamic data structures, concurrent threads, recursion, etc). The past five years have witnessed progress in that direction too, founded on the application of static analysis and abstraction/refinement methods to extract a finite-state model from a given C or Java program, which is then submitted to a model checker, and, in case of errors, concretized executions are derived from the abstract ones of the model's. Examples of relevant projects in this area are Bandera[4] and Java PathFinder[5] (for Java programs), BLAST[6] at Berkeley (most of C language) and SLAM[7] at Microsoft Redmond (applied to driver verification). That the techniques are making inroads into the software development practices is evident from the words of Bill Gates (April 18, 2002): "Things like even software verification, this has been the Holy Grail of computer science for many decades but now in some very key areas, for example, driver verification were building tools that can do actual proof about the software and how it works in order to guarantee the reliability."

8.2 Expected advances

Despite the successes of model checking in the past 20 years or so, the size and complexity of of industrial designs and software systems is such that their comprehensive verification is well beyond reach. The challenges pertinent to ubiquity are: *mobility* and *dynamic* scenarios (some progress has been recently made for the pi-calculus in [59]); *infinite-state* systems (parametric verification, push-down systems); *scalability* of the

[4] `http://www.cis.ksu.edu/santos/bandera/`
[5] `http://ase.arc.nasa.gov/visser/jpf/`
[6] `http://www-cad.eecs.berkeley.edu/~rupak/blast/`
[7] `http://research.microsoft.com/SLAM/`

methods to industrial scale systems (e.g. via hierarchical verification, compositionality, deductive methods); and *real software* (pointers and pointer arithmetic, dynamic data structures and the heap, procedures and recursion). In theory, we expect progress to be made in all those, with the successful methods to be developed in practice so that infinite-state model checking becomes widely accepted in industry. In turn, industrial demands will enforce greater reusability, more standardization and interconnectivity of tools.

8.3 Leading groups

- CMU (Clarke)
- Rice University (Vardi)
- Weizmann (Pnueli)
- VERIMAG
- IRST Trento
- LSV, ENS Cachan
- Oxford (Roscoe, FDR)
- Cambridge (Gordon)
- KSU (Bandera)
- MS Redmond (SLAM)
- NASA Ames (Java PathFinder)
- Stanford (Dill, Murphy)
- Berkeley (Henzinger, Mocha)
- University of Pennsylvania (Alur, Mocha)
- LIAFA, Paris 7 (Bouajjani)
- Uppsala (Jonsson)
- Edinburgh/Stuttgart (Esparza, Moped)

References

[1] M. Abadi and A.D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148:1–70, 1999.

[2] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, 1996.

[3] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.

[4] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann, 1999.

[5] S. Abramsky. Retracing some paths in process algebra. In U. Montanari and V. Sassone, editors, *Proceedings of CONCUR 96*, volume 1119 of *Lecture Notes in Computer Science*, pages 1–17. Springer-Verlag, 1996.

[6] S. Abramsky. A structural approach to reversible computation. In *Proceedings of LCCS 2001: International Workshop on Logic and Complexity in Computer Science*, 2001.

[7] S. Abramsky. Algorithmic game semantics: a tutorial introduction. In H. Schwichtenberg and R. Steinbruggen, editors, *Proof and System-Reliability*. Kluwer, 2002.

[8] S. Abramsky and R. Coecke. Physical traces: Quantum vs. classical information processing. In *Proceedings of CTCS 2002*, Electronic Lecture Notes in Computer Science, 2002.

[9] S. Abramsky, K. Honda, and G. McCusker. A fully abstract game semantics for general references. In *Proceedings of LiCS*, 1998.

[10] S. Abramsky and R. Jagadeesan. Games and full completeness for multiplicative linear logic. *Journal of Symbolic Logic*, 59:543–574, 1994.

[11] S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF. *Information and Computation*, 163:409–470, 2000.

[12] S. Abramsky and G. McCusker. Call-by-value games. In *Proceedings of CSL '97*, 1997.

[13] S. Abramsky and G. McCusker. Linearity, sharing and state. In P. O'Hearn and R. D. Tennent, editors, *Algol-like languages*, pages 317–348. Birkhauser, 1997.

[14] S. Abramsky and G. McCusker. Full abstraction for idealized Algol with passive expressions. *Theoretical Computer Science*, 1999.

[15] S. Abramsky and G. McCusker. Game semantics. In U. Berger and H. Schwichtenberg, editors, *Computational Logic*, pages 1–56. Springer Verlag, 1999.

[16] S. Abramsky and P.-A. Melliès. Concurrent games and full completeness for multiplicative-additive linear logic. In *Proceedings of LiCS*, 1999.

[17] Samson Abramsky and Radha Jagadeesan. New foundations for the geometry of interaction. *Information and Computation*, 111:53–119, 1994.

[18] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[19] R. Anderson. *Security Engineering*. Wiley, 2001.

[20] R. Anderson. TCPA / Palladium frequently asked questions, 2002. http://www.cl.cam.ac.uk/ rja14/tcpa-faq.html.

[21] A. Aziz, V. Singhal, F. Balarin, R. Brayton, and A. Sangiovanni-Vincentelli. It usually works: The temporal logic of stochastic systems. In P. Wolper, editor, *Proc. 7th International Conference on Computer Aided Verification (CAV'95)*, volume 939 of *LNCS*, pages 155–165. Springer, 1995.

[22] C. Baier, E. Clarke, V. Hartonas-Garmhausen, M. Kwiatkowska, and M. Ryan. Symbolic model checking for probabilistic processes. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *Proc. ICALP'97*, volume 1256 of *LNCS*, pages 430–440. Springer, 1997.

[23] C. Baier, J.-P. Katoen, and H. Hermanns. Approximate symbolic model checking of continuous-time Markov chains. In J. Baeten and S. Mauw, editors, *Proc. 10th International Conference on Concurrency Theory (CONCUR'99)*, volume 1664 of *LNCS*, pages 146–161. Springer, 1999.

[24] C. Baier and M. Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *Distributed Computing*, 11(3):125–155, 1998.

[25] J.A. Bergstra and J.W. Klop. Algebra for communicating processes with abstraction. *Theoretical Computr Science*, 7:77–121, 1985.

[26] A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In P. Thiagarajan, editor, *Proc. 15th Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 1026 of *LNCS*, pages 499–513. Springer, 1995.

[27] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proc. TACAS'99*, volume 1579 of *LNCS*, pages 193–207. Springer, 1999.

[28] B. Blanchet. From secrecy to authenticity in security protocols. In *9th International Static Analysis Symposium (SAS'02)*, volume 2477 of *Lecture Notes in Computer Science*, pages 242–259. Springer, 2002.

[29] M. Blaze, J. Feigenbaum, and A.D. Keromytis. Keynote: Trust management for public-key infrastructures. In *Proc. of Security Protocols*, volume 1550 of *Lecture Notes in Computer Science*, pages 59–63. Springer, 1999.

[30] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proc. of IEEE Symposium on Security and Privacy*, pages 164–173. IEEE Press, 1996.

[31] Val Breazu-Tannen, Peter Buneman, Shamim Naqvi, and Limsoon Wong. Principles of Programming with Collection Types. *Theoretical Computer Science*, 149:3–48, 1995.

[32] Peter Buneman, Sanjeev Khanna, Keishi Tajima, and Wang-Chiew Tan. Archiving scientific data. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, Madison, Wisconsin, 2002.

[33] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proc. 5th Annual IEEE Symposium on Logic in Computer Science (LICS'90)*, pages 428–439. IEEE Computer Society Press, 1990.

[34] M. Burrows, M. Abadi, and R.M. Needham. A logic of authentication. *Proceedings of the Royal Society of London A*, 426:233–271, 1989.

[35] M. Burrows, M. Abadi, and R.M. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

[36] M. Carbone, M. Nielsen, and V. Sassone. Towards a formal model for trust. Technical Report RS-03-4, BRICS, University of Aarhus, 2003.

[37] L. Cardelli, G. Ghelli, and A. D. Gordon. Ambient groups and mobility types. In *Proc. IFIP TCS*, volume 1872 of *Lecture Notes in Computer Science*, pages 333–347. Springer, 2000.

[38] L. Cardelli, G. Ghelli, and A.D. Gordon. Mobility types for mobile ambients. In *Proc. of ICALP'99*, volume 1644 of *Lecture Notes in Computer Science*, pages 230–239. Springer, 1999.

[39] L. Cardelli and A.D. Gordon. Mobile ambients. In *Proc. of FoSSaCS'98*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155. Springer, 1998.

[40] L. Cardelli and A.D. Gordon. Anytime, anywhere: modal logics for mobile processes. In *Proc. of POPL'00*. ACM Press, 2000.

[41] L. Cardelli and A.D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240:177–213, 2000.

[42] I. Castellani. Process algebras with localities. In J. Bergsta, I. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*, pages 947–1045. Elsevier, 2001.

[43] Sirish Chandrasekaran and Michael J. Franklin. Streaming Queries over Streaming Data. In *VLDB Conference*, Hong Kon, August 2002.

[44] W. Charatonik, A. Gordon, and J.-M. Talbot. Finite-control mobile ambients. In *Proc. of ESOP'02*, volume 2305 of *Lecture Notes in Computer Science*, pages 295–313. Springer, 2002.

[45] Y.-H. Chu, J. Feigenbaum, B. LaMacchia, P. Resnick, and M. Strauss. Referee: Trust management for Web applications. *Computer Networks and ISDN Systems*, 29:953–964, 1997.

[46] J. Clark and J. Jacob. A survey of authentication protocol literature. Unpublished report. University of York, 1997.

[47] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In A. Emerson and A. Sistla, editors, *Proc. 12th International Conference on Computer Aided Verification (CAV'00)*, volume 1855 of *LNCS*, pages 154–169. Springer, 2000.

[48] E. Clarke, O. Grumberg, and D. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, 1994.

[49] E.M. Clarke and E.A. Emerson. Design and Synthesis of Synchronization Skeletons using Branching Time Temporal Logic. In D. Kozen, editor, *Proceedings of the Workshop on Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71, Yorktown Heights, New York, May 1981. Springer-Verlag.

[50] S. Cluet. Designing OQL: Allowing objects to be queried. *Information Systems*, 2(3), 1998.

[51] B. Frantz C.M. Ellison, B. Lampson, R.L. Rivest, B.M. Thomas, and T. Ylonen. *SPKI certificate theory*, volume 2693. RCF, 1999.

[52] E.F. Codd. A Relational Model for Large Shared Databanks. *Communications of the ACM*, 13(6):377 – 387, June 1970.

[53] E. Cohen. TAPS: A first-order verifier for cryptographic protocols. In *13th IEEE Computer Security Foundations Workshop*, pages 144–158. IEEE Computer Society Press, 2000.

[54] S. Conchon and F. Le Fessant. Jocaml: Mobile agents for objective caml. In *Proc. of ASA/MA'99*. IEEE Press, 1999.

[55] Microsoft Corporation. Microsoft "Palladium": A business overview, 2002. http://www.microsoft.com/presspass/features/2002/jul02/0724palladiumwp.asp.

[56] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995.

[57] K. Crary, D. Walker, and G. Morrisett. Typed memory management in a calculus of capabilities. In *Proc. POPL 99*, pages 262–275. ACM Press, 1999.

[58] K. Crary and S. Weirich. Resource bounds certification. In *Proc. of POPL 00*, pages 184–198. ACM Press, 2000.

[59] Mads Dam. Proving properties of dynamic process networks. *Information and Computation*, 140(2):95–114, 1998.

[60] V. Danos and R. Harmer. Probabilistic game semantics. In *Proc. IEEE Symp. LICS'00*. Comp. Soc. Press, 2000.

[61] Susan Davidson, Chris Overton, Val Tannen, and Limsoon Wong. Biokleisli: A digital library for biomedical researchers. *Journal of Digital Libraries*, 1(1), 1997.

[62] L. de Alfaro, T. Henzinger, and R. Jhala. Compositional methods for probabilistic systems. In K. Larsen and M. Nielsen, editors, *Proc. 12th International Conference on Concurrency Theory (CONCUR'01)*, volume 2154 of *LNCS*, pages 351–365. Springer, 2001.

[63] D. Deavours and W. Sanders. An efficient disk-based tool for solving very large Markov models. In R. Marie, B. Plateau, M. Calzarossa, and G. Rubino, editors, *Proc. 9th International Conference on Modelling Techniques and Tools (TOOLS'97)*, volume 1245 of *LNCS*, pages 58–71. Springer, 1997.

[64] R. DeLine and M. Fähndrich. Enforcing high-level protocols in low-level software. In *Programming Language Design and Implementation*, pages 59–69, 2001.

[65] D. Dolev and A.C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT–29(2):198–208, 1983.

[66] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI certificate theory, 1999. RFC 2693.

[67] C.M. Ellison. Home network security. *Intel Technology Journal*, November 2002. http://developer.intel.com/technology/itj/2002/volume06issue04/.

[68] Javier Esparza. Decidability of model checking for infinite-state concurrent systems. *Acta Informatica*, 34(2):85–107, ???? 1997.

[69] C. Fournet and G. Gonthier. The reflexive CHAM and the join-calculus. In *Proc. of POPL'96*, pages 372–385. ACM Press, 1996.

[70] C. Fournet, G. Gonthier, J.-J. Lévy, L. Maranget, and D. Rémy. A calculus of mobile agents. In *Proc. of CONCUR'96*, volume 1119 of *Lecture Notes in Computer Science*, pages 406–421. Springer, 1996.

[71] C. Fournet, J.-J. Lévy, and A. Schmitt. An asynchronous distributed implementation for mobile ambients. In *Proc. IFIP TCS*, volume 1872 of *Lecture Notes in Computer Science*, pages 348–364. Springer, 2000.

[72] Michael J. Franklin. Challenges in Ubiquitous Data Management. In R. Wilhiem, editor, *Informatics: 10 Years Back, 10 Years Ahead.* Springer-Verlag, 2001. LNCS #2000.

[73] P. Gardner. From process calculi to process frameworks. In *Proc. 11th International Conference on Concurrency Theory*, volume 1877 of *Lecture Notes in Computer Science*, pages 69–88. Springer Verlag, 2000.

[74] R. German. *Performance Analysis of Communication Systems: Modeling with Non-Markovian Stochastic Petri Nets.* John Wiley and Sons, 2000.

[75] D. R. Ghica and G. McCusker. Reasoning about idealized Algol using regular languages. In *In* Proc. ICALP'00, pages 103–116. Springer-Verlag, 2000.

[76] J.Chr. Godskesen, T. Hildebrandt, and V. Sassone. A calculus of mobile resources. In *Proc. of CONCUR '02*, volume 2421 of *Lecture Notes in Computer Science*, pages 272–2872. Springer, 2002.

[77] Goetz Graefe and William J. McKenna. The Volcano Optimizer Generator: Extensibility and Efficient Search. *ICDE*, pages 209–218, 199.

[78] Todd J. Green, Gerome Miklau, Makoto Onizuka, and Dan Suciu. Processing XML Streams with Deterministic Automata. In *Proceedings of ICDT*, 2003.

[79] J. Guttman. Filtering postures: Local enforcement for global policies. In *IEEE Computer Society Symposium on Research in Security and Privacy*, pages 120–129, 1997.

[80] S. Hallem, B. Chelf, Y. Xie, and D. Engler. A system and language for building system-specific, static analyses. In *2002 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'02)*, pages 69–82, 2002.

[81] H. Hansson and B. Jonsson. A logic for reasoning about time and probability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[82] R. Harmer and G. McCusker. A fully abstract game semantics for finite nondeterminism. In *Proc. IEEE Symp. LICS'99*. Comp. Soc. Press, 1999.

[83] S. Hart, M. Sharir, and A. Pnueli. Termination of Probabilistic Concurrent Programs. *ACM Transactions on Programming Languages and Systems*, 5:356–380, 1983.

[84] M. Hennessy, M. Merro, and J. Rathke. Towards a behavioural theory of access and mobility control in distributed systems. In *Proc. of FOSSACS 03*, 2003.

[85] M. Hennessy and J. Riely. Type-safe execution of mobile agents in anonymous networks. In *Proceedings Workshop on Mobile Object Systems*, pages 378–390, 1998.

[86] M. Hennessy and J. Riely. Resource access control in systems of mobile agents. *Information and Computation*, 173:82–120, 2002.

[87] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57(1):94–124, August 1998.

[88] H. Hermanns, J. Meyer-Kayser, and M. Siegle. Multi terminal binary decision diagrams to represent and analyse continuous time Markov chains. In B. Plateau, W. Stewart, and M. Silva, editors, *Proc. 3rd International Workshop on Numerical Solution of Markov Chains (NSMC'99)*, pages 188–207. Prensas Universitarias de Zaragoza, 1999.

[89] J. Hillston. *A Compositional Approach to Performance Modelling*. PhD thesis, University of Edinburgh, 1994.

[90] C.A.R. Hoare. An axiomatic basis for computer programming. *Communications of ACM*, 12, 1969.

[91] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[92] M. Hofmann. A type system for bounded space and functional in-place update. *Nordic Journal of Computing*, 7(4):258–289, 2000.

[93] K. Honda and N. Yoshida. Game-theoretic analysis of call-by-value computation (extended abstract). In *Proc. of ICALP'97*. Springer-Verlag, 1997.

[94] J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF. *Information and Computation*, 163:285–408, 2000.

[95] IBM Corporation and Microsoft Corporation. Security in a web services world: A proposed architecture and roadmap, April 2002. Version 1.0.

[96] A. Igarashi and N. Kobayashi. Resource usage analysis. In *Proc. of POPL 02*, pages 331–342. ACM Press, 2002.

[97] C. Ip and D. Dill. Better verification through symmetry. *Formal Methods In System Design*, 9(1-2):41–75, 1996.

[98] Matthias Jarke and Jürgen Koch. Query Optimization in Database Systems. *ACM Computing Surveys*, 16(2):111–15, 1984.

[99] T. Jim, G. Morrisett, D. Grossman, M. Hicks, J. Cheney, and Y. Wang. Cyclone: A safe dialect of c. In *USENIX Annual Technical Conference*, 2002.

[100] W. Knottenbelt and P. Harrison. Distributed disk-based solution techniques for large Markov models. In B. Plateau, W. Stewart, and M. Silva, editors, *Proc. 3rd International Workshop on Numerical Solution of Markov Chains (NSMC'99)*, pages 58–75. Prensas Universitarias de Zaragoza, 1999.

[101] N. Kobayashi, B.C. Pierce, and D.N. Turner. Linearity and the pi-calculus. *ACM Transactions on Programming Languages and Systems*, 21(5):914–947, 1999.

[102] J. Laird. A categorical semantics of higher-order store. In *Proceedings of CTCS 2002*, Electronic Lecture Notes in Computer Science, 2002.

[103] J. D. Laird. *A semantic analysis of control*. PhD thesis, University of Edinburgh, 1998.

[104] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, 1992.

[105] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *Software Tools for Technology Transfer*, 1(1+2):134–152, 1997.

[106] R. Lazic and D. Nowak. A unifying approach to data-independence. In C. Palamidessi, editor, *Proc. 11th International Conference on Concurrency Theory (CONCUR'00)*, volume 1877 of *LNCS*, pages 581–595. Springer, 2000.

[107] F. Levi and D. Sangiorgi. Controlling interference in ambients. In *Proc. of POPL'00*, pages 352–364. ACM Press, 2000.

[108] T. Lindholm and F. Yellin. *The Java Virtual Machine Specification*. Addison-Wesley, 1997.

[109] H.-W. Loidl, O. Shkaravska, and L. Beringer. Preliminary investigations into a bytecode logic for grail. Available at http://www.dcs.ed.ac.uk/home/mrg, 2003.

[110] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using CSP and FDR. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer, 1996.

[111] K. MacKenzie. A virtual machine platform for resource-bounded computation. Available at http://www.dcs.ed.ac.uk/home/mrg, 2003.

[112] P. Malacaria and C. Hankin. Non-deterministic games and program analysis: an application to security. In *Proc. LICS'99*, pages 443–452. Comp. Soc. Press, 1999.

[113] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceshinis. *Modelling with Generalized Stochastic Petri Nets*. John Wiley and Sons, 1995.

[114] J. McCarthy et al. *LISP 1.5 Programming Manual*. MIT Press, 1956.

[115] K. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[116] M. Merro and V. Sassone. Typing and subtyping mobility in boxed ambients. In *Proc. of CONCUR 02*, volume 2421 of *Lecture Notes in Computer Science*, pages 304–320. Springer, 2002.

[117] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer Verlag, 1980.

[118] R. Milner. Bigraphical reactive systems. In *Proc. 12th International Conference on Concurrency Theory*, volume 2154 of *Lecture Notes in Computer Science*, pages 16–35. Springer Verlag, 2001.

[119] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. *Information and Computation*, 100(1):1–77, 1992.

[120] R. Milner, M. Tofte, and R. Harper. *The Definition of Standard ML*. MIT Press, 1990.

[121] G. Morrisett, D. Walker, K. Crary, and N. Glew. From system f to typed assembly languages. In *Proc. of POPL'98*, pages 85–97. ACM Press, 1998.

[122] A. S. Murawski and C.-H. L. Ong. Exhausting strategies, joker games and full completeness for IMLL with unit. In *Proc. 8th CTCS*, volume 29 of *Electronic Lecture Notes in Computer Science*, page 31, 1999.

[123] A. S. Murawski and C.-H. L. Ong. Discrete games, light affine logic and PTIME computation. In *Proc. CSL'00*, volume 1862 of *Lecture Notes in Computer Science*, pages 427–441. Springer-Verlag, 2000.

[124] N. Nagaratnam, P. Janson, J. Dayka, A. Nadalin, F. Siebenlist, V. Welch, I. Foster, and S. Tuecke. The security architecture for open grid services, 2002. Version 1. http://www.globus.org/ogsa/Security/OGSA-SecArch-v1-07192002.pdf.

[125] G. Necula. Proof-carrying code. In *24th ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages*, pages 106–119. ACM Press, 1997.

[126] G Necula, S. McPeak, and W. Weimer. CCured: type-safe retrofitting of legacy code. In *Symposium on Principles of Programming Languages*, pages 128–139, 2002.

[127] R. De Nicola, G. Ferrari, R. Pugliese, and B. Venneri. Types for access control. *Theoretical Computer Science*, 240(1):215–254, 2000.

[128] P.W. O'Hearn and D.J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, 1999.

[129] L.C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.

[130] C.A. Petri. Kommunikation mit automaten. Technical Report 2, Institut fur Instrumentelle Mathematik, Bonn, 1962.

[131] B.C. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–454, 1996.

[132] B.C. Pierce and D.N. Turner. A programming language based on the π-calculus. Technical Report 476, CSCI, Indiana University, 1997.

[133] J. Pincus. User interaction issues in defect detection tools, August 2001. http://research.microsoft.com/specncheck/docs/pincus.ppt.

[134] G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Århus University, 1981.

[135] A. Pnueli. The temporal logic of programs. In *Proc. FOCS*, pages 46–77. IEEE, 1977.

[136] C. Priami. Stochastic pi-calculus. *The Computer Journal*, 38(7):578–589, 1995.

[137] J. Queille and J. Sifakis. Specification and Verification of Concurrent Systems in CESAR. In *Proc. 5th International Symposium on Programming*, pages 337–351, 1982. Available as Volume 137 of *LNCS*.

[138] S. Rajan, N. Shankar, and M. Srivas. An integration of model-checking with automated proof checking. In P. Wolper, editor, *Proc. Computer-Aided Verification (CAV'95)*, volume 939 of *LNCS*, pages 84–97. Springer, 1995.

[139] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems*. McGraw-Hill, 2002.

[140] P.V. Rangan. An axiomatic basis of trust in distributed systems. In *Symposium on Security and Privacy*. IEEE Press, 1988.

[141] A. Ravara and V.T. Vasconcelos. Typing non-uniform concurrent objects. In *Proc. of CONCUR'00*, volume 1877 of *Lecture Notes in Computer Science*, pages 474–488. Springer, 2000.

[142] J. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proc. of LICS'02*, pages 55–74. IEEE Press, 2002.

[143] R.L. Rivest and B. Lampson. SDSI – a simple distributed security infrastructure, 1996. http://theory.lcs.mit.edu/ rivest/sdsi10.html.

[144] J.A. Robinson. A machine-oriented logic based upon the resolution principle. *Journal of ACM*, 12(1):23–41, 1965.

[145] B. Schneier. *Secrets and Lies*. Wiley, 2000.

[146] P. Sewell. Global/local subtyping and capability inference for a distributed pi-calculus. In *Proc. of ICALP'98*, volume 1443 of *Lecture Notes in Computer Science*, pages 695–706. Springer, 1998.

[147] U. Shankar, K. Talwar, J. S. Foster, and D. Wagner. Detecting format string vulnerabilities with type qualifiers. In *10th USENIX Security Symposium*, 2001.

[148] D.X. Song. Athena: a new efficient automatic checker for security protocol analysis. In *12th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, 1999.

[149] Lincoln Stein. The Distributed Annotation System. `http://www.biodas.org`.

[150] W. J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton, 1994.

[151] D. Teller, P. Zimmer, and D. Hirschkoff. Using ambients to control resources. In *Proc. of CONCUR'02*, volume 2421 of *Lecture Notes in Computer Science*, pages 288–303. Springer, 2002.

[152] A. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proc. London Math. Soc*, 42:230–265, 1937.

[153] A. Valente and D. Sangiorgi. A distributed abstract machine for safe ambients. In *Proc. of ICALP 2001*, volume 2076 of *Lecture Notes in Computer Science*, pages 408–420. Springer, 2001.

[154] M. Vardi. Automatic verification of probabilistic concurrent finite state programs. In *Proc. FOCS'85*, pages 327–338, 1985.

[155] J. Vitek and G. Castagna. Seal: A framework for secure mobile computations. In *Internet Programming Languages*, number 1686 in Lecture Notes in Computer Science, pages 47–77. Springer, 1999.

[156] D. Wagner, J. S. Foster, E. A. Brewer, and A. Aiken. A first step towards automated detection of buffer overrun vulnerabilities. In *Network and Distributed System Security Symposium*, pages 3–17, 2000.

[157] S. Weeks. Understanding trust management systems. In *Proc. of IEEE Symposium on Security and Privacy*. IEEE Press, 2001.

[158] M. Weiser. Ubiquitous computing. http://www.ubiq.com/hypertext/weiser/UbiHome.html.

[159] N. Yoshida. Graph types for monadic mobile processes. In *Proc. of FST&TCS'96*, volume 1180 of *Lecture Notes in Computer Science*, pages 371–386. Springer, 1996.

[160] S. Yovine. Kronos: A verification tool for real-time systems. *Software Tools for Technology Transfer*, 1(1/2):123–133, 1997.

[161] Clement T. Yu and C. C. Chang. Distributed query processing. *ACM Computing Surveys*, 16(4):399–433, 1984.