Nieuwsbrief van de Nederlandse Vereniging voor Theoretische Informatica

Mieke Bruné, Jan Willem Klop, Jan Rutten (redactie)*

Inhoudsopgave

*CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands. Email: mieke@cwi.nl.

1 Van de Redactie

Beste NVTI-leden,

Graag bieden wij u hierbij het vijfde nummer aan van de jaarlijkse NVTI-Nieuwsbrief. Bij het samenstellen hebben we weer de formule van de vorige vier nummers gevolgd. Zo vindt u naast het programma van de jaarlijkse Theoriedag en de bijgewerkte ledenlijst ook weer enkele bijdragen van collega's met een korte inleiding in hun speciale gebied van expertise. Evenals voorgaande jaren zouden deze Nieuwsbrief en de Theoriedag niet tot stand hebben kunnen komen zonder de financiele steun van onze sponsors: GEB-NWO (vroeger SION), Elsevier Publishing Company, en de onderzoekscholen IPA en OZL. Namens de NVTI gemeenschap onze hartelijke dank voor deze middelen die ons voortbestaan mogelijk maken!
De redactie,
Mieke Bruné (mieke@cwi.nl)
Jan Willem Klop (jwk@cwi.nl)
Jan Rutten (janr@cwi.nl)

2 Samenstelling Bestuur

Prof.dr. J.C.M. Baeten (TUE)
Dr. H.L. Bodlaender (UU)
Prof.dr. J.W. Klop (VUA/CWI) voorzitter
Prof.dr. J.N. Kok (RUL)
Prof.dr. J.-J.Ch. Meyer (UU)
Prof.dr. G.R. Renardel de Lavalette (RUG)
Prof.dr. G. Rozenberg (RUL)
Dr. J.J.M.M. Rutten (CWI) secretaris
Dr. L. Torenvliet (UvA)

3 Van de voorzitter

Geacht NVTI-lid,

Ook dit jaar heeft het Bestuur zich beijverd om een interessante Theoriedag te organiseren, met prominente sprekers uit binnen- en buitenland. De Theoriedag zal gehouden worden op vrijdag 23 maart, in het Jaarbeurs Congrescentrum Utrecht. We hopen en vertrouwen erop dat het programma voor velen van u interessant is. Graag tot ziens op 23 maart in Utrecht!
Jan Willem Klop, voorzitter NVTI

4 Theoriedag 2001

Vrijdag 23 maart 2001, Jaarbeurs Utrecht

Het is ons een genoegen u uit te nodigen tot het bijwonen van de Theoriedag 2001 van de NVTI, de Nederlandse Vereniging voor Theoretische Informatica, die zich ten doel stelt de theoretische informatica te bevorderen en haar beoefening en toepassingen aan te moedigen. De Theoriedag 2001 zal gehouden worden op vrijdag 23 maart 2001, in het congrescentrum Jaarbeurs Utrecht, op enkele minuten loopafstand van CS Utrecht, en is een voortzetting van de reeks jaarlijkse bijeenkomsten van de NVTI die zes jaar geleden met de oprichtingsbijeenkomst begon.

Evenals vorige jaren hebben wij een aantal prominente sprekers uit binnen- en buitenland bereid gevonden deze dag gestalte te geven met voordrachten over recente en belangrijke stromingen in de theoretische informatica. Naast een wetenschappelijke inhoud heeft de dag ook een informatief gedeelte, in de vorm van een algemene vergadering waarin de meest relevante informatie over de NVTI gegeven zal worden, alsmede presentaties van de onderzoekscholen.

Programma

09.30-10.00: Ontvangst met koffie
10.00-10.10: Opening
10.10-11.00: Lezing Prof.dr. R. Milner (University of Cambridge)
Titel: The Flux of Interaction
11.00-11.30: Koffie
11.30-12.20: Lezing Prof.dr. A. Siebes (Universiteit Utrecht)
Titel: BioInformatics
12.20-12.50: Presentatie Onderzoeksscholen (OZL, IPA, SIKS)
12.50-14.10: Lunch (Zie beneden voor registratie)
14.10-15.00: Lezing Prof.dr. D. Harel (The Weizmann Institute of Science, Israel)
Titel: On the Aesthetics of Diagrams
15.00-15.20: Thee
15.20-16.10: Lezing Prof.dr. K. Apt (CWI, Universiteit van Amsterdam)
Titel: A Primer on Constraint Programming
16.10-16.40: Algemene ledenvergadering NVTI

Samenvattingen van de voordrachten
Prof.dr. R. Milner:
The Flux of Interaction

The lecture will be about a simple graphical model for mobile computing. Graphical or geometric models of computing are probably as old as the stored-program computer, possibly older. I don't know when the first flowchart was drawn; but since then, many other graphical models have followed. In particular, in the 1960s Petri nets made a breakthrough in understanding synchronization and concurrent control flow.

There are now many process calculi concerned with mobility, both of process links and of the processes themselves. While these calculi were evolving, I have tried with colleagues to distill their shared mobile geometry into a notion of action graph and an associated calculus. These are based upon a notion of molecule, a node in which further graphs may nest. A signature determines a set of molecule types, and a set of reaction rules determines what configurations of molecules can react. Depending on the choice of signature, a molecule and its contents may represent a data constructor, a cryptographic key, a message, a physical location, a lambda-abstraction, a program script, an administrative region, a term of the pi-calculus, an ambient (in the sense of Cardelli and Gordon), and so on. In the lecture I shall outline the behavioural theory of action graphs.

Prof.dr. A. Siebes:
BioInformatics

The end of the Human Genome project is the beginning of an even bigger challenge: proteomics. We know the book, but what does it mean? Using a few concrete projects, I will highlight some of the interesting computer science problems that need to be solved before the biologists can solve their problems.

Prof.dr. K. Apt:
A Primer on Constraint Programming

The last four years I have been studying constraint programming. It took me some time to understand the subject but I believe I can now explain its essence to a congenial audience in one hour.

An interesting aspect of constraint programming is that in some areas like combinatorial optimization and numerical analysis it occasionally led to an improvement of the state of the art solutions that were obtained by means of the conventional methods. Another is that it allows us to view some well-known techniques, like the resolution method or Gaussian elimination as instances of constraint programming techniques.

To understand why it is the case we explain the concepts of constraint satisfaction problems, constraint propagation, and constraint solvers and put them together in a coherent framework.

Finally, we clarify the special nature of constraint programming languages by focusing on the use of variables. In computer science a variable stands for a place holder for a value, in mathematics for an unknown. In most constraint programming languages both uses are present.

Prof.dr. D. Harel:
On the Aesthetics of Diagrams

Given the extensive move towards visual languages and visual interfaces in the design and usage of computerized systems, the need for algorithmic procedures that produce clear and eye-pleasing layouts of complex diagrammatic entities arises in full force. The main part of this talk addresses a modest, yet still very difficult version of the problem, in which the diagrams are merely general undirected graphs with straight-line edges. We describe work done with a series of students over the last 12 years, starting from a system based exclusively on simulated annealing, and culminating with a far more powerful approach, based on a multi-level scheme, which can deal successfully and rapidly with extremely large graphs containing thousands of nodes. We then turn to a problem of different nature, the layout of "blob" hierarchies (nested rectangular shapes), with applications to window systems, web-page design, and higraph-based languages, such as object models and statecharts.

Lidmaatschap NVTI

Aan het lidmaatschap zijn geen kosten verbonden; u krijgt de aankondigingen van de NVTI per email of anderszins toegestuurd. Wilt u lid van de NVTI worden: u kunt zich aanmelden bij het contactadres beneden (M. Bruné, CWI), met vermelding van de relevante gegevens: naam, voorletters, affiliatie indien van toepassing, correspondentieadres, email, URL, telefoonnummer.

Lunchdeelname

Het is mogelijk aan een georganiseerde lunch deel te nemen; hiervoor is aanmelding verplicht. Dit kan per email of telefonisch bij Mieke Bruné (mieke@cwi.nl, 020-592 4249), tot een week tevoren (15 maart). De kosten kunnen ter plaatse voldaan worden; deze bedragen f 25,-. Wij wijzen erop dat in de onmiddellijke nabijheid van de vergaderzaal ook uitstekende lunchfaciliteiten

gevonden kunnen worden, voor wie niet aan de georganiseerde lunch wenst deel te nemen.

5 Mededelingen van de onderzoekscholen

Hieronder volgen korte beschrijvingen van de onderzoekscholen:

- Instituut voor Programmatuurkunde en Algoritmiek
- Landelijke Onderzoekschool Logica

5.1 Institute for Programming research and Algorithmics

The research school IPA (Institute for Programming Research and Algorithmics) educates researchers in the field of programming research and algorithmics. This field encompasses the study and development of formalisms, methods and techniques to design, analyse, and construct software systems and components. IPA has three main research areas: Algorithmics & Complexity, Formal Methods and Software Technology. In 2000, the composition of IPA was unchanged. Researchers from eight universities (University of Nijmegen, Leiden University, Eindhoven University of Technology, University of Twente, Utrecht University, University of Groningen, Vrije Universiteit Amsterdam, and the University of Amsterdam), the CWI and Philips Research (Eindhoven) participated.

IPA has two multi-day events per year, which focus on upcoming new subjects. In 2000, the Spring Days were on UML, the Fall Days on Applied Algorithmic Design. UML (Unified Modeling Language) has rapidly become a de facto standard in the software industry. The existence of a widely used and powerful language offers new possibilities for connecting academic research to industrial practice. The Spring Days sought to explore these possibilities. As it turns out, the UML standard gave rise to many interesting research questions which are also of industrial significance, i.e. with respect to the implementation of the standard in development support tools. These Fall Days featured the Algorithmics research in IPA that deals with "traditional algorithms" (as opposed to the research in Natural Computation that addresses genetic algorithms, neural nets and quantum computing). These algorithms have a wide range of applications, and new applications are constantly being developed. Designing these algorithms raises many interesting research questions on which quite a few people in IPA are working, often in close collaboration with industry. The Fall Days highlighted this work on Applied Algorithm Design, with days dedicated to different application areas, such as logistics, multi-media (algorithms for storage and retrieval, quality of service), and digital networks (algorithms for resource management, Web-algorithms and security).

On the European front, we saw the build-up of the European Educational Forum. IPA cooperates in EEF with BRICS (Denmark), TUCS (Finland) and UKII (United Kingdom). 2000 saw the addition of Italian, German and French consortia, so that we can truly speak of a research school of European dimensions. EEF activities included the Trends School on Formal Methods and Performance Analysis, organised by our Twente partners, the Foundations School in Deduction and Theorem Proving in Edinburgh and the School on Foundations of Security Analysis and Design in Bertinoro, Italy. Besides, we sponsored the Fifth Dutch Proof Tools Day, held in Gent Belgium.

Ph.D. Defenses in 2000

K.M.M. de Leeuw. *Cryptology and Statecraft in the Dutch Republic.*
Faculty of Mathematics and Computer Science, UvA

T.E.J. Vos. *UNITY in Diversity. A stratified approach to the verification of distributed algorithms.* Faculty of Mathematics and Computer Science, UU

W. Mallon. *Theories and Tools for the Design of Delay-Insensitive Communicating Processes.*
Faculty of Mathematics and Natural Sciences, RUG

W.O.D. Griffioen. *Studies in Computer Aided Verification of Protocols.*
Faculty of Science, KUN

P.H.F.M. Verhoeven. *The Design of the MathSpad Editor.*
Faculty of Mathematics and Computing Science, TUE

J. Fey. *Design of a Fruit Juice Blending and Packaging Plant.*
Faculty of Mechanical Engineering, TUE

M. Franssen. *Cocktail: A Tool for Deriving Correct Programs.*
Faculty of Mathematics and Computing Science, TUE

P.A. Olivier. *A Framework for Debugging Heterogeneous Applications.*
Faculty of Natural Sciences, Mathematics and Computer Science, UvA

E. Saaman. *Another Formal Specification Language.*
Faculty of Mathematics and Natural Sciences, RUG

Activities in 2001

In 2001, the Spring Days will be on April 18-20 in Heeze, on the topic of Security. Further details will become available on our recently reconstructed web-site, see:
http://www.win.tue.nl/cs/ipa/activities/springdays2001/. Further, we will have our basic courses on Software Technology, in the first half of the year, and on Algorithms and Complexity in the second half.

In EEF, there is the Trends School on Software Architecture in Finland in August, and the Foundations School on Logical Methods in Denmark in June (*http://www.brics.dk/LogicsSchool01/*). Further, there is the School on Foundations of Wide Area Network Programming in Lipari and the School on Process Algebras in Bertinoro, both in Italy
(*http://lipari.dmi.unict.it/Lipari/index.asp* and *http://www.cs.unibo.it/~aldini/sfm01pa/*).

Addresses

Visiting address
Eindhoven University of Technology
Main Building HG 7.17
Den Dolech 2
5612 AZ Eindhoven
The Netherlands

Postal address
IPA, Fac. of Math. and Comp. Sci.
Eindhoven University of Technology
P.O. Box 513
5600 MB Eindhoven
The Netherlands

tel. (+31)-40-2474124 (IPA Secretariat)
fax (+31)-40-2475361
e-mail ipa@tue.nl url http://www.win.tue.nl/cs/ipa/

5.2 Dutch Research School in Logic (OzsL), door: Jan van Eijck

The Dutch Research School in Logic (OzsL) is active in three main areas: mathematical logic, logic in linguistics and philosophy, and logic in computer science. Formal participants in the School are the University of Amsterdam, the Free University in Amsterdam, the University of Utrecht, the University of Groningen, and Tilburg University. In addition, there are numerous associate members, to cater for the need of those who have active scientific links with the OzsL community, while politicial reasons argue against full participation. The general policy of the school is to foster cooperation rather than competition with neighbouring schools, and associate membership is open for all our neighbours.

International cooperation agreements exist with Stanford University, the University of Edinburgh, the University of the Saarland, and the University of Stuttgart. Funding is available for visitor exchanges within this international network, and regular international workshops take place within the network.

The Ph.D. courses offered by the school fall in two categories: courses that are part of the 'school week curriculum', and master classes. School weeks are offered twice a year. To give an idea of the contents, here is a recent sample:

Autumn 2000 School Week, Nunspeet The program consisted of the following:

- A one-day event where staff members gave short short accounts of their current research interests: *Adult Accolade "{".*
- A tutorial on co-algebra.
- A tutorial on finite model theory and modal logic.
- A special session on Logic in Linguistics with presentations by a logician and a computational linguist, and discussion with a sample of Dutch linguists.
- An afternoon with Samson Abramsky, with an overview of recent work on logic and games in theoretical computer science.
- A Logic and Games event, with presentations by school members interested in applications of game theory.
- A one-day closing event where PhD students gave short short accounts of how their research is going: *Accolade "}".*

Further details can be found in the web archive of the School, at address
`http://www.ozsl.uva.nl/archive.html`.

The yearly Accolade Event, that always takes place in combination with the Autumn School Week is an occasion where PhD students within the school present their work to the outside world in an informal setting. The purpose of Accolade is to inform the community about how individual Ph.D. projects were progressing, and to stimulate mutual interest. Accolade has as its single aim to inform the Dutch logic community about how the Ph.D. projects within the School are going, irrespective of whether these projects are in an initial, intermediate or final stage of research. Participants and audience are enthousiastic about the formula, where PhD students get useful response in a supportive setting.

Accolade for Adults is a recent addition to the School activities. This complement event to Accolade for PhDs, also known under various silly nicknames, is meant to create an opportunity for staff members within the School to give brief outlines of their research, for an audience consisting of their colleagues and the PhD students of the School.

OzsL issues LIN ('Logic in the Netherlands'), a newsletter for the Dutch logic community (in the broadest possible sense) that appears at rather irregular intervals, both on paper and electronically. Subscription is free of charge; please send an email to the OzsL Office Manager dr Peter Blok at `pblok@wins.uva.nl` to subsribe. Further information about the school and its activities is available electronically, at `http://www.ozsl.uva.nl`.

OzsL is deeply involved in European Summer School activities in logic: the well known *ESSLLI* (European Summer School in Logic, Language and Information) meetings. The next (13th) ESSLLI School takes place in Helsinki, 13–24 August 2001. Further information can be obtained from `http://www.helsinki.fi/esslli`. Here is a quote from the web page:

> The main focus of ESSLLI is the interface between linguistics, logic and computation. Courses, both foundational, introductory and advanced, cover a wide variety of topics within six areas of interest: Logic, Computation, Language, Logic and Computation, Computation and Language, Language and Logic. Previous summer schools have been highly successful, attracting around 500 participants from Europe and elsewhere. The school has developed into an important meeting place and forum for discussion for students, researchers and IT professionals interested in the interdisciplinary study of Logic, Language and Information. In addition to courses, workshops and evening

lectures there will be special events, a student session and a social program. The number of courses offered is over 50.

Finally, OzsL collaborates with VvL, the Dutch Association for Logic (Vereniging voor Logica) in organizing activities for a broader community with an interest in logic. See the web site of VvL, at `http://www.cwi.nl/vvl`.

6 Wetenschappelijke bijdragen

6.1 Specifying Internet applications with *DiCons*: J.C.M. Baeten, H.M.A. van Beek, S. Mauw

Eindhoven University of Technology

Specifying Internet applications with *DiCons*

J.C.M. Baeten
Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands
josb@win.tue.nl

H.M.A. van Beek
Eindhoven Embedded Systems Institute (EESI), Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands
harm@win.tue.nl

S. Mauw
Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands
sjouke@win.tue.nl

Keywords
Internet applications, language design, distributed consensus, *DiCons*.

ABSTRACT
It is not easy to build Internet applications with common techniques, such as CGI scripts and Perl. Therefore, we designed the *DiCons* language, which supports the development of a range of Internet applications at the appropriate level of abstraction. In this paper we discuss the design of *DiCons*, we give an overview of the tool support and we explain the language by means of an example.

1. INTRODUCTION
Some trends concerning the development of new Internet applications can be observed. First of all, the Internet and applications of Internet are developed with a tremendous speed. The first to come with an interesting application sets the standard for that application area. Many new services are realized, for example applications which support auctions or voting via Internet.

Secondly, large portals replaced the *old style* search engines. They provide functionality that goes beyond mere guidance through the Internet. The longer that the visitor stays at the portal site and the more often that he uses functionality provided by the portal, the higher the income from advertisements will be. Therefore, portals must offer interesting applications and must keep their functionality up to date. This does not only imply that portals must maintain a large set of applications, but also that they must be able to rapidly develop new services. Short *time to market* is an important asset.

The third observation is that the number of commercial transactions on the Internet is growing. Security and dependability are important factors at all levels of interaction. Apart from proper use of cryptographic techniques, this also requires that the protocols by which information is exchanged are correct. A voting system, e.g., must guarantee that the winner is actually the candidate that has received most support.

To appear in the conference Proceedings of the 16th ACM Symposium on Applied Computing (SAC 2001), Las Vegas, USA, March 2001.

We want to be able to quickly develop secure and dependable Internet applications. Some problems that occur are, firstly, that several languages are involved, such as html, cgi scripts, and other scripting languages. Secondly, the level of abstraction of the language used often does not correspond to the level on which we think about an application: there is no C-primitive for filling out a Web form. Thirdly, current practices do not lend themselves to validation or verification.

Thus, our goal is to develop a language at the right level of abstraction, that is amenable to (formal) validation or verification. In order to make the problem more concrete and the solution more feasible, we limit the class of applications we consider. First of all, we consider applications where several users strive to reach a common goal without having to meet. We call this kind of applications *distributed consensus applications*. A central location on an Internet server should support this. We are interested in asynchronous communication, as exemplified by the sending of e-mails and Web forms. Users do not communicate directly but only communicate with the central application. Finally, we only want to use standard techniques, so the user does not require special programs, software or plug-ins. An Internet connection, e-mail and a Web browser should suffice, on any hardware/software platform.

In this paper a new specification language *DiCons* (*Distributed Consensus*) is introduced to specify Internet applications for *distributed consensus*. Major characteristic of this class of protocols is that a number of users strive to reach a common goal (e.g. make an appointment, evaluate a paper, select a "winner"). The problem is that the users do not want to physically meet to solve their goal, nor will there be any synchronized communications between the users. A central system, viz. an Internet application, must be used to collect and distribute all relevant information.

This class of applications was the starting point for developing our language. The language must both be expressive enough and concrete. In order to be applicable to an appropriate range of problems, it must have the right expressive power. The language must be concrete enough, such that automatic generation of an executable is feasible.

Typical examples of applications that our research targets at, are: Meeting scheduler, election support system, auction, and gift selection. These examples have in common that they support a task which is algorithmically simple but requires many interactions. This task is taken over by a central application, handling all interactions with the users. In this paper, we illustrate this by the example of a gift selection system.

The purpose of this paper is to give a description of the language *DiCons* and the tools developed for it. To this end, we first give an overview of the design decisions we took in order to arrive at this language and its prototype tools. Next, we illustrate the use of *DiCons* by the example of the gift selection system. After that, we compare our approach with other methods and techniques. Finally, we finish with some concluding remarks and ideas for the further development of the language and tools.

2. DESIGN OF *DICONS*

In this section we will discuss the considerations that led to the current design of the *DiCons* language and we describe the basic ingredients of *DiCons*.

2.1 Restrictions

In order to not have to face the complete problem of writing Internet applications in general we restrict our problem setting in several ways. First of all, we focus on a class of applications which is amenable to formal verification with respect to behavioral properties. This means that the complexity of the application comes from the various interactions between users and a system, rather than from the data being exchanged and transformed. Implications for the design of the language are that the primitive constructs are *interactions*, which can be composed into complex behavioral expressions. Furthermore, it implies that the development of the language and its formal semantics must go hand in hand. Nevertheless, we will not discuss semantical issues in the current paper.

A further restriction follows from the assumption that although the users work together to achieve some common goal, there will be no means for the users to communicate directly with each other. We assume a single, central application that follows a strictly defined protocol in communication with the users.

The last consideration with respect to the design of *DiCons* is that we want to make use of standard Internet technology only. Therefore, we focus on communication primitives such as e-mail and Web forms. This means that a user can interact with the system with a standard Web browser, without the need for additional software such as plug-ins. Of course, it must be kept in mind that the constructs must be so general as to easily support more recent developments, such as ICQ or SMS messages. Currently, we only consider asynchronous communication between client and server.

2.2 Overview of language constructs

Bearing above considerations with respect to the application domain and available technology in mind, we come to a description of the basic constructs of *DiCons*. We will first list the language ingredients and later discuss these in more detail, without precisely defining their syntax and semantics. The example in Section 4 will serve to show the flavor of the *DiCons* syntax and the way in which the language can be used.

users and roles The first observation is that, since an application may involve different users, the application must be able to identify users. Moreover, since different users may want to use the system in the same way, it must be possible to group users into so-called *roles*.

interactions We have to identify the communication primitives, which we will call *interactions*. They form the basic building blocks of the behavioral descriptions. Interactions are abstract descriptions which are identified by their name and may carry input and output parameters.

behavior A number of interactions with the same user may be combined to form a *session*. Sessions and interactions can be composed into complex behavioral descriptions which define an *application*.

presentations The abstract interactions are represented to the user by means of concrete communication means, such as e-mail and Web forms. This is called the *presentation* of an interaction.

data In order to transform (user) data and keep state information, we need a means to define and manipulate data (expressions, variables, data structures, etc.)

2.3 Users and roles

A user is an entity that can interact with the system. A user has three attributes: a name (for reference), an e-mail address (in case e-mail communication is desired), and a password (in case user authentication is needed). Users are grouped according to their role. Users with the same role are offered the same interaction behavior. In *DiCons* roles can be defined and variables can be declared which denote users with a given role.

2.4 Interactions

The basic problem when defining the interaction primitives is to determine the right level of abstraction. Taking, e.g., an http request as a primitive interaction will lead to programs which are too detailed. On the other hand, if we would define a complete user session as a primitive interaction, we could not deal with the variety of different sessions that occur in an application.

In order to get a feeling of the level of abstraction which is optimally suitable, look at Figure 1. In this drawing we sketch a typical scenario of an Internet application which is called the *Meeting Scheduler* (see [22]). This is an application which assists in scheduling a meeting by keeping track of all suitable dates and sending appropriate requests and convocations to the intended participants of the meeting.

The drawing is a so-called *Message Sequence Chart* (MSC, see [17]), which is a standardized visual language, especially suited for requirements engineering. The example shows

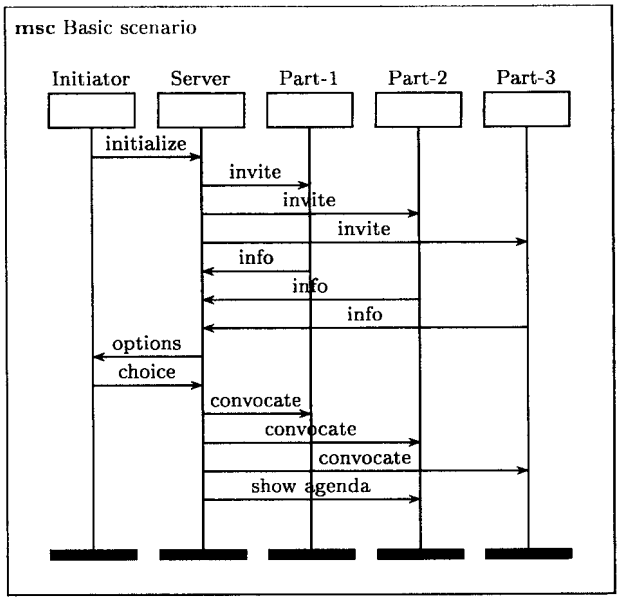


Figure 1: A scenario of an Internet application

that we have two roles, viz. initiator and participant. In this scenario, there is only one user with role initiator, while there are three users with role participant. The MSC shows that the initiator starts the system by providing it with meeting information. Next, the system sends an invitation to the participants who reply by stating which dates suit them. After collecting this information, the system informs the initiator about the options for scheduling the meeting and awaits the choice made by the initiator. Finally, the system informs the participants about the date and offers the users to have a look at the agenda. Only participant 2 is interested in the agenda.

This example nicely shows at which level of detail one wants to specify such an application. The arrows in the diagram represent the basic interaction primitives. First, look at the *invite* messages. Since the participants don't know that they will be invited for a meeting, the initiative of this interaction is at the server side. The way in which a server can actively inform a client is by sending an e-mail. This interaction only contains information transmitted from the server to the user. The messages *options* and *convocate* are also implemented as e-mails.

Next, look at message *info*. This interaction is initiated by the user and is best implemented as a Web form supplied by the server, on request of the user and filled in by the user. The message *choice* also stands for a Web form being filled in.

The last message, *show agenda* contains information sent by the server to the user, on request of the user. This is simply the request and transmission of a non-interactive Web page.

Finally, we look at the first message, *initialize*. The initiator has to supply the system with various kinds of information, such as a list of proposed dates and a list of proposed participants. This will probably be implemented as a dialogue between the user and the system in the form of a series of Web forms. This is called a *session*.

We summarize the three basic interaction schemes in Figure 2. Notice that the third scheme, the session, consists of a series of more primitive interactions. It starts with a client requesting a form and submitting it after having it filled in. This is the interaction which starts the session. Next, comes a series of zero or more submissions of Web forms. These are interactions which come in the middle of a session. And, finally, the session ends with the server sending a simple Web page after the last submission of the client.

In *DiCons* we have constructs for these five interaction primitives. We have used a naming scheme for the interaction primitives which is based on their properties. First, we make a distinction based on the flow of information. If the information goes from the server to the client, we call this a *server push*, while if the information flows to the server, we call this a *server pull*. Notice that we reason from the viewpoint of the server in this respect.

The second distinction which we make is on which party takes the initiative for the interaction. Still reasoning from the viewpoint of the server we consider an *active* communication, which means that the server takes the initiative, a *reactive* communication, which means that the client takes the initiative, and a *session oriented* communication, which means that the communication is a response from the server to a prior submission of a Web form by the client.

Finally, notice that we extend the interaction primitives with parameters to express which information is being transmitted. An output parameter denotes information sent by the server to the client, while an input parameter is a variable in the data space of the server which will contain the information sent by the client to the server.

The notation for our communication primitives is given below.

active server push The server takes the initiative to send information (for o_i $(0 \leq i \leq n)$ output parameters):

mail to *client* $\leftarrow$ message($o_0, \ldots, o_n$)

reactive server push The server sends a Web page on request of the client (for o_i $(0 \leq i \leq n)$ output parameters):

client $\leftarrow$ message(**out** $o_0, \ldots,$ **out** o_n)

reactive server pull The server sends a Web form on request of the client. After that, the client submits the filled in form. This interaction denotes the starting of a session. (for i_k $(0 \leq k \leq m)$ input parameters, o_k $(0 \leq k \leq n)$ output parameters and v_k $(0 \leq k \leq p)$ input/output parameters):

start session of *client* $\rightarrow$ message(**in** $i_0, \ldots,$ **in** i_m, **out** $o_0, \ldots,$ **out** o_n, **var** $v_0, \ldots,$ **var** v_p)

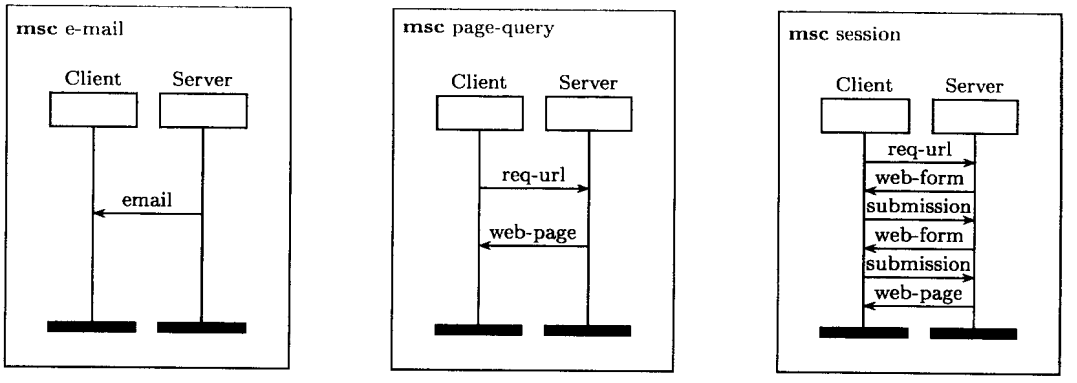


Figure 2: Interaction primitives

session-oriented server pull The server sends a Web form to the client as a response to a prior form submission by the client. After that, the client submits the filled in form. This interaction is repeated in the middle of a session. (for i_k $(0 \leq k \leq m)$ input parameters, o_k $(0 \leq k \leq n)$ output parameters and v_k $(0 \leq k \leq p)$ input/output parameters):

session of *client* $\rightarrow$ message(**in** $i_0, \ldots$, **in** i_m, **out** $o_0, \ldots$, **out** o_n, **var** $v_0, \ldots$, **var** v_p)

session-oriented server push The server sends a non-interactive Web page to the client in response to a prior form submission by the client. This interaction is the last interaction of a session. (for o_i $(0 \leq i \leq n)$ output parameters):

end session of *client* $\leftarrow$ message(**out** $o_0, \ldots$, **out** o_n)

Please notice that in our list of interaction primitives we did not mention the *active server pull*. The reason for this is simply that with standard Internet technology this interaction cannot be implemented. A Web server cannot take the initiative to obtain information from a client.

2.5 Behavior

Now that we have defined the basic interaction primitives, we can discuss the means to compose them into sessions and applications. An application describes the protocol to be executed by the server. A number of standard programming language constructs are supported in *DiCons*. We mention the following: sequential composition (denoted by a semi-colon), conditional branching (if-then-else-fi), repetition (for-all-do-od, and while-do-od, which is a parallel repetition), Since in most applications that we have studied users have to react before a given deadline, we have included a time-out construct in *DiCons* (until-do-od, which means that the body of this expression may execute until the given deadline). Finally, in order to manipulate the internal state of the application, we have assignments to variables and procedure calls. A session is simply a program fragment with the requirement that execution starts with a session-start interaction and ends with a corresponding session-end interaction.

2.6 Presentations

The interactions which are composed into a *DiCons* application are abstract in the sense that they only carry a name and possibly some parameters. Additional information is needed to determine how the interaction is implemented. In case of an e-mail, we need to specify the addresses of the sender and the receiver, the subject field, the body text and the places where the values of the output parameters must be filled in.

In case of a Web form, we must also define the fields where the user can type in values which are stored in the input parameters of the interaction. Furthermore, *DiCons* supports the inclusion of Java scripts which can put syntactic restrictions on the input provided by the user. Other supported features are pull-down selection menus, submit buttons, radio buttons and check boxes.

2.7 Data

Storing and manipulating data occurs at several places in a *DiCons* application. Therefore, a well equipped data language must be part of *DiCons*. Many programming languages have been developed to support the manipulation of data, so, rather than developing our own dedicated data language, we decided to include an existing language, namely Java [12]. The main reason for selecting Java, lies in its popularity in the Internet community, but also implementation issues made us decide for Java (because we use Java servlets, see Section 3).

In order to make *DiCons* as independent from the chosen data language as possible, we have defined the language in such a way that the included data language and the other parts of the language are orthogonal. Java fragments are only allowed in the definition of functions and procedures. Interaction with the other parts of the language takes place by calling these functions. In this way, Java can be easily substituted by other languages, such as C.

3. TOOLS FOR *DICONS*

We make use of several existing (Internet) techniques. First of all, we make use of Java servlets [16, 28]. These servlets generate HTML pages and HTML forms [25]. If data constraints are included into Web form, these constraints are checked by a piece of JavaScript code [14] which makes use of the regular expression, specified in the Perl/JavaScript regular expression syntax [11].

3.1 JavaCC

To implement a parser we have chosen to use the Java parser generator *Java Compiler Compiler (JavaCC)* [23]. This choice is made because we are specifying an Internet application and Java is the Internet specification language par excellence. JavaCC is a parser generator that produces parsers in Java from grammar specifications written in a lex/yacc-like manner.

We have implemented a package of classes which specify the different parts of the language: roles, types, variables, functions, interactions, sessions and the execution. After parsing an application, an object of type *DiConsApplication* is created. This object consists of different objects, all specifying one part of the application. These objects all have a method to convert that specific part of the application to a piece of Java code. By putting all these pieces together we get a Java application, viz. a Java servlet. This servlet can be compiled to Java byte-code by using a regular Java compiler. The file containing Java byte-code can be interpreted by a Web server.

3.2 Technical Aspects

In this section we will discuss some aspects of our specification which are non-trivial to implement. Since we cannot have multiple executions of one single servlet simultaneously we have to implement some kind of instance management and session management. The problem is that we want to be able to start several instances of some *DiCons* application. These instances must run independently and have disjoint state spaces. Within an instance of an application, several users may start parallel or overlapping sessions. Such sessions share the same data space.

3.2.1 Instance management

The servlet API does not implement instance management in the way we need it. It implements sessions using client-side cookies. Since one instance might concern more than one client, cookies cannot solve our instance management.

We introduce a new class *ServletInstance* in which all data concerning one instance can be stored. Furthermore, we add a variable containing the collection of available instances to our servlet. Each instance gets its own unique identifier. If one accesses the servlet without referring to an instance, a new instance is created. During a session, all Web forms are extended with a hidden variable containing the corresponding instance identifier. Submitting a Web form now results in including this identifier in the posted data. One can also call a servlet using a rewritten URL. This URL is extended with a query string containing an instance identifier. Calling the servlet like this results in continuing an instance if this is possible. If this instance does not exist the instance identifier is ignored and a new instance with a new, unique identifier is create. This identifier is composed of a letter I followed by eight randomly chosen digits.

Each time a servlet is called it checks whether an instance identifier is passed. If so, it tries to load that instance's data and continue the instance's execution. Otherwise, a new instance is created.

3.2.2 Session management

Session management support is built into the servlet API by using cookies. However, these techniques do not answer our needs. By using cookies we do not have the ability to run multiple simultaneous sessions using one and the same Web browser. Though this is not such a big shortcoming, one can turn off cookie usage in most of the Web browsers. This cookie problem can quite easily be solved by implementing sessions in the same way as we implemented instances.

We introduce a new class *ServletSession*. Sessions specified in the session part of the application are implemented as subclasses of this class. Sessions all have a session identifier which is unique for the instance it takes part in. This identifier is composed of a letter S followed by eight randomly chosen digits.

Again, we extend Web forms with a hidden variable containing the session identifier. Since sessions are started with a reactive pull and continued with session-oriented pulls it is not needed to use rewritten URLs within one session: pulls always return a Web form. Each instance contains a variable in which the collection of its active sessions is stored.

A result of this way of implementing sessions is that parallel sessions within one instance are automatically implemented. However, we do have to take care that parallel sessions do not interfere while accessing instance dependent data. This is prevented by synchronizing data access.

Each time a servlet is called it checks whether a session identifier is passed. If so, it checks whether the session occurs in the corresponding instance and continues the session if that is possible. If the session cannot be found or if no session identifier is passed, a new session is started.

4. EXAMPLE: THE GIFT SELECTION

In this section we give an example of a way to distribute gifts over invitees for a marriage. We specify an Internet application via which this distribution takes place. We do not give a full specification. Instead, we give parts of the specification which will be sufficient to get an idea of the way in which the different part of the application are specified.

First of all, we have to specify which roles are applicable to the problem. An initiator must specify which gifts can be given away and who are invited for the marriage. The invitees must be able to select a gift they want to donate to the bridal couple. This means that we have two roles: *Initiator* and *Invitee*.

```
role
  Initiator;
  Invitee;
end role
```

Next, we specify which variables and functions we make use of. We need a variable to refer to the initiator and one to refer to the invitees. Furthermore, the gifts (of type String) are stored in a variable. We make use of a deadline before which the invitees have to select the gift they want to give.

```
var initiator: Initiator;
var invitees: set of Invitee;
var gifts: set of String;
var deadline: Deadline;
```

Functions are also specified in the data part. The bodies of the functions are specified in the Java language. We can use the variables specified before. Variables which represent a set are implemented as objects of the Java *Vector* class. Therefore, we can make use of the methods of this class in the bodies of the functions. Some examples of functions we have specified are given below.

```
function process_selection(gifts: set of String,
     gift: String): String
  = java
     if (!gift.equals("") &&
         (gifts.indexOf(gift)>-1)) {
       gifts.removeElement(gift);
       return "yes";
     } else
       return "no";
   end java;

function gifts_left(gifts: set of String): Boolean
  = java
     return !gifts.isEmpty();
   end java;
```

Next, we specify the Web pages/forms and e-mails we use to interact with the users. We declare the kind of each interaction and the role a user must have to interact. A Web page/form is specified by its title and body, an e-mail by its sender, receiver, subject and contents. We use plain text and references to input/output parameters. A Web form which we use to ask the initiator to insert a deadline is given below. In the interaction, a regular expression is added to check the syntax of the text which is typed out in the input field. If the text does not answer the syntax, a message is shown and the text must be altered until it does satisfy the syntax.

```
session of Initiator → set_deadline(
     in deadline: Deadline) =
 { title:
     text: "Gift selection";
   body:
     text: "Insert deadline (dd-mm-yyyy hh:mm:ss):";
     input: deadline
            check "/^\d\d-\d\d-\d\d\d\d
                    \d\d:\d\d:\d\d$/"
            else "Incorrect date format.";
 };
```

A specification of an e-mail is given below. The e-mail is sent to each invitee. He is asked to visit the application's URL, log in and select a gift. A "\n" specifies a line break.

```
mail to Invitee ← invitation_email(
     out initiator: Initiator, out deadline: Deadline,
     out invitee: Invitee, out gifts: set of String) =
 { from:
     output: initiator.email;
   to:
     output: invitee.email;
   subject:
     text: "Invitation for gift selection.";
   contents:
     text: "Hello ";
     output: invitee.name;
     text: ",\n\nYou are invited to select a gift.
            \n\nVisit the following url:\n\n";
     output: URL;
     text: "\n\nThe gifts are:\n";
     output: gifts;
     text: "\n\nDeadline before which you have to
            select your gifts:\n";
     output: deadline;
     text: "\n\nUse the following name and password
            to log in:\nName:";
     output: invitee.name;
     text: "\nPassword: ";
     output: invitee.password;
     text: "\n\ngreetings, ";
     output: initiator.name;
 };
```

After specifying all interactions, we have to specify the different sessions. Each session has a name. A session is specified by a sequence of (inter)actions. We have to specify two sessions.

First of all, we specify the *initialization* session. In this session the initiator is asked to insert all relevant data which is needed for the gift selection, i.e. his name and e-mail address, the set of gifts, the set of invitees and the deadline before which the sessions with the invitees must take place.

```
initialization =
  { start session of initiator → set_initiator(initiator);
    while incorrect_deadline(deadline) do
      session of initiator → set_deadline(deadline);
    od;
    s = yes();
    while equals_yes(s) do
      session of initiator →
          add_invitee(invitees, invitee, s);
      process_invitee(invitees, invitee);
    od;
    s = yes();
    while equals_yes(s) do
      session of initiator → add_gift(gifts, gift, s);
      process_gift(gifts, gift);
    od;
    for all j ∈ invitees do
      mail to j ←
          invitation_email(initiator, deadline, j, gifts);
    od;
    end session of initiator ← thank_you_initiator();
  };
```

Furthermore, we specify the *selection* session. In such a session an invitee is asked to select a gift. First, the invitee has to log in using his name and password (this is indicated by the attribute *authenticate from*). After selecting a gift, a check is done. If the gift is still available it is removed from the set of available gifts and attributed to the invitee. If it has been attributed to another invitee a new gift must be selected.

```
selection =
  { start session of invitee →
        authenticate from invitees;
    session of invitee → select_gift(gifts, gift);
    s = process_selection(gifts, gift);
    while equals_no(s) do
      session of invitee → again_select_gift(gifts, gift);
      s = process_selection(gifts, gift);
    od;
    end session of invitee ← thank_you_invitee(gift);
  };
```

Finally, we have to specify in which order the sessions must take place. The application starts with the *initialization* session. After that, *selection* sessions can take place as long as the deadline has not been reached and gifts are available for distribution.

```
application

  session initialization;
  until deadline do
    while gifts_left(gifts) do
      session selection;
    od;
  od;

end application
```

This example has been implemented and can be executed as a Java Servlet. The *while* construction which is used in the final part of the specification is implemented as a parallel composition. This means that a number of *selection* sessions can be executed in parallel. Since it is possible to select a gift which, in the meantime, has been selected by another invitee in a parallel session, we have added the check to the *selection* session.

5. RELATED WORK

We introduced a specification language for a specific class of Internet applications, viz. applications for distributed consensus. There are many different languages to specify Internet applications, but as far as we know, none of them is specifically designed to develop such applications. We will discuss some of them and show in what way they agree with or differ from *DiCons*.

Closest to our work is the development of the Web-language *Mawl*, [1, 19]. This is also a language that supports interaction between an application and a single user, and adds a state concept to HTML. Mawl provides the control flow of a single session, but does not provide control flow across several sessions (the only thing that persists across sessions are the values of global variables). This is a distinguishing feature of *DiCons*: interactions involving several users are supported. On the other hand, Mawl does allow several sessions with a single user to exist in parallel, using an atomicity concept to execute sequences of actions as a single action. Mawl does not use Java servlets.

Groupware is a technology designed to facilitate the work of groups. This technology may be used to communicate, cooperate, coordinate, solve problems, compete, or negotiate. Groupware can be divided into two main classes: asynchronous and synchronous groupware. Synchronous groupware concerns an exchange of information, which is transmitted and presented to the users instantaneously by using computers. An example of synchronous groupware is chatting via the Internet. On the other hand, asynchronous groupware is based on sending messages which do not have to be read and replied to immediately. Examples of asynchronous groupware that can be specified in *DiCons* are work-flow systems to route documents through an office and group calendars for scheduling projects. More information on groupware can be found in [27].

Visual Obliq [3] is an environment for designing, programming and running distributed, multi-user GUI applications. Its interface builder outputs code in an interpreted language called *Obliq* [5]. Unlike *DiCons* applications, Obliq applications do not have to run on one single server: an application can be distributed over several so-called sites. After setting up a connection, sites can communicate directly. In this way, an application can be partitioned over different servers. Another difference with respect to *DiCons* is that a client has to install a special interpreter to view Visual Obliq applications whereas *DiCons* makes use of standard client-side techniques like HTML pages which can be viewed using a Web browser. In [4], embedding distributed application in a hypermedia setting is discussed and in particular how applications generated in the Visual Obliq programming environment are integrated with the World Wide Web. Here,

a Web browser is used to refer to a Visual Obliq application, but it must still be viewed using an interpreter.

Collaborative Objects Coordination Architecture (COCA) [21] is a generic framework for developing collaborative systems. In COCA, participants are divided into different roles, having different rights like in *DiCons*. Li and Muntz [20] used this tool to build an online auction. A COCA Virtual Machine runs at each client site to control the interactions between the different clients. On the other hand, any client connected to the Internet can communicate with a *DiCons* application without having to reconfigure his machine.

The *Describing Collaborative Work Programming Language (DCWPL)* [7] helps programmers to develop customizable groupware applications. DCWPL does not concern the computational part of an application. As in *DiCons*, this part is specified in a computational language like Java, Pascal or C^{++}. A DCWPL application also runs on an interpreter, here called control engine. DCWPL is based on synchronous groupware in contrast to *DiCons* in which the asynchronous aspect is more important.

Further, there are languages that allow to program browsing behaviour. These, for instance, allow to program the behaviour of a user who wants to download a file from one of several mirror sites. For so-called *Service Combinators* see [6, 18]. A further development is the so-called *ShopBot*, see [8].

Our implementation is based on existing Internet programming techniques, viz. Java servlets and HTML. In Udell's book on groupware [27] an Internet vote is implemented using a Java servlet. Also in [24] an election servlet is presented. Furthermore, there are commercial voting servlets put on the market. One of them can be found at [9]. To set up an Internet auction one can use commercial software like *rAuction*, which can be found at [26].

Other useful Internet programming techniques are Active Server Pages (ASP) [15] and Java Server Pages (JSP) [13]. We can extend these techniques with customized tags for distributed consensus. However, these techniques are library-based and therefore not as suitable for formal verification as our language-based *DiCons* technique.

6. CONCLUSIONS

We designed a language that supports the development of Internet applications at the right level of abstraction. Although we have done several experiments with the language, we plan to gain more experience, by using the language for larger applications. This will probably show options for refining and extending *DiCons*. Sample specifications of a voting system, an auction system and a Meeting Scheduler already indicated some useful extensions. We mention: atomic regions (to support mutual exclusion, as in Mawl [1, 19]), database coupling (for processing information available at the system, as in Strudel [10]), and style sheets (to give the Web forms a more professional appearance).

Since the communications with the users of the system are under dynamic control, based on the system state, *DiCons* supports personalized and adaptive interactions.

Our choice to base *DiCons* and its support tools on existing and readily available Internet technology, makes it very easy to use. Nevertheless, the language and tools can be easily extended to support more advanced communication schemes.

One of the motivations for designing *DiCons* was that it would allow for the development of formally verified Internet applications. Therefore we prefer a language-based approach to a library-based approach. Up to now, we have not gained experience with formal verification of *DiCons* programs. Current research is focussed on finalizing the formal semantics for the behavioral part of *DiCons* and to experience with formal validation based on this semantics.

We implemented a compiler to compile *DiCons* specifications into Java Servlets. Except for generating a Servlet, the compiler checks a specification on its syntax and static semantics.

More information on *DiCons*, its compiler and some working examples can be found in [2] or at *http://pc32.eesi.tue.nl/*.

7. REFERENCES

[1] D. L. Atkins, T. Ball, G. Bruns, and K. Cox. Mawl: A domain-specific language for form-based services. *IEEE Transactions on Software Engineering*, 25(3):334–346, May/June 1999. Special Section: Domain-Specific Languages (DSL).

[2] H. v. Beek. Internet protocols for distributed consensus – the *DiCons* language. Master's thesis, Eindhoven University of Technology, Aug. 2000.

[3] K. Bharat and M. H. Brown. Building distributed, multi-user applications by direct manipulation. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, Groupware and 3D Tools, pages 71–81, 1994.

[4] K. Bharat and L. Cardelli. Distributed applications in a multimedia setting. In *Proceedings of the First International Workshop on Hypermedia Design*, pages 185–192, Montpellier, France, 1995.

[5] L. Cardelli. Obliq A language with distributed scope. SRC Research Report 122, Digital Equipment, June 1994.

[6] L. Cardelli and R. Davies. Service combinators for web computing. *IEEE Transactions on Software Engineering*, 25(3):309–316, May/June 1999.

[7] M. Cortes and P. Mishra. DCWPL: A programming language for describing collaborative work. In *Proceedings of ACM CSCW'96 Conference on Computer-Supported Cooperative Work*, Language Support for Groupware, pages 21–29, 1996.

[8] R. B. Doorenbos, O. Etzioni, and D. S. Weld. A scalable comparison-shopping agent for the world-wide web. In W. L. Johnson and B. Hayes-Roth, editors, *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 39–48, Marina del Rey, CA, USA, 1997. ACM Press.

[9] Virtua – fastvote support, 1997–1999. http://www.virtua.com/fastvote/, Virtua Communications Corporation.

[10] M. Fernández, D. Suciu, and I. Tatarinov. Declarative specification of data-intensive Web sites. *ACM SIGPLAN Notices*, 35(1):135–148, 2000.

[11] J. Friedl and A. Oram. *Mastering Regular Expressions: Powerful Techniques for Perl and Other Tools (O'Reilly Nutshell).* O'Reilly & Associates, Inc., first edition, Jan. 1997.

[12] J. Gosling, B. Joy, G. Steele, and G. Bracha. *The Java Language Specification.* Java Series. Addison-Wesley, second edition, June 2000.

[13] M. Hall. *Core Servlets and JavaServer Pages.* Sun Microsystems Press/Prentic Hall PTR, June 2000.

[14] N. Heinle and R. Koman. *Designing with JavaScript.* O'Reilly & Associates, Inc., May 2000.

[15] A. Homer, D. Sussman, and B. Francis. *Professional Active Server Pages 3.0.* Wrox Press Inc, Sept. 1999.

[16] J. Hunter and W. Crawford. *Java Servlet Programming.* O'Reilly & Associates, Inc., 981 Chestnut Street, Newton, MA 02164, USA, 1998.

[17] ITU-TS. *ITU-TS Recommendation Z.120: Message Sequence Chart (MSC).* ITU-TS, Geneva, 1997.

[18] T. Kistler and H. Marais. WebL — a programming language for the Web. *Computer Networks and ISDN Systems*, 30(1–7):259–270, Apr. 1998.

[19] D. Ladd and J. Ramming. Programming the web: An application-oriented language for hypermedia service programming. In *Proc. 4th WWW Conf., WWW Consortium*, pages 567–586, 1995.

[20] D. Li and R. Muntz. Building online auctions from the perspective of coca. *Submitted to HICSS-33*, Jan. 2000.

[21] D. Li and R. R. Muntz. COCA: Collaborative objects coordination architecture. In *Proceedings of ACM CSCW'98 Conference on Computer-Supported Cooperative Work*, Infrastructures for Collaboration, pages 179–188, 1998.

[22] S. Mauw, M. Reniers, and T. Willemse. Message Sequence Charts in the software engineering process. In *Handbook of Software Engineering and Knowledge Engineering, S.K. Chang, editor.* World Scientific, 2001. To appear.

[23] Metamata home page: Javacc documentation. http://www.metamata.com/JavaCC/docs/, Fremont, California.

[24] L. O'Brien. Vox populi. *Java Pro Magazine*, June 1999.

[25] D. Raggett, A. L. Hors, and I. Jacobs. Html 4.01 specification. Technical report, W3C User Interface Domain Recommendation, Dec. 1999.

[26] Siteoption home page. http://www.siteoption.com/, SiteOption.com, Green Cove Springs, USA.

[27] J. Udell. *Practical Internet Groupware.* O'Reilly & Associates, Inc., Oct. 1999.

[28] A. Williamson. *Special Edition Using Java Servlet API.* Que Corporation, Indianapolis, IN, USA, 1997.

6.2 Performance Evaluation of Integrated-Services Networks: Sem Borst

CWI, Amsterdam / Eindhoven University of Technology, e-mail: Sem.Borst@cwi.nl

Introduction

The world of communications is experiencing an era of fascinating development. Over the past decade, the Internet has been expanding at an unprecedented rate, in terms of traffic volume, the number of users, as well as the range of applications. Initiated as an eccentric computer network used by a niche community, the Internet has transformed into a massive communication and information medium which has captured a central role in society. The use of wireless services has been showing equally turbulent growth. In just a few years, mobile phones have evolved from impractical devices into popular gadgets which have found ubiquitous usage.
The success of the Internet and the proliferation of wireless services have raised extremely high expectations for their future evolution. Network operators strongly anticipate further expansion, fueled by the advance of all-optical networking as well as the convergence of wireless and Internet access, along with a fundamental trend towards service integration. The future Internet is expected to accommodate a variety of new services with more stringent Quality-of-Service requirements than the current data applications. Next-generation wireless networks are designed to support new high-rate data applications, in addition to the voice calls and messaging services which are the dominant source of traffic in current systems. Eventually, these developments are believed to result in the consolidation of a wide variety of services on a common platform.

Streaming versus elastic traffic
The integration of several services on a single infrastructure offers significant operational advantages. Besides the inherent scaling efficiencies, a second benefit lies in the greater flexibility. History shows how difficult it is to predict the popularity of future applications with any degree of certainty. In view of the intrinsic uncertainty, network operators prefer to install a network infrastructure which can support a wide range of services.
While offering potential synergies, however, the co-existence of heterogeneous services also raises several challenging problems. Different applications may not only have drastically different *traffic characteristics*, but also extremely diverse *Quality-of-Service requirements*. In order to describe the most fundamental differences, it is convenient to make a broad distinction between *streaming* traffic and *elastic* traffic. Streaming traffic is produced by audio and video applications for both real-time communication and reproduction of stored sequences (or *'traces'*). Elastic traffic, on the other hand, results from the transfer of digital documents such as Web pages, files, e-mails, where the transmission rate is adaptable depending on the levels of congestion in the network. Thus, the rate and duration are measures of Quality-of-Service, as opposed to intrinsic characteristics of the traffic as in the case of streaming applications.
Within the category of streaming traffic, one can also draw a distinction whether the transmission rate is approximately constant over time or highly fluctuating. In case of voice traffic for example, fixed-size packets are transmitted at fixed time intervals, producing a constant bit stream. In encoded video, however, the bit rate may vary significantly, depending on the amount of activity in the scenes. In both these cases, Quality-of-Service is mainly determined by the integrity of the original signal. Thus, it is crucial that the signal is not distorted as the packets flow through the network, making small packet delay and low loss crucial Quality-of-Service requirements. For elastic traffic, on the other hand, it is not so much the delay of individual packets that is important, but the total transfer delay of the document that determines the Quality-of-Service as perceived by the users.

Circuit-switched versus packet-switched networks
Current public telephone networks have been highly customized for carrying voice traffic, and provide only limited capabilities for supporting data applications. Specifically, commercial telephone

systems are designed as *circuit-switched* networks, which means that the transmission capacity of each link is (logically) divided into several *circuits*. A call occupies exactly one circuit on each of the links on the path from origin to destination.
A circuit-switched network is perfect for carrying homogeneous constant-rate traffic, such as voice. It is less ideal for supporting heterogeneous or variable-rate traffic, such as video or data. To accommodate non-uniform and time-varying rate requirements, it is necessary that the transmission capacity is not statically partitioned into fixed chunks, but rather can be flexibly shared among various users. This has motivated the design of *packet-switched* networks such as the Internet, where users can freely adjust the transmission rate by varying the number of packets transmitted per unit of time. The packets are then transported as individual entities through the network, and *multiplexed* with packets from other users.

Connection-oriented versus connection-less networks
Within the category of packet-switched networks, one can draw a further distinction. In a *connection-oriented* network, an explicit connection is set up before any packet is transmitted. Once the connection has been set up, all the packets are labeled with a unique connection-identifier used by the switches in the network for packet forwarding. A major example of a connection-oriented packet-switching technology is ATM (Asynchronous Transfer Mode), which was pushed by the telephone industry as the technology of choice for building multi-service networks. Not surprisingly therefore, the ATM 'philosophy' resembles that of current telephone networks, and inherited concepts, if only in name, as *virtual circuit.*
In contrast, in a *connection-less* network there is no explicit notion of a connection, although conceptually one can still think of a connection in relation to user behavior. All packets are treated as separate entities containing complete address information used by the routers for packet forwarding. The most prominent example of a connection-less packet switching technology is IP (Internet Protocol).
There are various trade-offs associated with both approaches. Typically, the set of active connections is significantly smaller than the set of potential addresses of end hosts. As a result, stamping a connection-identifier on a packet introduces less overhead than adding the complete network address of the receiver. As a related advantage, the table look-up involved in packet forwarding is usually much faster in a connection-oriented network than in a connection-less network. The downside of the connection-oriented approach lies in the substantial amount of overhead involved in setting up a connection, which is only compensated for by the higher forwarding capacity in case of long transfers. The majority of the transfers in the Internet however are extremely short, e.g. a mouse click, consisting of just one packet.

Traffic control mechanisms
There are also less tangible but perhaps more important differences related to traffic control issues. Broadly speaking, a connection-oriented approach is well-suited for *centralized* traffic control by the network, whereas a connection-less approach relies on *distributed* control by the users. For example, the connection set-up in a connection-oriented network includes an admission control procedure to check if there are sufficient resources available before a connection is admitted. In addition, the connection-identifier allows the switches in the network to distinguish between packets of various connections, and use *policing* and discriminatory *scheduling* algorithms.
The inability to recognize connections in a connection-less network leaves limited scope for admission control or packet scheduling. Instead, the users must adapt their transmission rates based on congestion feedback signals provided by the routers.

Queueing theory
As described above, packet delay, packet loss, and blocking probabilities are crucial Quality-of-Service measures. Queueing theory provides a useful paradigm for evaluating such performance characteristics. The basic queueing model consists of a single resource (representing a link in a communication network) which provides service (transmission) to randomly arriving requests (packets or calls). The randomness of the traffic process reflects the stochastic fluctuations and

the intrinsic uncertainty in the transmission behavior of the traffic sources.
In the remainder of paper, we give a (selective) overview of the use of queueing-theoretic models and techniques in evaluating the performance characteristics of communication networks. In Section 6.2, we describe some classical blocking models, which provide a useful framework for capacity planning and traffic engineering in circuit-switched networks. In Section 6.2, we review some canonical queueing models, which play a central role in examining various performance issues in packet-switched networks. Both types of models also arise in the performance evaluation of *wireless* networks, where the air-interface and user mobility create an additional degree of complexity.

Blocking models

In this section, we discuss a few representative blocking models. Blocking (or *loss*) models offer a useful methodology for designing and evaluating the performance of circuit-switched networks. For further background we recommend [11], [12], [13], [22].

Erlang-B model

The classical loss model consists of a single link with a capacity of C circuits. Calls arrive as a Poisson process of rate λ, and have independent and exponentially distributed holding times with mean $1/\mu$. Define $\rho := \lambda/\mu$ as the amount of offered traffic. Each call requires exactly one circuit for the duration of the holding time. An arriving call that finds all circuits occupied, is blocked. Blocked calls are assumed to be lost (as an alternative, *retrial* models explicitly account for redialing phenomena).
The number of busy circuits (i.e. the number of calls in progress) behaves as a stochastic process $\mathbf{N}$ with a *birth-death* structure. The equilibrium distribution of $\mathbf{N}$ is determined by

$$\Pr\{\mathbf{N} = n\} = F^{-1}\frac{\rho^n}{n!}, \qquad n = 0, 1, \ldots, C,$$

with $F = \sum_{n=0}^{C} \frac{\rho^n}{n!}$ denoting a normalization constant. In particular, the probability that all circuits are occupied at an arbitrary epoch in equilibrium is

$$B = \Pr\{\mathbf{N} = C\} = F^{-1}\frac{\rho^C}{C!} = \frac{\frac{\rho^C}{C!}}{\sum_{n=0}^{C}\frac{\rho^n}{n!}}. \tag{1}$$

Because of the 'PASTA' property (Poisson Arrivals See Time Averages) [30], this is also the probability that an arriving call finds all circuits occupied, i.e., the blocking or loss probability. Remarkably enough, the above equilibrium distribution in fact applies for any holding-time distribution with mean $1/\mu$, which is referred to as *insensitivity*.
The above model is known as the Erlang-B model (B for blocking) or Erlang loss model, named after the Danish mathematician A.K. Erlang who introduced the model in the early 20th century for evaluating blocking probabilities (i.e. the probability of a busy signal) in telephone exchanges. Equation (1) gives the blocking probability B as a function of the link capacity C and the offered traffic ρ. In Quality-of-Service provisioning, one is primarily interested in the inverse relation. For example, what is the minimum capacity $C_{\min}$ that is required in order to achieve a certain target blocking probability B for a given amount of traffic ρ? Or, what is the maximum amount of traffic $\rho_{\max}$ that can be supported for given capacity C, without exceeding a certain blocking threshold B? Typical design values for B are on the order of $10^{-2} - 10^{-3}$. These inverse problems may be efficiently solved using recursive and convexity properties of the expression in (1).

Multi-class Erlang-B model

In the above model, we considered homogeneous traffic. In multi-service networks, connections may have distinct characteristics, in terms of holding-time distributions or capacity requirements. The above model may be extended to heterogeneous traffic as follows. Consider a single link of C circuits, which is offered traffic from K classes. Class-k calls arrive as a Poisson process of rate λ_k, and have independent and exponentially distributed holding times with mean $1/\mu_k$, $k = 1, \ldots, K$. Define $\rho_k := \lambda_k/\mu_k$ as the offered traffic associated with class-k calls. Each class-k call requires b_k circuits for the duration of the holding time. If an arriving class-k call finds less than b_k free circuits, then it is blocked.

Denote by $(\mathbf{N}_1, \ldots, \mathbf{N}_K)$ the link occupancy at an arbitrary epoch in equilibrium, with $\mathbf{N}_k$ representing the number of class-k calls in progress. Define $S := \{(n_1, \ldots, n_K) : \sum_{k=1}^{K} b_k n_k \leq C\}$ as the set of feasible link occupancies. Then

$$\Pr\{(\mathbf{N}_1, \ldots, \mathbf{N}_K) = (n_1, \ldots, n_K)\} = G^{-1} \prod_{k=1}^{K} \frac{\rho_k^{n_k}}{n_k!}, \qquad (n_1, \ldots, n_K) \in S,$$

with $G := \sum_{(n_1,\ldots,n_K)\in S} \prod_{k=1}^{K} \frac{\rho_k^{n_k}}{n_k!}$ denoting a normalization constant.

Unfortunately, the numerical evaluation of the normalization constant G is not a routine exercise. The cardinality of the set S, and hence the number of terms in the summation, grows rapidly with C and K. Therefore, a brute-force evaluation of G is prohibitively demanding for all but the smallest values of C and K. Sophisticated Monte-Carlo methods have been devised for evaluating G in a computationally efficient manner.

The blocking probability for class-k calls may be expressed as

$$B_k = \sum_{i=C-b_k+1}^{C} \pi_i,$$

with π_i denoting the probability that exactly i circuits are occupied at an arbitrary epoch in equilibrium. The probabilities π_i may be obtained through summation of the link occupancy probabilities, which however implicitly contain the complicated normalization constant G. The Kaufman-Roberts recursion provides an alternative for calculating the probabilities π_i directly through a simple recurrence relation, bypassing the numerical evaluation of G. In addition, advanced asymptotic techniques have been proposed for approximating the blocking probabilities B_k. Not surprisingly, the blocking probability B_k is higher for classes with larger capacity requirements b_k, since larger values of b_k induce a more stringent admission criterion. The discrepancy in blocking probabilities may be undesirable from a performance perspective. Trunk reservation provides a simple scheme for removing the bias, using class-defined parameters $t_1, \ldots, t_K$, with the interpretation that an arriving class-k call is only admitted if there are at least $b_k + t_k$ free circuits (or *'trunks'*). Taking values $t_k = T - b_k$ levels the admission criterion for all classes, and hence equalizes the blocking probabilities.

Instead of removing undesirable bias, a trunk reservation strategy may also be used to actively discriminate among calls of various classes by setting higher trunk reservation values for low-priority calls. It turns out that essentially any degree of differentiation in the blocking probability may be achieved by using even relatively small trunk reservation parameters.

Loss networks

In the above model, we allowed for heterogeneous traffic, but still focused on just a single link. Any practical communication network consists of several links, with connections requiring capacity on each of the links of a path from origin to destination. The above model may be generalized to a network setting as follows. Consider a network consisting of L links, with the l-th link comprising

C_l circuits, which is offered traffic from M classes. Class-m calls arrive as a Poisson process of rate λ_m, and have independent and exponentially distributed holding times with mean $1/\mu_m$. Define $\rho_m := \lambda_m/\mu_m$ as the offered traffic associated with class-m calls. Each class-m call requires b_{lm} circuits on link l for the duration of the holding time, $l = 1, \ldots, L$. Thus, a class defines a set of capacity requirements on each of the links in the network. If an arriving class-m call finds less than b_{lm} free circuits on some link l, then it is blocked. A typical example is $b_{lm} = b_m$ for all $l \in \mathcal{L}_m$, and $b_{lm} = 0$ for all $l \notin \mathcal{L}_m$, with $\mathcal{L}_m$ representing the *route set* of class-m calls, by which we mean the collection of links traversed on the path from origin to destination.
Denote by $(\mathbf{N}_1, \ldots, \mathbf{N}_M)$ the network occupancy at an arbitrary epoch in equilibrium, with $\mathbf{N}_m$ representing the number of class-m calls in progress. Define $S := \{(n_1, \ldots, n_M) : \sum_{m=1}^{M} b_{lm} n_m \leq C_l \text{ for all } l = 1, \ldots, L\}$ as the set of feasible network occupancies. Then

$$\Pr\{(\mathbf{N}_1, \ldots, \mathbf{N}_M) = (n_1, \ldots, n_M)\} = H^{-1} \prod_{m=1}^{M} \frac{\rho_m^{n_m}}{n_m!}, \qquad (n_1, \ldots, n_M) \in S,$$

with $H := \sum_{(n_1,\ldots,n_M)\in S} \prod_{m=1}^{M} \frac{\rho_m^{n_m}}{n_m!}$ denoting a normalization constant. Note the striking similarity with the equilibrium distribution for the multi-class Erlang-B model, with the intricacies of the network structure entirely encapsulated in the set S. Again, the numerical evaluation of the normalization constant is not a trivial task, which has motivated the construction of fixed-point approximations for the blocking probabilities.
In a network context, trunk reservation serves an additional purpose in supplementing the routing algorithm. In the above model, we tacitly assumed a *fixed* routing mechanism, where the route between origin and destination is fixed. An alternative is a *dynamic* routing algorithm, where the route is dynamically selected, based on the current load conditions in the network. Evidently, a dynamic routing algorithm may produce an improvement in performance by balancing the load more efficiently across the network. However, a dynamic routing mechanism is potentially vulnerable to a rather subtle, self-propelling phenomenon which may cause severe performance degradation. The selection of longer and longer routes in overload conditions wastes capacity, which may result in *bi-stability*, with the network oscillating between two operating regimes. Trunk reservation provides an effective mechanism to prevent the route lengths from spiraling upward in overload situations by denying access to non-direct calls.

Queueing models

In this section, we highlight a few characteristic queueing models. Queueing models play a crucial role in analyzing various performance measures in packet-switched networks. For further background material we refer to [10], [23], [27], [31].

Traffic hierarchy
In order to describe traffic processes in packet-switched networks, it is helpful to adopt a three-level hierarchy, consisting of the packet level, the burst level, and the flow level. The flow level (or call or connection or session level) is the highest level, where user sessions are initiated and terminated, although that may not be explicitly signaled in connection-less networks. The packet level (or cell level in case of fixed-size packets) is the lowest level, where individual packets are transmitted. The burst level is an intermediate level, which arises when traffic has *bursty* characteristics, meaning that packets tend to be generated in clumps, as is usually the case in packet-switched networks. For example, in a phone call, activity periods (talk spurts), or *bursts*, each consisting of a sequence of back-to-back packets, alternate with silence periods (pauses). During a web session, the user may click on several items, each of which triggers the transmission of a group of packets to deliver the requested information.

Typically, the above hierarchy induces a separation of time scales. The call-level dynamics evolve at a relatively slow time scale, e.g. minutes to hours. The packet-level dynamics occur at an extremely fast rate, e.g. milli to micro-seconds. The burst level corresponds to some intermediate time scale, but may also be close to either the packet or call level.

The M/G/1 queue

The M/G/1 queue consists of a single resource or server, which handles service requests from arriving customers. Along with several variants, the M/G/1 queue is by far the most intensively studied queueing model. Customers arrive as a Poisson process of rate λ, and require generally distributed service times $\mathbf{B}$ with mean $\beta^{(1)}$ and second moment $\beta^{(2)}$. Define $\rho := \lambda\beta^{(1)}$ as the traffic intensity. For stability, we assume that $\rho < 1$ so that the server is able to handle all the traffic. An arriving customer that finds the server idle, is taken into service immediately. Otherwise, the customer waits for service. Customers are served in order of arrival, which is also known as the First-Come First-Served (FCFS) discipline.

The above representation M/G/1 is the Kendall notation which provides a useful convention for the systematic classification of queueing models, named after the English mathematician Kendall. The Kendall notation has the form A/B/C/D, with the A indicating the interarrival time distribution (M for the *Memoryless* property of the Poisson process), the B denoting the service time distribution (with G standing for general), C for the number of servers, and an optional fourth field D defining the number of waiting positions (assumed infinite unless specified otherwise).

Due to the memoryless property of the Poisson process, the queue length $\mathbf{N}$ (including the customer in service) observed just before departure epochs evolves as a Markov chain. Denote by $\mathrm{E}[z^{\mathbf{N}}]$ the probability generating function (PGF) of the equilibrium distribution of $\mathbf{N}$. The notion of a PGF may seem somewhat obscure, but provides an extremely convenient 'calculus' for manipulating with (sums of independent) discrete random variables. At the same time, the PGF contains all the relevant information regarding the distribution of the random variable. In particular, the mean of the random variable may be obtained by differentiating the PGF, and setting $z = 1$. Higher (generalized) moments of the random variable may be determined by considering higher-order derivatives.

Now observe that there is a simple relation between the queue length at two successive departure epochs: the queue length increases by the number of arrivals during the service time, minus one, unless the first departure left the system empty. This recurrence relation may be used to derive that

$$\mathrm{E}[z^{\mathbf{N}}] = \frac{(1-\rho)(1-z)\beta(\lambda(1-z))}{\beta(\lambda(1-z)) - z},$$

$|z| \leq 1$, with

$$\beta(s) := \mathrm{E}[\mathrm{e}^{-s\mathbf{B}}] = \int_{t=0}^{\infty} \mathrm{e}^{-st}\mathrm{dPr}\{\mathbf{B} < t\}$$

denoting the LST of the service time distribution. LST's may be viewed as the conceptual counterparts of PGF's for continuous random variables with similar convenient properties. Because of an up- and down-crossing argument and the PASTA property mentioned earlier, $\mathbf{N}$ is also the queue length at an arbitrary epoch in equilibrium. Differentiating w.r.t. z, and setting $z = 1$, we find that the mean queue length at an arbitrary epoch in equilibrium is

$$\mathrm{E}\mathbf{N} = \frac{\lambda^2\beta^{(2)}}{2(1-\rho)} + \rho.$$

Note that ρ is the fraction of time that the server is busy, and hence the mean number of customers in service. Thus, the mean number of waiting customers in equilibrium, excluding the customer in service, is

$$\mathrm{E}\mathbf{L} = \frac{\lambda^2\beta^{(2)}}{2(1-\rho)}.$$

The mean waiting time **EW** of an arbitrary customer is related to **EL** via

$$\mathbf{EL} = \lambda \mathbf{EW}, \tag{2}$$

yielding the Pollaczek-Khintchine formula

$$\mathbf{EW} = \frac{\lambda \beta^{(2)}}{2(1-\rho)}, \tag{3}$$

which is one of the classical results in queueing theory.
The above formula conveys two fundamental insights. First of all, the mean waiting time increases steeply as the traffic intensity ρ approaches 1. Second, for given arrival rate λ and mean service time $\beta^{(1)}$, the waiting time increases with the variability of the service time as measured by the second moment $\beta^{(2)}$. These two insights in fact apply in a much broader context.
The relation (2), known as Little's law or Little's formula [16], is one of the most fundamental and celebrated results in queueing theory. It establishes a universal relation between the mean *number* of customers in a system, the mean amount of *time* the customers spend in the system, and the arrival rate into the system. A simple heuristic proof of Little's result proceeds as follows. Suppose that every customer in the system pays 1 euro per unit of time. Then the mean amount of revenue per unit of time may be evaluated as $\lambda \mathbf{EW}$, but may equivalent be expressed as **EN**. Note that this argument does not involve any particular property of the M/G/1 queue, and it is not surprising therefore that Little's result applies in a very general setting.

Extensions to the M/G/1 queue

As evidenced by the above analysis, the M/G/1 queue admits a detailed analysis of several crucial performance measures. Unfortunately, however, the applicability of the M/G/1 queue in analyzing packet-level performance is somewhat limited. As described earlier, packets usually arrive in an extremely bursty fashion, rather than as a Poisson process as in the M/G/1 model. Thus, in order to examine packet-level performance, one needs to consider more complicated arrival models, such as compound Poisson processes, Markov-modulated models, or even more general models such as Batch Markovian Arrival Processes (BMAP's). These models may be viewed as generalizations of the M/G/1 queue, and can be numerically analyzed using matrix-analytic techniques, see [17], [18] for further details.
The M/G/1 queue is better suited for investigating flow-level performance issues. At the flow or burst level, it is usually not unreasonable to assume a Poisson arrival process. However, the order of service at the flow or burst level is typically not First-Come First-Served, but better modeled by the Processor Sharing (PS) discipline, meaning that all present flows receive an equal share of the capacity [19]. In that case, the waiting time is not a meaningful performance measure, since an arriving flow does not wait, but starts to receive service immediately. Instead, one can consider the mean transfer delay or sojourn time of an arbitrary flow, which is given by

$$\mathbf{ET} = \frac{\beta^{(1)}}{1-\rho}. \tag{4}$$

If we compare the above formula with that for the mean waiting time in case of FCFS, then there are two immediate observations. As before, the mean delay increases sharply as the traffic intensity approaches 1. However, for given arrival rate λ and mean service time $\beta^{(1)}$, the variability of the service time has no longer any effect on the mean delay. This insensitivity property is quite crucial, as we will see later.

Fluid queues

Fluid queues provide a versatile approach for modeling bursty traffic processes. Suppose that one focuses on the burst level in the traffic hierarchy described earlier. At that time scale, the flow-level dynamics evolve relatively slow, and the number of active flows is nearly static. On

the other hand, the packet-level dynamics occur extremely fast, and the stream of packets may approximately be viewed as continuous fluid. This elegant concept has created a strong interest in fluid models as a paradigm for capturing burst-level traffic phenomena.
A classical fluid queue may be described as follows [2]. Consider a link of unit rate, fed by the superposition of N stochastically identical independent On-Off sources. Each source alternates between exponentially distributed On-periods with parameter μ, and exponentially distributed Off-periods with parameter λ. During an On-period, a source generates traffic at some constant rate r. Thus, the mean rate is $\rho := \lambda r/(\lambda+\mu)$. In order for the queue to be stable and non-trivial, we assume $N\rho < 1 < Nr$. Denote by $\mathbf{V}$ the buffer content at an arbitrary epoch in equilibrium. Then

$$\Pr\{\mathbf{V} > x\} = \sum_{i=1}^{P} \alpha_i \mathrm{e}^{-\zeta_i x},$$

with $\alpha_1, \ldots, \alpha_P$ and $\zeta_1, \ldots, \zeta_P$ positive coefficients.
The above model was introduced to describe packetized voice traffic, with the On-periods modeling talk spurts, and the Off-periods corresponding to silence periods. In addition, On-Off sources provide a popular (approximate) model for describing the 'worst-case' output of traffic shaping devices such as leaky buckets. The model may be extended to include several activity levels or activity periods with a general Markovian structure [26], as would be suitable to characterize encoded video.

Asymptotic results

In Quality-of-Service provisioning, one is typically interested in extremely small probabilities (typical design values for packet loss fractions are on the order of $10^{-6} - 10^{-9}$), or systems with an extremely large population of users. Although such dimensions complicate a conventional queueing analysis, they usually facilitate the derivation of accurate asymptotic results using large-deviations techniques, in particular the computation of *effective bandwidths* [6], [8], [14], [25].
Consider N stochastically identical independent traffic sources sharing a link of rate $C \equiv Nc$ endowed with a buffer of size $B \equiv Nb$. Thus, the coefficients b and c represent the the link rate and buffer size per source, respectively. Denote by $\mathbf{V}_N^c$ the buffer content per source as a function of N and c. The above scaling was first proposed in [28]. Let $A(t)$ be a random variable representing the amount of traffic generated by an individual source in a time interval of length t. Let $\alpha_t(s) := \frac{1}{t}\log \mathrm{E}[\mathrm{e}^{sA(t)}]$ be the log-moment generating function of $A(t)$, and let $\alpha(s) := \lim_{t\to\infty} \alpha_t(s)$. There are two relevant asymptotic regimes, (i) 'large-buffer' (large-b) asymptotics, and (ii) 'many-sources' (large-N) asymptotics.

Large-buffer asymptotics
Here, the size of the buffer is scaled up, while the number of sources is held fixed. It may be shown that

$$\lim_{b\to\infty} \frac{1}{Nb} \log \Pr\{\mathbf{V}_N^c > b\} = -\eta(c),$$

with

$$\eta(c) := \sup\{\theta : c \geq \frac{\alpha(\theta)}{\theta}\},$$

which suggests the following approximation for large values of b,

$$\Pr\{\mathbf{V}_N^c > b\} \approx \mathrm{e}^{-Nb\eta(c)}.$$

The above approximation may be used for admission control purposes. A typical Quality-of-Service requirement is that the probability of buffer overflow does not exceed some small value ϵ, which is ensured if $\Pr\{\mathbf{V}_N^c > b\} \leq \epsilon$. Using the above approximation, the latter constraint may be translated into $\eta(c) \geq \delta$, or equivalently, $c \geq \phi(\delta)/\delta$, with $\delta := -\log(\epsilon)/Nb$. Hence, $\phi(\delta)/\delta$ may be interpreted as the effective capacity requirement, or *effective bandwidth* per source. Even though the microscopic traffic activity varies over time, the notion of effective bandwidths allows one to

treat the capacity requirements at a macroscopic scale as *constant* for the purpose of admission control.

Many-sources asymptotics
Here, the number of sources grows large, while the amount of buffer space per flow remains constant. It may be shown that

$$\lim_{N\to\infty} \frac{1}{N} \Pr\{\mathbf{V}_N^c > b\} = -\gamma(b,c),$$

with

$$\gamma(b,c) := \inf_{t>0} \sup_{s} [s(b+ct) - t\alpha_t(s)],$$

which suggest the following approximation for large values of N,

$$\Pr\{\mathbf{V}_N^c > b\} \approx \mathrm{e}^{-N\gamma(b,c)}.$$

In this case, the effective bandwidth per source may be calculated as $c \geq \sup_{t>0} \inf_{s} \left[\frac{b}{t}\left(\frac{\delta}{s}-1\right) + \frac{\alpha_t(s)}{s}\right]$ for certain δ. It turns out that the notion of effective bandwidth generally remains valid when heterogeneous sources are multiplexed, so that the *admissible region* can be described through a simple linear constraint, exactly as in the blocking models discussed earlier. Thus, the notion of effective bandwidths allows the arsenal of techniques developed for circuit-switched models to be used in examining call-level performance issues in packet-switched networks.

Queues with heavy tails

Over the past few years, empirical findings have triggered a strong interest in fluid queues with non-Markovian activity periods. Extensive measurements have shown that bursty traffic behavior may extend over a wide range of time scales, and manifest itself in *self-similarity* and *long-range dependence* [3], [15], [21], citeWillinger. Self-similarity means that the traffic process shows fractal behavior, and looks similar when observed on various time scales. Long-range dependence is a closely related phenomenon implying that correlations extend over long time periods. The occurrence of these phenomena is commonly attributed to *heavy-tailed* characteristics in the activity patterns. For example, several studies have suggested that file sizes typically follow a Pareto distribution [9], which means that the tail probabilities exhibit relatively slow *polynomial* decay, i.e. $\Pr\{\mathbf{S} > x\} \sim ax^{-\nu}$ as $x \to \infty$. A typical value that is quoted is $\nu \approx 1.7$, which implies that the first moment exists, but that the second moment is infinite! It turns out that queues with heavy-tailed activity periods behave fundamentally different from queues with traditional *light-tailed* traffic processes, where the tail probabilities decay at an *exponential* rate, i.e., $\Pr\{\mathbf{S} > x\} \sim bx^{-\mu}$ as $x \to \infty$. We refer to [7], [20], [24] for further details. The exact asymptotics for fluid queues with heavy-tailed On-periods were recently obtained in [5], [32].
Although the occurrence of heavy-tailed traffic characteristics is widely acknowledged, the practical implications for network performance remain a matter of debate. If we consider the Pollaczek-Khintchine formula (3), then we find that the mean delay in an M/G/1 queue with FCFS is infinite when the second moment of the service time is infinite. However, if we look at formula (4) for the mean delay in case of Processor Sharing, then the second moment of the service time has no effect. Thus, we can immediately conclude that the service discipline plays a crucial role in assessing the impact of heavy-tailed traffic characteristics. The insensitivity property which we observed in blocking models suggests that the effect of heavy-tailed traffic also depends on the buffer size and what performance measure is considered. Detailed results on the role of scheduling in assessing the impact of heavy-tailed traffic may be found in [1], [4].

References

[1] Anantharam, V. (1999). Scheduling strategies and long-range dependence. *Queueing Systems* **33**, 73–89.

[2] Anick, D., Mitra, D., Sondhi, M.M. (1982). Stochastic theory of a data-handling system with multiple sources. *Bell Syst. Techn. J.* **61**, 1871–1894.

[3] Beran, J., Sherman, R., Taqqu, M.S., Wilinger, W. (1995). Long-range dependence in variable-bit-rate video traffic. IEEE *Trans. Commun.* **43**, 1566–1579.

[4] Borst, S.C., Boxma, O.J., Jelenković, P.R. (2000). Reduced-load equivalence and induced burstiness in GPS queues with long-tailed traffic flows. CWI Report PNA-R0016. Submitted for publication.

[5] Borst, S.C., Zwart, A.P. (2000). A reduced-peak equivalence for queues with a mixture of light-tailed and heavy-tailed input flows. SPOR-Report 2000-04, Eindhoven University of Technology. Submitted for publication.

[6] Botvich, D.D., Duffield, N.G. (1995). Large deviations, the shape of the loss curve, and economies of scale in large multiplexers. *Queueing Systems* **20**, 293–320.

[7] Boxma, O.J., Dumas, V. (1998). Fluid queues with heavy-tailed activity period distributions. *Computer Communications* **21**, 1509–1529.

[8] Courcoubetis, C., Weber, R.R. (1996). Buffer overflow asymptotics for a buffer handling many traffic sources. *J. Appl. Prob.* **33**, 886–903.

[9] Crovella, M., Bestavros, A. (1996). Self-similarity in World Wide Web traffic: evidence and possible causes. In: *Proc. ACM Sigmetrics '96*, 160–169.

[10] Gross, D., Harris, C.M. (1985). *Fundamentals of Queueing Theory* (John Wiley & Sons, New York).

[11] Kelly, F.P. (1979). *Reversibility and Stochastic Networks* (John Wiley & Sons, Chichester).

[12] Kelly, F.P. (1986). Blocking probabilities in large circuit-switched networks. *Adv. Appl. Prob.* **23**, 473–505.

[13] Kelly, F.P. (1991). Loss networks. *Ann. Appl. Prob.* **1**, 319–378.

[14] Kelly, F.P. (1991). Effective bandwidths at multi-service queues. *Queueing Systems* **9**, 5–16.

[15] Leland, W.E., Taqqu, M.S., Willinger, W., Wilson, D.V. (1994). On the self-similar nature of Ethernet traffic (extended version). IEEE/ACM *Trans. Netw.* **2**, 1–15.

[16] Little, J.D.C. (1961). A simple proof of $L = \lambda W$. *Oper. Res.* **9**, 383–387.

[17] Neuts, M.F. (1981). *Matrix Geometric Solutions in Stochastic Models* (John Hopkins University Press, Baltimore).

[18] Neuts, M.F. (1990). *Structure of Stochastic Matrices of M/G/1 Type and Their Applications* (Marcel Dekker, New York).

[19] Núñez Queija, R. (2000). *Processor-Sharing Models for Integrated-Services Networks.* PhD Thesis, Eindhoven University of Technology.

[20] Park, Willinger, W. (eds.) (2000). *Self-Similar Network Traffic and Performance Evaluation* (John Wiley & Sons, New York).

[21] Paxson, V., Floyd, S. (1995). Wide area traffic: the failure of Poisson modeling. IEEE/ACM *Trans. Netw.* **3**, 226–244.

[22] Ross, K.W. (1995). *Multiservice Loss Models for Broadband Telecommunication Networks* (Springer, Berlin).

[23] Shwartz, A., Weiss, A. (1995). *Large Deviations for Performance Analysis: Queues, Communication, and Computing* (Chapman & Hall, London).

[24] Sigman, K. (ed.) (1999). *Queueing Systems* **33**. Special Issue on Queues with Heavy-Tailed Distributions.

[25] Simonian, A., Guibert, J. (1995). Large deviations approximations for fluid queues fed by a large number of on/off sources. *IEEE J. Sel. Areas. Commun.* **13**, 1017–1027.

[26] Stern, T.E., Elwalid, A.I. (1991). Analysis of separable Markov-modulated rate models for information-handling systems. *Adv. Appl. Prob.* **23**, 105–139.

[27] Walrand, J. (1988). *An Introduction to Queueing Networks* (Prentice Hall, Englewood Cliffs NJ).

[28] Weiss, A. (1986). A new technique of analyzing large traffic systems. *Adv. Appl. Prob.* **18**, 506–532.

[29] Willinger, W., Taqqu, M.S., Sherman, R., Wilson, D.V. (1997). Self-similarity through high-variability: statistical analysis of Ethernet LAN traffic at the source level. IEEE/ACM *Trans. Netw.* **5**, 71–86.

[30] Wolff, R.W. (1982). Poisson arrivals see time averages. *Oper. Res.* **30**, 223–231.

[31] Wolff, R.W. (1989). *Stochastic Modeling and the Theory of Queues* (Prentice Hall, Englewood Cliffs NJ).

[32] Zwart, A.P., Borst, S.C., Mandjes, M. (2000). Exact asymptotics for fluid queues fed by multiple heavy-tailed On-Off flows. SPOR Report 2000-14, Eindhoven University of Technology. Shortened version in: *Proc. IEEE INFOCOM 2001*, to appear.

6.3 VerifiCard A European Project for Smart Card Verification: Bart Jacobs, Hans Meijer and Erik Poll

Computing Science Institute, University of Nijmegen
{Bart.Jacobs,Hans.Meijer,erikpoll}@cs.kun.nl http://www.verificard.org

Abstract

The next generation of smart cards will be used for services where security is a key issue. Reliability and trust are necessary for a large scale adoption and success of these smart cards. New validation techniques are needed, based on well-defined mathematical models, using special tools for mathematically proving correctness, going well beyond testing. A EU-funded consortium 'VERIFICARD' of 5 academic and 2 industrial partners, coordinated by Nijmegen, will work on the correctness of crucial components of the chosen (JAVACARD) platform and of individual applications. This brief note will give an introduction to smart cards and their importance in computer science today. Additionally, it will give an impression of the work done in Nijmegen on these topics.

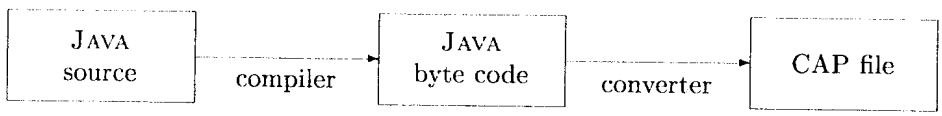
Introduction

Smart cards are small devices carrying a computer chip. They come in various shapes: plastic cards in the form of credit cards (used as, for instance, electronic purses) or SIM-cards in mobile telephones.
The next generation of smart cards will be used for services where security is a key issue, like authenticated access to computer networks, e-commerce, m-commerce, high value wire-less (GSM- or UMTS-based) services, loyalty programs and digital signing. The correct functioning of these cards must be absolutely guaranteed. Potentially malicious application programs must be identified. A virus on a smart card is a nightmare scenario. Therefore, new validation techniques are needed, based on well-defined mathematical models, using special tools (theorem provers and model checkers) for mathematically proving correctness, going well beyond testing.
This offers both a challenge and an opportunity to the field of formal methods in general, and of JAVA-program verification in particular, as the leading smart card platform – JAVACARD – is based on JAVA. The challenge is to show that formal verification techniques are applicable to practical programs, in a real-life context, and thus may develop from an academic discipline into an industrially relevant field. The opportunity follows from the fact that smart cards are still small enough to bring their software within the reach of modern verification tools.
This is a very important point. The formalization, specification and verification of the JAVACARD platform is certainly not a trivial task, yet the platform is small enough to make a *complete* formal treatment feasible. Moreover, since smart cards are mass-produced and open to any kind of attack, the smart card industry as well as the card issuers have a profound interest in an error-free and secure implementation of the platform and its applets. This makes JAVACARD a unique chance for formal methods to prove their importance for the software industry. For a comprehensive overview of applications of formal methods to JAVACARD and smart cards, see [Har00].

JavaCard

Smart cards contain a microprocessor chip which can execute small application programs called *applets*. The cards that are in use today feature a single applet, written in machine code, which is 'burnt' in ROM and therefore fixed forever. In contrast, the new generation of smart cards can hold several applets in their memory at the same time ("multi-application"), and new applets can be downloaded after the cards have been issued ("post-issuance"), allowing services to be updated or new services to be added without replacing the card. Moreover, applets are not written in machine code specific to the particular chip used, but are written in a high-level language which is compiled to byte code that is interpreted by a virtual machine on the smart card.
The amount of memory available on smart cards is rather restricted, typically in the order of 10 kilobytes. It consists of a ROM containing a run-time environment, persistent updateable

memory (such as EEPROM) for information which has to be preserved when power is removed, and transient memory for temporary workspace.
A card is used by inserting it in (or bringing it sufficiently close to) a smart card reader. This smart card reader will generally be an input/output device of some computer system, which communicates with the card's system by exchanging byte sequences called Application Protocol Data Units (APDUs). An APDU may contain an order to select a certain applet, or an order for the currently selected applet. The card responds with APDUs containing results, error messages, etc. This and other aspects of smart cards are standardized in ISO 7816.
Applets are written in a high level language and are designed to run on a standardized open platform. The applets for smart cards can be programmed in a language JAVACARD, which is a simplified version of the popular language JAVA. JAVACARD, like JAVA, is owned by Sun, but is freely available and described in open standards.
There are currently three standards for smart cards: JAVACARD[1], MultOS, and Windows for Smart Cards. JAVACARD is the most open standard, with its specifications publicly and freely available. All currently operational implementations of multi-application cards use the JAVACARD platform, and all major players in the smart card market are producing JAVACARD-based products. MultOS is slowly moving in the direction of JAVACARD. Windows for Smart Cards is developed by Microsoft. Altogether, the choice for JAVACARD seems obvious.
The JAVACARD language is a subset of JAVA developed with the limited memory and computation power of smart cards in mind. For instance, it only supports the types `byte`, `short`, and optionally `int` with their basic operations, but not `double` and `float`, and only allows one-dimensional arrays. It supports most control flow constructs, as well as exception throwing and catching. However, its Application Programming Interface (API) is quite small, see below.
The source code of a JAVACARD applet is translated into byte code by a standard JAVA compiler. The resulting byte code is checked by a *byte code verifier* to verify certain properties related to e.g. typing, and to see that the applet only uses features supported by JAVACARD. Since JAVA byte code contains redundant information (which for instance enables full reverse-compilation), and JAVACARD applets should be as small as possible, the byte code is further converted by a *cap converter* into a Converted Applet (CAP-file), which is the form loaded and installed on the card.



An applet's CAP file is interpreted by the JAVACARD Virtual Machine (JCVM). The JCVM resides in the card's ROM, together with the JAVACARD Run-time Environment (JCRE), which is an implementation of the JAVACARD API, and the applets which are loaded when the card is manufactured. One of these applets may be designed to load and install *post-issuance* applets into persistent memory.
Applets are separated from each other by a *firewall* mechanism. No object created by one applet can be accessed by a different applet, unless such access is explicitly requested and granted, via a so-called *shareable object.*
Whenever the card is powered-up in a card reader it is reset to a consistent state. In order to preserve consistency, all updates of persistent memory are subject to a `begin/commit/abort`-transaction mechanism.
Altogether, the JAVACARD API currently comprises 18 classes for exceptions, 16 interfaces for working with cryptographic keys and only 10 fundamental classes. Among the latter are a class `APDU` for communication with the card reader, classes `PIN` and `OwnerPIN` for working with PIN-codes, an abstract class `Applet` for applets, and a class `JCSystem`, which contains methods for transaction handling and for handling requests for object sharing between applets maintaining the applet firewall.
For general background information on smart cards, see [HNSS00], and for more specific information on JAVACARD, see [Che00].

[1]See the URL of the JAVACARD Forum `http://www.javacardforum.org`

Formal methods for smart cards in VerifiCard

The VERIFICARD project will work on formalization of the JAVACARD platform (mainly: what is on the card) and of applets, in several different ways.

Platform specification and verification providing formal descriptions of the full JAVACARD smart card platform, starting from the open JAVACARD standards. This formalization includes that of the JAVACARD language, virtual machine and run-time environment (API) and is used to develop and formally establish the soundness of several components of the JAVACARD platform, in particular the byte code verifier and cap converter, and possibly also the compiler. It is also needed as a basis for techniques of applet verification.

Since the existing informal specification of JAVACARD is partly given at source code level and partly at byte code level, the platform specification and verification will be done at both levels.

At the source code level, a formal operational semantics of the JAVACARD language and API will be provided, for proving the compiler correct. As an axiomatic semantics (Hoare logic) is better suited to proving logical (safety) properties of applets using theorem provers, such a semantics is provided as well. The soundness and completeness of the axiomatic semantics w.r.t. the operational one will be proved.

At the byte code level, a formalization of the JCVM will be provided, and in doing so, a formal operational semantics of JAVA byte code. This formalization will be used for a verification of the compiler, and the construction of a certified byte code verifier and cap converter.

Applet specification and verification developing methods and tools for the validation of aspects of applets, in particular security aspects. The applets run on the JAVACARD platform, and therefore these methods need to be based on the platform formalisation.

Different styles of specification will be explored, namely logic methods using Hoare logic and algorithmic methods using temporal logic, in order to investigate the types of problems for which they are best suited. The case studies provided by the industrial partners will drive this exploration (instead of looking for problems which are well solved by a chosen method).

The algorithmic methods will be based on an operational semantics at byte code level. A JAVACARD Temporal Logic Specification Language will be developed which should take care of typical aspects of JAVACARD like object sharing and firewalls, transactions, transient objects, *etc.* Three techniques for the verification of such aspects will be considered: compositional proof techniques (allowing to specify and verify applets individually), abstract interpretation and model checking.

The logic methods will be based on an axiomatic semantics at source code level. A JAVACARD Interface Specification Language will be developed which is similar to a subset of JML described below. Three approaches to verification are investigated: verification at the syntactic level and at the semantic level, the latter with and without using a representation of the global object store. For verification at the semantic level applets are translated into the logic of some theorem prover, using the LOOP-tool described below.

Applications applying and testing the techniques developed in the other parts, in particular using realistic case studies supplied by industrial partners and members of the End-User Panel.

In particular a banking case study and a mobile communication (GSM) case study will be developed which will include examples of hostile applets to be detected as such by the methods and tools developed earlier. Also, a set of security properties will be provided which are to be verified by these methods and tools. As a further application, the API will be verified using the specification developed earlier.

In the project 5 academic and 2 industrial partners participate. The academic partners are the University of Nijmegen (the project coordinator), INRIA in France, the Technical University of Munich, the University of Hagen in Germany and SICS, the Swedish Institute of Computer Science. The industrial partners are the French companies Gemplus and Bull, who are both deeply involved in the development of (JAVACARD) smart cards.
The project also has an End User Panel representing different industries with an interest in smart card security. The End User Panel currently consists of TNO (The Netherlands), Deutsche Telekom, France Telecom, Setec Oy (Finland), Trusted Logic (France), IBM Zurich, INTEGRI (Belgium) and Ericsson (Sweden).
At source code level, Munich will work on the operational and axiomatic semantics of JAVACARD. Nijmegen and Hagen will develop a complete formal specification of the JAVACARD API, which ideally should become a standard reference. All this will form the basis for the specification and verification of JAVACARD applets at source code level using theorem provers by Nijmegen, Hagen, and INRIA. Here Nijmegen will use the LOOP-tool – discussed in more detail below –, Hagen will use the Jive tool developed there, and INRIA will use Coq.
At byte code level, INRIA, Munich, and Gemplus will work on formalisation of the JAVACARD virtual machine and a certified byte code verifier. INRIA will also develop a certified cap-converter, and Munich will start work on a certified compiler. The formalisation of the virtual machine will be used by INRIA and SICS for verification of JAVACARD applets at byte code level using modelchecking, abstract interpretation, and compositional proof techniques.

The Java Modelling Language (JML)

Since new applets can be downloaded to the new generation of cards, card issuers are very worried about controlling which applets are allowed on their cards. They can use digital signatures to ensure that only applets that they signed can be installed. But this leaves them with the problem that they want to know for sure that their applets behave correctly. This leads to old-fashioned program verification, which, these days, is done with modern (verification) tools and formal specification languages. This is the topic that Nijmegen (and also Hagen and INRIA) will work on within the VERIFICARD consortium.
The LOOP-group in Nijmegen (see [Loop]) uses the behavioural specification language for Java called JML [LBR99b, LBR99a]. JML is designed primarily by Gary Leavens in cooperation with Compaq SRC and Nijmegen. We give a brief account of its features without going into any detail. JML is designed to specify the *behaviour* of JAVA classes and interfaces and their objects. The basic idea is that pre- and postconditions be specified for each method of a class, and an invariant for the class as a whole, which describes properties of the class' fields, but may be considered as an additional pre- and postcondition for the class' methods.
One may specify several 'behaviours' for a method. Each behaviour has a precondition given as a `requires`-clause and a postcondition given as an `ensures`-clause. A behaviour may have additional clauses, like a `modifiable`-clause which specifies (non-local) variables which may be changed in that behaviour, or a `signals`-clause with a condition, which indicates that the behaviour may result in throwing an exception when that condition holds. A behaviour may be qualified as 'normal', in which case it should return normally, or 'exceptional', in which case it should signal an exception. JML knows many other qualifying modifiers.
The invariants and pre- and postconditions are written as (Boolean) expressions in standard JAVA-syntax, extended with various constructs like `forall` and `exists`. The reason for choosing standard syntax is to encourage JAVA-programmers to specify their code as they write it. Extensions include quantifiers and set comprehension, logical implication, a pseudo-variable `\result` indicating a method's returned value, and more such pseudo-variables and pseudo-operators like `\old`: the pre-value of a modifiable variable v may be referred to in postconditions as `\old(`v`)`.
A specification may use *model fields*, i.e. specification-only variables, e.g. to capture implicit variables whose values could be inferred from explicit JAVA variables. The initial value of such model fields may be specified. A specification may also directly call methods of 'pure' classes, i.e. without side effects. This allows for the abstraction of (mathematical) details in a JAVA setting.

```
/*@ behavior
  @    requires: src != null &&  srcOff >= 0 &&
  @              srcOff+length <= src.length &&
  @              dest != null && destOff >= 0 &&
  @              destOff+length <= dest.length &&
  @              length >= 0;
  @  modifiable: dest[destOff..destOff+length-1],
  @              TransactionException.systemInstance.reason;
  @     ensures: \forall (short i) 0 <= i && i < length
  @                   ==> dest[destOff+i] == \old(src[srcOff+i]);
  @     signals: (TransactionException e)
  @                  e.getReason() == TransactionException.BUFFER_FULL;
  @*/
public static final native short arrayCopy(byte[] src,
                                           short srcOff,
                                           byte[] dest,
                                           short destOff,
                                           short length)
throws ArrayIndexOutOfBoundsException,
       NullPointerException,
       TransactionException;
```

Figure 1: JML specification for `arrayCopy` from `Util`

Specifications are written in separate files or embedded in JAVA code as special comments.
As an example, a JML specification of a method `ArrayCopy` of the JAVACARD API class `Util` is given in Fig. 1. The intended behaviour of `arrayCopy(src, srcOff, dest, destOff, length)` is to copy the items from `src[srcOff..srcOff+length-1]` to `dest[destOff..destOff+length-1]`.

A subtle point is the need for `\old(...)` in the `ensures` clause, i.e. the postcondition, of the `arrayCopy` method. This is needed to correctly specify the behaviour in the case that aliasing occurs, i.e. the case that `src == dest`. We have to refer to the original entries of the `src` array in the postcondition. The informal (javadoc) specifications of the JAVACARD API explicitly state that this is what happens if `src` and `dest` are aliases.
The `TransactionException` that may be thrown by `arrayCopy` is typical for JAVACARD, and may arise because of the limited available resources on smart cards. It may occur when an overflow arises in a special transaction buffer that is used to enable rollback of operations in case of failure (*e.g.* when the card is prematurely removed from the card reader.) Further examples of JML specifications for classes from the JavaCard API are discussed in [PBJ00, PBJ01].
Specifications as those in Fig. 1 provide a basis to formally prove that a given method implementation (*e.g.* in an applet, possibly involving an `arrayCopy`) satisfies its specification. Especially the `modifiable` clauses are useful to control the side-effects that applets can have (and thus the possible damage that they can do).

The LOOP-tool

The LOOP-tool (where LOOP stands for Logic of Object Oriented Programming) is developed in Nijmegen, partly in cooperation with the Technical University of Dresden. It translates JML-programs (i.e., JAVA programs annotated with JML-specifications as outlined above) into a description of their semantics in the higher-order logic used by the theorem provers PVS and Isabelle/HOL, see [BJ00]. In fact, a JAVA-class is modelled as a co-algebra [JR97, Rei95, Jac99] mapping a state into a sum of products of JAVA types and the state space. The theorem provers PVS or Isabelle/HOL can then be used that JAVA-programs meet their JML-specifications. This

is a non-trivial, highly interactive activity for which a special tailor-made Hoare logic has been developed [JP00]. The first steps in the verification of JML-annotated JAVA-programs using the LOOP tool are described in [BJP01].

Summary

The JAVACARD platform offers a unique opportunity for formal methods to show their value for real-life software development: the platform is non-trivial but not too large, and an error-free and secure implementation is of vital importance not only for the smart card industry and card issuers, but also for society at large.
The VERIFICARD project aims to provide methods and tools for the complete specification and verification of applets written for the JAVACARD platform used in next-generation smart cards. To this end the project will develop

- a formalization of
  - the JAVACARD language at source level, by an operational semantics and an axiomatics semantics, and of the JAVACARD Application Programming Interface (API)
  - the JAVACARD Virtual Machine, giving an operational semantics of the JAVACARD byte code level
- certified tools for developping JAVACARD programs, including
  - a byte code verifier
  - a cap converter
  - a JAVA compiler
- methods and tools for the specification and verification of applets, at byte code level and source code level, using theorem proving, abstract interpretation, and model checking.

These formalizations, specifications and verification methods will be evaluated using non-trivial characteristic case studies (and the reference implementation of the JAVACARD API).

References

[BJ00] J. van den Berg and B. Jacobs. The LOOP compiler for Java and JML. Techn. Rep. CSI-R0019, Comput. Sci. Inst., Univ. of Nijmegen. To appear at TACAS'01., 2000.

[BJP01] J. van den Berg, B. Jacobs, and E. Poll. Formal Specification and Verification of JavaCard's Application Identifier Class. In I. Attali and T. Jensen, editors, *Proceedings of the JavaCard Workshop*, LNCS. Springer, 2001.

[Che00] Z. Chen. *Java Card Technology for Smart Cards.* The Java Series. Addison-Wesley, 2000.

[HNSS00] U. Hansmann, M.S. Nicklous, T. Schäck, and F. Seliger. *Smart Card Application Development Using Java.* Springer, 2000.

[Har00] P. Hartel. Formalising Java safety – An overview. In J. Domingo-Ferrer, D. Chan, and A. Watson, editors, *Fourth Smart Card Research and Advanced Application Conference (CARDIS'2000)*, pages 114-134. Kluwer Acad. Publ., 2000.

[Jac99] B. Jacobs. Coalgebras in specification and verification for object-oriented languages. Newsletter 3 of the Dutch Association for Theoretical Computer Science (NVTI), 1999.

[JP00] B. Jacobs and E. Poll. A logic for the Java Modeling Language JML. Techn. Rep. CSI-R0018, Comput. Sci. Inst., Univ. of Nijmegen. To appear at FASE'01., 2000.

[JR97] B. Jacobs and J. Rutten. A tutorial on (co)algebras and (co)induction. *EATCS Bulletin*, 62:222–259, 1997.

[LBR99a] G.T. Leavens, A.L. Baker, and C. Ruby. JML: A notation for detailed design. In H. Kilov and B. Rumpe, editors, *Behavioral Specifications of Business and Systems*, pages 175–188. Kluwer, 1999.

[LBR99b] G.T. Leavens, A.L. Baker, and C. Ruby. Preliminary design of JML: A behavioral interface specification language for Java. Techn. Rep. 98-06, Dep. of Comp. Sci., Iowa State Univ. (`http://www.cs.iastate.edu/~leavens/JML.html`), 1999.

[Loop] Loop Project. `http://www.cs.kun.nl/~bart/LOOP/`.

[PBJ00] E. Poll, J. van den Berg, and B. Jacobs. Specification of the JavaCard API in JML. In J. Domingo-Ferrer, D. Chan, and A. Watson, editors, *Smart Card Research and Advanced Application*, pages 135–154. Kluwer Acad. Publ., 2000.

[PBJ01] E. Poll, J. van den Berg, and B. Jacobs. Formal specification of the JavaCard API in JML: the APDU class. *Comp. Networks Mag.*, 2001. To appear.

[Rei95] H. Reichel. An approach to object semantics based on terminal co-algebras. *Math. Struct. in Comp. Sci.*, 5:129–152, 1995.