# 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 de NVTI Nieuwsbrief van 1999 aan. Naast de nuttige adreslijst (in bijgewerkte versie) vindt u ook als vast element de statuten. Ook het programma van de Theoriedag 1999 vindt u in dit nummer. Voorts vindt u bijdragen van de onderzoekscholen IPA, OZL en SIKS. Het doet ons genoegen dat ook dit jaar enkele van onze collega's bereid zijn gevonden een korte bijdrage te schrijven over een thema dat naar wij hopen en verwachten relevant en interessant is voor een groot deel van onze gemeenschap. Ook dit jaar is de Theoriedag, van 19 maart, alsmede dit Nieuwsbrief-nummer, weer tot stand gekomen dankzij externe steun en sponsoring. Hiervoor danken wij SION, de onderzoekschool IPA, en Elsevier's Publishing Company.

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)
Prof.dr. J.W. Klop (VUA/CWI) voorzitter
Prof.dr. J.N. Kok (RUL)
Prof.dr. G. Rozenberg (RUL)
Dr. J.J.M.M. Rutten (CWI) secretaris
Dr. J. Torenvliet (UvA)

## 3 Van de voorzitter

Geacht NVTI-lid,

Graag wil ik u kort informeren over het reilen en zeilen van de NVTI, in dit derde nummer van de Nieuwsbrief. Wat onze activiteiten betreft zal de Theoriedag van vrijdag 19 maart nauwelijks aan uw aandacht ontsnapt kunnen zijn. Verder is het NVTI bestuur bezig om ook andere initiatieven te ontplooien waarover we zullen rapporteren in de Algemene ledenvergadering op de Theoriedag. Het NVTI bestuur is bezig enkele wijzigingen te ondergaan. Zoals op de vorige Theoriedag een jaar geleden aangekondigd, is Grzegorz Rozenberg als voorzitter opgevolgd door ondergetekende, met Jan Rutten als secretaris. Ook hier danken we Grzegorz voor zijn grote inzet en het vele werk dat hij het afgelopen decennium in zijn voorzittersperiode verricht heeft voor de NVTI, in vorige incarnaties nog bekend als VTI en WTI. We hopen het heilige vuur van de theoretische informatica naar zijn voorbeeld te blijven verzorgen en door te geven - en zo mogelijk nog aan te wakkeren. Tot ziens op 19 maart!

Jan Willem Klop, voorzitter NVTI

## 4 Theoriedag 1999

### Vrijdag 19 maart 1999, Vergadercentrum Hoog Brabant Utrecht

Het is ons een genoegen u uit te nodigen tot het bijwonen van de Theoriedag 99 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 99 zal gehouden worden op vrijdag 19 maart 1999, in het Vergadercentrum Hoog Brabant te Utrecht (in CS Utrecht, op drie minuten loopafstand van de trein), en is een voortzetting van de reeks jaarlijkse bijeenkomsten van de NVTI die vier 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, waaronder: -Embedded systemen, -Quantum computing, -Semantiek. Naast deze 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. G. Plotkin, Universiteit van Edinburgh
11.00-11.30: Koffie
11.30-12.20: Lezing Prof.dr. F. Vaandrager, Katholieke Universiteit Nijmegen
12.20-12.50: Presentatie Onderzoekscholen (OZL, IPA, SICS)
12.50-14.10: Lunch
14.10-15.00: Lezing Prof.dr. I. Damgaard, BRICS, Aarhus 15.00-15.20: Thee
15.20-16.10: Lezing Dr. H. Buhrman, CWI, Amsterdam
16.10-16.40: Algemene ledenvergadering NVTI

### Lidmaatschap NVTI

Alle leden van de voormalige WTI (Werkgemeenschap Theoretische Informatica) zijn automatisch lid van de NVTI geworden. Aan het lidmaatschap zijn geen kosten verbonden; u krijgt de aankondigingen van de NVTI per email of anderszins toegestuurd. Was u geen lid van de WTI en wilt u lid van de NVTI worden: u kunt zich aanmelden bij het contactadres beneden (M. Brune', 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 in het Vergadercentrum Hoog Brabant 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 (12 maart). De kosten kunnen ter plaatse voldaan worden; deze bedragen (ongeveer) f 27,50. 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.

### Abstracts van de voordrachten Abstract Syntax and Variable Binding

Prof.dr. G. Plotkin, University of Edinburgh

Abstract Syntax is needed whenever one writes systems dealing with languages - at least programming languages or logical languages. The situation is well understood in what might be called the context-free case. There, abstract syntax can, for example, be modelled via initial multi-sorted algebras, and represented in programming languages via recursive types. In the case of languages with variable binding the situation is less clear, and various proposals have been made. We advocate an indexed — in fact, presheaf-theoretic — approach where the indices are contexts. This yields an algebraic view and seems also to support a programming language recursive type view. We hope to show how this can be encapsulated in a suitable type theory.

F. Vaandrager, KUN **Root Contention in IEEE 1394**
Prof.dr. F. Vaandrager, Katholieke Universiteit Nijmegen

The model of probabilistic I/O automata of Segala and Lynch is used for the formal specification and analysis of the root contention protocol from the physical layer of the IEEE 1394 ("FireWire") standard. In our model of the protocol both randomization and real-time play an essential role. In order to make our verification easier to understand we introduce several intermediate automata in between the implementation and the specification automaton. This allows us to use very simple notions of refinement rather than the more general but also very complex simulation relations which have been proposed by Segala and Lynch. (Joint work with Marielle Stoelinga)

**Unconditionally Secure Cryptography; was Shannon too Pessimistic?**
Prof.dr. I. Damgaard, BRICS, Aarhus

Most of the cryptography used today, on the Web, in homebanking systems etc. is based on computational assumptions, such as hardness of factoring integers. Assumptions we do not (yet) know how to prove. It is a common misconception that there is nothing to do about this. In fact, it has been believed since 1948 that Shannon showed that unconditionally secure cryptography cannot be useful in practice. For instance, it is only possible to communicate n bits in perfect secrecy if sender and receiver already share at least n secret bits (and these can only be used once!)

However, Shannons pessimistic results only hold in a particular scenario, namely when one assumes that an adversary trying to attack our system has complete and totally reliable information on every message communicated.

This assumption breaks down in many natural scenarios, for example:

- if an adversary cannot eavesdrop or control all players in a network
- if the communication channels used are noisy (as most real channels are)
- if quantum communication is used (since a piece of communicated quantum information cannot be measured completely reliably)

In all these cases, recent research has shown that essentially any cryptographic task can be done with unconditional security. For instance, two players who do not share any secrets initally can establish a shared secret under the nose of an unbounded eavesdropper - something which is impossible in Shannon's world.

**Quantum Communication Complexity**
Dr. H. Buhrman, CWI, Amsterdam

The new paradigm of quantum computing makes use of quantum mechanical effects to speed up computation. It has been shown by Shor that factorization of a number M can be done in polynomial time on a quantum computer. In comparison the best known classical algorithms take close to exponential time. Whereas classical computers operate on bits, quantum algorithms make essential use of bits in superposition: qubits.

Qubits can—just as classical bits—be used to code information. A fundamental result in quantum information theory by Kholevo (1973) shows that k qubits can not contain more information than k classical bits. Nevertheless it can be shown that communication via qubits can drastically reduce the communication cost in the setting of communication complexity.

We will give an overview of recent results obtained in this area as well as a short introduction to quantum computing.

## 5 Mededelingen van de onderzoekscholen

Hieronder volgen korte beschrijvingen van de onderzoekscholen:

- Instituut voor Programmatuurkunde en Algoritmiek;
- Landelijke Onderzoekschool Logica;
- School voor Informatie- en KennisSystemen;

### 5.1 Institute for Programming research and Algorithmics

In 1998, the composition of the research school IPA (Institute for Programming research and Algorithmics) 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.

In the standard curriculum the following activities took place: the IPA Spring-days, dedicated to the subject Parallelism, the Fall-days which focussed on Software Renovation, and two basic courses, Software Technology and Algorithmics, designed to provide Ph.D. students with an overview of IPA's main research areas.

Besides that, IPA was involved in the organisation of the Third Dutch Proof Tools Day and the international workshop User Interfaces for Theorem Provers which was held in conjunction with the workshop 'Calculemus and Types '98'. Working together with the research schools BRICS (Denmark) and TUCS (Finland) in the European Educational Forum (EEF), the Synergos Summerschools on 'Crypthography and Data Security' and 'Specification, Refinement and Verification' were realized.

Finally, we organised two so-called Feedback Meetings, designed to bring people from academia and industry together to discuss controversial issues in areas which are of interest to both communities. One meeting was dedicated to Software Architecture, the other to Software Renovation.

#### Activities in 1999

Together with BRICS and TUCS, IPA has organised a series of summerschools sponsored by the EC in the TMR program "Synergos" in the last few years. In the coming months the last two schools in this series will take place. The penultimate Synergos Summerschool, on Logic and Computation, will be realised by EEF in cooperation with a newly formed British inter-university research school, the UK Institute of Informatics (UKII).

On the homefront, IPA will organise at least the Spring-days on Probabilistic Methods in Computer Science, the Fall-days and two standard courses: Formal Methods and Software Technology, each covering one of IPA's main research areas. To stay informed on the activities of IPA, you can download the IPA newsletter from our Web-site.

IPA **Spring-days** on **Probabilistic Methods**
*April 7-9, 1999, De Brug, Mierlo, The Netherlands.*
This seminar discusses applications of Probabilistic Methods in the different research areas within IPA, in particular: Verification, Performance Analysis, and Algorithmics.
See: http://www.win.tue.nl/cs/ipa/activities/lentedagen99.html

**Synergos Summer School** in **Logic and Computation**
*April 10-13, 1999, Heriot-Watt University, Edinburgh.*
This school is organised by the UKII in cooperation with the EEF. The list of invited speakers includes Samson Abramsky and Bob Constable.
See: http://www.cee.hw.ac.uk/~fairouz/workshop2.html

**Synergos Summer School** in **Semantics of Computation**
*May 3-7, 1999, University of Arhus, Denmark.*
This school is the last in a series of "Summer Schools on the Foudations of Computer Science", based on the Handbook of Logic in Computer Science (eds. Ambramsky, Gabbay and Maibaum, Oxford University Press).
See: http://www.brics.dk/Activities/99/SemanticsSchool/

## Addresses

| **Visiting address** | **Postal address** |
|---|---|
| Eindhoven University of Technology | IPA, Fac. of Math. and Comp. Sci. |
| Main Building HG 7.17 | Eindhoven University of Technology |
| Den Dolech 2 | P.O. Box 513 |
| 5612 AZ Eindhoven | 5600 MB Eindhoven |
| The Netherlands | The Netherlands |

tel. (+31)-40-2474124 (IPA Secretariat)
fax (+31)-40-2463992
e-mail ipa@win.tue.nl url http://www.win.tue.nl/cs/ipa/

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

The Dutch Research School in Logic 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 forbid 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, in Spring and in Autumn. To give an idea of the contents, here is an overview of the Autumn 1998 School Week in Nunspeet. The curriculum consisted in tutorials in logic and category theory, logic and constraint solving, logic and linguistics, and logic and philosophy. Lecturers were Jaap van Oosten (Basic Category Theory), Krzysztof Apt (Constraints: Logic, Programming and Applications), Michael Kohlhase (Higher Order Unification with NL Applications), and Theo Kuipers (Formal Methods in the Philosophy of Science).

In combination with the Autumn School Week the yearly Accolade Event takes place: an occasion where PhD students within the school present their work to the outside world in an informal setting. Recently, the character of this event has changed. It was felt by the OzsL Board that previous Accolade meetings had to strike an uneasy compromise between informing the community about how individual Ph.D. projects were progressing on one hand, and functioning as an internal workshop where our Ph.D. students could fine-tune their conference presentation skills in a friendly and supportive setting. Therefore the Board decided to separate these two functions, and to relegate the second to the regular colloquia.

Accolade New Style now 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. The first time it was held, Accolade New Style was a tremendous success.

This year in Spring, a complement event to Accolade is planned. The AXE (Accolade Adult XXX-Rated Event) will create an opportunity for staff members within the School to give brief outlines of their research. AXE will take place on Thursday April 1 (no joke!) at Uil-OTS, Trans 10, Utrecht. (Trans 10 is in the centre of town.)

Other important upcoming events in which the School is heavily involved are the ASL Logic Colloquium '99, in Utrecht, August 1–6, and the European Summer School in Logic, Language and Information (ESSLLI), in Utrecht, August 9–20, with several ESSLLI related workshops connected to it. Logic Colloquium '99 offers tutorials by leading researchers in set theory, model theory, category theory and computational logic (respectively Greg Hjorth, Anand Pillay, Ieke Moerdijk, Jan Willem Klop), invited lecturers by Samson Abramsky (Edinburgh), Alessandro Andretta (Torino), Sergei Artemov (Moscow), Lev Beklemishev (Moscow), Peter Cholak (Notre Dame, Indiana), Deirdre Haskell (Worcester, MA), Dale Miller (Philadelphia), Andrei Morozov (Novosibirsk). Jan Rutten (Amsterdam), Patrick Speissegger (Toronto), Steve Todorcevic (Paris), Andreas Weiermann (Muenster), plus contributed talks. Deadline for submission of a contributed talk is April 2, 1999. `http://www.cwi.nl/lc99` gives more information.

ESSLLI is a two-week roller coaster of lectures, workshops, tutorials, courses, gossip, dinners, disco parties and socializing. See `http://esslli.let.uu.nl/` for further information.

## 5.3 School voor Informatie- en KennisSystemen SIKS, door: R.J.C.M. Starmans

### Inleiding

De School for Information and Knowledge Systems (SIKS) zag begin 1996 het licht. Het initiatief tot de oprichting was drie jaar eerder genomen door een groep onderzokers op het terrein van kunstmatige intelligentie, database theorie en software engineering. Zij trokken zich enige dagen terug op het Vlaamse platteland om de haalbaarheid te onderzoeken van een nieuwe onderzoekschool. Deze zou zich moeten toeleggen op fundamenteel en toepassingsgericht ondezoek op het gebied van de informatica, meer in het bijzonder op het terrein van informatie- en kennissystemen. Daarnaast moest zij borg staan voor een hoogwaardige promovendi-opleiding en bovenal een eigen en herkenbare plaats innemen ten opzichte van andere samenwerkings- verbanden binnen de informatica in Nederland.
Het benodigde universitaire draagvlak bleek aanwezig en de initiatieven resulteerden in 1994 in een formeel voorstel tot oprichting van onderzoeschool SIKS. Twee jaar later werd conform de WHW en overeenkomstig het verkavelingsplan van de Stichting Informatica Onderzoek Nederland (SION) een interuniversitaire samenwekingsovereenkomst gesloten. SIKS kon van start onder het penvoerderschap van de Vrije Universiteit van Amsterdam. De Universteit Utrecht, de Technische Universiteit Delft, de Technische Universiteit Eindhoven, de Universiteit Twente, de Rijksuniversiteit Leiden en de Universiteit Maatricht traden eveneens Ook met het Centrum voor Wiskunde en Infortica, de Erasmus Universiteit Rotterdam en de Katholieke Universiteit Brabant werd een samenwerkingsverband aangegaan.

### KNAW-erkenning in 1998

Al spoedig groeide het besef dat accreditatie door de Erkenningscommissie van de KNAW een belangrijke waarborg vormt voor de continuëteit van de In 1997 besloot het bestuur daarom na overleg met de programmaleiders dat per 1 januari 1998 een erkenningsaanvraag bij de KNAW moest worden ingediend. Een grondige herbezinning op de positie van de school was het gevolg. Zo werd bij voorbeeld de missie van SIKS, zoals indertijd verwoord in het oorspronkelijke oprichtingsvoorstel, ingrijpend herzien. Ook koos SIKS er nadrukkelijk voor het onderzoek te centreren rond een vijftal researchfocussen: knowledge science, coöperatieve systemen, requirements engineering en formele specificatie van IKS, multimedia en tot slot architecturen van informatiesystemen. De keuze en invulling van deze aandachtsgebieden werden ingegeven door interne ontwikkelingen binnen de wetenschapsgebieden waarop SIKS actief is, maar sloten bovendien nauw aan bij externe beleidsinitiatieven en onderzoeksagenda's van met name NWO en SION. Ondertussen werd hard gewerkt om een solide onderwijsprogramma van de grond te krijgen, het ondezoeksprogramma te versterken, de interne organisatie van SIKS te verbetren en bovenal de herkenbaarheid van de school te bevorderen door een duidelijke positionering ten opzichte van de twee andere Nederlandse onderzoekscholen op het terrein van de informatica. Uitgaande van het oordeel van de Erkenningscommissie Onderzoekscholen lijkt SIKS in deze opzet te zijn geslaagd. In mei 1998 was de erkenning door de KNAW een feit. De commissie betoonde zich cotent met de missie van de school, die naar haar oordeel "complementair is aan die van de beide andere onderzoekscholen op het terrein der informatica". Waardering was er ook voor het onderwijsprogramma. De commissie acht dit "goed gestructureerd" en voorzien van een "welomschreven opleidingsdoelstelling". Speerpunten hierin vormen een homogeniseringsfase en een basisfase die een gemeenschappelijke basis moeten bewerkstelligen bij alle SIKS-promovendi. Voornoemde basisfase krijgt gestalte via een achttal door SIKS zelf ontwikkelde cursussen, waarin onder meer kennismodelleren, systeemmodelleren, databases, combinatoriek, intelligente systemen en interactieve systemen centraal staan. Hoge verwachtingen heeft de commissie van de vijf researchfoci. Zij verwacht dat het onderzoek zich gedurende de komende erkenningsperiode voor een belangrijk deel zal richten op deze thema's en meent bovendien dat dit de coherentie van het onderzoeksprogramma ten goede zal komen. Het bestuur van de school dient volgens de commissie dan ook via centrale aansturing erop toe te zien dat dit alles ook daadwerkelijk plaatsvindt.

## Activiteiten

Ofschoon het weinig bevreemding zal wekken dat 1998 voor SIKS voor een belangrijk deel in het teken stond van de KNAW-erkenning, vonden de reguliere activiteiten uiteraard gewoon doorgang. Een aantal zal hier kort de revue passeren. Om te beginnen werd een tweetal landelijke promovendicursussen georganiseerd, die ook voor promovendi van andere scholen en (in beperkte mate) voor externe deelnemers werden opengesteld. Eveneens in mei 1998 vond aan de Katholieke Universiteit Brabant een SIKS-themadag plaats gewijd aan "Natuurlijke taal en IKS". In oktober werd in Nijmegen de SION-databasedag georganiseerd aan de Katholieke Universiteit Nijmegen in samenwerking met SIKS. Ook kon in 1998 een groeiend aantal promoties binnen de school worden gesignaleerd, zowel van eerste-geldstroom promovendi, als van promovendi die door de industrie worden bekostigd. Om die reden is SIKS van start gegaan met een eigen dissertatie-reeks, waarin in het vervolg de SIKS-proefschriften zullen worden opgenomen. De activiteiten van SIKS die in 1999 staan gepland zijn deels een voortzetting c.q. uitbreiding van het bestaande programma, maar zullen tevens nadrukkelijk aansluiting zoeken bij de aanbevelingen van de erkenningscommissie. Van 31 mei tot en met 4 juni 1999 staan de cursussen Combinatoire methoden en Intelligent Sytems op het programma. Van 29 november tot 3 december volgen dan Systeemmodelleren en Kennismodelleren, die eveneens tot het basisprogramma van de school behoren. In september 1999 zal een internationaal wetenschappelijk symposium worden georganiseerd over een nog vast te stellen thema. Voorts vindt in november onder auspiciën van SIKS en in aansluiting op de organisatie van BNAIC 98 in Maastricht een databasedag plaats, alsmede een masterclass voor promovendi en een doctoraalconsortium. Het is de bedoeling wetenschappelijke en meer gespecialiseerde onderwijsactiviteiten zoveel mogelijk te laten aansluiten bij de research focussen. Voor meer informatie over SIKS en haar activiteiten kunt u de site van de onderzoekschool raadplegen (www.siks.nl) of contact opnemen met het Bureau van SIKS: 030-2534083, email: office@siks.nl.

# 6 Wetenschappelijke bijdragen

## 6.1 Model-based specification of design Patterns, door: the SOOP-working group

# Model-based specification of design patterns

SOOP-working group[1][2]
Department of Computing Science
Eindhoven University of Technology

## Motivation

A considerable interest in design patterns has sprouted in the last couple of years. This interest has been stimulated by the appearance of some excellent books on the subject. A notable example is the book by Gamma et al. [1]. At the time the first ideas of patterns in software engineering were developed, a couple of essential ingredients were available. First, the mechanisms of behavioral and structural abstraction as offered by object-oriented methods (i.e. encapsulation and inheritance). Second, the availability of a notation to communicate the patterns (i.e. OMT [2] as proper predecessor of UML [3]), and last but not least, relevant design-experience with object-oriented systems. The latter is important because patterns are supposed to capture experience on how to solve certain often-occurring problems in system design. Many of the patterns described by Gamma are intended for improving a design by changing class dependencies (possibly without affecting resulting object dependencies) such that groups of classes are de-coupled from others. For example, the observer pattern can be used to make problem specific classes ("concrete subject") independent of the user interface ("concrete observer") such that the changes in the user interface will not affect the implementation of a problem solution. It is interesting to observe that such patterns lean very heavily on inheritance and the related extended object substitutability.

Although the book of Gamma is impressive in its clarity, it has the drawback that it describes the intended application of patterns informally and intuitively. The descriptions of how a pattern works are typically operational. Although they can be understood well by the human reader, such an informal approach hampers specification and implementation of a level of tool-support for design patterns that goes beyond (trivial) diagram manipulation. In addition, it is very difficult to establish correctness of a design or implementation as a whole and the correct application of a pattern in particular. Therefore, the SOOP working group in Eindhoven has set itself the goal to develop a specification formalism that helps to formally specify patterns. The result of the attempt should be the construction of tools supporting pattern application and ultimately the establishment of a formal pattern language.

## Requirements for a specification formalism

When one tries to develop a specification formalism that is to be used in practice, one must consider the generally accepted design approach and try to adapt the formalism to it, rather than adapt the design approach to the formalism. Following the principle of direct mapping (see Meyer [4]), design of an object-oriented program starts from a model of the problem domain. This model is successively refined into an implementation adding user-interface classes, adding auxiliary container classes that group objects in a way that facilitates efficient inter-object navigation, and applying design patterns to improve the modularity of the class structure. In our opinion, a specification formalism must tie in with this approach, i.e. it must parallel the decomposition of an object-oriented model and its refinement from analysis to implementation. In fact, decomposition of the implementation and specification should be isomorphic, i.e., yield a one to one correspondence between certain specification expressions and classes in the implementation. A consequence is that a specification of a class's behavior must only involve specifications of classes on which it directly depends. Specification fragments can then be taken from a context and can be reused elsewhere, just as implementation fragments can. Design patterns can be seen as such specification fragments.

## Model based specification

We follow the approach of *Design by Contract* [4,5], particularly advocated by Meyer. A contract is a specification of benefits and obligations in interactions between classes. In each interaction one class is the client, the other the supplier. A contract can be specified as pre- and post-conditions on operations

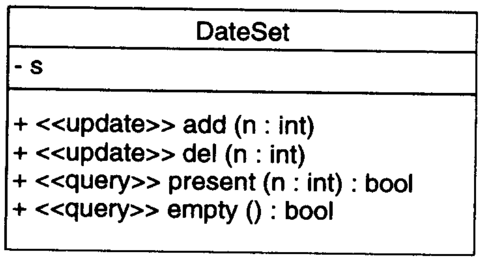
---

[1] Structured Object-Oriented Programming; email: `soop_l@win.tue.nl`;

[2] members: Lex Bijlsma, Rik van Geldrop, Louis van Gool, Kees Hemerik, Kees Huizing, Ruurd Kuiper, Onno van Roosmalen, Jaap van der Woude, Gerard Zwaan.

offered on the interface of a class. A pre-condition specifies the conditions that must be realized before the operation is invoked by an instance of the client class and the post-condition specifies the obligations of the invoked instance of the class, the supplier.
There are several methods to specify contracts. We think that the most intuitive and concise way to express them is by way of a *model*. A model is a mathematical way to describe the state of an object. It must be detailed enough to express the result of method invocations. A model is intuitively appealing because it is very close to a description in terms of implementation variables. Although it may give a bias towards the actual implementation of the specified behavior, it does not force a particular implementation upon the programmer. In actual fact, rules can be given that relate a model and the method specifications to a specification of behavior in terms of implementation variables.

Example


DateSet
- s
+ <<update>> add (n : int)
+ <<update>> del (n : int)
+ <<query>> present (n : int) : bool
+ <<query>> empty () : bool

Consider a class DateSet depicted in the above UML diagram. The purpose of objects of this class is to keep a number of dates of the month, e.g. to be used for certain appointments. We adopt the following model for this class.

**model** : $s \in \wp(Z)$

The restriction on the values of dates in s is introduced using a, so called, *model invariant*.

**inv** $I(s)$ : $(\forall i : i \in s : 1 \le i \le 31)$

Naturally, the possible use and implementation of this class entirely depends on the offered operations and their specified behavior, and not on this model.

$\{1 \le n \le 31\}$ DateSet::add (n : int) $\{$ modify( $\{s\}$ ) $\wedge\ s = s_{\bullet} \cup \{n\}$ $\}$
$\{1 \le n \le 31\}$ DateSet::del (n : int) $\{$ modify( $\{s\}$ ) $\wedge\ s = s_{\bullet} \setminus \{n\}$ $\}$
$\{$ true $\}$ DateSet::present(n : int) : bool $\{$return $(n \in s)\}$
$\{$ true $\}$ DateSet::empty() : bool $\{$ return $(s = \varnothing)\}$

The modify clause is a shorthand for the restriction that all other variables of the (current) model except the ones specified in the set, must be left unchanged by the operation. Absence of this clause implies no restrictions. However, queries may only return values and the restriction modify($\varnothing$) is always implicitly assumed. The dotted variable, e.g. $s_{\bullet}$ , indicates the old value of s, i.e. the value in the precondition. Consider now an implementation of DateSet. We assume that a boolean array is used to keep track of which days belong to the set s . Hence a[i-1] has the value true if $i \in s$ . This relation can be expressed in a, so called, *implementation invariant.*

**inv** $Im(a,s)$ : $(\forall i : 1 \le i \le 31 : a[i-1] = i \in s)$

The pre- and post-conditions applicable to an operation m() implemented using this representation, i.e. in terms of the implementation variables must satisfy the following refinement rule for any of its operations m() with pre- and post-conditions $Pre_m$ and $Post_m$ :

$$\{ (\exists s :: I(s) \wedge Im(a,s) \wedge Pre_m(s)) \}\ \text{DateSet::m()}\ \{ (\exists s :: I(s) \wedge Im(a,s) \wedge Post_m(s)) \} \quad (1)$$

From this rule we can easily derive the following pre- and post-conditions for, e.g., the operation DateSet::add in terms of the implementation variables a[i].

$\{1 \le n \le 31\}$ DateSet::add(n : int) $\{$ modify(a[n-1]) $\wedge$ a[n-1] $\}$

This suggests the simple implementation of setting a[n-1] equal to true.
End of example

An identical relationship to the refinement rule (1) exists between specifications of methods in subclasses of classes. In that case the implementation invariant in (1) has to be replaced by the *subtype invariant* which relates the variables of the sub-class to those of the super-class. Meyer [4] provides some rules as to how pre- and post-conditions and invariants may change under inheritance and when invariants should hold. In general it is too restrictive to require that the invariant holds during the complete execution of a method. The usual scheme is that every method has to restore the invariant at the end of the method and that it may assume that the invariant holds at the beginning. In the presence of **re-entrancy**, however, this scheme is too weak. In the next section we will deal with this problem.

**Proof obligations and re-entrancy**

For simplicity, in this section we assume that the model and the implementation are identical, so we leave out the implementation invariant and only mention the model invariant I. Furthermore, we assume that all constructors establish the invariant. Consider the following general form of a method with its specification and implementation:

$$\{ Pre_m \}\ m()\ \{ Post_m \}\ :\ \{Q_1\}\ S1;\ \{ R_1\}\ o1\text{->}m1();\ \{ Q_2 \}\ S2;\ \{ R_2\}\ o2\text{->}m2();\ldots\ Sn\ \{R_n\}$$

The proof obligations for methods are, to begin with:

1. $( Pre_m \wedge I_{this} ) \rightarrow Q_1$ and $R_n \rightarrow ( Post_m \wedge I_{this} )$
2. $\{ R_i \}$ oi->mi() $\{ Q_{i+1} \}$
3. $\{Q_i\}$ Si $\{ R_i \}$

Since various forms of mi() may be invoked depending on the dynamic type of the object that o1 refers to, the following rules apply. Assume that the static type of o1 is C and D is a sub-type (sub-class) of C we write D<:C. Since, by hypothesis, that the subtype invariant between D and C is trivial, we find from the refinement rule (1):

$$Pre_{C::mi} \rightarrow Pre_{D::mi} \quad \text{and} \quad ( Post_{D::mi} \wedge I_D ) \rightarrow ( Post_{C::mi} \wedge I_C )$$

This is intuitive because D::mi() can be actually invoked wherever mi() appears. This states that the post-condition and invariants must be strengthened in such subclasses and preconditions must be weakened.

As follows from proof obligation 1, the invariant may be assumed when a method starts and need not be established by the caller. The reason is that the client object will be found to satisfy its invariant automatically if every operation on the object satisfies the invariant. This reasoning, however, is not necessarily correct. The problem here is **re-entrancy**. Assume in the above program sketch that the object o invokes operations of other objects and that ultimately control re-enters the object, this, executing m(). At that point we have no guarantee that the invariant holds unless we demand that:

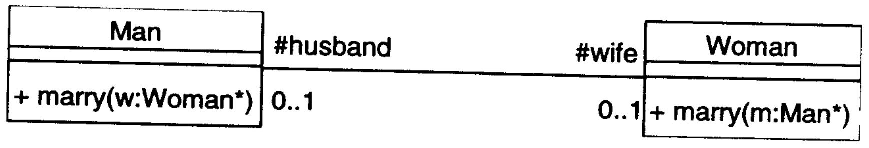
$$R_i \rightarrow I_{this}$$

Thus it seems necessary to demand some stronger property of the invariant namely that it holds whenever control leaves an object.

**Vulnerability of invariants**

Another problem is possible sensitivity of invariants with respect to modifications in other objects. It is possible that the invariant $I_{this}$ is formulated in terms of properties of other object this depends on, e.g. the object referred to by o in the above program. This is no problem in the above method m() because we explicitly require restoration of the invariant by execution of this method. However, if methods of the object referred to by o are invoked directly, i.e. without going through this, it can no longer be guaranteed that the invariant $I_{this}$ holds when entering m(). We say that $I_{this}$ is **vulnerable** to changes in the objects o can refer to.

We consider the *forward-backward problem* as described by Meyer [4]. It concerns a hypothetical variant of asymmetric marriage relations.



We assume asymmetric model for men and women.

Man: **model** wife : Woman*
**inv** M : wife ≠∅ → wife->husband = this
Woman: **model** husband : Man*

Thus it is always required that if a man has a wife, his wife has him as a husband. The converse is not required. Although the following method implementation seems to preserve the invariant

{ w≠∅ } w->marry(this); wife = w; {modify( {wife} ) ∧ wife = w ∧ wife->husband=this},

even in the strong sense (at invocation of w->marry(this) ) it is not preserved in general. For example, the execution of the following program fragment

```
Man m1,m2;
Woman w;
m1.marry(&w);
w.marry(&m2); { m1.wife->husband = &m2 } ,
```

that may appear anywhere, will break the invariant of the object m1. Note that introduction of a tighter specification for the marry operation in Woman would not solve the problem because new methods can be introduced in derived classes that could destroy the invariant of men in a similar fashion. We have to solve this problem because we do not want correctness of a class-implementation to depend on code that can appear just anywhere and could even be added in later program extensions. The solution we propose is to introduce a so-called *post-invariant* that restricts changes in women, i.e. must be added as a conjunction to all methods' post-conditions.

Woman: **model** husband : Man*
**post** P: husband. ≠∅ ∧ husband.->wife = this → husband = husband.

Furthermore we have to restrict calls to Woman::marry to those women who are not someone's wife by strengthening the precondition:

{ m≠∅ ∧ (husband=∅ ∨ husband->wife≠this)}
Woman::marry(Man* m)
{ modify( {husband} ) ∧ husband = m }

With this specification of Woman it can now be shown that the invariant for men can not be broken by methods of Woman. Note the appearance of the dot-notation in the post-invariant. Note that we have indeed arrived at a situation where we only need to consider the local view from class Man to satisfy our proof obligations. This is what we want to achieve in general. It would mean that proofs can be provided class by class in an incremental fashion. The feasibility of this goal has not yet been established.

Note that in the specification of the class Man other possible side-effects on women, e.g. a change of a possible variable name is not specified. Since the modify clause is defined as only referring to local variables, it does not forbid these side-effects. We could extend the semantics of the modify clause along these lines, but then again we would end up with a non-local specification. Therefore, we propose to formulate restriction locally by explicitly specifying which methods may be called. For example the post-condition of Man::marry method we extend to:

modify( {wife} ) ∧ calls( {w->marry} ) ∧ wife = w ∧ wife->husband=this

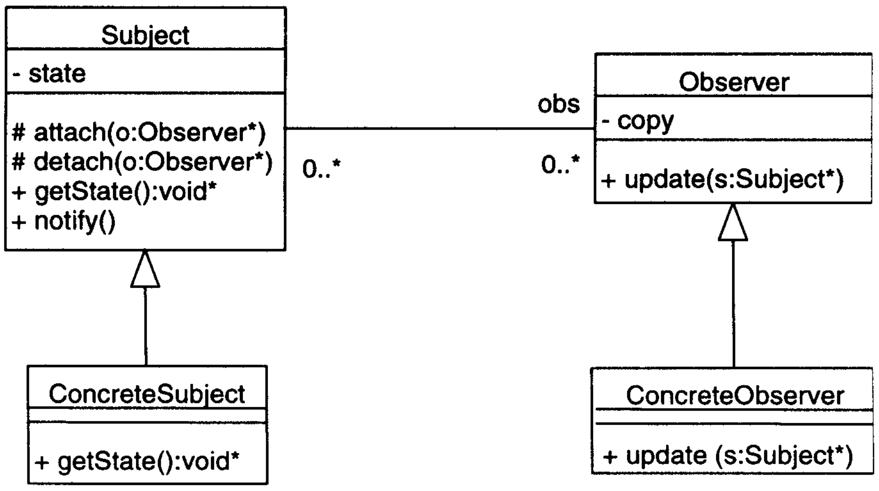
The calls-clause implies the possible (external) modifications as expressed in the modify-clauses of the callable methods. Note that calls( {w->marry} ) does not imply that w->marry will definitely be invoked.

Tracing the calls-clauses it is possible to establish whether re-entrancy is at all possible or not. Consider our husbands and wives. It can be seen that it is not possible to enter any instance of the class Man before the Man::marry() method is finished. In this case we can then relax the condition that the invariant is maintained in the strong sense and even the following implementation can be proved correct.

{ $w \neq \varnothing \wedge$ (husband=$\varnothing \vee$ husband->wife$\neq$this)}
wife = w; w->marry(this);
{modify( {wife} ) $\wedge$ calls( {w->marry} ) $\wedge$ wife = w $\wedge$ wife->husband=this }

**The Observer pattern**

We will briefly show how the sketched approach can be applied to the Observer pattern as described in [1]. In addition to the classes Subject and Observer, the pattern describes concrete versions of these classes that can be derived from them. In this process some methods must be defined



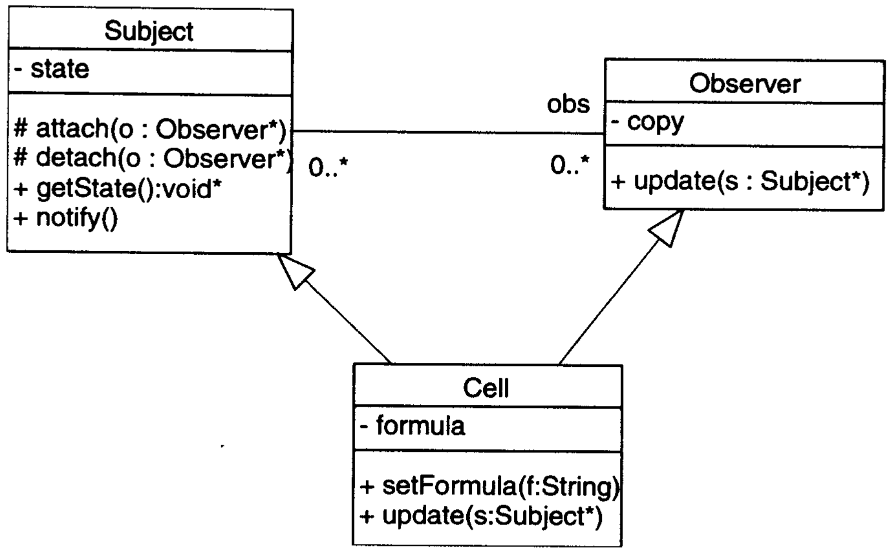
Subject **model** state : void*
obs $\in \wp$( Observer*)
**post** ( $\forall$o : o $\in$obs : o->copy(this) =state )

{ true } Subject::attach(o:Observer*) { modif( {obs} ) $\wedge$ calls( {o->update} ) $\wedge$ obs = obs. $\cup$ {o} }
{ true } Subject::detach(o:Observer*) { modif( {obs} ) $\wedge$ obs = obs. \ {o} }
{ true } Subject::getState() : void* { modif($\varnothing$) $\wedge$ calls($\varnothing$) $\wedge$ return(state) }
{ true } Subject::notify() { modif($\varnothing$) $\wedge$ calls( { o->update | o$\in$obs } ) }

Observer **model** copy $\in$ Subject $\rightarrow$ void*

{ this $\in$ s->obs }
Observer::update(s:Subject*)
{ modif( {copy(*s)} ) $\wedge$ calls( {s->getState} ) $\wedge$ copy(*s)=s->state }

A spreadsheet cell fulfills the roles of a subject and an observer simultaneously. This is readily implemented by multiple inheritance. The model for class Cell contains variables corresponding to the models of both Subject and Observer.



Cell: **model** state : Number
obs : $\wp$ (Observer)
copy : Cell $\rightarrow$ Number
formula : (Cell $\rightarrow$Number) $\rightarrow$ Number
**inv** state = formula(copy) $\wedge$ ($\forall$p: this$\in$FREEVAR(p.formula) $\rightarrow$p$\in$obs)
**post** ($\forall$o: o$\in$obs: o->copy(this) = state)

Just consider the specification for Cell::update :

{ this $\in$ s->obs }
Cell::update(s : Subject*)
{ modif( {copy(*s)} ) $\wedge$ calls( {s->getState, notify } ) $\wedge$ copy(*s)=s->state }

Note that according to the modify-clause in Subject, Subject::state may not be changed by Subject::notify. In cell, however, Cell::state must be kept equal to cell::formula(copy) which is modified by Cell::update. A cyclic dependency of a cell to itself that would lead to such self-modifications is excluded by this specification. Hence, an implementation of Cell would inlcude an invariant that forbids cyclic dependencies.

**References**

[1] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *"Design Patterns, Elements of ReusableObject-Oriented Software"*, Addison-Wesley, 1995.

[2] J.Rumbaugh, M. Blaha, W. Premerlani and F. Eddy, *"Object Oriented Modeling and Design"*, Prentice Hall, 1991.

[3] Grady Booch, James Rumbaugh and Ivar Jacobson; *The Unified Modeling Language User Guide*, Reading Massachusetts: Addison-Wesley, 1995.

[4} Bertrand Meyer, *"Object Oriented Software Construction"*, second edition, Prentice Hall, 1997.

[5] C. Szyperski, *"Component Software, Beyond Object-Oriented Programming"*, Addison Wesley, 1997.

## 6.2 Coalgebras in Specification and Verification for Object-Oriented Languages, door: Bart Jacobs

# Coalgebras in Specification and Verification for Object-Oriented Languages

Bart Jacobs*

Department of Computer Science, University of Nijmegen,
P.O. Box 9010, 6500 GL Nijmegen, The Netherlands.
Email: bart@cs.kun.nl URL: http://www.cs.kun.nl/~bart

**Abstract**

The aim of this short note is to give an impression of the use of coalgebras in specification and verification for object-oriented languages. Particular emphasis will be given to the rôle of coalgebraic operations in describing state-based systems. At the end some active research topics in coalgebra will be sketched, together with pointers to the literature.

## 1 Introducing coalgebras

One of the new, recent developments in the area of theoretical computer science is the emergence of the field of coalgebra. One formal way of introducing coalgebras is as duals of algebras—in categorical style[1]. As such, coalgebras have been around for several decades. What is recent is the recognition that coalgebras form the underlying structure of various forms of dynamical systems / automata / transition systems. Such dynamical structures have been used for a long time in various situations in mathematics and computer science, but it is only now that people are beginning to identify them as coalgebras: it is fair to say that many people use coalgebras, but do not realise that they do.

Coalgebras may thus be seen as prototype dynamical systems. Such systems often display infinite behaviour, notably when they are supposed to be running forever, like operating systems. Coalgebras capture the idea of a system as consisting of a set of states (often called the state space), together with several operations on states. The state space should be seen as a black box, and the operations as providing interaction with the outside world. Think of yourself as sitting behind your computer: the computer has an internal state, given by, among other things, the contents of its memory cells and its registers. This state is not directly observable, but the screen provides certain observable information about your computer's state. The keyboard provides you with operations for modifying the state of your computer. You are not the only one modifying the state: the pulses of your computer's internal clock are bringing about constant state changes. The idea is that the computer (without its clock) can be seen as a coalgebra: it has a state space and operations for observation and modification.

In the context of dynamical systems / automata / transition systems there are familiar notions of invariance, bisimilarity, and behaviour. An invariant is a predicate on states, which, once it is true, remains true no matter how the state is

*Research Fellow of the Royal Netherlands Academy of Arts and Sciences.

[1] A dualisation of a concept named "X" is usually called "coX" in category theory. Examples are limit and colimit, product and coproduct, monad and comonad, *etc.*

changed. Bisimilarity is the relation on states which contains all pairs of observationally indistinguishable states. The behaviour of a system can be described in terms of the possible observations after series of consecutive state changes, starting in a particular state. Two states are then bisimilar if and only if they give rise to the same observable behaviour. These notions of invariance, bisimilarity and behaviour are given a precise and uniform mathematical foundation in the theory of coalgebras. As such they can be studied at an appropriate level of abstraction. This coalgebraic framework not only provides an abstract system theoretic setting, but also associated proof techniques. Of particular importance is the technique to prove equality of (generally infinite) behaviour via bisimulation relations.

Formally, a coalgebra consists of two parts: a set and a function acting on that set. The set is the state space, and will typically be written as Self. The function (or operation) is of the following special form.

$$\mathsf{Self} \longrightarrow \boxed{\cdots \ \mathsf{Self} \ \cdots} \tag{1}$$

The box on the right-hand-side is some expression involving Self[2]. To be precise, it is the result of applying what is called a "functor" to Self. This right-hand-side describes the interface of the coalgebra to the outside world. Below we shall see several examples of how this works. What is typically coalgebraic is that the function has Self as domain, on the left-hand-side. This reflects the idea that operations act on states. In particular, there is no way to construct a new state from scratch, via such operation of a coalgebra. Initial states thus have to be given as additional structure.

(As an aside, algebras can be described as consisting of a set, say Self, together with a function pointing in the reverse direction:

$$\boxed{\cdots \ \mathsf{Self} \ \cdots} \longrightarrow \mathsf{Self} \tag{2}$$

Hence algebras allow us to construct elements in Self. In general, algebras are appropriate for describing (finite) data types, whereas coalgebras are appropriate for state-based dynamical systems, displaying possibly infinite behaviour.)

Let us see some examples of coalgebras. Consider a (silly) system consisting of a light bulb together with a button. The idea is that if the button is pushed twice, the light changes from on to off, and vice-versa. The state space will thus have to incorporate information about whether the light is on or off, and about whether the next push will result in a change of light. One way of realising such a system is to take as state space (set of states):

$$S_1 = \{0,1\} \times \{0,1\}$$

Then, for a state $(x, y) \in S_1$, the first component $x \in \{0,1\}$ tells whether the light is on or off, and the second component $y$ tells whether a next push will result in a change of light. We thus define two operations on $S_1$, capturing the light bulb:

$$\mathsf{light}_1 \colon S_1 \longrightarrow \{0,1\} \qquad \text{is} \qquad (x,y) \longmapsto x$$

and the push button:

$$\mathsf{push}_1 \colon S_1 \longrightarrow S_1 \qquad \text{is} \qquad (x,y) \longmapsto \begin{cases} (x,1) & \text{if } y = 0 \\ (1-x,0) & \text{if } y = 1. \end{cases}$$

[2]In positive position.

The two operations $\mathsf{light}_1, \mathsf{push}_1$ can be combined into a single function, giving an example of a coalgebra:

$$S_1 \xrightarrow{\langle \mathsf{light}_1, \mathsf{push}_1 \rangle} \{0,1\} \times S_1 \tag{3}$$

Notice that the operation $\mathsf{light}_1$ is for observation, and $\mathsf{push}_1$ is for modification.

A completely different realisation of this same system consists of a state space

$$S_2 = \mathbb{N}$$

with light operation:

$$\mathsf{light}_2 \colon S_2 \longrightarrow \{0,1\} \qquad \text{is} \qquad n \longmapsto \begin{cases} 0 & \text{if } \exists m \in \mathbb{N}.\ m \text{ is even, and} \\ & \qquad n = 2m \text{ or } n = 2m+1 \\ 1 & \text{otherwise.} \end{cases}$$

The realisation of the push button is now very simple:

$$\mathsf{push}_2 \colon S_2 \longrightarrow S_2 \qquad \text{is} \qquad n \longmapsto n+1.$$

This second realisation forms a coalgebra, with the same interface as the first realisation:

$$S_2 \xrightarrow{\langle \mathsf{light}_2, \mathsf{push}_2 \rangle} \{0,1\} \times S_2 \tag{4}$$

But the second state space $S_2$ is clearly different from the first one ($S_1$). For a user on the outside, these differences are not noticeable: both systems display the same observable behaviour.

The idea that will be elaborated in these notes is that coalgebras share certain essential aspects with classes in object-oriented languages. A class has an implicit state space, given, for example, by the Cartesian product of the types of its instance variables (both public and private). Also, it has certain operations (called methods in this context) for observing or possibly modifying the state. The types of the instance variables and of the methods together determine the interface of the class, corresponding to the box in (1), see Section 3 for more details. As a suggestive example, we describe the first realisation (3) as a class in Java.

```
class LightPush {
  private boolean light_status, next_change;
  public boolean light() {
    return light_status;
  }
  public void push() {
    if ( next_change )
      { light_status = !light_status; }
    next_change = !next_change;
  }
}
```

Now that we have seen some examples of coalgebras, we proceed with their use in specification and verification.

## 2 Coalgebraic class specifications

In the previous section we have given an informal description of a system with a light and a push button, such that pushing the button twice changes the status of the light. What would be useful is a formal way of describing such a system. This can be done in what is called a coalgebraic specification. It is a specification based on coalgebras, that may be understood as a specification of a class in an object-oriented language. It consists of several coalgebraic operations (which may be combined into a single coalgebra operation), together with certain assertions. The latter restrict the behaviour of the operations in an appropriate manner. Additionally, a coalgebraic specification may contain (parametrised) initial states, typically written as new, with their own assertions. For example, the light bulb system mentioned earlier can be specified as:

ATTRIBUTE

$\quad \mathsf{light}\colon \mathsf{Self} \longrightarrow \{0,1\}$

METHOD

$\quad \mathsf{push}\colon \mathsf{Self} \longrightarrow \mathsf{Self}$

ASSERTION

$\quad \forall x\colon \mathsf{Self}.\ \mathsf{light}(\mathsf{push}(\mathsf{push}(x))) = 1 - \mathsf{light}(x)$

CONSTRUCTOR

$\quad \mathsf{new}\colon \mathsf{Self}$

CREATION

$\quad \mathsf{light}(\mathsf{new}) = 0 \wedge \mathsf{light}(\mathsf{push}(\mathsf{new})) = 0$

The distinction between attributes and methods is not essential at this stage, but the idea is that an attribute is an operation that is used for observation and that does not change the state. Notice that this specification does not tell us anything about the internal structure of the state space Self. In coalgebraic specification this structure is irrelevant, since the state space is considered to be a black box. The specification only requires the presence of certain operations, displaying certain behaviour. How that behaviour is realised internally should not be part of a specification. For example, both the state spaces $S_1$ and $S_2$ from the previous section may be used to obtain models of this specification. The light and push operartions from the above specification are interpreted as the functions $\mathsf{light}_1$ and $\mathsf{push}_1$ on $S_1$, or as $\mathsf{light}_2$ and $\mathsf{push}_2$ on $S_2$. The initial state new from the specification is interpreted as $\mathsf{new}_1 = (0,0) \in S_1$, or as $\mathsf{new}_2 = 0 \in S_2$. It is not hard to see that the assertion and creation condition from the specification hold, both for $S_1$ and for $S_2$.

We have seen in (1) that the operation of a coalgebra has a simple domain (*viz.* Self), but a structured codomain. This gives great expressivity. For instance, a *partial* operation from Self to Self can be described coalgebraically as

$$\mathsf{Self} \longrightarrow 1 + \mathsf{Self}$$

where 1 is a singleton set, say $1 = \{*\}$, and $+$ is disjoint union. Thus, $1 + \mathsf{Self}$ is the "lift of Self", obtained by adding an extra element. Next, suppose one wishes to specify some store system with an operation to read certain data, say in a set $D$, from the (reading position in the) store. This reading should proceed in such a way that reading means at the same time removing the element that is being read from the store. The coalgebraic way to describe such an operation is as a function

$$\mathsf{Self} \xrightarrow{\mathsf{read}} 1 + (D \times \mathsf{Self})$$

Notice that this read is regarded as a partial operation: if it produces an output in the singleton set $1 = \{*\}$ this means that the store is empty and that there is nothing to read. If the store is non-empty, the read operation produces a pair $(d, y)$

consisting of the data $d \in D$ that has been read, together with the successor state $y$, which is the store state from which $d$ is removed.

Let us turn to assertions in coalgebraic specifications. In principle ordinary logical (first- or higher-order) language may be used here, with one important restriction: coalgebraic specifications should not contain equations between states. This goes against the underlying idea that states are basically inaccessible, so that one is never able to establish actual equality of states. What should be used instead of equality on states is bisimilarity $\leftrightarrow$. Bisimilarity is behavioural indistinguishability, and is equality "as far as we can see". For example, in a system with a do and an undo operation, say both of type Self $\to$ Self, one should *not* write an assertion

$$\forall x\colon \mathsf{Self}.\ \mathsf{undo}(\mathsf{do}(x)) = x \tag{5}$$

but:

$$\forall x\colon \mathsf{Self}.\ \mathsf{undo}(\mathsf{do}(x)) \underline{\leftrightarrow} x \tag{6}$$

Writing an equality as in (5) severely limits the possible models of the specification. For example, it excludes "history" models in which it is internally recorded which operations have been applied. In that case one can still have validity of the assertion (6), if the internal history record is not observable.

There is more to be said about bisimilarity in specifications.

1. The theory of coalgebras provides for each interface (formally, for each functor) a tailored definition of bisimilarity[3] which precisely captures indistinguishability for that interface. This is used in each coalgebraic specification: the operations are given first, and determine the interface, and thus the precise formulation of bisimilarity $\underline{\leftrightarrow}$ for the specification. This relation $\underline{\leftrightarrow}$ can then be used in the subsequent assertions.

2. Bisimilarity $\underline{\leftrightarrow}$ on Self is not enough in assertions, and must be "lifted" to a new relation $\underline{\underline{\leftrightarrow}}$, called observational equality, on arbitrary type expressions. This is done by induction on the structure of these type expressions, in such a way that $\underline{\underline{\leftrightarrow}}$ is equality on constants, and $\underline{\leftrightarrow}$ on Self. On constructed types, say via $\times$ and $+$, the relation $\underline{\underline{\leftrightarrow}}$ is structurally determined by $\underline{\underline{\leftrightarrow}}$ on the components. To see an example, consider the previously mentioned read operation of a store system. Suppose the system also has a insert operation, of type Self $\to$ Self$^D$, for inserting an element from $D$ into the store—where Self$^D$ is the set (or type) of functions from $D$ to Self. If the system under consideration is a LIFO queue, one expects an assertion of the form

$$\forall x\colon \mathsf{Self}, d\colon D.\ \mathsf{read}(\mathsf{insert}(x)(d)) \underline{\underline{\leftrightarrow}} (d, x)$$

It tells that reading after inserting an element $d$ in a state $x$ produces output containing the element $d$ together with a state which is bisimilar to the original state $x$. This involves observational equality $\underline{\underline{\leftrightarrow}}$ lifted to the type expression $1 + (D \times \mathsf{Self})$. For elements $z, w\colon 1 + (D \times \mathsf{Self})$, $z \underline{\underline{\leftrightarrow}} w$ means: either both $z$ and $w$ are in $1 = \{*\}$ and thus equal, or both $z$ and $w$ are in $D \times$ Self and their first components in $D$ are equal and their second components in Self are bisimilar.

Once a coalgebraic specification has been written down, it can be used for (at least) three important purposes. First, one can describe a concrete (intended) model of the specification, *e.g.* to test its consistency[4]. This involves defining a concrete

[3]and also of invariance.

[4]A more demanding challenge is identify the final model of the specification, given by all possible behaviours. It can be used to see that unintended behaviour does not occur in models of the specification.

coalgebra (like in Section 1) together with initial states, and showing that the required assertions hold. Experience shows that such checks often reveal errors in the specification. Secondly, one can do theory development for a coalgebraic specification. This means deriving consequences from the assertions, using ordinary logical rules, plus some rules for bisimilarity. Important to establish are invariants of specifications, telling for example that a certain integer attribute has a positive value. Such invariants express properties that are supposed to hold forever, and trying to prove them sometimes reveals unforeseen behaviour. Thirdly, one can establish refinements between coalgebraic specifications. In that case, one defines "relative models" of one (abstract) specification, assuming a model of another (concrete) specification.

## 3 Coalgebraic program verification

We shall sketch some ingredients of the use of coalgebras in program verification for the object-oriented programming language Java. What we tell here is not very specific for Java, and may apply, in slightly modified form, to other (imperative) languages.

The foundation of coalgebraic program verification for Java involves modeling Java classes as suitable coalgebras, acting on a common global memory. The basic components of Java classes are instance variables, methods and constructors. All of these are modeled as coalgebraic operations[5]—as was already suggested in Section 1. An instance variable, say `boolean b`, gives rise to an operation $b\colon \mathsf{Self} \to \mathsf{bool}$. Methods and constructors give rise to more complicated operations. We concentrate on methods—or more generally, on statements—with return type `void`, which do not produce a result. Statements in Java may hang, *e.g.* because of an infinite while loop or recursion. Also, they may terminate abruptly, *e.g.* because of an exception, say thrown by a division by zero. But other forms of abrupt termination can also occur, caused by a return, a break or a continue statement. And of course a statement can also terminate normally. Coalgebraically, all these different result options can be captured naturally. Statements are coalgebras of the form

$$\mathsf{Self} \xrightarrow{\mathsf{statement}} \mathsf{StatResult(Self)}$$

where $\mathsf{StatResult(Self)}$ is an abbreviation:

$$\mathsf{StatResult(Self)} \;=\; 1 + \mathsf{Self} + \mathsf{StatAbnormal(Self)}$$

and where $\mathsf{StatAbnormal(Self)}$ is another abbreviation, to be described shortly. The first $+$-component $1 = \{*\}$ in $\mathsf{StatResult(Self)}$ corresponds to non-termination (hanging); in this case no next state is produced. The second component $\mathsf{Self}$ corresponds to normal termination; in this case a successor state in $\mathsf{Self}$ is produced. The third component $\mathsf{StatAbnormal(Self)}$ corresponds to abrupt termination; it contains the four possible forms of abrupt termination for statements in Java:

$$\begin{aligned}&\mathsf{StatAbnormal(Self)}\\ &\quad= \;(\mathsf{Self} \times \mathsf{Exception}) + \mathsf{Self} + (\mathsf{Self} \times (1 + \mathsf{String})) + (\mathsf{Self} \times (1 + \mathsf{String}))\end{aligned}$$

The first of the (four) $+$-components in $\mathsf{StatAbnormal(Self)}$ corresponds to an exception abnormality; in that case the state in which the exception occurred is returned,

[5]Even constructors are coalgebras in this approach, since they modify the global state space, in allocating storage for the newly created object.

together with (a reference to) the relevant exception object. The second component captures a return abnormality; then, only the state is returned, appropriately tagged. The third and fourth option deal with break and continue abnormalities. In Java both the break and the continue statement may occur with or without label, given as string. Hence, a break or continue abnormality consists of a state, possibly with a string. Thus, all possible termination options of statements can be captured in an appropriate type, which is used as codomain type of statements, like in the general pattern (1).

Expressions in Java, say of return type Out, are handled similarly. They are captured as coalgebras of the form

$$\mathsf{Self} \xrightarrow{\mathsf{expression}} \mathsf{ExprResult}(\mathsf{Self}, \mathsf{Out})$$

where ExprResult(Self, Out) is an abbreviation:

$$\mathsf{ExprResult}(\mathsf{Self}, \mathsf{Out}) \quad = \quad 1 + (\mathsf{Self} \times \mathsf{Out}) + \mathsf{ExprAbnormal}(\mathsf{Self})$$

with ExprAbnormal(Self) incorporating abrupt termination for expressions[6]. In the normal termination case—the second component of ExprResult(Self, Out)—both a state in Self and a value in Out are returned. The state that is produced captures the possible side-effect of the expression.

The interface of a class in Java can be described as a type, in a similar manner. We only give an example, involving a simple class:

```
class A {
  int i;
  void m1(boolean b) { ... }
  float m2(float x, byte a) { ... }
}
```

For such a class we construct an "interface type", containing[7] the interface types of the members of this class in a Cartesian product.

$$\begin{array}{l} \mathsf{AIFace}(\mathsf{Self}) \\ \quad = \quad \mathsf{int} \times (\mathsf{Self}^{\mathsf{int}}) \times (\mathsf{StatResult}(\mathsf{Self})^{\mathsf{boolean}}) \times (\mathsf{ExprResult}(\mathsf{Self}, \mathsf{float})^{(\mathsf{float} \times \mathsf{byte})}) \end{array}$$

(The second $\times$-component $\mathsf{Self}^{\mathsf{int}}$ corresponds to a generated assignment operation for the instance variable `i`. It takes an integer, say $a$, and yields a new state in which the value of `i` is set to $a$, and everything else is unchanged.)

A coalgebra of the form

$$\mathsf{Self} \longrightarrow \mathsf{AIFace}(\mathsf{Self}) \tag{7}$$

then incorporates all the operations of the Java class. A further step is to link the method bodies—occurring in the above class A only as "..."—into this approach. This is done by using suitable predicates on coalgebras like (7), telling that the methods correspond to the translated method bodies (which are not discussed here). These predicates are much like the assertions used in the coalgebraic specifications in the previous section, since they take the form: "method call = method body".

Once this translation is completed, it can be used to reason with standard type theoretic and logical means about the original Java class, possibly making use of proof tools. Typical properties to be proved are normal / abnormal termination

[6]which can only happen because of an exception.
[7]For simplicity we omit the default constructor.

resulting in certain outcomes, for individual methods, and invariants, for classes as a whole. Given the present space restrictions, details have to be skipped. What we would like to emphasise is the natural coalgebraic representation of classes, via a precise interface type as codomain, leading to tailor-made definitions of bisimilation and invariance, for particular (Java) classes. In alternative approaches based on automata or transition systems, the structure of the interface to the outside world is usually lost in an unstructured set of actions.

## 4 Conclusions and pointers

We have sketched some ideas behind the use of coalgebras in specification and verification for state-based systems, described in object-oriented style. Emphasis was put on the rôle of coalgebraic operations, having the state space as domain, and a structured type as codomain. Several exemplary systems were described using such operations.

We conclude with a brief survey of some research lines in the new field of coalgebra, together with selected pointers to the rapidly growing literature. The paper [JR97] forms an introduction to the field of coalgebras. Part of the ongoing research may be found in the proceedings of the first two workshops on coalgebras [JMRR98, JR99]—selected papers from the first of which will appear in a special issue of *Theoretical Computer Science.*

### Structure on coalgebras

The abstract study of products, coproducts, homomorphisms *etc.* for coalgebras can be done as "universal coalgebra" [Rut99] or also as a categorical study [JPT$^+$98]. There is a famous result in universal algebra, due to Birkhoff [Bir35], stating that a class of algebras is equationally definable if and only if it is closed under subalgebras, quotients and products. A dual result for coalgebras is investigated in [GS98, Roş98].

### Coalgebras and data type theory

Early work on using coalgebras in data type theory to capture infinite data structures occurs in [AM82, Hag87, Hag89]. The experimental programming language Charity only contains algebras and coalgebras [CF92]. Programs can be constructed via various laws for initial algebras and final coalgebras, as studied *e.g.* in [MFP91, Par98]. As an example, the theory of the data type of both finite and infinite sequences is developed on a coalgebraic basis in [HJ99a]. Sometimes, initial algebra and final coalgebra solutions of data type equations coincide, giving rise to special theories [Fre91, Fre92].

### Reasoning principles for coalgebras

For initial algebras there is a standard reasoning principle called induction. The analogous, but dual, principle for final coalgebras is called coinduction. It can also be formulated in terms of bisimulations [Fio96, TR98]. Coinduction is useful for reasoning about (infinite) behaviours of systems, by reducing global arguments over infinite structures to single-step arguments—like induction does. The coinduction principle is currently formalised in the proof tools Coq [BBC$^+$97] and Isabelle [Pau97], and can be encoded in second order type theory [Wra89, RP93].

### Coalgebraic specification

The area of coalgebraic specification of classes (in object-oriented languages) was introduced in [Rei95], and continued in [Jac96c, Jac95, Jac96b, Jac97]. In [HHJT98] a formal language CCSL for coalgebraic class specifications is described, together with a translation tool for generating logical theories for the proof tool PVS [ORSvH95]. Connections with (specifications in) hidden algebra (see [GM96]) are investigated in [Mal96, Cîr98].

### Coalgebraic program verification

Coalgebras can provide a basis for Java program verification, in the style described in Section 3. An elaboration of such a semantics, together with an associated Hoare logic, is described in [HJ99b]. A tool that translates Java classes to logical theories (in PVS [ORSvH95]) may be found in [JvdBH+98]. A major case study is presented in [HBJ99]. In this context it is worth pointing out that the recent textbook [Rey98] on program semantics systematically describes denotations $[\![ c ]\!]_{\mathrm{comm}}$ of programs $c$ as coalgebras, acting on a state space of mappings from variables to values—without, by the way, ever calling these denotations coalgebras.

### Coalgebras and continuous mathematics

Infinite sequences play an important rôle in analysis. Describing these coalgebraically, leads to a new, uniform framework in which one can describe analytic functions and various associated results [PE98], including solutions to differential equations [Pav98]. Such solutions give rise to coalgebras in hybrid systems [Jac96a], combining discrete and continuous dynamics. Also, probabilistic bisimulation for Markov processes may be studied from a coalgebraic perspective [VR98, DEP98].

### Coalgebras and modal logic

Modal logic [Eme90, Gol92] may be seen as the logic of dynamics. Therefore, it combines very well with coalgebras, as witnessed by several recent studies [Mos99, Rößl99, Kur98].

### Coalgebras and non well-founded set theory

In the original monograph [Acz88] on non well-founded sets, coalgebras are already prominently present, since they are fundamental to the idea of a non well-founded set. Further investigations occur in [BM96, TR98].

### Coalgebras and automata / transition systems

Automata and transition systems form well-established structures for describing dynamical systems. They form instancs of coalgebras. As such, they are studied in [Rut98, TR98, JPT+98].

## Acknowledgements

Thanks to Joachim van den Berg, Marieke Huisman, Erik Poll, Jan Rutten and Hendrik Tews for commenting on the first draft.

## References

[Acz88] P. Aczel. *Non-well-founded sets.* CSLI Lecture Notes 14, Stanford, 1988.

[AM82] M.A. Arbib and E.G. Manes. Parametrized data types do not need highly constrained parameters. *Inf. & Contr.*, 52:139–158, 1982.

[BBC+97] B. Barras, S. Boutin, C. Cornes, J. Courant, J.-Chr. Filliâtre, E. Giménez, H. Herbelin, G. Huet, C. Muñoz, C. Murthy, C. Parent, C. Paulin-Mohring, A. Saïbi, and B. Werner. The Coq Proof Assistant User's Guide Version 6.1. Technical Report 203, INRIA Rocquencourt, France, May 1997.

[Bir35] G. Birkhoff. On the structure of abstract algebras. *Proc. Cambridge Phil. Soc.*, 31:433–454, 1935.

[BM96] J. Barwise and L.S. Moss. *Vicious Circles: On the Mathematics of Non-wellfounded Phenomena.* CSLI Lecture Notes 60, Stanford, 1996.

[CF92] J.R.B. Cockett and T. Fukushima. About Charity. Technical Report 92/480/18, Dep. Comp. Sci., Univ. Calgary, 1992.

[Cîr98] C. Cîrstea. Coalgebra semantics for hidden algebra: parametrised objects and inheritance. In F. Parisi Presicce, editor, *Recent Trends in Data Type Specification*, number 1376 in Lect. Notes Comp. Sci., pages 174–189. Springer, Berlin, 1998.

[DEP98] J. Desharnais, A. Edalat, and P. Panangaden. A logical characterization of bisimulation for labeled markov processes. In *Logic in Computer Science*, 1998.

[Eme90] E.A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 997–998. Elsevier/MIT Press, 1990.

[Fio96] M.P. Fiore. A coinduction principle for recursive data types based on bisimulation. *Inf. & Comp.*, 127(2):186–198, 1996.

[Fre91] P.J. Freyd. Algebraically complete categories. In A. Carboni, M.C. Pedicchio, and G. Rosolini, editors, *Como Conference on Category Theory*, number 1488 in Lect. Notes Math., pages 95–104. Springer, Berlin, 1991.

[Fre92] P.J. Freyd. Remarks on algebraically compact categories. In M.P. Fourman, P.T. Johnstone, and A.M. Pitts, editors, *Applications of Categories in Computer Science*, number 177 in LMS, pages 95–106. Cambridge Univ. Press, 1992.

[GM96] J.A. Goguen and G. Malcolm. An extended abstract of a hidden agenda. In J. Meystel, A. Meystel, and R. Quintero, editors, *Proceedings of the Conference on Intelligent Systems: A Semiotic Perspective*, pages 159–167. Nat. Inst. Stand. & Techn., 1996.

[Gol92] R. Goldblatt. *Logics of Time and Computation.* CSLI Lecture Notes 7, Stanford, 2$^{nd}$ rev. edition, 1992.

[GS98] H.P. Gumm and T. Schröder. Covarieties and complete covarieties. In B. Jacobs, L. Moss, H. Reichel, and J. Rutten, editors, *Coalgebraic Methods in Computer Science*, number 11 in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 1998.

[Hag87] T. Hagino. A typed lambda calculus with categorical type constructors. In D.H. Pitt, A Poigné, and D.E. Rydeheard, editors, *Category and Computer Science*, number 283 in Lect. Notes Comp. Sci., pages 140–157. Springer, Berlin, 1987.

[Hag89] T. Hagino. Codatatypes in ML. *Journ. Symb. Computation*, 8:629–650, 1989.

[HBJ99] M. Huisman, J. van den Berg, and B. Jacobs. A case study in class library verification: Java's Vector class. Manuscript, 1999.

[HHJT98] U. Hensel, M. Huisman, B. Jacobs, and H. Tews. Reasoning about classes in object-oriented languages: Logical models and tools. In Ch. Hankin, editor, *European Symposium on Programming*, number 1381 in Lect. Notes Comp. Sci., pages 105–121. Springer, Berlin, 1998.

[HJ99a] U. Hensel and B. Jacobs. Coalgebraic theories of sequences in PVS. *Journ. of Logic and Computation*, 1999. To appear.

[HJ99b] M. Huisman and B. Jacobs. Java program verification via a Hoare logic with abrupt termination. Manuscript, 1999.

[Jac95] B. Jacobs. Mongruences and cofree coalgebras. In V.S. Alagar and M. Nivat, editors, *Algebraic Methodology and Software Technology*, number 936 in Lect. Notes Comp. Sci., pages 245–260. Springer, Berlin, 1995.

[Jac96a] B. Jacobs. Coalgebraic specifications and models of deterministic hybrid systems. In M. Wirsing and M. Nivat, editors, *Algebraic Methodology and Software Technology*, number 1101 in Lect. Notes Comp. Sci., pages 520–535. Springer, Berlin, 1996. The full version is to appear in *Theor. Comp. Sci.*

[Jac96b] B. Jacobs. Inheritance and cofree constructions. In P. Cointe, editor, *European Conference on Object-Oriented Programming*, number 1098 in Lect. Notes Comp. Sci., pages 210–231. Springer, Berlin, 1996.

[Jac96c] B. Jacobs. Objects and classes, co-algebraically. In B. Freitag, C.B. Jones, C. Lengauer, and H.-J. Schek, editors, *Object-Orientation with Parallelism and Persistence*, pages 83–103. Kluwer Acad. Publ., 1996.

[Jac97] B. Jacobs. Invariants, bisimulations and the correctness of coalgebraic refinements. In M. Johnson, editor, *Algebraic Methodology and Software Technology*, number 1349 in Lect. Notes Comp. Sci., pages 276–291. Springer, Berlin, 1997.

[JMRR98] B. Jacobs, L. Moss, H. Reichel, and J. Rutten, editors. *Coalgebraic Methods in Computer Science (CMCS'98)*, number 11 in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 1998. Available from URL: `http://www.elsevier.nl/locate/entcs/volume11.html`.

[JPT+98] P.T. Johnstone, A.J. Power, T. Tsujishita, H. Watanabe, and J. Worrell. An axiomatics for categories of transition systems as coalgebras. In *Logic in Computer Science*. IEEE, Computer Science Press, 1998.

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

[JR99] B. Jacobs and J. Rutten, editors. *Coalgebraic Methods in Computer Science (CMCS'99)*, number 19 in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 1999. Available (soon) from URL: `http://www.elsevier.nl/locate/entcs/volume19.html`.

[JvdBH+98] B. Jacobs, J. van den Berg, M. Huisman, M. van Berkum, U. Hensel, and H. Tews. Reasoning about classes in Java (preliminary report). In *Object-Oriented Programming, Systems, Languages and Applications*, pages 329–340. ACM Press, 1998.

[Kur98] A. Kurz. Specifying coalgebras with modal logic. In B. Jacobs, L. Moss, H. Reichel, and J. Rutten, editors, *Coalgebraic Methods in Computer Science*, number 11 in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 1998.

[Mal96] G. Malcolm. Behavioural equivalence, bisimulation and minimal realisation. In M. Haveraaen, O. Owe, and O.J. Dahl, editors, *Recent Trends in Data Type Specification*, number 1130 in Lect. Notes Comp. Sci., pages 359–378. Springer, Berlin, 1996.

[MFP91] E. Meijer, M. Fokkinga, and R. Paterson. Functional programming with bananas, lenses, envelopes and barbed wire. In J. Hughes, editor, *Functional Programming Languages and Computer Architecture*, number 523 in Lect. Notes Comp. Sci., pages 215–240. Springer, Berlin, 1991.

[Mos99] L.S. Moss. Coalgebraic logic. *Ann. Pure & Appl. Logic*, 1999. To appear.

[ORSvH95] S. Owre, J.M. Rushby, N. Shankar, and F. von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Trans. on Softw. Eng.*, 21(2):107–125, 1995.

[Par98] A. Pardo. Monadic corecursion—definition, fusion laws, and applications—. In B. Jacobs, L. Moss, H. Reichel, and J. Rutten, editors, *Coalgebraic Methods in Computer Science*, number 11 in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 1998.

[Pau97] L.C. Paulson. Mechanizing coinduction and corecursion in higher-order logic. *Journ. of Logic and Computation*, 7:175–204, 1997.

[Pav98] D. Pavlović. Guarded induction on final coalgebras. In B. Jacobs, L. Moss, H. Reichel, and J. Rutten, editors, *Coalgebraic Methods in Computer Science*, number 11 in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 1998.

[PE98] D. Pavlović and M. Escardó. Calculus in coinductive form. In *Logic in Computer Science*, pages 408–417. IEEE, Computer Science Press, 1998.

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

[Rey98] J.C. Reynolds. *Theories of Programming Languages*. Cambridge Univ. Press, 1998.

[Roş98] G. Roşu. A Birkhoff-like axiomatizability result for hidden algebra and coalgebra. In B. Jacobs, L. Moss, H. Reichel, and J. Rutten, editors, *Coalgebraic Methods in Computer Science*, number 11 in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 1998.

[Röß99] M. Rößiger. From modal logic to terminal coalgebras. Manuscript, 1999.

[RP93] J.C. Reynolds and G.D. Plotkin. On functors expressible in the polymorphic typed lambda calculus. *Inf. & Comp.*, 105:1–29, 1993.

[Rut98] J. Rutten. Automata and coinduction (an exercise in coalgebra). In D. Sangiorigi and R. de Simone, editors, *Concur'98: Concurrency Theory*, number 1466 in Lect. Notes Comp. Sci., pages 194–218. Springer, Berlin, 1998.

[Rut99] J. Rutten. Universal coalgebra: a theory of systems. *Theor. Comp. Sci.*, 1999. To appear.

[TR98] D. Turi and J. Rutten. On the foundations of final semantics: non-standard sets, metric spaces and partial orders. *Math. Struct. in Comp. Sci.*, 8(5):481–540, 1998.

[VR98] E.P. de Vink and J.J.M.M. Rutten. Bisimulation for probabilistic transition systems: a coalgebraic approach (extended abstract). In

P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *International Colloquium on Automata, Languages and Programming*, number 1256 in Lect. Notes Comp. Sci., pages 460–470. Springer, Berlin, 1998.

[Wra89] G.C. Wraith. A note on categorical datatypes. In D.H. Pitt, A. Poigné, and D.E. Rydeheard, editors, *Category Theory and Computer Science*, number 389 in Lect. Notes Comp. Sci., pages 118–127. Springer, Berlin, 1989.

## 6.3 Intelligent Agents and their Behaviour, door: Catholijn M. Jonker en Jan Treur

# Intelligent Agents and their Behaviour

Catholijn M. Jonker[1] and Jan Treur[1,2]

[1]Vrije Universiteit Amsterdam, Department of Artificial Intelligence
De Boelelaan 1081a, 1081 HV Amsterdam
URL: http://www.cs.vu.nl/~{jonker,treur}, Email: {jonker,treur}@cs.vu.nl, Tel: 020-44477{43,63}

[2]Centrum voor Wiskunde en Informatica, Kruislaan 413, 1098 SJ Amsterdam

**Abstract** In this paper an introduction is given to agent notions and a short survey of literature. Behavioural characteristics of agents and their interaction play an important role. This emphasis on behaviour and interaction imposes specific requirements on the techniques used during the development of an agent or multi-agent system.

## 1 Introduction

The nature of many processes in today's world is distributed, as is the information used and produced by those processes. Furthermore, nowadays applications are more and more dynamic of nature. Different systems (either human or automated) are responsible for different parts of a process: the combination of the processes defines its effect. Users expect dedicated assistance from the applications they use: in a co-operative manner the applications have to intelligently anticipate, adapt, and actively seek ways to support users. The common denominator for such applications is software agent technology. In response to these requirements, technological developments contributed by researchers from different fields have made it possible to support such processes, their co-ordination, and the co-ordination between such processes and their human users.

## 2 Definitions of Agent Notions

The term agent has become popular, and has been used for a wide variety of applications, ranging from simple batch jobs (termed agent because they can be scheduled in advance to perform tasks on a remote machine) and simple email filters, to mobile applications (termed agent because they can move themselves from computer to computer), to intelligent assistants (because they present themselves to human users as believable characters that manifest intentionality and other aspects of a mental state normally attributed only to humans), to large, open, complex, mission critical systems (such as systems for air traffic control). Agents' abilities vary significantly, depending on their roles, capabilities, and their environment. To describe these abilities different notions of agenthood have been introduced. A good overview of the discussions on these themes can be found in (Franklin and Graesser, 1996). In (Wooldridge and Jennings, 1995) the *weak notion of agent* was introduced that is often used as a reference (see also (Jennings and Wooldridge, 1998a)).

## 2.1 Weak Notion of Agent

The weak notion of agent is a notion that requires the behaviour of agents to exhibit the following four types of behaviour:

- Autonomous behaviour
- Responsive behaviour (also called reactive behaviour)
- Pro-active behaviour
- Social behaviour

*Autonomy* relates to control: although an agent may interact with its environment, the processes performed by an agent are in full control of the agent itself. Jennings and Wooldridge (1998a) define *autonomous* behaviour as:

> ... the system should be able to act without the direct intervention of humans (or other agents) and should have control over its own actions and internal state.

This means that an agent can only be requested to perform some action, and, as Jennings and Wooldridge (1998a) state:

> The decision about whether to act upon the request lies with the recipient.

Examples of autonomous processes are: any process control system (like thermostats, missile guiding systems, and nuclear reactor control systems), software deamons (e.g., one that monitors a user's incoming email and obtains their attention by displaying an icon when new, incoming email is detected), operating systems.

A lot of processes that exhibit autonomous behaviour are being termed agents. However, if such agents do not exhibit flexible behaviour, they are not considered intelligent agents. An intelligent agent is defined in (Jennings and Wooldridge, 1998a) to be a computer system that is capable of flexible autonomous actions in order to meet its design objectives. Intelligence requires flexibility with respect to autonomous actions, meaning that intelligent agents also exhibit responsive, social, and pro-active behaviour.

An agent exhibits *responsive* (or *reactive*) behaviour if it reacts or responds to new information from its environment. Jennings and Wooldridge define responsive behaviour as follows:

> Agents should perceive their environment (which may be the physical world, a user, a collection of agents, the Internet, etc.) and respond in a timely fashion to changes that occur in it.

A barometer is a simple example of a system that exhibits responsive behaviour: It continually gets new information about the current air pressure and responds to this new information by adjusting its dial.
*Pro-active* behaviour is defined by (Jennings and Wooldridge, 1998a) as follows:

> Agents should not simply act in response to their environment, they should be able to exhibit opportunistic, goal-directed behaviour and take the initiative where appropriate.

Pro-active behaviour is the most difficult of the required types of behaviour for an agent defined according to the weak agent notion. For example, pro-active behaviour can occur

simultaneously with responsive behaviour. It is possible to respond to incoming new information in an opportunistic manner according to some goals. Also initiatives can be taken in response to incoming new information from the environment, and thus this behaviour resembles responsive behaviour. However, it is also possible to behave pro-actively when no new information is coming in from the environment. This last type of behaviour can by now means be called responsive behaviour. A more elaborate comparison between responsive behaviour and pro-active behaviour can found in (Jonker and Treur, 1998a).
An agent exhibits *social* behaviour if it communicates and co-operates with other agents. Jennings and Wooldridge define social behaviour as follows:

> Agents should be able to interact, when they deem appropriate, with other artificial agents and humans in order to complete their own problem solving and to help others with their activities.

An example of an agent that exhibits social behaviour is a car. It communicates with its human user by way of its dials (outgoing communication dials: speed, amount of fuel, temperature) and its control mechanisms like the pedals, the steer, and the gears (incoming communication). It co-operates with its human user, e.g., by going in the direction indicated by the user, with the speed set by that user.

## 2.2 Other Notions of Agenthood

Agents can also be required to have additional characteristics. In this section three of these characteristics are discussed: adaptivity, pro-creativity, and intentionality.

*Adaptivity* is a characteristic that is vital in some systems. An adaptive agent learns and improves with experience. This behaviour is vital in environments that change over time in ways that would make a non-adaptive agent obsolete or give it no chance of survival. This characteristic is modelled often in simulations of societies of small agents, but also, for example, in adaptive user interface agents.

*Pro-creativity* is of similar importance to find agents that satisfy certain conditions. The chance of survival is often measured in terms of a fitness function. This characteristic is modelled often in simulations of societies of small agents (see the literature in the area of Artificial Life). A computer virus is a very infamous form of a pro-creative agent.

According to Dennett (1987) an *intentional system* is an entity

> ... whose behaviour can be predicted by the method of attributing beliefs, designs and rational acumen.

Mentalistic and intentional notions such as *beliefs*, *desires*, *intentions*, *commitments*, *goals*, *plans*, *preference*, *choice*, *awareness*, may be assigned to agents. The *stronger notion of agenthood* in which agents are described in terms of this type of notions provides additional metaphorical support for the design of agents.

## 2.3 Relation to other disciplines

One of the most important aspects of agents is their behaviour. In the past, behaviour has been studied in different disciplines. In Cognitive Psychology the analysis of human behaviour is a major topic. In Biology, animal behaviour has been and is being studied extensively. Around

1900 a discussion took place about the manner in which observed animal behaviour can be interpreted in order to obtain an objective and testable description; for an overview, see (Allen and Bekoff, 1997; Vauclair, 1996). A risk of taking the intentional stance (Denett, 1987) as a perspective to explain behaviour, is that explanations are generated that make use of (a large number of) mental concepts that cannot be tested empirically. Therefore the *principle of parsimony* was introduced, stating that 'in no case may we interpret an action as the outcome of the exercise of a higher psychical faculty, if it can be interpreted as the outcome of the exercise of one which stands lower in the psychological scale'; see (Morgan, 1894).

Building further on this perspective *behaviourism* was developed, e.g., (Gibson, 1960; Skinner, 1935; Watson, 1919). In this approach animal behaviour is explained only in terms of a black box that for each pattern of *stimuli* (input of the black box) from the environment generates a *response* (output of the black box), that functionally depends on the input pattern of stimuli; i.e., if two patterns of stimuli are offered, then the same behaviour occurs if the two patterns of stimuli are equal. This view was also extended to human behaviour. Because of the underlying black box view, behaviourism discouraged reference to internal (mental) concepts, states and activities of organisms: any speculation about the internal functioning of the black box (i.e., the processes that might mediate between sensory inputs and behavioural outputs) was forbidden; cf. (Vauclair, 1996), p. 4. For example, to avoid a reference to assumed internal constructs such as memory, in literature on animal behaviour the term *delayed response behaviour* is used to indicate observable behaviour the animal shows in response to a stimulus which (at least partly) lies in the history (and not in the current instance of time); e.g., (Tinklepaugh, 1932; Vauclair, 1996).

## 3 Primitive Agent Concepts

The notions of agenthood as discussed in Section 2 are highly abstract notions. In order to analyse or design agents that respect one or more of the above agent notions, these primitive concepts serve as as vocabulary. Two classes of primitive notions are distinguished: those used to describe the behaviour of agents in terms of their external ("observable" or public) states and interactions (Section 3.1), and those used to describe the behaviour of agents in terms of their internal ("non-observable" or private) states, and processes (Section 3.2). In Section 3.3, as an illustration an example agent is discussed in terms of these concepts: an elevator agent.

### 3.1 External primitive concepts

Two types of interaction of an agent with its environment are distinguished, depending on whether the interaction takes place with an agent or with something else (called an *external world*), for example a database, or the material world. For each of these two types of interaction a specific terminology is used.

***A. Interaction with the external world***

Two primitive types of interaction with the external world are distinguished. The first type of interaction, observation, does change the information the agent has about the world, but does not change the world state itself, whereas the second type, performing an action, does change the world state, but does not change the information the agent has about the world. Combinations of these primitive types of interaction are possible; for example, performing an action, and observing its results.

- *Observation*
  In which ways is the agent capable of observing or sensing its environment? Two types of observation can be distinguished, passive and active. *Passive*: the agent passively receives the results of observations without taking any initiative or control to observe. *Active*: the agent actively initiates and controls which observations it wants to perform; this enables the agent to focus its observations and limit the amount of information acquired.
- *Execution of actions in the external world*
  The agent is capable of making changes to the state of its environment by initiating and executing actions.

***B. Communication with other agents***

Communication changes the information state of a destination agent, but not the world. If the communication process itself is incorporated in the agent's state, also the information state of the source agent can be changed. From the perspective of one agent, two directions of communication are distinguished, which can occur in combination.

- *Outgoing communication*
  Is the agent capable of communicating to another agent?
- *Incoming communication*
  Is the agent capable of receiving communication from another agent?

## 3.2 Internal primitive concepts

A number of internal primitive agent concepts often occurs in the literature on agents. These concepts refer to the agent's own internal state. Note that these concepts *can* be used to analyse or design an agent with a specific type of externally observable behaviour, but no logical implication exists that these are the only concepts that can be used to explain or realise such behaviour.

***A. World Model***

Does the agent create and maintain a model of the external world based on its observations of this world, on information about the world communicated by other agents, and its own knowledge about the world?

***B. Agent Models***

Does the agent create and maintain models of other agents in its environment based on its observations of these agents as they behave in the external world, on information about these agents communicated by (other) agents, and knowledge about agents?

***C. Self Model***

Does the agent create and maintain a model of its own characteristics, internal state, and behaviour?

***D. History***
Does the agent create and maintain a history of the world model, or agent models, or self model, or own and group processes?
***E. Goals***
Does the agent represent, generate, and use explicit goals in its processing?
***F. Plans***
Is the agent capable of representing, generating, and using its own plans of action?

***G. Group Concepts***
Which concepts referring to a group of agents does the agent have that allow it to co-operate with other agents? *Joint Goals*: is the agent capable of formulating or accepting and using goals for a group of agents, i.e., goals that can only be achieved by working together? *Joint Plans*: is the agent capable of representing, generating, and using plans of action for joint goals, i.e., involving which actions are to be performed by which agents in order to achieve a certain joint goal? *Commitments*: is the agent able to commit to joint goals and plan? *Negotiation Protocol*: does the agent represent and use negotiation protocols? *Negotiation Strategies*: does the agent represent and use negotiation strategies?

## 3.3 Example analysis: an elevator

The agent concepts introduced in Sections 3.1 and 3.2 are illustrated by analysing an elevator from the agent perspective using these basic concepts.

### 3.3.1 External primitive concepts

The external agent concepts can be used to express external (observable) behaviour of the elevator, both in interaction with the external (physical) world and in interaction with other (human) agents.

***A. Interaction with the external world***
*Observation*
An elevator is capable of receiving observation results on

- the presence of objects between the doors (often both by a visual and a tactile sensor)
- the total weight of its contents
- possibly, its position in the building (at which floor)

These observations are passive: no decision is made by the agent as to when the observations are to be performed.
*Performing actions*
It performs actions in the world like moving itself (and people) from one position to another and opening and closing doors.

***B. Communication with other agents***
*Incoming*
It receives communication from users with information about where they want to go, or where they want to be picked up (by buttons that have been pushed).

*Outgoing*

It communicates to a user information on the floor (by lighting buttons) and information about overload (by sounding beeps).

### 3.3.2 Internal primitive concepts

The internal agent concepts can be used to analyse the internal behaviour of the elevator. Since this behaviour is non-observable from outside, and we do not know how an elevator is built, this analysis has a more speculative character than the analysis of the external behaviour.

***A. World Model***

Elevators need to know at which floor they are. They may maintain this knowledge themselves based on the actions (going two floors up, going one floor down) they perform. Another possibility is that the elevator immediately observes where it is. Furthermore, the elevator needs to know how much weight its physical self is capable of transporting.

***B. Agent Models***

The agent information of the user goals (where they want to go) may be maintained as well.

***C. Self Model***

The agent might have an explicit representation of when it needs maintenance.

***D. History***

It does not need to know what actions it performed previously in order to perform its task now. It might have an explicit representation of when it has last received maintenance.

***E. Goals***

Modern elevators seem to make use of the explicit goals (adopted from the goals communicated by the users).

***F. Plans***

The goals are used to determine which actions to perform. They may even make plans for reaching these goals: determine the order of actions, for example when one of the users has the goal to be at a highter floor and another on a lower floor.

***G. Group Concepts***

The elevator co-operates with its users. Sometimes the elevator can also co-operate with other elevators so that they could strategically distribute themselves over the floors, to do this they need joint goals, joint plans, commitments to each other, and negotiation knowledge.
*Joint goals*: The goals adopted from the goals communicated by the users are joint

### 3.3.3 Types of behaviour

The following types of elevator behaviour can be observed.

***Autonomy***

As soon as it is activated, no system or human is controlling its machinery, and (normally) it is not switched off and on by the user. The fact that it responds to the immediate stimuli of buttons being pushed is not the same as being controlled. The elevator has full control, e.g., of its motor, doors, and lights.

***Pro-activeness***

What are elevators doing when nobody is inside anymore and nobody has called for them?

The most simple elevators just stay where they are (most elevators take the initiative to close their doors then), but these days often elevators go to a strategic floor (e.g., the ground floor) if they finished bringing a person to his or her destination and do not have a new assignment.

***Reactiveness***

The elevator reacts to the immediate stimuli of buttons pushed; therefore, it shows reactive behaviour. Furthermore, elevators often show delayed-response behaviour in picking up people. People often have to wait for the elevator as the elevator picks up people on other floors, however, the elevator does not forget a signal and will, eventually, come to the requested floor.

***Social behaviour***

It communicates and co-operates with users and, sometimes, with other elevators (e.g., to distribute work load).

***Own adaptation and learning***

Simple elevators are not capable of adjusting their own behaviour to new situations, nor are they capable of learning. However, it is possible to conceive of more intelligent elevators that can learn the rush hours for the different floors.

## 4 Formal Techniques to Support the Design of Multi-Agent Systems

In this section some formal techniques used as a basis for agent development are discussed.

### 4.1 The behavioural perspective

The behaviour of the most simple systems is defined by a function that maps the input to the output. For example, a compiler is a such a system. Viewed from a Software Engineering perspective, modelling behaviour by a functional relation between input and output provides a system that can be described as a (mathematical) *function*

$$F : \text{Input\_states}\ '\ \text{Output\_states}$$

of the set of possible input states to the set of possible output states. Such a system is transparent and predictable. For the same input always the same behaviour is repeated: its behaviour does not depend on earlier processes; for example, no information on previous experiences is stored in memory so that it can be remembered and affect behaviour. Well-known traditional programming methods are based on this paradigm; for example, program specification and refinement based on preconditions and postconditions as developed in, e.g., (Dijkstra, 1976).

Reactive systems that are not a purely functional input-output dependency are among the most complex systems to design and implement (Pnueli, 1986). They maintain continuous interaction with their environment, and are expected to operate independently, typically over a long period of time. The behaviour of reactive systems is still functional, but at each point in time the function maps the current *and* (the history trace of) past input to the new output. For certain types of reactive systems, even the specialised software engineering techniques and tools for development fail, and new techniques are required.

The above motivates the use of agent technology for the development of the reactive systems that proved to be difficult to design. The notion of agenthood becomes even more important when systems need to be design that are more than reactive, i.e., systems whose behaviour is no longer a simple function of input to output at each point in time. The behaviour of such systems can only be described by a function from the current input and the current internal state to the new output and a new internal state. Previous processes may have led to internal storage of information in a the agent's internal state so that the same input pattern of stimuli can lead to different behaviour a next time it is encountered; the agent may be able to deliberate about it. Using agent terminology, the internal state might be based on experience, goals, beliefs, desires, and intentions. From a formal perspective the emphasis on behaviour suggests logics for dynamic phenomena such as temporal logics or dynamic logics.

## 4.2 Specification of design descriptions and specification of requirements

Given the characteristics discussed, to support the development of multi-agent systems, dedicated techniques and tools are needed. Formal modelling languages and logical foundations can play an important role as a basis for these techniques and tools. Techniques and tools need to address specification of *design descriptions* of agent systems, specification of *requirements*, and *formal analysis*. Each of these aspects imposes specific desiderata. A language to specify a (multi-agent) system architecture needs features different from a language to express (required) properties of such a system. The distinction between these specification languages follows the distinction made in the AI and Design community (Gero and Sudweeks, 1996, 1998) between the *structure* of a design object on the one hand, and *function* or *behaviour* on the other hand.

An example of a formal language (based on executable temporal logic, see (Barringer, Fisher, Gabbay, Owens, and Reynolds, 1996)) to specify designs of multi-agent systems is ConcurrentMetateM (Fisher, 1994). A formal language to specify compositional designs is used within the development method for multi-agent systems DESIRE; see (Brazier, Jonker and Treur, 1998) for the underlying principles, (Brazier, Dunin-Keplicz, Jennings and Treur, 1997) for an extensive case study, and (Brazier, Treur, Wijngaards and Willems, 1998) for temporal semantics based on three-valued states.

Languages to specify (required) behavioural properties of agents and multi-agent systems can be based on temporal logic. An example of a requirements engineering method based on a real-time temporal logic is ALBERT; see (Dubois, Du Bois, and Zeippen, 1995). In DESIRE requirements specification is based on temporal logic with three-valued states (Herlea, Jonker, Treur, and Wijngaards, 1999). Although having a formal basis for specification languages is useful, for example to be able to develop dedicated tools, specification in practice may involve informal, semi-formal and formal representations with both graphical and textual elements.

## 4.3 Verification

Design descriptions and requirements can be related in a formal manner: it is formally defined when a design description satisfies a requirements specification, and this formal relation is used to verify that the design description fulfills the requirements. For formal analysis, proof techniques can play an important role. For example, work on verification of reactive systems can be a useful starting point for verification of agent systems; e.g., (Manna and Pnueli, 1995; Roever, Langmaack and Pnueli, 1998). In the context of verification of agent systems, variants of temporal logic are often used to express behavioural properties, and to prove these properties; e.g., (Fisher and Wooldridge, 1997; Jonker and Treur, 1998a; Engelfriet, Jonker and Treur, 1999).

### 4.4 Formal techniques for specific aspects

For particular aspects of agent systems, formal techniques have been developed in isolation. A number of them are discussed. Formal techniques for *reasoning about actions and plans* have been studied for quite some time, for example (Sandewall, 1995). An influencial formal theory for *communication* is Speech Act Theory (Searle, 1969). Specific agent communication languages such as ACL (FIPA standard for agent communication) and KQML (Finin, Labrou, and Mayfield, 1997). Another specific aspect at the level of a multi-agent system as a whole is *negotiation.* Mathematical techniques from areas such as system theory, game theory, decision theory or economics are used and further developed, for example (Rosenschein and Zlotkin, 1994ab). For processes internal in an agent *epistemic reasoning* can be addressed by modal logics, e.g., (Fagin, Halpern, Moses, and Vardi, 1995; Meyer and Hoek, 1995), *defeasible reasoning processes* by techniques from nonmonotonic logic, e.g., (Engelfriet and Treur, 1995, 1996ab). Logics for *beliefs, desires and intentions* have been developed in (Rao and Georgeff, 1991; Rao, 1996), and temporal logic techniques for the *control of reasoning processes* has been addressed in (Treur, 1994). A problem with application of these techniques for different aspects of agents is their integration in one agent system. An example of a useful *integration technique* is temporalising a given logic, as addressed in (Finger and Gabbay, 1992).

### 4.5 Elements of a research agenda

Within the European Network of Excellence on Agent Systems (AgentLink, 1998) a special interest group on Methodologies and Software Engineering of agent Systems is organised by the Department of Artificial Intelligence of the Vrije Universiteit Amsterdam. On the first meeting a number of issues to be addressed were identified. For a full report, see (Treur, 1998). One of the issues put forward is the ungroundedness of many of the logical approaches in the literature: often it is not clear whether and how the formal concepts introduced can be related to actual running agent systems.

## 5 Domains of Application of Software Agents

The international conferences and exhibitions on the practical application of intelligent agents and multi-agent technology (e.g., PAAM'97 and PAAM'98) are an interesting medium to get a comprehensive overview of the domains of application of multi-agent systems. Of course, also most other agent workshops and conferences pay attention to applications of multi-agent systems. The following application areas are distinguished in (Jennings and Wooldridge, 1998):

*Industrial applications*

Process Control, Manufacturing, Design, Air Traffic Control.

*Commercial applications*

Information Management, Electronic Commerce, Business Process Management

*Medical applications*

Patient Monitoring, Health Care.

*Entertainment applications*

Games, Interactive theater and cinema.

*Sociological and Biological applications*

Artificial Life, Small societies, Simulation of evolution theories.

The compositional development method DESIRE has been and is being used for the development of multi-agent systems for a variety of application domains.

### 5.1 Agents in Electronic Commerce

For electronic commerce an application has been made that consists of *agents negotiating* on the price of electricity in order to balance the overall load of electricity use (Brazier, Cornelissen, Gustavsson, Jonker, Lindeberg, Polak, en Treur, 1998a). The verification of this multi-agent system is presented in (Brazier, Cornelissen, Gustavsson, Jonker, Lindeberg, Polak, en Treur, 1998b). Furthermore a generic architecture has been developed ontwikkeld for a *broker* agent (Jonker en Treur, 1998c) that connects the demands of users to the products of providers and then matches demands and offers. The matching can be driven by a demand made by a user, but can also be iniated by a provider pushing his produce using a 1-1 marketing strategy. In cooperation with the company CRISP this architecture has been applied in a project for virtual markets to create a prototype application for the brokering of automobiles (Albers, Jonker, Karami and Treur, 1999).

### 5.2 Intelligent Web-sites

Another interesting application area that can benefit from agent technology is the development of an intelligent web-site: a web-site that is controlled and maintained by agents that communicate with each other and with visiting users or agents. In conventional web-sites a visitor navigates through the statically structured web-site. In an intelligent web-site the visitor is welcomed by the maintaining *(host/hostess)* agents that provide the visitor as effectively as possible with the desired information (Jonker and Treur, 1999). These hosts can be effective because they make an effective use of accumulated knowledge about (the profile) of the visitor. The profile (which contains information about interests and preferences) is constructed by interaction with the visitor. In cooperation with Ordina Utopics an intelligent Web-site has been developed in insurance.

### 5.3 Project Co-ordination and Agenda Maintenance

In cooperation with Rabobank and Telemachos Planning a prototype multi-agent system has been developed that supports the effective exploitation of a call center (Brazier, Jonker, Jüngen en Treur, 1999). With its call center Rabobank is able to offer the client the possibility to contact the bank 24 hours a day. Without the multi-agent system the call center can immediately accomodate the simple requests of the client. However, for a more complex request of the client it should be possible to propose an appointment to the client. The multi-agent system enables the Rabobank to make the appointment real-time. For each local bank the multi-agent system has a Work Manager Agent that deals with the problem of *work flow management* (the procedure that needs to be executed in preparation of an appointment with a client). The Work Manager Agent communicates with the Personal Assistants of the employees of the local bank (one for each employee) that maintain the electronic agenda of the employees. A generic model for the dynamic creation and monitoring of cooperation projects is used to ensure *project coordination* (Brazier, Jonker en Treur, 1996).

### 5.4 Clarification Agents

In cooperation with DSM, a *clarification agent* has been designed for diagnostic processes. At any point in time during the diagnostic process the clarification agent is able to provide explanations regarding the behaviour of the system, regarding the questions posed by the system to the user, and regarding the conclusions drawn by the system (Brazier, Jonker, Treur, Wijngaards, 1999). Initiated by an interaction between the user and the clarification agent, the diagnostic process can go back a number of steps.

### 5.5 Adaptive and Evolutionary Agent Systems

A few interesting experiments have been made to design systems of intelligent agents that are capable adapting themselves (and / or others) on demand and even of creating new or killing off agents. The requirements posed on the adaptations of those agents were engineered from abstract, high level conceptual behavioural requirements. During the engineering process the primitive agent concepts were a key factor in creating transparent designs for the adaptation of the multi-agent system. Note that the agents capable of redesigning themselves and the multi-agent system in which they "live" maintain and create internal representations of those designs (Brazier, Jonker, Treur, and Wijngaards, 1998).

### 5.6 Applications of Agents in the Domain of Biology and Social Sciences

Next to the above mentioned application domains, also biology and the social sciences are interesting application domains for agent technology. For example, the *simulation* of reactive, pro-active and social *animal behaviour* in which animals are modelled as agents (Jonker en Treur, 1998c), the *simulation of societies* consisting of a great number of agents (Brazier, Eck en Treur, 1997) and the *simulation of a bactery* as an agent in which the emphasis of the model lies on the different levels of regulation of the biochemical processes within a cell (in cooperation with the department of Microbiology of the Vrije Universiteit Amsterdam).

## 6 Conclusions

In this article a short introduction is given to the characteristics of intelligent agents and concepts to express these characteristics. The introduction focusses mainly on the behavioural characteristics of agents and their interaction. It is discussed that the traditional functional input-output specification techniques are not adequate to model the more complex systems that have to show a number of agent characteristics. In recent years, therefore, formal languages have been and currently are being developed with which multi-agent systems can be specified and with which behavioural properties of those systems can be described. In particular, during development (e.g., requirements specification, specification of designs and verification), and agent-specific aspects like negotiation tools and techniques have been created.

A wide range of application domains illustrates the generic applicability of agent technology and, in particular, the compositional development method DESIRE developed at Department of Artificial Intelligence of the Vrije Universiteit Amsterdam that can be used to design agent applications. The characteristics of behaviour of agents play an important role in the design of these applications. Applying requirements engineering to analyse the domain of application and to formulate properties has proved its value in the design of multi-agent systems.

Annually, the Department of Artificial Intelligence of the Vrije Universiteit Amsterdam organises a course on the design of multi-agent systems. Further information on this course can be obtained through the web-site of the course (Brazier, Jonker and Treur, 1999).

## References

AgentLink (1998). European Network of Excellence on Agent Systems. URL: http://www.AgentLink.org.

Albers, M., Jonker, C.M., Karami, M., and Treur, J., (1999). An Agent-based Architecture for an Electronic Market Place. In: H.S. Nwana and D.T. Ndumu (eds.), *Proceedings of the Fourth International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, PAAM'99. The Practical Application Company Ltd. To appear, 1999.

Allen, C., and Bekoff, M., (1997). *Species of Mind: the philosophy and biology of cognitive ethology*. MIT Press.

Barringer, H., M. Fisher, D. Gabbay, R. Owens, and Reynolds, M., (1996). *The Imperative Future: Principles of Executable Temporal Logic*, Research Studies Press Ltd. and John Wiley & Sons

Bradshaw, J.M., (ed.) (1997). Software Agents. AAAI Press / The MIT Press, London, England.

Brazier, F.M.T., Cornelissen, F., Gustavsson, R., Jonker, C.M., Lindeberg, O., Polak, B., and Treur, J., (1998a). Agents Negotiating for Load Balancing of Electricity Use. In: M.P. Papazoglou, M. Takizawa, B. Krämer, S. Chanson (eds.), *Proceedings of the 18th International Conference on Distributed Computing Systems*, ICDCS'98. IEEE Computer Society Press, Los Alamitos, CA, pp. 622-629.

Brazier, F.M.T., Cornelissen, F., Gustavsson, R., Jonker, C.M., Lindeberg, O., Polak, B., and Treur, J., (1998b). Compositional Design and Verification of a Multi-Agent System for Load Balancing. In: *Proceedings of the Third International Conference on Multi-Agent Systems*, ICMAS-98, IEEE Society Press, Los Alamitos, CA, pp. 49-56.

Brazier, F.M.T., Dunin-Keplicz, B., Jennings, N.R., and Treur, J., (1995). Formal specification of Multi-Agent Systems: a real-world case. In: V. Lesser (Ed.), *Proceedings of the First International Conference on Multi-Agent Systems*, ICMAS'95, MIT Press, Cambridge, MA, pp. 25-32. Extended version in: *International Journal of Cooperative Information Systems*, M. Huhns, M. Singh, (eds.), special issue on Formal Methods in Cooperative Information Systems: Multi-Agent Systems, vol. 6, 1997, pp. 67-94.

Brazier, F.M.T., Eck, P.A.T. van, and Treur, J., (1997). Modelling a society of simple agents: from conceptual specification to experimentation. In: Conte, R., Hegselmann, R. and Terna, P. (eds.), *Simulating Social Phenomena, Proceedings of the International Conference on Computer Simulation and Social Sciences*, ICCS&SS'97, Lecture Notes in Economics and Mathematical Systems, vol. 456, Springer Verlag, pp. 103-107.

Brazier, F.M.T., Jonker, C.M., Jüngen, F.J., and Treur, J., (1999). Distributed Scheduling to Support a Call Centre: a Co-operative Multi-Agent Approach. In: *Applied AI Journal*, vol. 13 (Special Issue on Multi-Agent Systems), pp. 65-90.

Brazier, F.M.T., Jonker, C.M., and Treur, J., (1996). Modelling Project Coordination in a Multi-Agent Framework. In: *Proceedings of the Fifth Workshops on Enabling Technology for Colloborative Enterprises,* WET ICE'96, IEEE Computer Society Press, pp. 148-155.

Brazier, F.M.T., Jonker, C.M., and Treur, J., (1998). Principles of Compositional Multi-agent System Development. In: J. Cuena (ed.), *Proceedings of the 15th IFIP World Computer Congress, WCC'98, Conference on Information Technology and Knowledge Systems,* IT&KNOWS'98, IFIP, 1998, pp. 347-360.

Brazier, F.M.T., Jonker, C.M., and Treur, J., (1999). Design of Intelligent Multi-agent Systems: a five-day Course. URL: http://www.cs.vu.nl/~wai/demas.

Brazier, F.M.T., Jonker, C.M., Treur, J., and Wijngaards, N.J.E., (1998). An Agent Architecture for Dynamic Re-design of Agents. In: P. Rodgers and A. Huxor (eds.), *Proceedings of the AID'98 Workshop on Distributed Web-based Design Tools,* Key Centre of Design Computing, University of Sydney, Sydney, pp. 16.

Brazier, F.M.T., Jonker, C.M., Treur, J., and Wijngaards, N.J.E., (1999). On the use of shared task models in knowledge acquisition, strategic user interaction and clarification agents. In: *International Journal of Human Computer Studies.* In press.

Brazier, F.M.T., Treur, J., Wijngaards, N.J.E. and Willems, M., (1999). Temporal Semantics of Compositional Task Models and Problem Solving Methods. In: *Data and Knowledge Engineering,* vol. 29(1), 1999, pp. 17-42.

Cohen, P.R. and Levesque, H.J., (1990). Intention is choice with commitment, In: *Artificial Intelligence* 42, pp. 213-261.

Dennett, D.C., (1987). *The Intentional Stance.* MIT Press, Cambridge.

Dijkstra, E.W., (1976). *A discipline of programming.* Prentice Hall.

Dubois, E., Yu, E., and Petit, M., (1998). From Early to Late Formal Requirements. In: *Proceedings of* IWSSD'98. IEEE Computer Society Press.

Dubois, E., Du Bois, P., and Zeippen, J.M., (1995). A Formal Requirements Engineering Method for Real-Time, Concurrent, and Distributed Systems. In: *Proceedings of the Real-Time Systems Conference,* RTS'95.

Engelfriet, J., Jonker, C.M. and Treur, J., (1999). Compositional Verification of Multi-Agent Systems in Temporal Multi-Epistemic Logic. In: J.P. Mueller, M.P. Singh, and A.S. Rao (eds.), *Pre-proc. of the Fifth International Workshop on Agent Theories, Architectures and Languages, ATAL'98,* 1998, pp. 91-106. To appear in: J.P. Mueller, M.P. Singh, A.S. Rao (eds.), *Intelligent Agents V.* Lecture Notes in AI, Springer Verlag.

Engelfriet, J., and Treur, J., (1995). Temporal Theories of Reasoning. In: *Journal of Applied Non-Classical Logics,* 5, pp. 239-261.

Engelfriet, J., and Treur, J., (1996a). Specification of nonmonotonic reasoning. In: *Proceedings of the International Conference on Formal and Applied Practical Reasoning,* Springer-Verlag, Lecture Notes in Artificial Intelligence, vol. 1085, pp. 111–125.

Engelfriet, J., and Treur, J., (1996b). Executable temporal logic for nonmonotonic reasoning; *Journal of Symbolic Computation,* vol. 22, no. 5&6, pp. 615–625.

Fagin, R., Halpern, J., Moses, Y., and Vardi, M., (1995). *Reasoning about Knowledge.* Cambridge, MA, MIT Press.

Finger, M., and Gabbay, D., (1992). Adding a temporal dimension to a logic system. In: *Journal of Logic, Language and Information* 1, pp. 203–233.

Finin, T., Labrou, Y., and Mayfield, J., (1997). KQML as an Agent Communication Language. In: (Bradshaw, 1997), pp. 291-316.

Fisher, M., (1994). A survey of Concurrent METATEM — the language and its applications. In: D.M. Gabbay, H.J. Ohlbach (eds.), *Temporal Logic, Proceedings of the First International Conference,* Lecture Notes in AI, vol. 827, pp. 480–505.

Fisher, M., and Wooldridge, M., (1997). On the formal specification and verification of multi-agent systems. In: Huhns, M. and Singh, M. (eds.), *International Journal of Co-operative Information Systems, IJCIS* vol. 6 (1), special issue on Formal Methods in Co-operative Information Systems: Multi-Agent Systems, pp. 37–65.

Gero, J.S., and Sudweeks, F., eds. (1996) In: *Artificial Intelligence in Design '96*, Kluwer Academic Publishers, Dordrecht.

Gero, J.S., and Sudweeks, F., eds. (1998) In: *Artificial Intelligence in Design '98*, Kluwer Academic Publishers, Dordrecht.

Gibson, J.J., (1960). The concept of the stimulus in psychology. In: *American Psychology* 15, pp. 694-703.

Herlea, D.E., Jonker, C.M., Treur, J., and Wijngaards, N.J.E., (1999). Integration of Behavioural Requirements Specifications within Knowledge Engineering. In: *Proceedings of the European Knowledge Acquisition Workshop*, EKAW99, to appear.

Jennings, N.R., and Wooldridge, M., (1998). Applications of Intelligent Agents. In: Jennings, N.R., and M. Wooldridge (eds.), *Agent Technology: Foundations, Applications, and Markets*. Springer Verlag, pp. 3-28.

Jonker, C.M., and Treur, J., (1998a), Compositional verification of multi-agent systems: a formal analysis of pro-activeness and reactiveness. In: W.P. de Roever, H. Langmaack, A. Pnueli (eds.), *Proceedings of the International Workshop on Compositionality*, COMPOS'97, Lecture Notes in Computer Science, vol. 1536, Springer Verlag, Berlin Heidelberg, pp. 350-380.

Jonker, C.M., and Treur, J., (1998b), Agent-based Simulation of Reactive, Pro-active, and Social Animal Behaviour. In: *Proceedings of the 11th International Conference on Industrial and Engineering Applications of AI and Expert Systems*, IEA/AIE'98, vol. I). Lecture Notes in AI, vol. 1415, Springer Verlag, Berlin Heidelberg, pp. 584-595.

Jonker, C.M., and Treur, J., (1998c), A Generic Architecture for Broker Agents. In: Nwana, S.H., and Ndumu, D.T., (eds.), *Proceedings of the Third International Conference on Practical Applications of Agents and Multi-Agent Systems*, PAAM-98, The Practical Application Company Ltd., pp. 623-624.

Jonker, C.M., and Treur, J., (1999). Information Broker Agents in Intelligent Web-sites. In: *Proceedings of the 12th International Conference on Industrial and Engineering Applications of AI and Expert Systems*, IEA/AIE'99. Lecture Notes in AI, Springer Verlag, Berlin Heidelberg. In press.

Manna, Z., and Pnueli, A., (1995). *Temporal Verification of Reactive Systems: Safety*. Springer Verlag, Berlin-Heidelberg

Meyer, J.-J., Ch., and Hoek, W. van der, (1995). *Epistemic Logic for AI and Computer Science*. Cambridge Univerity Press.

Morgan, C.L., (1894). *An introduction to comparative psychology*. London: Scott, 1894.

PAAM-97, *Proceedings of the Second International Conference on Practical Applications of Agents and Multi-Agent System*, The Practical Application Company Ltd.

Nwana, S.H., and Ndumu, D.T., (eds.), (1998). *Proceedings of the Third International Conference on Practical Applications of Agents and Multi-Agent* System, PAAM-98. The Practical Application Company Ltd.

Pnueli, A., (1986). Applications of temporal logic to the specification and verification of reactive systems: A survey of current trends. In: J.W. de Bakker, W.P. de Roever and G. Rozenberg (eds.). *Current Trends in Concurrency*. Lecture Notes in Computer Science, vol. 224. Springer Verlag, Berlin-Heidelberg, pp. 510-584.

Rao, A.S., (1996). AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language. In: W. van der Velde, J.W. Perram (eds.), *Agents Breaking Away, Proceedings of the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, MAAMAW'96, Lecture Notes in AI, vol. 1038, Springer Verlag, pp. 42-55.

Rao, A.S., and Georgeff, M.P., (1991). Modeling rational agents within a BDI architecture. In: R. Fikes, and E. Sandewall (eds.), *Proceedings of the Second Conference on Knowledge Representation and Reasoning*, Morgan Kaufman, pp. 473-484.

Roever, W.P. de, Langmaack, H., and Pnueli, A., (eds.) (1998). *Proceedings of the International Workshop on Compositionality*, COMPOS'97, Lecture Notes in Computer Science, vol. 1536, Springer Verlag, Berlin Heidelberg.

Rosenschein, J.S., and Zlotkin, G., (1994a). *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers*, MIT Press.

Rosenschein, J.S., and Zlotkin, G., (1994b). Designing Conventions for Automated Negotiation, In: *AI Magazine*, Vol. 15-3, Fall 1994, pp. 29-46.

Sandewall, E., (1995) Features and Fluents.

Searle, J.R., (1969). *Speech Acts: an Essay in the Philosophy of Language*. Cambridge University Press. New York.

Skinner, B.F., (1935). The generic nature of the concepts of stimulus and response. In: *Journal of gen. Psychology* 12, pp. 40-65.

Tinklepaugh, O.L., (1932). Multiple delayed reaction with chimpanzees and monkeys. In: *Journal of Comparative Psychology*, 13, pp. 207-243.

Treur, J., (1994), Temporal Semantics of Meta-Level Architectures for Dynamic Control of Reasoning. In: L. Fribourg and F. Turini (ed.), *Logic Program Synthesis and Transformation-Meta-Programming in Logic, Proceedings of the Fourth International Workshop on Meta-Programming in Logic*, META'94. Springer Verlag, Lecture Notes in Computer Science, vol. 883, 1994. pp. 353-376.

Treur, J., (1998). Report on the First Meeting of the AgentLink Special Interest Group on Methodologies and Software Engineering of Agent Systems. URL: http://www.cs.vu.nl/~treur/SIG1report.html

Vauclair, J., (1996). *Animal Cognition*. Harvard Univerity Press, Cambridge, Massachusetts.

Watson, J.P., (1919). *Psychology from the standpoint of a behaviourist*. Philadelphia: Lippincott.

Wooldridge, M., and Jennings, N.R., (1995a), Agent theories, architectures, and languages: a survey. In: (Wooldridge and Jennings, 1995b), pp. 1-39.

Wooldridge, M., and Jennings, N.R., (eds.) (1995b), *Intelligent Agents, Proceedings of the First International Workshop on Agent Theories, Architectures and Languages*, Lecture Notes in AI, vol. 890, Springer Verlag.

## 6.4 Evolutionaire Sturing in Economische Strategie Ontwikkeling, door: Han La Poutré

# Evolutionaire Sturing in Economische Strategie Ontwikkeling

CWI-Themagroep
"Evolutionary Computation and Applied Algorithmics"

Han La Poutré *

De themagroep "Evolutionary Computation and Applied Algorithmics" is een jonge groep binnen het Centrum voor Wiskunde en Informatica (CWI). De groep richt zich op onderzoek en ontwikkeling op het gebied van evolutionaire algoritmen, neurale netwerken en discrete algoritmen, in het bijzonder met betrekking tot management-gerelateerde en economische problemen. In dit kader zullen van een van de onderzoeksgebieden van de groep, namelijk het kruisgebied van economie en evolutionaire computermethoden, hier enige achtergronden en ontwikkelingen worden geschetst.[1]

## Inleiding

Hedendaagse markten en ondernemingen kenmerken zich door een veel grotere dynamiek en diversiteit dan in het verleden. Oorzaken liggen ondermeer in de voortgang van de informatietechnologie en het steeds belangijker worden van de creatie en manipulatie van niet-materiële produkten (intangibles). Markten en ondernemingen zijn steeds meer complexe adaptieve systemen, gekenmerkt door snelle vraagveranderingen, onzekere marktwaarden, tijdelijke allianties en individualisering. Dit betekent dat aan de wijze van commerciële strategie-ontwikkeling, waaronder productiemanagement, marketing en financieel beleid, nieuwe eisen worden gesteld in termen van dynamiek, voorspelling en adaptiviteit. Deze nu, kunnen we tegemoettreden in het licht van evolutionaire processen en berekening met computers.

## Complexe Systemen in Ontwikkeling

De evolutietheorie neemt een belangrijke plaats in binnen de biologie: ontwikkelingen en veranderingen in de natuur blijken volgens evolutionaire regels te verlopen.

*Han La Poutré is onderzoeksleider van de themagroep "Evolutionary Computation and Applied Algorithmics" van het Centrum voor Wiskunde en Informatica (CWI) in Amsterdam; adres: CWI, Kruislaan 413, 1098 SJ Amsterdam; email: Han.La.Poutre@cwi.nl; tel. 020-5924082.

[1]Dit bevat tevens bewerkte en uitgebreide delen van een kort artikel van deze auteur, dat eerder is verschenen in: De Onvermijdelijke Culturele Revolutie, Stichting Maatschappij en Onderneming, ISBN 90-6962-143-6, mei 1998.

Recentelijk is de link met heel andere gebieden gelegd, zoals bijvoorbeeld natuurkundige, economische of neurale systemen. Dit heeft aanleiding gegeven tot fundamenteel onderzoek naar de evolutietheorie van abstracte systemen (S. Kauffman), de zogenaamde NK-systemen. Hierbij is bekeken hoe systemen zich aanpassen aan een bepaalde taak, door in een toestand te komen die optimaal is voor die taak. Gezien de complexiteit van het probleem, is het niet mogelijk om in één keer het juiste systeem te construeren of het direct in de juiste richting te sturen. Daarom wordt structureel het evolutionaire ontwikkelingsproces toegepast van gradueel veranderen (mutaties), waarbij de taakgeschiktheid de primaire drijfveer is voor het toelaten van een bepaalde mutatie. Op deze wijze is sturing van het systeem in een bepaalde richting mogelijk. Hiervan gebruik makend, blijken na enige tijd systemen te ontstaan met een structuur die in staat is tot een beheerst adaptief en gestructureerd gedrag: het evolutionaire selectiemechanisme optimaliseert zowel op taak als op adaptiviteit en flexibiliteit. In het bijzonder is een beheerste samenhang tussen elementen van het systeem aanwezig: het is "geordend, maar op de rand van chaos" en daarom flexibel. Het idee bestaat dat dit een meer algemene eigenschap is bij de evolutionaire ontwikkeling van systemen met dynamische omgevingen. Het "van boven af" geforceerd laten belanceren van een complex systeem (zoals een markt) in een dergelijke toestand lijkt echter buitengewoon moeilijk, laat staan het actief sturen naar een bepaalde einddoel, waarbij het preciese te volgen traject eveneens onduidelijk is. In dezelfde lijn zijn co-evolutionaire systemen onderzocht, waarin verschillende typen systemen met elkaar interageren en coöpereren. (In de natuur kunnen we denken aan verschillende diersoorten die samen een voedselketen vormen.) Ook hier vinden we adaptieve systemen met subtiele evenwichten tussen ordening en chaos.

Modelmatige, mathematisch analyse van en sturing in het adaptieve gedrag van systemen blijkt echter ingewikkeld en beperkt realistisch te zijn (J. Holland & J. Miller, S. Kauffman). Zo kunnen bijvoorbeeld traditionele economische modellen en expertsystemen moeilijk overweg met de complexiteit van dynamische markten waarin adaptieve agenten opereren. Een meer realistische modellering en aanpak is dus gewenst, waarbij de evolutionaire en zelfsturende principes in de ontwikkeling en dynamiek een belangrijke plaats dienen in te nemen. Evolutionaire ontwikkelmethoden kunnen daarom uitkomst bieden bij onderzoek, berekening en optimalisering.

## Evolutionaire Algoritmen

Evolutionaire algoritmen, ook genetische algoritmen genoemd, zijn berekeningsmethoden die gebaseerd zijn op de evolutieleer (J. Holland, D. Goldberg). Voor een bepaald probleem wordt een populatie van mogelijke kandidaatoplossingen gegenereerd, die achtereenvolgens herhaaldelijk wordt veranderend en verbeterd door middel van evolutionaire concepten: selectie, "survival of the fittest", mutatie en recombinatie. Doel is om goede, optimale oplossingen te creëren. Door selectie en "survival of the fittest" hebben kandidaatoplossingen met grote fitness (de kwaliteit ervan) een grotere kans om in de nieuwe populatie te komen; bij recombinatie worden uit twee kandidaatoplossingen twee nieuwe gemaakt, zoals in de natuur gebeurt (zoals bijvoorbeeld "crossover"). Recombinatie maakt het mogelijk om goede deeloplossingen te combineren tot een beter geheel: het overbrengen van een goed idee naar een nieuwe contekst en een vorm van innovatie.

Evolutionaire algoritmen (EAs) kunnen worden beschouwd als een evolutionair ontwikkelsysteem waarmee problemen kunnen worden opgelost. Hierbij is geen expliciete kennis nodig over hoe een goede oplossing moet worden gecreëerd: het EA bouwt de kennis over goede oplossingen impliciet op. Daarnaast is het mogelijk, een "black box" voor het EA te laten bepalen welke van twee kandidaatoplossingen de beste is (relatieve ordening); de manier waarop dat gebeurt en door wat of wie, is minder relevant. Dit maakt de methode aanzienlijk wijder toepasbaar, daar ze niet slechts numeriek gebaseerd is. Daarom zijn evolutionaire algoritmen vaak zeer geschikt voor problemen met ingewikkelde of vage criteria, of met variërende dan wel onnauwkeurige data of modellering.

## Evolutionaire Strategie-Ontwikkeling

De ontwikkeling van economische of commerciële strategieën is een nieuw veld voor gebruik van evolutionaire algoritmen. Ze openen de mogelijkheid om commerciële strategieën te ontwikkelen, die zowel gebaseerd kunnen zijn op concrete handels- en markinformatie als op data gegenereerd in simulatie-omgevingen, en waarbij het dynamische en niet-lineaire gedrag van een markt kan worden opgenomen. Traditioneel zijn economische systemen gemodelleerd door middel van rationeel handelende agenten met volledige informatie. Dit is echter een significante beperking. Agenten hebben vaak maar beperkte informatie, handelen uit eigenbelang en vaak niet "rationeel".

In een EA-populatie worden agenten met verschillende taakstellingen als strategieën gerepresenteerd. Agenten handelen hierbij binnen interactieschema's (zoals bijvoorbeeld tussen producenten en consumenten) en volgen hun eigen strategie; de opbrengsten daaruit bepalen de fitness per agent. Zo ontstaat een co-evolutionair systeem dat middels EA-concepten kan optimaliseren of stabiliseren. Agenten kunnen hierin leren van andere agenten (kennis- en informatie-uitwisseling) of kunnen op hen anticiperen (voorspellen). Deze mechanismen kunnen middels evolutionaire operatoren worden uitgedrukt. Zo kunnen informatie-uitwisseling of leerprocessen tussen economische agenten worden geimplementeerd door een combinatie van recombinatie- en selectie-operatoren. Het preciese model (semantiek) van dergelijke combinaties van operatoren hangt af van de uiteindelijke implementatie van de recombinatie-operator, het selectie-schema (elitist/unconditional replacement), generationele of andere EA technieken, etc. Soortgelijk wordt de interactie tussen agenten gemodelleerd; dit kan bijvoorbeeld een onderhandelings-model zijn, een coöperatie-probleem, of bepaling van eigen activiteiten (zoals de eigen productiehoeveelheid). Naast deze modellering, kunnen andere aspecten worden opgenomen, bijvoorbeeld in de vorm van restricties op mogelijke interactie-partners voor agenten.

Aldus kunnen meer realistische kenmerken aan agenten of hun omgevingen worden toegekend. Belangrijk aspect hierbij is dat een agent slechts beperkte capaciteiten heeft, zoals een beperkte hoeveelheid informatie, kennis en geheugen of beperkte gedragsmogelijkheden. Ander aspect is de verspreiding van informatie, kennis en gedrag door een systeem van agenten heen, via locale mechanismen (zoals recombinatie-operatoren). Met name in dynamische niet-materiële markten, zoals van services, informatie, entertainment, kennis en financiën, waarin irrationale groepsaspecten de doorslag kunnen geven en waar traditionele randvoorwaarden zoals materiële schaarste kunnen wegvallen, lijken dit belangrijke eigenschappen.

Op deze wijze kan het effect van een bepaalde individuele strategie op de ontwikkeling van het totale systeem worden bestudeerd (bijv. markten en Nash-equilibria), als wel de strategieën zelf worden geoptimaliseerd. Maar ook kunnen kwalitatieve afhankelijkheden van eigenschappen en mechanismen worden bestudeerd middels evolutionaire computersimulaties, en kan het resulterend dynamisch gedrag van een marktsysteem ("emergent gedrag") worden bepaald als gevolg van basis-eigenschappen van agenten en marktmechanismen. Voorbeelden zijn prijsbepaling in een markt met een beperkt aantal producenten, samenwerking van produktiebedrijven (process chains), electronic commerce, investeringsbeleid, of voorspelling van trends.

## Economische Strategieën, Electronic-Commerce, en Evolutionaire Methoden

In het bovenstaande zijn economische strategie-bepaling, evolutietheorie en berekening middels evolutionaire algoritmen samengekomen. Deze combinatie vormt een van de onderzoeksgebieden van de themagroep "Evolutionary Computation and Applied Algorithmics". Electronic commerce problemen vormen hierbij een belangrijk nieuw applicatie-gebied.

## Referenties

(Selectie)

J. Biethahn & V. Nissen (eds.), Evolutionary Algorithms in Management Applications, Springer-Verlag, Berlin, 1995.

D.D.B. van Bragt, C.H.M. van Kemenade, and J.A. La Poutré, The Influence of Evolutionary Selection Schemes on the Iterated Prisoner's Dilemma, Conference on Computational Economics and Finance '99, Boston, USA; forthcoming.

J. Bullard and J. Duffy, A Model of Learning and Emulation with Artificial Adaptive Agents, J. Economic Dynamics and Control 22 (1998), pp. 179-207.

H. Dawid, Adaptive Learning by Genetic Algorithms: Analytical Results and Applications to Economic Models, Springer Lecture Notes in Economics and Mathematical Systems 441, 1996.

D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, 1989.

J.H. Holland, Adaptation in Natural and Artifical Systems, MIT Press, Cambridge, 1992.

J.H. Holland & J.H. Miller, Artificial Adaptive Agents in Economic Theory, American Economic Review: Proceedings of the 103rd Ann. Meeting of the American Economic Association, 365-370, 1991.

S.A. Kauffman, The Origins of Order: Self-Organisation and Selection in Evolution, Oxford University Press, 1993.

## Pages Deleted

The following pages contain(ed) the memberlist of the NVTI. These pages have been deleted to protect the privacy of our members.

## 8 Statuten

**Artikel 1.**
1. De vereniging draagt de naam: "Nederlandse Vereniging voor Theoretische Informatica".
2. Zij heeft haar zetel te Amsterdam.
3. De vereniging is aangegaan voor onbepaalde tijd.
4. De vereniging stelt zich ten doel de theoretische informatica te bevorderen haar beoefening en haar toepassingen aan te moedigen.

**Artikel 2.**
De vereniging kent gewone leden en ereleden. Ereleden worden benoemd door het bestuur.

**Artikel 3.**
De vereniging kan niet worden ontbonden dan met toestemming van tenminste drievierde van het aantal gewone leden.

**Artikel 4.**
Het verenigingsjaar is het kalenderjaar.

**Artikel 5.**
De vereniging tracht het doel omschreven in artikel 1 te bereiken door
a. het houden van wetenschappelijke vergaderingen en het organiseren van symposia en congressen;
b. het uitgeven van een of meer tijdschriften, waaronder een nieuwsbrief of vergelijkbaar informatiemedium;
c. en verder door alle zodanige wettige middelen als in enige algemene vergadering goedgevonden zal worden.

**Artikel 6.**
1. Het bestuur schrijft de in artikel 5.a bedoelde bijeenkomsten uit en stelt het programma van elk van deze bijeenkomsten samen.
2. De redacties der tijdschriften als bedoeld in artikel 5.b worden door het bestuur benoemd.

**Artikel 7.**
Iedere natuurlijke persoon kan lid van de vereniging worden. Instellingen hebben geen stemrecht.

**Artikel 8.**
Indien enig lid niet langer als zodanig wenst te worden beschouwd, dient hij de ledenadministratie van de vereniging daarvan kennis te geven.

**Artikel 9.**
Ieder lid ontvangt een exemplaar der statuten, opgenomen in de nieuwsbrief van de vereniging. Een exemplaar van de statuten kan ook opgevraagd worden bij de secretaris. Ieder lid ontvangt de tijdschriften als bedoeld in artikel 5.b.

**Artikel 10.**
Het bestuur bestaat uit tenminste zes personen die direct door de jaarvergadering worden gekozen, voor een periode van drie jaar. Het bestuur heeft het recht het precieze aantal bestuursleden te bepalen. Bij de samenstelling van het bestuur dient rekening gehouden te worden met de wenselijkheid dat vertegenwoordigers van de verschillende werkgebieden van de theoretische informatica in Nederland in het bestuur worden opgenomen. Het bestuur kiest uit zijn midden de voorzitter, secretaris en penningmeester.

**Artikel 11.**
Eens per drie jaar vindt een verkiezing plaats van het bestuur door de jaarvergadering. De door de jaarvergadering gekozen bestuursleden hebben een zittingsduur van maximaal twee maal drie jaar. Na deze periode zijn zij niet terstond herkiesbaar, met uitzondering van secretaris en penningmeester. De voorzitter wordt gekozen voor de tijd van drie jaar en is na afloop van zijn ambtstermijn niet onmiddellijk als zodanig herkiesbaar. In zijn functie als bestuurslid blijft het in de vorige alinea bepaalde van kracht.

**Artikel 12.**
Het bestuur stelt de kandidaten voor voor eventuele vacatures. Kandidaten kunnen ook voorgesteld worden door gewone leden, minstens een maand voor de jaarvergadering via de secretaris. Dit dient schriftelijk te gebeuren op voordracht van tenminste vijftien leden. In het geval dat het aantal kandidaten gelijk is aan het aantal vacatures worden de gestelde kandidaten door de jaarvergadering in het bestuur gekozen geacht. Indien het aantal kandidaten groter is dan het aantal vacatures wordt op de jaarvergadering door schriftelijke stemming beslist. Ieder aanwezig lid brengt een stem uit op evenveel kandidaten als er vacatures zijn. Van de zo ontstane rangschikking worden de kandidaten met de meeste punten verkozen, tot het aantal vacatures. Hierbij geldt voor de jaarvergadering een quorum van dertig. In het geval dat het aantal aanwezige leden op de jaarvergadering onder het quorum ligt, kiest het zittende bestuur de nieuwe leden. Bij gelijk aantal stemmen geeft de stem van de voorzitter (of indien niet aanwezig, van de secretaris) de doorslag.

**Artikel 13.**

Het bestuur bepaalt elk jaar het precieze aantal bestuursleden, mits in overeenstemming met artikel 10. In het geval van aftreden of uitbreiding wordt de zo ontstane vacature aangekondigd via mailing of nieuwsbrief, minstens twee maanden voor de eerstvolgende jaarvergadering. Kandidaten voor de ontstane vacatures worden voorgesteld door bestuur en gewone leden zoals bepaald in artikel 12. Bij aftreden van bestuursleden in eerste of tweede jaar van de driejarige cyclus worden de vacatures vervuld op de eerstvolgende jaarvergadering. Bij aftreden in het derde jaar vindt vervulling van de vacatures plaats tegelijk met de algemene driejaarlijkse bestuursverkiezing. Voorts kan het bestuur beslissen om vervanging van een aftredend bestuurslid te laten vervullen tot de eerstvolgende jaarvergadering. Bij uitbreiding van het bestuur in het eerste of tweede jaar van de cyclus worden de vacatures vervuld op de eerstvolgende jaarvergadering. Bij uitbreiding in het derde jaar vindt vervulling van de vacatures plaats tegelijk met de driejaarlijkse bestuursverkiezing. Bij inkrimping stelt het bestuur vast welke leden van het bestuur zullen aftreden.

**Artikel 14.**

De voorzitter, de secretaris en de penningmeester vormen samen het dagelijks bestuur. De voorzitter leidt alle vergaderingen. Bij afwezigheid wordt hij vervangen door de secretaris en indien ook deze afwezig is door het in jaren oudste aanwezig lid van het bestuur. De secretaris is belast met het houden der notulen van alle huishoudelijke vergaderingen en met het voeren der correspondentie.

**Artikel 15.**

Het bestuur vergadert zo vaak als de voorzitter dit nodig acht of dit door drie zijner leden wordt gewenst.

**Artikel 16.**

Minstens eenmaal per jaar wordt door het bestuur een algemene vergadering bijeengeroepen; één van deze vergaderingen wordt expliciet aangeduid met de naam van jaarvergadering; deze vindt plaats op een door het bestuur te bepalen dag en plaats.

**Artikel 17.**

De jaarvergadering zal steeds gekoppeld zijn aan een wetenschappelijk symposium. De op het algemene gedeelte vaan de jaarvergadering te behandelen onderwerpen zijn

a. Verslag door de secretaris;

b. Rekening en verantwoording van de penningmeester;

c. Verslagen van de redacties der door de vereniging uitgegeven tijdschriften;

d. Eventuele verkiezing van bestuursleden;

e. Wat verder ter tafel komt. Het bestuur is verplicht een bepaald punt op de agenda van een algemene vergadering te plaatsen indien uiterlijk vier weken van te voren tenminste vijftien gewone leden schriftelijk de wens daartoe aan het bestuur te kennen geven.

**Artikel 18.**

Deze statuten kunnen slechts worden gewijzigd, nadat op een algemene vergadering een commissie voor statutenwijziging is benoemd. Deze commissie doet binnen zes maanden haar voorstellen via het bestuur aan de leden toekomen. Gedurende drie maanden daarna kunnen amendementen schriftelijk worden ingediend bij het bestuur, dat deze ter kennis van de gewone leden brengt, waarna een algemene vergadering de voorstellen en de ingediende amendementen behandelt. Ter vergadering kunnen nieuwe amendementen in behandeling worden genomen, die betrekking hebben op de voorstellen van de commissie of de schriftelijk ingediende amendementen. Eerst wordt over elk der amendementen afzonderlijk gestemd; een amendement kan worden aangenomen met gewone meerderheid van stemmen. Het al dan niet geamendeerde voorstel wordt daarna in zijn geheel in stemming gebracht, tenzij de vergadering met gewone meerderheid van stemmen besluit tot afzonderlijke stemming over bepaalde artikelen, waarna de resterende artikelen in hun geheel in stemming gebracht worden. In beide gevallen kunnen de voorgestelde wijzigingen slechts worden aangenomen met een meerderheid van tweederde van het aantal uitgebrachte stemmen. Aangenomen statutenwijzigingen treden onmiddellijk in werking.

**Artikel 19.**

Op een vergadering worden besluiten genomen bij gewone meerderheid van stemmen, tenzij deze statuten anders bepalen. Elk aanwezig gewoon lid heeft daarbij het recht een stem uit te brengen. Stemming over zaken geschiedt mondeling of schriftelijk, die over personen met gesloten briefjes. Uitsluitend bij schriftelijke stemmingen worden blanco stemmen gerekend geldig te zijn uitgebracht.

**Artikel 20.**

a. De jaarvergadering geeft bij huishoudelijk reglement nadere regels omtrent alle onderwerpen, waarvan de regeling door de statuten wordt vereist, of de jaarvergadering gewenst voorkomt.

b. Het huishoudelijk reglement zal geen bepalingen mogen bevatten die afwijken van of die in strijd zijn met de bepalingen van de wet of van de statuten, tenzij de afwijking door de wet of de statuten wordt toegestaan.

**Artikel 21.**

In gevallen waarin deze statuten niet voorzien, beslist het bestuur.