FACS Europe

The Newsletter of the BCS Formal Aspects of Computing Science Special Interest Group and Formal Methods Europe.

Series I Vol. 2, No. 1 Summer 1995

Contents

Editorial

This issue sees a number of changes. One is that we say farewell to Jawed Siddiqi, with appreciation for his work with Chris Roast in keeping the Newsletter going through the changes and difficulties of the last few years. The Newsletter team is now myself as editor, with assistance from Chris Roast and Margaret West, and from a number of others who have taken responsibility for trying to get material regularly on particular topics. If you have anything to offer, please do not wait for any of us to ask you - send it in, electronically if possible (email to *FACS@lut.ac.uk* or to me at *a.wrightson@hud.ac.uk*) or by snailmail to:

FACS/FME Newsletter
c/o Ann M Wrightson
School of Computing and Mathematics
University of Huddersfield
Queensgate
Huddersfield
HD1 3DH
UK

Tel: (01484) 472758
Fax: (01484) 421106

Another change is the return of John Cooke as Chair of BCS-FACS, with Tim Denvir taking on the role (from the FACS point of view) of liaison with FME, to complement his role in FME itself. One of my hopes for the Newsletter is that it can be more interesting and relevant to our industrial readership (I've had some hard words said to me about it being only marginally better than the FACS Journal for that...); that depends a lot, of course, on those readers letting us know what they are interested in, so please talk to us and help make it a useful source of news and views for everyone seriously involved with "formal aspects" and "formal methods".

Events

There are a number of FACS events planned for the next year or two. There are announcements for the 1995 Xmas Workshop, the Seventh Refinement Workshop and a workshop on Formal Aspects of the Human-Computer Interface, at the back of this issue.

Other events being planned include a joint event with the BCS Requirements Engineering SIG, and a workshop on developments in Theorem-proving. If you have any other ideas for events, please email them to FACS@lut.ac.uk, or talk to any of us...

Contributions Welcome...

Contributions to the Newsletter on any relevant topic are welcome. Please send them electronically, in LaTeX or TeX form if you can; next best is plain ASCII. Otherwise please send A4 copy fit to reproduce by fast photocopying (i.e. no paste-ups), with 300dpi laserprint or equivalent a minimum standard. We will not convert WP formats or type up manuscripts. We will not reproduce extensive notices of events which are also available electronically; please send a short notice (max 1 page) with pointers to more extensive information where available. Please always include a postal or telephone contact for those without email.

Contributions express the opinions of contributors, not of FACS, FME or any organization with which they are associated (unless they say otherwise!).

Letters are welcome and should be sent to the Editor.

Note *from Margaret West*

After Tom Maibaum's interesting talk at the AGM [reported within—Ed.] I mentioned work by Toby Walsh on "abstraction". Just before I dashed off for the train one or two people asked me his email and I now have it: *tw@aisb.ed.ac.uk*.

An example of a published paper on his work is: F Giunchiglia and T Walsh, Abstraction in AI, AISB Quarterly (73): 22-26, 1990.

BCS-FACS Xmas Workshop 1994

The BCS-FACS 1994 Xmas workshop was held at Imperial College, London on 19-20 December 1994, on the theme of **Concurrency**. It was organized on behalf of BCS-FACS by Tim Denvir *timdenvir@cix.compulink.co.uk* and Richard Mitchell *Richard.Mitchell@brighton.ac.uk*. The workshop provided a good spread of topics, including recent work based on well established specification languages, reviews of broader areas of work, and reports of work in progress.

This report is based on my [Ed] notes for most of it; thanks to John Cooke for notes on the last half-day. Mike Ingleby took a more raffish view - his report follows this one!

1 Formulating Rely/Guarantee-conditions

Cliff Jones, cliff@computer-science.manchester.ac.uk

The starting point of Cliff Jones' talk was a shift in emphasis in considering invariants, from establishing invariants to make things safe, to thinking about permitted perturbations of desired conditions. This was brought together with the idea of two parallel processes together satisfying a 4-tuple made up of preconditions, rely-conditions, guarantee-conditions, and post-conditions; with a formalization of the idea of one parallel process tolerating the interference of both the environment, and the activity of the other process. This led on to evolutionary predicates, as a dynamic aspect of the observable state-transitions which end up conjoined with the other post-conditions. (Tom Maibaum suggested a similarity between evolve, and dynamic constraints in databases, and another interesting similarity between the guarantee aspects, and frame conditions as used in temporal logic and AI.)

An implementation using Fischer-Galler forests had an interesting concurrent feature in itself which displayed a strong analogy with garbage-collection. A process collapsing linear paths to retain the significant branching structure can be working in garbage-collection style behind the scenes of other tree operations, without harmful interference. Cliff informed us he had a PhD student willing to prove it!

Then we came back to rely and guarantee conditions, with concern expressed about implicit 'do nothing' conditions, the granularity choices implicit in many development methods, and about the difficulty of realizing code implementing the specification conditions needed to satisfy interference tolerance.

2 Compilation and Transformation in a Parallel Extension of TDF

Tom Lake, Tom.Lake@glossa.co.uk

This was an account of a two-year project extending the intermediate language TDF to cover concurrency and distribution, carried out for DRA Malvern. As you would expect, the extensions to TDF are independent of explcit synchronization mechanisms such as thread scheduling or busy-wait, and can be compiled for radically different control architectures. The level and style of abstraction is similar to that used in CSP, though the concurrency operators are not identical. The behaviour necessary for concurrent interaction is characterized using guarantees of progress, giving independent progress for each cluster of operations contained in a different gop.

6 Higher-Order Processes and Their Models

Matthew Hennessy, matthewh@cogs.sussex.ac.uk

Matthew Hennessy presented the motivation for higher-order process algebra as a natural progression of ideas in process algebra. In the beginning were pure process algebras, typically applied to analysis of abstract communication structures, then came augmentation with data to address applications to analysing distributed systems. Thence, via the λ-calculus (considered as a very primitive process algebra which allows processes to be sent and received as values), to higher-order process algebra, where values are processes. Applications here include understanding new-generation programming languages such as CML and FACILE, and specification languages for distributed systems with dynamic topologies, such as π-calculus and CHOCS. The picture then emerged in more detail, with the presentation of an operational semantics, and a discussion of the extension to higher order of the characterization of 'may' and 'must' by traces and acceptance trees respectively.

7 Specifications, Programs and Operating Systems: A Question of Politeness

Tom Maibaum, tsem@doc.imperial.ac.uk

Tom Maibaum focussed on emergent properties of a component in a system, which are not predictable from the component's specification, but emerge from consideration of the behaviour of the system as a whole. Using (to the sound of faint groans from here and there in the audience) the dining philosophers as an example, the problem emerged from an overall need for no philosopher to starve, combined with there being no guarantee internal to any philosopher that his or her current bout of eating would eventually stop. The detailed development was in the context of the categorial model of compositionality developed with José Fiadeiro and others.

The main motivation is the construction of a specification of a system from the specifications of the components. Components synchronize on some of their actions; synchronization is defined in terms of shared components. The key notions characterized are part-of (the system specification), and communication and synchronization, using a generalization of property-preserving morphisms between theories in first-order logic. The system comes out as a colimit, with a coincidence of objects in the software engineering and categorial sense, and interpretations between theories as the morphisms. That the category is finitely co-complete amounts to being able to find a specification which is a minimal interpretation of the intended meaning of the diagram. All properties of a component are in the system (via a property-preserving morphism). And because of the colimit property, any other system specification is an image of this one. The main function of category theory here is to capture a notion of composition from components. (This stuff, reviewed in this presentation, was presented by José Fiadeiro in the 1993 BCS-FACS Xmas workshop - no apologies, essential background for what follows...)

Previous work relating specifications to process models, and on characterizing refinement (reification, implementation) doesn't give a framework which accommodates the emergent properties as above; something is missing. How is the system to enforce the desired behaviour, when combining components written to the component specifications? Options include redesigning the components to be sociable, or having an environment which enforces the sociability (eg takes forks off philosophers who eat too long). What is missing from the formalism, to support reasoning about these meta-assumptions, is the formal representation of *programs* and their relations to specifications. Taking this through (I really do have to cut this one short somehow...) leads to a situation where systems made up from components lose permissions and gain obligations, compared to the components, and this accounts neatly for emergent properties. To represent formally the notion of correctness between a system specification (made up of parts) and a system (made up of programs corresponding to those parts), there is a translation back from a program (resp. system) to a(nother) specification. Correctness comes from there being an appropriate structure of morphisms between this and the original specification, represented as a pretty layered commu-

3 Synchronous Logic Systems

David Gilbert, drg@cs.city.ac.uk

A different view of concurrency here, from the world of logic programming. This approach has arisen using the concepts of synchronous communication and shared logic variables. Different from process algebras—using shared variables rather than ports—like CIRCAL in some respects, and inherently multi-party. The approach was presented in detail using the examples of a 2-place buffer, a queue, and an expedited data queue (as in earlier version of LOTOS). There is an interpreter in Sixtus Prolog. Asynchronous parallel constraints are becoming popular in the logic programming world, and work in progress includes reformulation in a framework of constraints (the interpreter will be moved into the constraint formalism too). There is also a forthcoming paper on algebraic and compositional semantics, and work in hand on tools based on the semantics, aimed at the logic programming world.

4 A Model of Synchronous Behaviour with Duration

Mike Shields, m.shields@mcs.surrey.ac.uk

Introduced as 'a look at what I've been playing with over the summer', this was an exhilarating ride through a fresh look at underlying concepts for interval logics. Mike started from a view of actions which considers them as uninterruptable, taking an exact amount of time, and with an action having a set time to wait for release of necessary resources after another action acts on a component of state common to both. The behaviour is modelled by a function from a set of actions to sets of open intervals in the non-negative reals, built iteratively by derivation rules, obtaining a direct characterization of the set of behaviours via a transition system. Concurrency is modelled by a delay of zero. The points where an action is enabled come out as a set of instantaneous points, plus an open interval after the earliest time after which there is no (known) disabling of the action. An alternative representation has some similarity (not yet explored) to a timed Petri net, and can also be transformed to a state-transition representation showing things such as lack of deadlocks.

An entertaining presentation, well scheduled to dispel any tendency to sleep after lunch.

5 Rigorous Development of Concurrent O-O Systems in VDM++

Kevin Lano, kcl@doc.imperial.ac.uk; Stephen Goldsack, sjg@doc.imperial.ac.uk

This talk placed itself firmly in the context of various explorations of extensions of VDM and Z (there are about 8 extensions of Z, and 3 of VDM, accommodating principally concurrency and objects), and also in the context of the AFRODITE project. It was a long tutorial-style presentation. The O-O side was presented by Kevin Lano, concurrency by Stephen Goldsack.

The method invocation protocol is based on Ada's rendezvous facility, with the semantics allowing a delay between the request and the corresponding action. Inheritance has two distinct sides to it: representation inheritance, which provides straightforward inheritance of instance variables, and controlled inheritance, which controls which methods are inherited, and takes that inheritance relationship out of the subtype hierarchy. There is also support for a form of aggregation, of the 'a car has 4 wheels' variety.

VDM++ classes have extra bits for synchronization and thread characterization, the latter partly similar to 'answer' in Ada. Synchronization of passive objects (i.e. objects without 'thread' parts) is a problem, addressed by having a default synchronization of mutual exclusion, which can be overridden, and by employing guard predicates as used in DRAGOON.

There was much more, but hopefully these few points adequately convey the flavour of this work.

tative diagram showing the three layers involved. For this to work, there needs to be a functor between programs and specifications - but functors between nice programming formalisms and nice specification formalisms don't always exist when you want them, eg because there is no way to calculate the corresponding morphism between specifications, where you have clearly related programs. This can be remedied by strengthening the system program until it *can* be calculated—thus paving the way for taking the fork off the philosopher, or alternatively yielding a fairness condition on the program, which interestingly leads to an explicit reference to the *other* component in the interaction when considering obligations. Rely/guarantee conditions seem a natural way of taking this further.

There was much more, but I hope this gives enough to grasp the idea. It was also interesting to hear that there is related current work on agents working according to rules of cooperation being undertaken by a series of interdisciplinary workshops including philosophers and lawyers, with a special volume of Studia Logica due out sometime in 1995.

8 Verification of Properties of LOTOS Specifications: A Logic and a Proof System

Carron Kirkwood, carron@dcs.glasgow.ac.uk

Carron Kirkwood described how to check LOTOS specs by a variety of means. The main questions were how to formulate correctness in a LOTOS spec, and how to show when you've got it. One approach was behavioural, looking at whether an implementation in a concrete level of LOTOS satisfied a specification in a more abstract level. The verification tool PAM from the University of Sussex helped with this. Another approach used was property testing, using simulation and other testing techniques to ascertain reachability and other characteristics. A third was to use logic, describing abstract properties. Other work in this vein has used CTL, but this was found to be rather weak, so they moved to a modal μ-calculus as used with CCS. One non-trivial problem found here was that current LOTOS semantics is not valid on open terms, and these are needed to reason about intermediate states and parts of the system.

An important conclusion was that the approaches used were not alternatives to be evaluated to find the best one, but complementary techniques with different virtues suited to different examples. There are papers in preparation, and some material is available by ftp—please ask for details by email as above.

9 Using Z to reason about real time a la TLA and Unity

Andy Evans, A.Evans@leeds-metropolitan.ac.uk
Mike Hinchey, Michael.Hinchey@computer-lab.cambridge.ac.uk

Andy Evans spoke to a joint paper co-authored with Mike Hinchey and cryptically entitled "God Rest Ye Merry Gentlemen ...". It described an attempt to extend earlier work (on the use of Z in the specification and verification of concurrent processes—reported in the proceedings of FME94) in which they propose the use of a non-decreasing integer state variable to model real-time.

The idea is to specify the action of the system in Z and then extend the specification by incorporating a real-time variable; and subsequently to to consider safety and liveness properties.

They are in good company in trying to 'add time' in this way. Unfortunately, although they do mention TLA, they seem to have missed the recent paper by Abadi and Lamport (TOPLAS 16(5), pp1543-1571) in which time is represented by a REAL variable in extended TLA. That paper goes on to consider safety and weak and strong fairness.

10 Galois Connections in Local Reasoning in Control Systems

Michael Ingleby, m.ingleby@hud.ac.uk

Next we heard a spirited presentation from Michael Ingleby of Huddersfield and BR Research. This work is derived from the study of (potentially very large but) finite railway routing problems. In order to recover from errors, there is a requirement that safety is transitive—in the sense that, under 'normal' operation all successor states of a safe state are also safe. Solving the related graph-based problem is of exponential complexity but the application of clustering techniques often reduces this to more resonable proportions.

Constraints on available time, and the non-standard use of familiar 'fm' notation, unfortunately resulted in this interesting sortie into Universal Algebra being rather rushed and somewhat confusing. The derived results can be used to partition certain systems—in effect they isolate the 'prime' or 'orthogonal' bases from which the system in composed. These results were then applied to the railway routing problem introduced earlier.

I hope this approach can be more fully explained to the FACS community and its relevence further investigated.

11 Using inheritance to specify extendible synchronisation policies for concurrent distributed systems

M Ben-Gershon et al

After a short break for refreshments Kevin Lano spoke for the second time. He presented joint work with Ben-Gershon and Goldsack, the aim of which is to support the incremental construction of concurrent systems using ideas from DRAGOON, the synchronisation classes of McHale et al., VDM++ (sic), and extended C++.

Like the earlier presentation by Andy Evans, this work seeks to separate the various features of a specification—in this case the synchronisation concerns are 'isolated'. The main proof technique is induction over events.

The work is ongoing and the talk consisted mainly of illustrative examples.

12 Action and Control Structures

Philippa Gardner, pag@dcs.edinburgh.ac.uk

The workshop was concluded with a survey of the work being undertaken by Robin Milner and others at Edinburgh. (Robin, as many of you will know, has by now moved to Cambridge, and this survey was presented by Philippa Gardiner.)

In general terms this is the search for a uniform framework for process algebra. More specifically, the group is working on 'molecular forms' as a generic structure related to CCS, Petri nets, simple λ-calculus and the π-calculus, and which is isomorphic to term algebra.

(Incidently, it is hoped that in the near future FACS will be able to run a meeting that will deal in more detail with the π-calculus.)

Thus far, molecular forms—perceived as a general notion of hierachical 'processes'—have nice pictorial representations but no acceptable textual syntax has been found. Related work is concerned with the development of equational theories for various different variants of action calculi.

By its very nature this concluding presentation could not be very technical, and that was appropriate. Phillipa gave a very entertaining and stimulating overview of the work by herself and colleagues at DCS/LFCS that was a fine ending to most enlightening workshop; well up to the standards we have come to expect of the 'FACS Christmas Event'.

BCS Christmas Meeting on Formal Aspects of Computing Science

Michael Ingleby

The meeting was a symposium of theoretical ideas relating to concurrent systems, and heard presentations on process algebras, formalisation of concurrency in VDM and Z, interval logic, etc.. The offerings were necessarily superficial : in computing there is no clear consensus favouring one style of reasoning about concurrent actions over others, so each proponent had to proselytise before adherents to another style.

From a professional theorist's point of view, the most interesting presentation was that of T.Maibaum (Imperial) based on a categorical approach to objects/agents imbedded in systems. Most computing scientists have some acquaintance with the branch of universal algebra dealing with categories, functors and universal objects, so the topic was suited to the audience. In mainstream mathematics, the historical name for this type of algebra is 'the theory of abstract nonsense': Maibaum used abstract nonsense to approach the notion of 'emergent property' of a system of interacting agents . A system property is emergent if it results from the coordinated actions of many agents and not derivable from actions of single agents. In theoretical mechanics, emergent properties are known historically as 'cooperative phenomena' and they are manifest when a large piece of matter undergoes a phase change (gas-to-liquid, ferromagnet-paramagnet etc.). Given the scientific culture of the participants, Maibaum wisely avoided such hard examples and concretised discourse around the dining philosophers problem. This gave the discussion the flavour of a Professorenseminar on deontic logic, which the naive empiricists present found *too salty*.

My own, briefer theorising was made more difficult by having a basis in a branch of universal algebra antipodal to category theory, and virtually unknown to English-speaking computer scientists : the theory of Galois connections. This I reviewed quickly and superficially outlining a German way of using it to define the concepts emerging from a data-context, and to present these as a concept lattice. The naive empiricists present found the offering, like German food, *too bland*. In a misguided attempt to accommodate their tastes, I showed how formal concept analysis can be used to decompose a large population of concurrently acting agents into localities - small cliques sharing system resources intimately and interacting only weakly with other cliques. The naive empiricists present then found this *too spicy*, particularly when I added some of the combinatorics of proof to the mixture.

No symposium of this type would be complete without generous servings of VDM and Z, but the organisers worked hard to integrate these boring old staples into the rest of the menu. Cliff Jones (Manchester) developed VDM formulations of rely and guarantee conditions on agent behaviour - much as an American chef might assume that if pre- and post-conditions are good, then more must be better. The same sort of assumption led Lano and Goldsack (Imperial) to offer a VDM++ for the development of object-oriented systems with concurrency. Most of the naive empiricists present gobbled these up like *steak and chips*, but a few wanted specification without development, an obligatory *vegetarian* dish of Z for real-time agents with timed actions (Hinchey and Evans, Cambridge and Leeds). Actually, they were upstaged by M. Shields (Surrey) who had already modelled actions with overlapping durations in a way which allows mathematical properties of overlapping real intervals to be used in discharging proof obligations. Slipping some real mathematics into a Z specification is a bit like dropping *raw horsemeat into a dish of nut*

cutlets – a few of us were amused, but the naive empiricists didn't notice, munching on the Z regardless.

The second-most-interesting presentation came as *dessert*, and just about everyone enjoyed it. Perhaps this was because it was served by Philippa Gardner (Edinburgh) who is much prettier and better dressed than Tom Maibaum [Hmmm??—Ed.] but more than just a pretty face. The dessert recipe from R.Milner probably had something to do with it, too: CCS, a pinch of modal mu-calculus and an updating of the behavioural equivalences pioneered by Milner and Anderson. The organisers should have allowed more time to savour this tasty stuff.

My account of the meeting omits some significant but unadventurous offerings which have already been described by Ann Wrightson (Huddersfield), official BCS taster for the naive empiricist tendency. There was the computational equivalent of tripe and onions and a lot more besides. Only someone from a tripe-eating region like Normandy or the industrial north of England could convincingly fake enjoyment of this. But in true Christmas spirit, Ann consumed everything that was put in front of her, showing the determination of a true north-countrywoman to eat it if it is free and whinge afterwards. Exemplary conduct of this nature should be mentioned in dispatches: she took enormous risks with her figure to put FACS into the guidebooks.

Postscript, on Wishful thinking...

What faith is placed in formal aspects of computing! I was recently reading some papers on traceability of requirements, and came across this quotable quote in a paper by Gotel and Finkelstein published in the proceedings for Requirements Engineering '93 (publ.IEEE).

> **Requirements traceability** *refers to the ability to describe and follow the life of a requirement, in both a forwards and backwards direction (i.e. from its origins, through its development and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases).*
>
> ...
>
> Any problems [in post-requirements-specification traceability] are an artefact of informal development methods. These can be eliminated by formal development settings, which automatically transform an RS [requirements specification] into an executable, and replay transformations following change.

Putting that last paragraph in the context of the preceding definition, and putting the whole lot into the categorial framework for compositionality used by Tom Maibaum in his workshop talk, what do I get? Well, it looks like a working assumption that requirement fragments *and their interrelationships* can be traced, forward and back, right through all layers of development to a working program, provided you use some magic called 'formal development methods'. That is, that there is nooooo problem in having a merry chain of functors embodying the traceability of requirement structures step by step through to the behaviour of the machine instance fulfilling the requirement in action. *And* that the structure given by these functors is not just there to trace requirements, but directly governs the animation (= implementation?) of the RS.

Wow!

Engineering Models for Software Development

1 Introduction

This is a report on the talk given by Tom Maibaum at the 1995 FACS AGM, from my (Ed) notes; comments and questions from the audience are in [brackets].

2 Context

Have Formal Methods had their chance and muffed it? Quite a few people appear to think so, first in the UK, and also now in Europe. One result of this is that eg there's a growing feeling that you can't *say* you'll be using category theory in a research proposal any more - you *will*, of course, but the focus expected has shifted in the direction of industrial relevance, so you dress it to suit.

A very basic question raised by this is: what is/should be the role of mathematics? Comparison with other branches of engineering, concerning their view of mathematics and how it is used, leads to the conclusion that we in software engineering haven't yet developed a view of 'our' mathematics which makes it *accessible to engineers* in the same way that other disciplines have done.

On the issues of image and takeup, perhaps we can learn from comparing formal methods with other techniques which *have* been taken up enthusiastically, eg O-O. This apparently grew up from nowhere and succeeded overnight (though there was actually a lot of predecessor work to the currently fashionable approaches, from the 50s on, lots of it academic). [And the take up was not always wise - my sole experience of redundancy came from the failure of a project which decided to use the new O-O/GUI technology relying on early (1989) PC application development tools and lots of handcrafted glue. There must have been more like this, and I guess there have been some FM instances too. An interesting question is why O-O appears to have survived that stage without a bad name...? Ed] Are academics perhaps studying the wrong problems? Quite likely, but the problem lies elsewhere.

3 Engineering and Mathematics

Nearly all engineering students hate maths, and like it when they finish their studies and don't have to do it any more! [Question: UK only?] More UK/US, but quite true overall, even in Germany. *But* I can't imagine other engineers working without a very solid *foundation* in the relevant maths - yet software engineers do just that. Degree programmes in computing often don't have much maths, and there are even still companies who take on graduates in other disciplines entirely, to train as software developers. [Question: What kinds of software developers are you meaning here - eg do you include a business analyst or similar?] The software developers I mean are those that other engineers would say were engineers.

Engineering is based on scientific theory, and on mathematics, in different ways. Scientific theory underpins engineering methods in that scientific theories are models of physical phenomena, and engineering deals with physical phenomena. These theories are expressed using mathematics. But most engineers' use of maths doesn't happen like that. You may in theory be dealing with, usually, a set of differential equations, but in practice you use a *standard model* which is an encapsulated representation of some maths which is known to be effective in particular situations. They

are *simplified* representations which are valid in the usual domain of use. Systems are designed to behave like such ideal models - the model is *not* an empirical description of its behaviour.

Manipulations of engineering models are simple and concrete; a model uses an intuitive representation of a physical system which is adequate for the purpose required. (Example: the use of 'poles' to describe a class of mathematical functions used in control engineering; practical design discussions often involve highly concrete discussions based on 'moving the poles' to get the right characteristics.) A clue here, to be taken up later, is the similarity of such uses of models to the idea of *abstract interpretation* in computer science.

Engineers generally are skilled in choosing and using the right model; this is a key skill of engineering design. Nowadays, CAD systems support this by providing full mathematical capability behind the model. In software engineering/formal methods, we don't have such models available - where we're using maths, we're often doing the equivalent of trying to use the system of differential equations of the theory directly each time.

Do other engineers have any similar problems to SE? Yes. Instrument engineers have much the same problems *in the large* as software engineers; their small-scale problems use well-defined models, and are pretty routine, yet when they need to put together a lot of sensors, have communication between them, etc., they run into similar problems to those we know on large systems of software.

There are also general problems in the mathematics of systems - the maths needed to build a bridge is pretty well known, but building a transportation system, eg for a city or region, with all its various bridges, signals, roads of various capacities and speeds, is still not well understood. Engineers in general do not have good models to support development in the large/complex.

4 Methods and Models

In SE, it's been suggested that to overcome some of the problems of using formal methods, use *rigour* instead of formality. The problem with this is that it needs greater, deeper mathematical skills to really work rigorously without the support of a formal system.

We also need to look at informal methods. The problem is that they *are* informal, and connexions between them and formal concepts have not been properly made. And this last is why despite their similarity in some respects, these are not engineering models, because they lack the connexions that engineering models need to the underlying science and mathematics of the field. But don't forget that the informal methods we have are useful, where they are useful, because they are *meaningful* to the people using them, and this is good, though not good enough.

We need something else which is more akin to real engineering models. So, for SE: what science? what mathematics? and how to connect them? what kind of cookbooks do we need?

What mathematics? - here it is clear that logic provides for SE the equivalent of continuous mathematics in other engineering disciplines. Just as differential equations pop up everywhere, so theories/specifications in SE.

What science?—not so clear! Models of computation? (and if so, of what kind? domain theory? chemical abstract machine?) ... Is O-O the beginnings (the witchcraft before the science) of a scientific theory of development in the large? ... [Cognitive science?]

Do we already have any genuine engineering models? There are a few candidates, things which are used over and over again in a well-founded way. One is type theories and type checking; another is relations (as used in relational databases).

'Patterns' as emerging in the O-O world, and 'Software Architecture', may be the beginnings of other models.

Other models may well grow from what we have now as reusable paradigms for modelling interrelationships. These are structures which are useful in a number of contexts, appear to be independent of the various formalisms and contexts in which they are used, and also look to be more generally applicable. Here are four areas looking promising in this way.

Synchronization and interaction as (co)limits. There followed a discussion of this approach centred on the dining philosophers. (And in case you've wondered what the small curly

print is in the middle of the table on Tom's usual slide for this, it says "Manduco ergo cogito ergo sum".) [See the report of Tom Maibaum's talk at the Xmas workshop 1994, for more on this approach - Ed] This is a candidate for a scientific paradigm of what 'configuring a system' means, which could be a basis for engineering models for particular purposes. Diagrams alone are *not* sufficient; you need further information to support them.

For example, specifying using objects can use the idea that an object embodying a computation is made up of computation plus interfaces (methods). This can then be developed as a model where a computation is a composition of a kernel-body and the interfaces available to it, with links between objects provided via the interfaces (*some* constructions can then collapse a link via an interface to a link direct kernel to kernel, to allow simpler connection within an aggregate). In the category of object-specifications, the system then appears as the colimit of its configuration diagram.

Structure for systems via clients, servers and interfaces.

Reification/refinement/implementation understood using interpretations and conservative extensions.

KIDS is the only successful transformation system I know of. It takes a particular view of transformation systems, and uses some knowledge-base principles to structure the transform set based on problem classes. There is no interactive theorem-proving, but things are identified which can be tackled by the system's proof-tools. From the US DoD involvement in the project (inspired by their large logistics problems) the domain is scheduling and planning. They have produced some significant programs using novel algorithms, for this domain.

The thinking here is in terms of *theorems and interpretations* rather than the traditional fare of transforms where programs are terms. A new system 'Specware' is also being developed, with category theory explicit as the underlying theory for modularity and composition.

Views and perspectives via *something* - category theory again? There needs to be some understanding of them in terms of engineering models, or modules of large systems. [I think *situation theory* is the key here, probably; you think of a system as made of human and machine agents perceiving and acting in situations (paying attention to the information flow between situations), and the theories/specifications describing the agents are then constructed from this picture - Ed]

5 Conclusion

All this is promising, but needes to be taken much further to provide true engineering models for SE. The key feature of an engineering model is that it takes an abstract view of only certain aspects of an artefact, and allows simple, concrete-feeling use of that abstract view, in a way that in SE terms supports an abstract interpretation back into the full theory. We have various things which do parts of this, but need to develop this approach much further to support sound SE.

[There was quite a bit of lively discussion at the end, which I didn't note - probably because I was too busy talking myself! Thanks to Tom for an interesting and thought-provoking contribution to the AGM - Ed]

Formal Methods and Object-orientation: A Historical Perspective

K. Lano (kcl@doc.ic.ac.uk)

Formal methods and object-orientation are two of the most significant current ideas in software engineering, with the latter in particular having moved successfully from research into industrial practice. Object-orientation and formal techniques are highly compatible in a number of ways:

- object-orientation encourages the creation of abstractions, and formal techniques provide the means to precisely describe such abstractions;
- object-orientation provides the structuring mechanisms and development disciplines needed to scale up formal techniques to large systems;
- formal techniques allow a precise meaning to be given to complex object-oriented mechanisms such as aggregation or statecharts.

In this article we will give some of the history behind the interaction of these two areas, and the current state of the field.

1 The Origins of Object-orientation and Formal Methods

Object-orientation first appeared as a programming concept in the Simula language of the 1960's, which originated the concept of a class, inheritance and methods. The explosion of interest in object-orientation came however in the 1980's, with the development of Smalltalk and then C++, and the widespread adoption of object-oriented programming in industry (even COBOL now has "object-oriented" dialects).

Similarly, formal methods have been researched and applied for over 25 years, having their origin in the work of Dijkstra and Hoare on program verification, and Scott, Stratchey and others on program semantics. A major step was the development of mathematical languages for program specification, of which the most established are the Z and VDM notations. The Z language was initially created by J. R. Abrial (his paper "Data Semantics" from 1974 could be considered the birth of this language), and expanded and industrially applied by many others in the 1980s, with the Programming Research Group at Oxford being the focus of much key work. The VDM language was initially developed by the IBM research laboratories in Vienna, and the University of Manchester group led by Cliff Jones became the focus of research and tool development efforts, leading to the Mural toolkit, and ISO standardisation.

Algebraic notations such as OBJ, PLUSS and LARCH were also developed, but remained less widely used in industry than the "model-based" notations of Z and VDM.

The development of object-orientation has not been entirely separate from the development of formal techniques. Indeed, both formal methods and object-orientation take inspiration from the concept of abstract data types, and the Eiffel language has adopted many ideas from formal specification (such as invariants and pre and postconditions).

However the main work in combining formal and object-oriented methods has taken place in the 1990s, in two main areas:

- the addition of object-oriented structuring to formal notations such as VDM, Z and algebraic languages;

- the addition of formal notations and concepts to object-oriented analysis and design methods.

These will be considered in the following sections.

2 The Application of Object-orientation to Formal Methods

The limitations of Z and VDM for specifying large systems in a modular manner led to various investigations aimed at adding structuring mechanisms to these languages. For example, [20] proposed adding a "chapter" mechanism to decompose Z specifications into modules, and [13] identified a similar (object-based but not object-oriented) approach using a combination of HOOD and Z. Abrial and others at BP Research also developed an object-based extension of Z, the B Abstract Machine Notation [10], which has had major applications in safety-critical transport systems [3]. The SmallVDM language [17, Chapter 10] combined an object-based structuring style derived from Smalltalk with VDM notation.

Researchers and practitioners were led in the direction of object-oriented mechanisms because these seemed to naturally complement model-based specification languages [8]. The first fully object-oriented extension of Z was the Object-Z language [4]. This featured the key aspects of object-oriented structuring:

- encapsulation of data and operations on that data into named modules (*classes*) which also define types;
- the possibility of creating subclasses of classes via *inheritance*, and the ability to use operations polymorphically between a subclass and its superclasses which contain a particular operation;
- the ability to use instances of a class (ie, *objects*) within another class – the concept of class *composition*. The class C containing instances of class D is termed a *client* of D, whilst D is termed a *supplier* of C.

It however initially restricted composition to be acyclic (class A could not contain an instance of class B if B contained an instance of A, or other cyclic situations). Object-orientation was found to provide advantages in terms of ease of education [22] and in terms of convergence with domain descriptions.

Other object-oriented extensions of Z were also developed in 1989–91, such as Z^{++} [14], MooZ [19], OOZE [1] and ZEST [8]. Of these, OOZE is distinctive as the only one to be based on an algebraic specification approach, using OBJ and FOOPS as its foundation but with a Z-like syntax. In the VDM world, the Fresco language was developed as a means of providing formal specification and verification support for Smalltalk development [23].

A number of applications of these languages were carried out, in the telecommunications and process control fields. The work of BT using ZEST is of particular note in this respect. The scope and power of the languages were extended as more challenging application areas were investigated. Thus concepts of object identity and cyclic composition structures were introduced in Z^{++} and Object-Z [15]. The VDM^{++} language represented a significantly more ambitious approach, based on ideas from VDM, Smalltalk and the DRAGOON Ada extension [2], it included Ada-style concurrency and synchronisation features, traces (as in CSP), and real-time aspects taken from the ideas of Hayes [11]. As with Fusion and ZEST, the involvement of a major company (in this case CAP Gemini) provided direct routes to significant industrial applications [9]. The toolset for VDM^{++}, called *Venus*, extends an OMT CASE tool (*Lov*) with facilities to translate between OMT object models and sets of VDM^{++} classes, and with facilities for the writing, analysis and animation (via prototyping) of VDM^{++} specifications.

Real-time and concurrency aspects were incorporated into Z^{++} in a more declarative manner via the use of real-time logic [16].

A significant drawback with the use of object-oriented structuring is the degree to which this complicates reasoning about specifications. Until recently, there were no formal semantics for an object-oriented specification language. This has now been remedied, with an axiomatic semantics and reasoning system being provided for Object-Z [21] and Z^{++} [16], and a denotational semantics for MooZ [18].

3 The Application of Formal Methods to Object-orientation

Whilst this work on formal object-oriented languages was being carried out, other groups were investigating the converse process, of how formal specification can supplement object-oriented development, or help to clarify the semantics of object-oriented notations and concepts.

Examples of such work include formalisation of the OMG's core object model using Z [12], formalisation of the OOA notation using Z, and the Fusion [5] and Syntropy [7] methods. The work on OOA uncovered apparent weaknesses in its semantics, such as the lack of any relationship between the set of object identities of instances of a subtype to those of its supertypes.

Fusion grew out of work by Coleman, Dollin and others at Hewlett Packard Labs on formal enhancement of object-oriented notations such as statecharts [6]. It provides semi-formal notations (structured English) for operation pre-conditions, post-conditions and invariants, and uses the OMT object-model notation and Booch object interaction diagram notation, together with other notations, to provide a rigorous method for sequential systems. It is now used quite extensively within Hewlett Packard, across 15–20 divisions and in areas such as printer technology, network management software and test software. Other companies have taken up Fusion in the USA and Europe.

Syntropy is a more recent method in the same direction. It combines the statechart and object model notations of OMT with object interaction diagrams, and allows the use of Z notations on these diagrams to specify pre and post conditions, invariants and constraints. A partial formal semantics for the notation is also provided. Unlike Fusion, a treatment of concurrency is provided. Syntropy grew out of many years of industrial experience in consultancy, and has already been commissioned for use in financially critical applications in the UK.

4 Conclusions

The development of highly integrated formal and object-oriented methods, with supporting tools, appears to be an effective way of introducing formal methods into industry. Tools such as Venus allow users of object-oriented methods to extend the rigour of their developments all the way to fully formal and proved specification and refinement steps. The trend towards greater rigour in object-oriented languages and methods also makes it easier for formality to enter industrial practice.

The great number of object-oriented variants of formal languages may seem to be a hinderance to their uptake, however these languages share many common concepts and features, so that learning one does not imply a great retraining cost if an alternative language becomes used instead.

In the UK there is a working group, EROS, involving both industrialists and researchers, which holds regular discussions on formal object-oriented approaches, including the standardisation of formal object-oriented languages. A number of conferences and workshops on the topic have also been held, with the ECOOP conference being of particular relevance. The 1993 BCS FACS Christmas meeting was on this topic. The proceedings of this meeting are to appear in a revised and updated form as a book in the Springer-Verlag FACIT series in 1995, and gives a useful overview of current work in the field. The book [17] gives a collection of case studies in various formal object-oriented notations, and there are projected books on VDM^{++} and on formal object-oriented development in general.

References

[1] A J Alencar and J A Goguen. OOZE: An object-oriented Z environment. In P America, editor, *ECOOP '91 Proceedings*, volume 512 of *Lecture Notes in Computer Science*, pages 180–199. Springer-Verlag, July 1991.

[2] W D Atkinson, J P Booth, and W J Quirk. Modal action logic for the specification and validation of safety. In *Mathematical Structures for Software Engineering*. The Institute of Mathematics and its Applications Conference Series 27, Clarendon Press, 1991.

[3] J Bowen and V Stavridou. Safety-critical systems, formal methods and standards. *Software Engineering Journal*, 8(4):189–209, July 1993.

[4] D Carrington, D Duke, R Duke, P King, G A Rose, and G Smith. Object-Z: An object-oriented extension to Z. In *Formal Description Techniques, II (FORTE'89)*, pages 281–296. North-Holland, 1990.

[5] D Coleman, P Arnold, S Bodoff, C Dollin, H Gilchrist, F Hayes, and P Jeremaes. *Object-oriented Development: The FUSION Method.* Prentice Hall Object-oriented Series, 1994.

[6] D Coleman, F Hayes, and S Bear. Introducing objectcharts or how to use statecharts in object-oriented design. *IEEE Transactions on Software Engineering*, 18(1), January 1992.

[7] S Cook and J Daniels. *Designing Object Systems: Object-Oriented Modelling with Syntropy.* Prentice Hall, Sept 1994.

[8] E Cusack. Object-oriented modelling in Z. In P America, editor, *ECOOP '91 Proceedings*, Lecture Notes in Computer Science. Springer-Verlag, 1991.

[9] E Durr and E Dusink. The role of VDM^{++} in the development of a real-time tracking and tracing system. In J Woodcock and P Larsen, editors, FME '93, Lecture Notes in Computer Science. Springer-Verlag, 1993.

[10] H Haughton and K Lano. *B Abstract Machine Notation: A Reference Manual.* McGraw-Hill, 1995.

[11] I J Hayes and B Mahony. A case study in timed refinement: A mine pump. *IEEE Software*, 18(9), September 1992.

[12] I Houston. Formal Specification of the OMG Core Object Model. Technical report, IBM UK, Hursely Park, 1994.

[13] P L Iachini and R Di Giovanni. HOOD and Z for the development of complex software systems. In *VDM and Z, VDM 90*, volume 428 of *Lecture Notes in Computer Science*, pages 262–289. Springer-Verlag, 1990.

[14] K Lano. Z^{++}, an object-oriented extension to Z. In J Nicholls, editor, *Z User Meeting, Oxford, UK*, Workshops in Computing. Springer-Verlag, 1991.

[15] K Lano. Refinement in object-oriented specification languages. In D Till, editor, *6th Refinement Workshop*. Springer-Verlag, 1994.

[16] K Lano. Reactive system specification and refinement. In *TAPSOFT '95*, volume 915 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.

[17] K Lano and H Haughton. *Object-Oriented Specification Case Studies (First Edition).* Prentice Hall, 1993.

[18] T Lin. A Formal Semantics for MooZ, PhD Thesis. Technical report, DI/UFPE, Recife/PE, Brazil, 1994.

Cont. on p.20

How did software get so reliable without proof?

Notes from a lecture by C. A. R. Hoare given at South Bank University, 24/11/94, by A.M. Wrightson

1 Surprising reliability...

One starting point comes from Dijkstra about 1968: how to tell a chip is still working? (i.e. testing reliability after installation). No one is interested in that any more - they do keep working. Also, no one runs a system that long, keep replacing them - so that problem has gone away. [I suspect it will come back with eg processors embedded in car control - AMW]

A second starting point: decrease in reliability with increasing size of software. Problem originally posed in context of programs around 10k lines, known to be unreliable because of bugs etc.—and fear of error rates potentially exploding with greater size. Again, the predicted fortune has not come to pass. We have large software, in daily use—and these are of the more difficult kinds to build right, and are also subject to evolutionary changes. Yet they do in fact work, with adequate reliability to deliver services, and with very few accidents caused by software error.

Interesting picture from a recent survey (McKenzie) on computer related deaths. Yes there are some, not many (a few 100s) deaths attributable to systems involving computers. For most of these, the survey shows the mistake occured before building the software, typically at requirements gathering or formulation, or are due to evolution of the environment of operation of the system. For example, warmaking software used for peacekeeping, with different priorities and parameters implied. Deaths from software errors are very rare, probably in tens not hundreds. Best documentation is on errors in settings on X-ray treatment machine, even here the picture is unclear.

So even very large (26M lines) programs can run continuously and reliably, for example telephone switching software, while being under development all the time. These programs do contain errors, and in principle are safety-critical, at least commercially critical. (When an exchange on US eastern seaboard went down, the operating company lost a few billion dollars.) There is more surprise in their success than in their failure.

Big software such as telephone systems is mostly constructed by physicists, engineers, electricians, without computer science concepts. NO knowledge of proof, using developed intuition about how programs will work. There is much intuition about dynamic operation of physical systems used by engineers in visualizing design decision consequences for physical artefacts. The same kind of intuition develops in programming. The lack of specifically spatial intuition requirements is an advantage in writing programs.

Program testing has improved, eg by simulation of the operating environment. Testing is very effective in leaving a program with only rare bugs (An IBM fault report analysis showed the likelihood of a second customer finding the same fault in mature software as about 1 in 5Kyr. That seems to be so—I don't know why. There's much the same experience in hardware—if it passes initial tests a device will survive well in use. Software is *more* reliable in use than it has any right to be! Testing is very efficient on personal workstations, and lots of testing is done. Amazingly, customers will *pay* for the privilege of finding bugs—β-test—no other branch of engineering managed to put this one over on their customers. (Reference here to a short story by E. M. Forster, "The Machine Stops".) There are many small faults. They *can* have nasty consequences. Small symptoms may be of larger problems [and emergent properties? AMW].

2 Observations from software engineering practice

Most of the code of the large programs (operating systems, GUIs, telephone exchanges) is put in to recover from operator or hardware error. If there is a software error it will have similar symptoms, and will be recovered to a safe system state. The transaction may well go wrong, but the operator can recover that too. Also, the operator never knew it *was* a software fault: "you must have pressed the wrong key"—how many 'operator errors' are latent software errors, or interaction effects?

The original assumption/conjecture that errors are cumulative is wrong, and in practice the reverse is the case, because the bigger systems have evolved, by layered construction, over a long time, with time to settle in at each stage. Lower layers 'learn', adapt to how they are used—higher levels only use lower levels in certain ways, and the normal process of bug removal leads to 'machine learning' over the whole structure.

Software is thus *as reliable as its highest layer*, this is the easiest to write, so can deliver high reliability albeit on an insecure basis.

An analogy from a natural system—genetics, our only good example of a discrete natural system, and it has similarities. In principle, a one-molecule change could lead to didaster, yet there is considerable resilience in the genetic code plus how it is expressed. Again this has come about throug evolution, through may layers, adding new material, using old subsystems—and there is dead code, no-one knows why it's there or what it does.

Software developers have had it easy for 15 years. Most software can ignore resource limitations. Much has been too expensive in resources for the hardware when first delivered—then the hardware caught up, again and again. The factors involved are incredible–a physical analogue would be meeting a need for 'a tap which doesn't drip' by a device weighing 3 tons (and letting through very little when on).

An interesting technique from the telecoms industry has a scavenging process loose in the system, known as a 'software audit'. It looks at activation records, data structures etc. and uses a test of plausibility on the data (right parameters, ranges of values, relationships between them). If there is anything suspicious, the structure in question is reinitialized. Possible analogy here with human immune system. Could be called 'Run-time testing of assertions'—but the people involved don't know the concept. An improvement in length of operation without failures, from a few hours, to a few years, reported with this technique. Using assertions in the development would protect against these errors—wouldn't it? (and there are some companies who don't need this technique...).

Another phenomenon: programmers in large development teams write lots of code to protect their own modules against external changes, and against misuse. These include checks for errors in the code which uses their code, and in the code it uses (i.e. layers above and below). This is defensive programming, essential for developers in large teams. This is a good contribution to software safety—what proportion of the code in really big systems is there just to do this?

Another technique widely practised is design reviews, code reviews, gatekeepers for release of code into a wider environment. These reviews are remarkably effective in all engineering disciplines, and are effective on code in particular.

3 Some hope from research (since 1968)

Modern programming languages do contribute to reliability, because of their simplicity—none of the telephone code looked at used jumps. High-level languages *with types* (eg specific to a domain) are easy to read accurately, because you can rely on the types, and this confidence is essential to really understanding code. For example, there is a received wisdom that you can't maintain systems of over 8M lines of C - is this why?

Dijkstra's famous 'goto' letter caused a furore—but now everyone does it. Strict type checking will follow (the basic research for this happened *before* Algol68!). It can take *very* long for research in this area to have an effect. Much work on O-O since 1967 has been inferior and obscured the

issues, compared to early work such as that which resulted in Simula67. Assertions (Turing 1947, Hoare 1997...) *do* work, can be useful in code reviews—review based on conviction rather than traces and examples would be more secure.

So what use is research? The future direction of CARH's research will explicitly go basic/scientific, making it clear what our understanding is of programs of various kinds, and of relationships between them. This to include not only procedural code, functional languages, logic programs, but also eg SQL, MATLAB, spreadsheets, visual basic.

How to understand the relationships between all of these (in principle, as compare-and-contrast, and in practice as interaction. What should a method look like for developing applications involving all of these? What aspects are independent, of which if these, and how to specify it? When computers are like this, how to see that interfaces are secure and don't cause problems? Surely there *is* an immediate practical engineering interest, but the research needs to be *in principle*, with simplification of models—and the actual motivation is curiosity.

[Interesting practical parallel here to CLiCS project concerns with integration of various underlying theories - AMW]

Looking now at what actual users of computers are doing, there is a noticeable contrast between small-scale use of programs and applications, and large projects. The scale contrast is similar to a plank bridge vs the Forth bridge. A view amongst students which inoculates against the approaches needed for Forth Bridge type projects is encouraged by playing with small-scale examples and packages.

Most people making big projects *have* to use a language which will be there in 2050AD—which C will be, whatever you think of it otherwise—a change of language will moreover be unthinkably expensive and risky. So, we need practical ways ahead, to get safe ways of using C, or getting high-level languages which translate securely into C. See paper [by CARH]: 'The High Cost of Programming Languages' - the cost is not quantifiable.

4 Discussion

Audience: In other engineering disciplines, new ways have eventually taken over, and even driven out old equipment.

CARH: A disadvantage, and a problem peculiar to software, is that new concepts need to propagate throughout people using it. In other areas, 1 or 2 people understanding a formula can bring about a process or procedure which many people can follow. In computing, lots of people have to learn concepts and techniques, and integrate them into practice, for a change to stick. Experience now is that efforts such as (part of) IBM using the B-tool can pay locally, but you get problems of acceptance later on—when in theory you should be getting the proper development payoff—what happens is that people don't like it because it's not a 'normal' product

Another thing to remember is that the crash of the US telephone system [mentioned earlier] was *caused by* (too much) defensive code. A localized overload led to a cascade of offloading which overloaded other modes in turn, leading to the crash. Moral: Analysis of recovery patterns should happen early, and also as an iterated check throughout a development—some basic mathematical techniques applied in this way would have prevented this situation.

Computers deliver such good value for money that their deficiencies are tolerated—and there is no sign of competition changing this.

Audience: [another question]

CARH: People don't spend less on computing as it gets cheaper, they use more and more.

There is a software engineering question to be answered here—Why does software get so large? What's defensive? What's dead? What's (needlessly) copied? It's a surprise that no-one is asking this question, and measuring carefully to get a good answer.

Audience: Is the increase linear, or what?

CARH: No-one observes actual software to see how it works. No-one removes code—so if stable, is linear, but depends on number of programmers.

There was a recent PICT group meeting on learning from disasters. London Ambulance project—it was crazy from start to finish, yet it's easy to imagine how it happened. Not *software* faults as such, perhaps a lack of humility, lack of management conception of the project as well, in being able to define an achievable thing. Perhaps "Never do anything with more than 6 months delivery" a good maxim for interactive systems. Also, ensure communication and connection between machines and databases *at places where things will need to adapt.* The 'legacies' in the minds of users are more permanent and dangerous than anything the other side of the glass screen.

Tim Denvir: As we understand some development, eg the transition from assembler to high-level language , then a bit later, automate it, and change from selling machine time to selling programs. More recently, 20 out of 25 projects at Praxis delivered feasibility trials or specifications, not programs. So software engineering is always "difficult", because the bit we are doing now, as the task for people rather than machines, is the bit we don't understand well enough to automate. And what we don't understand at present is requirements.

CARH: Agree.

Audience: Will O-O make systems more reliable?

CARH: No. For example, systems in C++ can run for a while, then crash. C++ can't have good garbage collection because of lack of type checking - you can overwrite pointers without deallocating the associated store. Pointers are as dangerous as jumps.

(From p.16)

[19] S R L Meira and A L C Cavalcanti. Modular object-oriented Z specifications. In *Z User Meeting 1990*, Workshops in Computing, pages 173–192. Springer-Verlag, 1991.

[20] A Sampaio and S Meria. Modular extensions to Z. In *VDM and Z*, volume 428 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

[21] G Smith. A logic for Object-Z. In *Z User Meeting '95*, Lecture Notes in Computer Science. Springer Verlag, 1995.

[22] P A Swatman. Using formal specification in the acquisition of information systems: Educating information systems professionals. In *Z User Meeting 1992*, Workshops in Computing. Springer-Verlag, 1993.

[23] A Wills. Capsules and types in Fresco: Program verification in Smalltalk. In P America, editor, *ECOOP '91 Proceedings*, volume 512 of *Lecture Notes in Computer Science*, pages 59–76. Springer-Verlag, 1991.

British Computer Society - Formal Aspects of Computing Science

BCS - FACS

BCS-FACS Christmas Workshop

Imperial College, London
18/19 December 1995

SEMANTICS

Call for Papers

The Xmas Workshop of the Formal Aspects Special Interest Group of the British Computer Society is a well-established event, which has traditionally served to the FACS community a varied and interesting diet of topics related to "formal aspects".

The aim of this workshop is to cover a wide range of current work in Semantics, including reviews, tutorials, and reports of work in progress. The two principal foci are Semantics of Computation, and Semantics as Meaning. Understanding computation could be seen as the longest or largest endeavour in real-world semantics in computing, and this provides a strong connection between the two aspects of the programme.

Semantics and Logic of Computation is of course currently very active and fruitful, especially with the input from CLiCS and other European funded projects. Keynote presentations will convey to the FACS audience substantial aspects of current work, and this will be complemented by shorter presentations on more specialized areas and work in progress. WE INVITE abstracts of presentations which are reviews, tutorials, or reports of work in progress, and hope to have all these kinds in the programme.

To complement this focus, WE INVITE AND ENCOURAGE contributions from a wide range of semantic enterprises concerned in some way with meaning in real-world or system development terms. Examples would include work concerned with data, interaction, real-time, application domain concepts etc. If the response permits, then a panel discussion involving a diversity of semantic outlooks will be a feature of the second day.

Tentative list of invited speakers includes Luke Ong, Andrew Pitts, and Alan Dix.

See overleaf : Information for Contributors

Information for Contributors:

ABSTRACTS (maximum of 2 pages) suitable for distribution at the workshop should be submitted to either organizer to arrive not later than the end of September. Authors will be informed whether their submission has been accepted by the end of October.

Co-organizers:

Kevin Lano
Department of Computing
Imperial College of Science Technology and Medicine
180 Queen's Gate
London
SW7 2BZ
UK
Tel: 0171 594 8246
Fax: 0171 581 8024
Email: kcl@doc.ic.ac.uk

Ann M Wrightson
School of Computing and Mathematics
University of Huddersfield
Queensgate
Huddersfield
HD1 3DH
UK
Tel: (01484) 472758
Fax: (01484) 421106
Email: scomaw@zeus.hud.ac.uk

First Announcement and Call for Papers

BCS FACS Group
The Seventh Refinement Workshop
Theory and Practice of System Design

3–5 July 1996
University of Bath, England

Formal techniques constitute the foundation of a systematic design. They have beneficial applications throughout the engineering process, from the capture of requirements through specification, coding and compilation, right down to the hardware which embeds the system into its environment. The use of refinement techniques in computing systems is increasing rapidly, as it provides the theoretical foundations for the design of reliable systems.

The workshop is devoted to considering the problems and the solutions in computing system design. Particular emphasis will be given to examination of how well formal refinement techniques for design, analysis and verification serve in relating theory to practical development of real time systems. Papers are invited which address theoretical or practical issues in the development and/or application of formal system specification, design, refinement and implementation. The organisers are aiming for a balanced mixture of theoretical and practical material, drawn from fully refereed papers as well as invited presentations. Some particular themes that might be addressed by authors are as follows:

- Requirement capture and analysis of safety-critical systems.
- Refinement techniques for real-time systems.
- Methods for large scale software development.
- Formal specification and design.
- Hardware/software codesign.
- Parallel software engineering
- Tools and techniques for parallel system development.
- Industrial case studies in the use of formal development techniques.
- Education and training in formal methods.
- Issues relating to the usability of formal notations.

Contributions need not be limited to these themes, but papers with clear industrial relevance are particularly encouraged.

The proceedings are likely to be published by Springer-Verlag as part of the BCS Workshop series. Details of the cost of the workshop and a provisional timetable will be distributed at the beginning of March 1996.

Depending on the availability of suitably mature (and formally based) software engineering tools to support refinement, a tool demonstration may also be possible. Anyone wishing to mount such a demonstration should contact the Workshop Chairman and provides a two page summary of the tool.

Dates For Authors

Authors are invited to submit full (draft) papers to the Workshop Chairman at the address below. The following dates have been set:

Submission of papers	15th November 1995
Notification of acceptance	15th January 1996
Camera-ready copy of papers for publication	15th April 1996

Organizing And Programme Committees

Mike Ainsworth	Praxis plc
John Cooke	Loughborough University
Roger Hale	SRI International
He Jifeng (Workshop Chairman)	Oxford University
Jeremy Jacob	York University
Victoria Stavridou	Royal Holloway and Bedford New College
David Till	City University
Peter Wallis (Local Organizer)	Bath University
Hussein Zedan	Liverpool John Moores University

Local Details

The workshop will be held at the University of Bath from 3rd July to 5th July 1996 with meals and accommodation available on the nights of 2nd to 5th June. The cost of the workshop is still to be decided and will be inclusive of proceedings, lunches and dinner.
The University is located in landscaped grounds on a plateau overlooking the Georgian City of Bath. The city is well suited for ease of access by road, rail or air.

Further Details From:

He Jifeng
Programming Research Group
Oxford University Computing Laboratory
11 Keble Road
Oxford OX1 3QD, England
Tel: +44 1865 283513 Fax: +44 1865 273839
Email: jifeng@comlab.ox.ac.uk

or

Peter Wallis
School of Mathematical Science
University of Bath
Claverton Down
Bath BA2 7AY England
Tel: +44 1225 826002 Fax: +44 1225 826492
Email: pjlw@maths.bath.ac.uk

BCS - FACS

CALL FOR PAPERS

Formal Aspects of The Human Computer Interface

BCS FACS Workshop

Sheffield Hallam University, Sheffield, UK.

10th - 12th September 1996

The field of Human Computer Interaction is becoming increasingly relevant owing to the growing number of unskilled users making greater demands upon interactive systems in a wide variety of contexts. In addition, the nature of interaction is developing rapidly with the incorporation of advanced graphical and multi-media techniques, and the potential for multi-modal interaction. Within this context, user requirements are making high demands upon the notations involved in formally specifying and analysing requirements for interactive systems. The use of formal methods and notations within software engineering has provided considerable benefits to improve the quality of software development. The aim of this workshop is to demonstrate and examine the benefits and limitations of such techniques within the analysis, specification and design of the human machine interface, and usability requirements in general.

Contributions are sought that focus upon either basic research addressing modelling techniques suitable for accommodating advanced interface requirements or the application of formal techniques within interface design. Examples include (but are not limited to) the following:

- Modelling human computer interaction (especially multi-media and multi-modal aspects of interaction)
- Modelling temporal aspects of interaction
- Unifying system and user models
- Improving the effectiveness, usability and acceptance of formal techniques
- The formalisation and analysis of usability requirements for interactive systems
- The specification of Graphical User Interfaces
- The application of formal techniques in interface design, refinement and verification
- The application of tools

The workshop is organised by BCS - FACS (Formal Aspects of Computing Science). Accepted papers will be invited to present their work, and it is planned that proceedings will be published by Springer-Verlag.

Panel: Proposals that focus on current controversies within a workshop topic are encouraged. Preference will be given to panels that consist of members who have a significant presence in the field and present a diversity of views.

Submission: Submissions should not exceed 6,000 words in length, typed doubled spaced, they should include a short abstract, a list of keywords, and full contact details for the principal author. Six copies should be sent to: *Dr. Chris Roast, Sheffield Hallam University, Sheffield, S1 1WB, United Kingdom.*

Attendance: The workshop will be suitable for researchers interested in the application of formal methods, and industrialists interested in the development of quality advanced interfaces.

Organisers: Chris Roast and Jawed Siddiqi, Computing Research Centre, Sheffield Hallam University, Sheffield, S1 1WB, UK (Tel: +44 (0)114-253-3768, Fax: +44 (0)114-253-3161). Further information on the workshop is available by email c.r.roast@shu.ac.uk and through the WWW URL http://pine.shu.ac.uk/~cmscrr/fahci.html.

Key Dates:
Expression of interest: 13th Nov. 1995 **Paper submission: 8th Jan. 1996**
Notification of acceptance: 8th Mar. 1996

Programme Committee:
G. D. Abowd
A. J. Dix
D. A. Duce
B. Fields
T. R. Green
M. D. Harrison
C. W. Johnson
S. Milner
P. Palanque
F. Paterno
C. R. Roast
J. I. Siddiqi
H. Thimbleby
D. Till
P. C. Wright
A. M. Wrightson


Virtual Library

Formal Methods

Z Notation

Announcement: Formal Methods and the World Wide Web

Information on formal methods, and Z notation in particular, is held as part of the World Wide Web (WWW) global hypermedia Virtual Library under the following URL (Uniform Resource Locator):

`http://www.comlab.ox.ac.uk/archive/formal-methods.html`

Please email *Jonathan.Bowen@comlab.ox.ac.uk* if you know of relevant on-line information which could be included.

Currently there are around 35 million people with Internet access and the number is doubling each year. WWW usage has been estimated to be increasing at 1% per *day*! The WWW Virtual Library formal methods page is accessed around 150 to 200 times a day from around the world (as of May 1995) and is mentioned in the September 1994 issue of *Scientific American*.

Technical information: The page may be accessed on the Internet by WWW client programs such as *netscape* or *mosaic* under X windows and *lynx* on ASCII terminals under Unix. Client programs are also available for use under MS-Windows on PCs and on Apple Macintosh computers. Contact your system manager if WWW is not accessible from your computer.

WWW pages include underlined phrases which are hyperlinks to other URLs. These may be anywhere in the world on the Internet computer network, accessible via anonymous FTP, NNTP (USENET news), Gopher, WAIS, Telnet, or WWW's own HTTP protocol, using HTML, a mark-up language based on SGML. As well as HTML format, files may be in PostScript (formatted documents), DVI (LaTeX output), GIF (colour images), XBM (monochrome images), JPEG (compressed colour images, especially photographs), MPEG (moving colour images), Sun audio (sounds), etc., and may be compressed using utilities such as *compress* and *gzip*. Different formats are handled by appropriate programs on the host machine.

Jonathan Bowen, Oxford University Computing Laboratory.
Email: *Jonathan.Bowen@comlab.ox.ac.uk*.

(Part of the OUCL Archive Service.)

Notice of Conference and Call for Participation

ZUM'95

9th International Conference of Z Users

Organized by the Z User Group
Sponsored by BT, Forbairt, Praxis and University of Limerick
Supported by BCS FACS and ESPRIT ProCoS-WG
7–8th September 1995
University of Limerick, Ireland

The 9th International Conference of Z Users (ZUM'95) will be held at University of Limerick in the West of Ireland in September 1995. The conference is being held outside the UK for the first time. Limerick (Shannon Airport) is easily reached by air, with several flights per day from London (Heathrow), Leeds/Bradford and Manchester, and direct flights from major cities in Europe and the US. The following activities are planned:

Monday – Wednesday, 4–6th September 1995	Tutorials
Thursday – Friday, 7–8th September 1995	Main meeting
Saturday, 9th September 1995	Educational issues session

The following invited speakers will give presentations as part of the main sessions of the conference:

Prof. David Lorge Parnas, McMaster University, Canada
Dr. John Rushby, SRI International, USA
Prof. Jeannette M. Wing, Carnegie Mellon University, USA

The conference will also include:

• Tool demonstrations • Exhibitions by publishers • Posters or leaflets

A conference dinner will be held at historic Dromoland Castle where the after dinner speech will be given by Prof. David Gries (Cornell University, USA)

Attendees will receive a copy of the proceedings, to be published by Springer-Verlag *Lecture Notes in Computer Science*, and a special Z issue of the *Information and Software Technology* journal as part of the delegate pack. To attend, please complete and return the booking form with the appropriate remittance. We look forward to seeing you at the meeting.

For on-line information on the conference, see:

`http://www.comlab.ox.ac.uk/archive/z/zum95.html`

BCS FACS
Department of Computer Studies
Loughborough University of Technology
Loughborough, Leicestershire
LE11 3TU
Tel: +44 509 222676
Fax: +44 509 211586
E-mail: FACS@lut.ac.uk

FACS Officers

Chair	John Cooke	D.J.Cooke@lut.ac.uk
Treasurer	Roger Stone	R.G.Stone@lut.ac.uk
Committee Secretary	Roger Carsley	roger@westminster.ac.uk
Membership Secretary	John Cooke	D.J.Cooke@lut.ac.uk
Newsletter Editor	Ann Wrightson	a.wrightson@hud.ac.uk
Liaison with BCS	Margaret West	mmwest@scs.leeds.ac.uk
Liaison with FME	Tim Denvir	timdenvir@cix.compulink.co.uk