Requirenautics Quarterly

The Newsletter of the Requirements Engineering Specialist Group of the British Computer Society

http://www.resg.org.uk

Issue 27 (November 2002)

RE-Locations

RE-Soundings

Editorial

Welcome to the bumper Christmas issue of RQ27. The big event since RQ26, of course, was the combined ICRE and RE conferences held, for the first time, this side of the Atlantic in Essen. If you were unlucky enough not to make it, you can at least read Ian Alexander's account later in this issue.

Meanwhile, books on RE continue to proliferate like requirements change requests. The astonishing thing is that the quality of almost all the books is so high. Now, having fed you the link, I can't pass up this opportunity to plug one of the new books, co-authored (with Richard Stevens) by our very own Ian Alexander and bringing to three the number of books on RE by members of the RESG committee. Ian and Richards' book is a cracker, as you will see from Suzanne Robertson's review. Get your request to Santa now.

Pete Sawyer
Computing Department, Lancaster University.

Chairman's message

I'm not sure if it's the outstanding efforts of our Publicity Officers, Juan Ramil and Sebastian Uchitel, or simply a revival of public interest in requirements engineering, but all the indications are that requirements are in fashion! The recent RE'02 conference in Essen, Germany, saw record attendance and the launch of several new RE books, including an excellent one by our very own RQ reporter, Ian Alexander. I am not sure when Ian had time to write the book, given his prolific contributions to this newsletter, including his insightful reviews of other authors' RE books!

RESG events have been very well attended, with over 100 attendees at the "Scenarios Work" half day meeting in July, and about 80 attendees and the RESG's first industrial networking event, "RE:FRESH". The calendar of events for the coming months is in this newsletter, so this is the time to make a note of these in your diaries!

As we approach the year end, it is time to send out RESG membership renewal requests. After a year of free subscription to encourage RESG membership, to consolidate RESG mailing lists, and to give Membership Secretary, Steve Armstrong, a chance to settle into his new role, we must now return to our usual charging of nominal membership fees to cover our costs. If our membership renewal process has worked correctly, you should have received with this copy of RQ a request to renew your membership. I would like to invite you to renew your membership for a further year, by returning the enclosed form with your payment. As always, membership of the RESG entitles you to receive this newsletter free of charge, and to attend many RESG and related events at significantly reduced rates.

As an added incentive to renew your membership, I am happy announce that we have negotiated, for RESG members, an additional substantial discount on personal subscriptions to the Requirements Engineering Journal published by Springer (£10 for a year's electronic subscription; £30 to have the hardcopy included as well!). We have also managed to negotiate substantial discounts on personal subscriptions to two other Springer journals, Cognition, Technology & Work, and Knowledge and Information Systems. Information about these offers is enclosed with your membership renewal.

One last comment on membership. The RESG offers Academic and Industrial Corporate memberships, which entitle member organisations to up to 15 named individual memberships, as well as an opportunity to include a half page advert in RQ. So, if there are a number of individuals in your organisation interested in RE, why not team up and join the RESG as a group. Only one person needs to act as the Corporate Contact, while the others receive RQ and other benefits hassle-free!

Finally, I would like to end by wishing all RESG members the Executive Committee's warmest Season's Greetings, and best wishes for a happy and successful New Year.

Merry Christmas, Happy Hanukah, and Eid Mubarak!

Bashar Nuseibeh
The Open University

RE-Treats

For further details of all events, see www.resg.org.uk
Next event organised by the group.

Using Formal methods to Understand Requirements better

A one-day symposium

Date: 09.15-17.00 6th November 2002
Location: Imperial College, London
Contact: Alessandra Russo.
(ar3@doc.ic.ac.uk)

This one-day event will focus on the use of formal models and formal reasoning to enable a better analysis and understanding of requirements specifications. The event will include a morning tutorial by Professor Axel van Lamsweerde on goal-oriented requirements engineering and an afternoon of presentations and panel discussion from leading European researchers and practitioners. Research presentations will illustrate recent developments in formally-based techniques for requirements engineering, providing a forum for open technical discussion. The practitioners' presentations will illustrate the use of some of these techniques in real software development projects.

The event will be of particular interest to researchers and students, as well as anyone interested in learning about and/or challenging what formal techniques can offer to the requirements engineering process.

Speakers at the event include leading academics and practitioners in Requirements Engineering:

- Professor Axel van Lamsweerde, Catholic University of Louvain, Belgium
- Professor Jeff Kramer, Imperial College London, UK
- Dr. Paolo Giorgini, University of Trento, Italy
- Dr. Anthony Hall, Praxis Critical Systems Limited, UK
- Tim Clement, Adelard, UK.

Security Requirements

Date: 29th January 2003
Location: City University, London
Contact: Elena Perez-Minana
(Elena.Perez-Minana@philips.com)

A master class in Security Requirements Engineering: a new event-format in which an information security expert and a practitioner discuss the security requirements of a practitioner's application in front of a live audience, of which you are invited to be part.

An event of relevance for all those concerned with the elicitation, analysis and validation of security requirements for computer applications.

Other events likely to be of interest to RESG members.

Mastering the Requirements Process

Date: 17-19th February 2003
Location: London
Contact: Jeanette Hall
(Jeanette@irmuk.co.uk)

A 3-day seminar and workshop. RESG members are entitled to a 10 percent discount.

Requirements Modelling

Date: 20-21st February 2003
Location: London

Contact: Jeanette Hall (Jeanette@irmuk.co.uk)

A 2-day seminar and workshop. RESG members are entitled to a 10 percent discount.

RE-Calls

Recent Calls for Papers and Participation

11th IEEE International Requirements Engineering Conference (RE'03)

8th - 12th September 2003, Monterey Bay, California USA

http://www.re03.org

The RE conferences are a platform for research to present novel results, for transfer of research results to industrial practice, and for the presentation of industrial experiences that can inform new research directions. Two kinds of technical papers can be submitted: research and experience. Topics of interest include, but are not restricted to:

- Requirements elicitation techniques
- Requirements validation techniques
- Requirements management and traceability
- Requirements evolution
- Requirements, software architecture and business architecture
- Requirements prioritizing and negotiation
- Combination of formal and informal specification techniques
- Requirements for high-assurance systems
- Making formal techniques usable
- RE for mechatronics systems
- Specification of quality attributes
- Requirements metrics
- Tool support for RE
- Prototyping, animation and execution of requirements
- Requirements for business systems (workflow, groupware, e-commerce systems)
- Requirements for web-based systems
- Requirements for ubiquitous computing
- Requirements for product families
- Requirements engineering case studies and experiences
- Cognitive, social and cultural factors in RE
- Requirements engineering education

Electronic submissions will be accepted at the RE'03 Paper submission site. Authors without web access must make advance arrangements with the Programme Chair at least one week before the deadline. Papers must not exceed 10 pages in length, and must in the IEEE CS Press Proceedings format (see http://computer.org/cspress/).

Accepted papers must be accompanied by a signed IEEE copyright release form. See the submission page for information on how to submit technical papers (research and experience), workshop proposals, panel and tutorial proposals, doctoral workshop papers, posters, research demos and industry track contributions. For any other queries, please contact info@re03.org. Revised and extended versions of a selection of the best papers will appear, depending upon focus, in a special issue of the *Requirements Engineering Journal* or *IEEE Software*.

Key Dates

- Paper abstract submissions (mandatory) 31st January 2003
- Full paper submissions 7th February 2003
- Notification sent to authors 11th April 2003
- Camera-ready papers received 13th June 2003
- Poster submissions 26th April 2003
- Workshop proposal submissions 29th March 2003
- Tutorial proposal submissions 29th March 2003
- Doctoral symposium submissions 29th March 2003
- Research demo submissions 26th April 2003
- Industry track submissions 1st March 2003

RE-Readings

Reviews of recent Requirements Engineering events.
All reports by Ian F. Alexander

Scenarios Work! Improving Requirements Engineering with Use Cases and Scenarios

RESG Event, July 10 2002, University College London.

Ian Alexander (Meeting Chairman) welcomed about 100 people to a packed lecture hall in the handsome victorian Cruciform Building (formerly University College Hospital – the corridors still smelt of carbolic!).

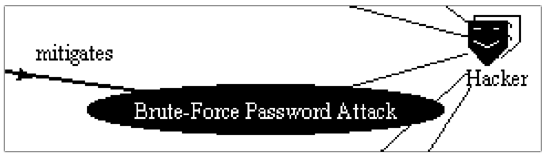
Roland Leibundgut (Zuhlke Engineering) introduced the subject with a presentation on Use Cases in the Project Lifecycle. He asked for a show of hands as to who had ever written a Use Case, and certainly a majority of the audience had.

A Use Case was not just a bubble on a diagram but could be as much as 80 pages of text; more than 3 pages was common. Use Cases were helpful in defining system boundaries, guiding development, specifying tests, writing end-user documents, as well as monitoring progress and planning iterations. They could only describe functional requirements. Advanced UML (Unified Modeling Language) features such as extensions and inheritance were rarely needed and often abused, e.g. for functional decomposition, which was definitely not the intention.

Alstom wanted to sell gigantic steam turbines over the web – in a photo, the humans were as small as UML stickmen. Turbines have a long life, so project documents had to be readable in 30 years' time. XML (eXtensible Markup Language, a development of SGML) was therefore chosen as a stable format – Leibundgut was unsure whether the .doc format would be around in 2030. The RUP (Rational Unified Process) style of use cases covered all the functions; NFRs were described in traditional requirements format. A good part ("80%") of the use cases were identified in the RUP inception phase; most were then defined in the elaboration phase, and all were completed in the construction phase – i.e. change was accommodated and iteration was expected. Traceability was therefore vital. Traces and documentation were handled with simple XML tools; obviously, with an open format, any compatible tools could later be deployed to read and analyse the requirements.

Ian Alexander (Independent Consultant) introduced the concept of Misuse Cases – use cases with hostile intent – originally created by Guttorm Sindre and Andreas Opdahl, though negative scenarios have been around since cavemen hunted Woolly Rhinoceroses across northern Europe and wondered around the campfire if what they were about to do was the last thing they'd ever do. He illustrated this with a cave-painting from the Dordogne, and worked examples in the automotive and railway domains.

These were negative scenarios, used to help elicit system goals and requirements. If a scenario is a sequence of actions leading to a goal desired by a person or organisation, then a Negative Scenario is a scenario whose Goal is desired by a hostile agent (not necessarily human), and desired not to occur by the organisation in question. In other words the approach is a way of thinking about intentional opposition to a system's goals. This leads naturally to thinking about ways of mitigating threats, and where necessary handling conflicts between goals. It was helpful when doing this to use a few stereotyped relationships between Use and Misuse Case goals: *threatens*, *mitigates*, *aggravates*, and *conflicts with*. These could be handled simply and elegantly with a few rules (e.g., a link from a Use Case to a Misuse Case is by default a mitigation).



He demonstrated the ability of Scenario Plus (a free Use/Misuse Case toolkit for DOORS) to represent and display a design trade-off model, filtering to show only the desired relationships.

We then adjourned for Tea – in an enormous queue – and vigorous discussion across the road in the Pearson Building.

Neil Maiden (City University) told some Tales from the Trenches: Using Scenarios to Specify an Air Traffic Control System – a topical matter, given the catastrophic failure a week earlier (a head-on collision of two airliners) in the skies over Lake Constance. He recounted how controllers in a workshop had laughed at the question 'and what if the controller was not present?' – perhaps the question seemed to them outside the boundaries of "the system", but of course 'the man in the loop' is a source of numerous classes of exception.

The Crews-Savre tool helps to generate and walk through scenarios. In Air Traffic Control (for Eurocontrol), the CORA-2 tool will be a redesigned conflict resolution assistant. Crews-Savre is the cornerstone for turning all CORA-2 scenarios into requirements. Other techniques being applied are i* agent dependency and goal modelling; creative design workshops (with musicians and chefs to get the ATCs' creative juices flowing; VOLERE-based requirements management; human activity modelling; and scenario-based impact analyses.

In such work you were not just specifying software; you were changing the work processes of the controllers, i.e. redesigning work in a Checklandian socio-technical system. The basic approach was to generate normal course scenarios, and then create numerous what-if? Exceptions as branches off those normal courses, using rules based on the type of action. For instance, if an action was classed as cognitive, then you could throw in all the generic cognitive exceptions such as memory recall failure, confused thinking, and so on. Maiden "raided taxonomies of errors" from HCI to identify 40 common domain-independent or generic exceptions. He also elicited (by laddering, and through scenarios) 130 air-traffic-specific exceptions. A facilitator/scribe team then plodded through the RUP-style use cases, asking broadly the same exception questions in each one.

Did the tools help to get more requirements? Yes; in 10 months before the walkthroughs, 247 operational requirements were gathered (i.e. 6 per week). 3 weeks of walkthroughs created an additional 134 new approved requirements (i.e. 45 per week). In the process 254 normal course events and over 3000 exceptions were considered. What is more, the new requirements were more tightly defined and more verifiable, and often replaced old brainstormed requirements.

But disappointingly, only 4 requirements came from domain-specific scenarios: 79 came from generic exceptions, and 51 from normal courses. In a way this is good news: standard questions will elicit almost all the requirements, perhaps. This suggests that the walkthroughs should have been done earlier to avoid timewasting. Another possibility is that the domain-specific exceptions were too narrowly defined (this might be a feature of the domain, as ATCs are notoriously procedural thinkers). In feedback, people said that they were impressed, and that the process was "exhaustive" – you can take this either way.

Steve Armstrong (Open University) presented some controversial views on The Abuse of Use Case Techniques. For instance:

- Scenarios introduce a time-ordering, but this is a design. "How might a given task unfold?" is a design question.
- UML Use Cases provide a structural view, in contrast to sequence (swimlane) and activity (flowchart) diagrams which provide a dynamic view. (Hmm, maybe, but isn't the time dynamic admittedly inherent in the courses of events embedded in use cases?)
- The Lending Library example is popular amongst academics teaching UML; but Jacobson never intended Use Case models to describe human activities in soft systems. (Hmm, mmm, but if something can apply to a system with software agents, why not hardware agents, robots, or even humans? Aren't they all systems?)
- UML brought something to the party, but there was sill a huge gap between developers and users, and better CASE (Computer-Aided System/Software Engineering) tools were still urgently needed. (OK, maybe that one's not too controversial.)
- One size did not fit all; Use Cases and Walkthroughs weren't enough.
- Awareness (of how people work, of the (socio-economico-politico-technical?) situation of any system) is all.

The meeting came to a close with a great deal of informal discussion which continued round the corner in the **Jeremy Bentham**. He was a distinguished pioneering physician, and he is evidently still frequently consulted for his opinion, as evidenced by the number of stethoscope-wielding visitors. A straw poll was taken on whether people at the meeting would like to attend informal Birds-of-a-Feather gatherings in London, and there was an overwhelming response, so David Bush will shortly be organising the first BoF. The speakers' slides are on the RESG website (www.resg.org.uk).

Using Formal Models to Understand Requirements Better

RESG Event, November 6 2002, Imperial College London.

We had two chairpersons for this event (or three if you count Alessandra's 6-weeks-from-term baby), **Bashar Nuseibeh** and **Alessandra Russo**. They were delighted to welcome an all-tickets-sold audience to Imperial College and the RESG.

Axel van Lamsweerde gave a morning tutorial on Goal-Oriented Requirements Engineering. He was on top form and gave an enjoyable and readily-understandable overview of RE from the point of view of goal modelling. Goals are the why; requirements and assumptions are the what, he said. Goals had been ignored by UML, but guru Fowler agreed they were needed. (Mind you, guru Cockburn treats use case titles as a hierarchy of functional goals.) Goals were prescriptive (optative in Jackson's parlance), unlike domain properties which were descriptive (indicative). In other words they were essentially high-level requirements. They didn't have to be functional; non-functional goals could be for safety, security and so on: what the Toronto i* people call softgoals. He argued that there wasn't much of a distinction between functional and non-functional goals/requirements, and it is true that there is an interplay between them. For instance, a security NFR causes various functions to come into being, e.g. to lock things and to record accesses. But that doesn't make the NFR the same as the functions that implement it, does it?

Agents always went with goals; whether human or machine, agents co-operated to satisfy goals. Perhaps more contentiously, goals had to be realizable, at least at finer levels of detail when you get down to defining exact logical conditions for them – low-level goals were thus implicitly equated with requirements; high-level goals were what some people call objectives.

Goal modelling could proceed top-down (the obvious approach) or any other way; ANDing subgoals explains the basic rationale for high-level goals; subsequent OR-abstraction allowed you to refine the alternatives. This analysis helped to discover conflicts, early in the life-cycle – long before design – by being declarative about what people wanted.

On the more formal side, you could show that a set of requirements was "complete" if for all goals, the requirements, assumptions and domain knowledge entailed the goals. Obviously this could only work if the goals were realizable 'user requirements'. Similarly, a requirement was only pertinent if it helped to satisfy a goal. But you absolutely couldn't show that a set of goals was complete with respect to the real world. Low-level goals could be verified; high-level objectives could in Herbert Simon’s unlovely phrase be ’satisficed’ to some degree of confidence.

Goals at a high level were generally more stable, as Annie Anton had observed, so in principle you might be able to use goals to document product evolution (e.g. in product lines). In theory you’d do this by providing new alternative ways to satisfy a stable goal. I think this may be a tall claim because old product features can always become obsolete; cars used to have to be able to tune to Medium Wave and to play Audio Cassette tapes, for instance; while it may be true that high-enough objectives (’play music’) are indeed stable, they aren’t very useful as product requirements.

Obstacles allowed you to reason about ’unexpected agent behaviour’ in a goal model. Formally, an obstacle obstructs a goal if and only if the presence of an obstacle in a domain meant that the goal could not be achieved; and that the domain definition did not exclude the possibility of the obstacle’s existence. Identifying and removing obstacles helps to make requirements more complete, more realistic, and hence more robust. However, this happy story does not take into account the possibility that malicious agents can be creatively hostile: there is an arms race between your obstacle-removing tactics and the subsequent moves of the other side. ’Obstacle’ sounds and is static; a more dynamic approach is to deal with threat and counter-threat as a game which won’t necessarily end, and in which new moves are expected.

Thoughts of obstruction led naturally to consideration of safety and security issues, including the famous counter-example of aircraft wheels aquaplaning on the runway to the simple assumption that the plane is on the ground and may use its reverse thrust if and only if its wheels are turning. Jawed Siddiqi (Sheffield Hallam) said this was all post-hoc, and asked how you would probe to elicit such things? Axel replied that indeed there was "no free lunch" – you get completeness only with respect to what you already know about a domain. "We use obstacles all the time, semi-formally – using patterns to provide proof", said Axel. (Anthony Hall said over a cup of tea that safety engineers are like old generals who always fight the last war; safety people always prevent the crash that happened last time.) Could goal-obstacle analysis then ever be predictive? Axel replied that the disastrous Uberlingen midair collision (July 2002) had in fact already been analysed in a large (>300 goal) model which had been completed back in March 2002. All the six or so "perversely chosen" obstacles that occurred simultaneously – there was only one controller, he was overloaded, the onboard collision warning system and the air traffic controller gave conflicting orders, etc – were all in the model but were in different goal trees. To be predictive, you'd have to be able to assign probabilities to complex combinations of obstacles.

Scenarios and goals offered complementary benefits; scenarios for elicitation and validation with people, goals for reasoning. Scenarios were excellently concrete and narrative, but were partial (like test coverage), procedural, and vulnerable to combinatorial explosion. But, said Axel, "we finish where the formal modelling guys usually start" – goal models formed a natural bridge between domains and systems.

In the afternoon we had four speakers and then a panel session.

Jeff Kramer (Imperial College) talked about Scenarios to Behaviour Models, and Back. He felt that behaviour modelling should be part of everyone's requirements process: you needed to go from requirements to models to make things definite. Scenarios were wonderful for elicitation and for explaining why bits of behaviour were needed, but they had no precise semantics, even when they were as neatly documented as a UML message sequence chart. A model was complementary to a set of scenarios; it could be analysed and checked; it could be animated to generate a trace (i.e. a scenario) or drawn as some picture of the system which you could then animate in a more visual way as a sort of simulation.

Equally, you could go in the other direction. "People find message sequence charts easier to produce than models", he said. So, how could you create models from scenarios? If you treat scenarios as sequences of activities, you can eaily put together a state machine. If scenario 1 goes ABCD and scenario 2 goes ABCE then state C has two exits, and so on. In general the resulting state machine or architecture is richer than the traces or scenarios as you start discovering implied scenarios. If scenario 3 goes EDFG then you discover that ABCEDFG is possible, as well as ABCDFG which you didn't know before. You may discover ways of getting into error states, or things you don't understand that you can check with your stakeholders – using animation of scenarios to give them a picture of what the model implies.

Like Axel, Jeff said there was "no free lunch" – the approach did not discover unknown event types. What it could do was discover negative scenarios, certain implied scenarios that shouldn't happen.

Axel van Lamsweerde asked about the quality of models if they were based on instances. Jeff Kramer replied that when you went to negative scenarios (after the initial model-building), you often needed to generalize: "that's an instance of the general case, one of a class of counterexamples".

Anthony Hall asked if you put multiple scenarios into one behaviour model? Jeff Kramer replied that the approach did exactly that. It could handle 10,000 states which was rich enough for significant problems. Finite state modelling was vulnerable to combinatorial explosion of scenarios but it was tractable.

Paolo Giorgini (Trento, Italy) talked about Reasoning with Goal Models in the Tropos Methodology. AND/OR trees (a la Axel) could not handle vague or partly-defined goals. You could have networks of goals with cycles of dependency (positive or negative), for instance. Tropos reasons about full or partial evidence that a goal is satisfied or denied. Intuitively these can be based on positive and negative relationships between goals held by actors. The reasoning is fully axiomatized and can be applied qualitatively or quantitatively. Tropos can propagate values (full, partial, none) or -1.0 .. + 1.0 across a 30-goal network in negligible time (1/100 second). You can then see which relationships are important.

Tropos - movement - first modelled the environment as Actors with dependencies on each other. Then it took the 'system actor' and modelled its dependencies – whether these represented functional or non-functional requirements – on actors in its environment. Then it decomposed the global system into subsystems and modelled the data, control (etc) dependencies between these. Finally, it analysed the details of input, output, and control. In this way it aimed to bridge the gap between the early-in-the-life-cycle approach of i* and the late-in-development approach of agent-oriented programming. KAOS was another early approach; UML mainly later. Tropos was unique in using the notion of agent throughout development.

Anthony Hall (Praxis) talked about Industrial Experience with Formal Requirements. Formality added precision and enabled exact reasoning. Did it really help, he asked? "It really makes the whole development more visible" he said. He explained the basis of the Reveal method, and gave some examples. Requirements Engineering was about showing that the Requirements were entailed by the Domain and the Specification, i.e. that the requirements would be satisfied. (In response to a question, he agreed that his requirements were Axel's goals, and his specifications were Axels requirements.) The domain could be modelled formally by analysis, with business rules, laws of physics and so on. By being precise, early, you could discover problems and reduce trouble during development. For instance, smart card issuers were naughty in assigning card IDs so you in fact needed to know the issuer, the ID, and some sort of hash function to identify the application correctly.

Similarly in requirements you often (but not always) needed to be formal, especially when safety or security were concerned. On the other hand, vague high-level objectives like improve throughput might well be acceptable. Precise requirements could be defined in Z, a logic notation. You could then reason about satisfaction and detect errors and missing requirements. Contrary to expectation, this was cheap; Z found as many errors as unit testing, but was five times cheaper. It found errors early too: many of the errors introduced by specification were removed during architecture or detailed design, and in a recent project only one specification error survived through to operations. Modestly but with some pride he quoted an authentic customer statement: "The system behaves impeccably as expected."

Axel van Lamsweerde said that he agreed 1000%, but what of people who advocated agile methods? Anthony Hall replied by quoting John Barnes: "Ada is not meant to make programming quick. It's meant to make programming slow. Slow is good". There was laughter.

Tim Clement (Adelard) talked about Dust-Expert™: an example of formal methods use in industry. Almost any powder like sawdust, flour or custard could explode if mixed with air and ignited by a spark – electrostatics were likely in a dust-filled factory. He showed photographs of a furniture factory with all its walls blown out. He had used VDM, another notation like Z, to define the specification of Dust-Expert precisely. Numerous useful properties of the system emerged from the formalism; for instance, the Graphical User Interface revealed all of the system's internal state, and essentially always (ok, not while the user was updating a field) exactly represented it. So you got properties like interface predictability and correctness of representation. Productivity was above average for safety-related software,and there were only 1.5 errors per 1000 lines of code (a total of 42 bugs). 31of these were discovered through testing, and the rest through field experience.

The **panel session** was chaired by Bashar Nuseibeh. He asked how to justify formal models in industry. Tim Clement said it was cheaper on maintenance. Anthony Hall said that if the customer cared if [a system] worked, it was the fastest and cheapest way, e.g. for embedded software. He agreed that if you didn't care, you didn't do it – a bank made a cold-blooded calculation that getting a website up 2 months early got you 200,000 extra customers, and it then cost you only £20M to fix the problems which was worth it. "We're a bank, we always do this."

Workshop on Goal Oriented Business Process Modeling (GBPM'02)

Part of HCI '02

September 2 2002, South Bank University, London

Ilia Bider (IbisSoft, Stockholm) led the 3rd GBPM workshop, in memory of his late collaborator, friend, and co-founder of the workshop, Dr Maxim Khomyakov. The first workshop looked at OO and BPM; the second at practical issues of modelling business processes; and this one focused on Goals – what BPM is all about.

Ian Alexander (Independent Consultant) drew the short straw, speaking first straight after the holiday season. He spoke about **Modelling the Interplay of Conflicting Goals**. A scenario was a sequence of activities to achieve a functional goal desired by an organization, often modelled nowadays as a Use Case. Hostile agents could also have goals which may directly or indirectly threaten the organization's goals; these can be modelled as 'Misuse Cases'. Interactions between Use and Misuse Cases include MC threatens UC; a subsidiary UC mitigates MC; a UC unintentionally aggravates an MC (while perhaps mitigating another); two UCs directly conflict (see previous item - Ed.).

Depicting agents – possibly systems or parts of the environment (like the weather) as UML stickmen was a useful anthropomorphism; it enabled people to reason with their social brains about agent's goals and how to counter them. The approach led to a simple way of depicting and agreeing on goals, which was useful in specific areas such as design trade-offs and the identification of safety and security requirements.

Gil Regev (Ecole Polytechnique de Lausanne) spoke about **Regulation-based Linking of Strategic Goals and Business Processes**. The basic approach is attractively simple: businesses can be considered as control systems intending to maintain a target value, or as organisms with homeostatic goals (such as keeping body temperature constant), influenced negatively by the environment. To my mind it was surprising that neither control nor homeostasis were mentioned; instead, Wiener, Weinberg, Anton, and Checkland were referenced on aspects of systems. Perhaps different communities use different but isomorphic languages to describe goal-oriented behaviour.

Constancy or stability sound static, but can be interpreted dynamically (said Ilia Bider) by talking about constancy of growth (e.g. having a Strategic Goal to increase market share). You then get an Operational Goal (e.g. increase sales geographically) which is related to the strategic goal through a Belief that it will help to achieve it – through the operation of a business process (e.g. set up sales forces in new areas).

Ilia Bider said both these talks were about the **Environment** and our understanding of it – which might be true or not; and we could find out if it were true by modelling the environment and simply by trying out your model in the world and seeing if it works. Hence there is feedback to bring our understanding closer to real life, which is itself a higher-level (regulative, if you like Kant) process applied to lower-level business process interventions.

Erik Perjons (Stockholm University) (in a paper co-authored with **Ilia Bider** and **Paul Johanneson)** spoke about **Goal-Oriented Patterns for Business Processes**, focussing on goals rather than on activities. He described a state-oriented approach to describe business process patterns. Quoting Fowler, a pattern is an idea useful in one context and hoped to be useful in others. Hammer & Champy define a BP as a partially-ordered set of activities to reach a goal. Different organisations may have different sequences, or somewhat different sets of activities, but shared goals and basically similar processes, so perhaps common patterns can be identified.

Approaches include input/output flow as in IDEF0, workflow sequences of activities as in UML Activity charts (flowcharts), agent-related workflow as in RAD, and state-flow looking at changes in the world caused by activities (a bit like IDEF3). This last is Perjons' favourite. Two BPs are similar if they have state spaces with the same topology, their goals are similar, and they have the same kind of valid movements in the state space. For instance, ordering increases the number of ordered goods (on the Y-axis); delivering increases the delivered goods (on the X-axis). To meet the goal that Ordered = Delivered, the final state must be on the line X=Y; and similarly for all other dimensions of the pattern. Goals thus become surfaces, not points, in the state space!

Ann Lindsay said that BPs were about what you know happens, but we need instead to learn to adapt, and BP modelling isn't helping people to adapt.

Ilia Bider said that to use patterns to synthesize processes, you needed to specialize them, adding procedures to move from one state to another, and adding constraints (e.g. you can't move back in time – you can return goods or deliver more, but not actually undeliver).

Dick Schaap (University of Groningen) spoke on defining a **Goal-Reached, Energy-Used Value pair as a Business Process Measure**. His aim is to describe rather more scientific ways of improving performance than the rather vague ideas in the literature such as 'eliminate, simplify, integrate, automate' (Peppard and Rowland).

His idea is to plot Goal Reached on the Y-axis, and Energy Used on the X-axis. If you have some uncertainty or variation in each, you'll get an elliptical GR/EU blob instead of a point for each activity. You can use a UML-like swimlanes chart to show the sequence of activities organized by actor, and then construct a table of goals reached (presumably one can tell if a goal is 100% reached, or less) and actor-time (a crude measure of energy) per activity. From a flowchart he ran some simulations to evaluate GR/EU results, getting a family of curves or lines. He admits that both Goal Reached and Energy Used can become complicated and unrealistic, so the challenge is to simplify them sufficiently. With multiple processes you also need to address issues of scheduling people and other resources to avoid conflicts. The ideal process clearly uses no energy and no time, i.e. is at the origin of the GR/EU chart.

Gil Regev said that in the 1950's, energy and information were seen to be different – an autopilot uses very little energy but can control a huge ship or plane. **Dick Schaap** said that the GR/EU chart makes visible what is happening in terms of resources applied and results obtained; you might have reasons (not visible on the chart) for going for goals quickly but at higher energy cost. **Ilia Bider** said that some goals have higher value, e.g. hitting the market first may be business-critical as **Ian Alexander** said was the case for developing new Jet Engines for specific aircraft types. And values might change; **Ann Lindsay** said the same was true in production – you might rush to meet a critical order for chocolates even if you wasted materials and labour. Value would have to be a third axis but Dick Schaap had not worked that out.

Ip-Shing Fan (Cranfield University) spoke about **a Process Model for Diverse Stakeholder Goals**. BPM projects were limited in various ways, but all aimed to abstract the organization of activities in an organization. Goals were driven by strategy, eg. quality, lead time, time to market, delivery reliability, design flexibility, volume flexibility, cost/price, innovation, and trustworthiness. These are extremely diverse competitive factors. Organizations differ in the factors they choose, and in their resulting success. It would be useful to understand the reasons for these variations.

Fan suggested that the largest cause of difference is people; the interaction of people with processes is complicated. Three perspectives on this are organization, people, and technology; these might be useful. But BPM's assumption that there is one single canonical process that everyone should follow is clearly false; so going from an As-Is model to a single new To-Be model – in classical BP modelling – is a limiting approach. People naturally desire variability in their work. Quality manuals indeed deliberately avoid (mechanism) details, sticking to (vague) business goals. Workflow is often too prescriptive. Instead, we need controlled diversity – not chaos but an allowed range of process variation. Fan is therefore working on ways of generating diverse models offering a choice of possible processes. TRIZ, the so-called theory of inventive problem solving, now available in 2 software forms (Invention Machine, and The Ideator), is being explored as a BPM tool. TRIZ was invented by Genrich Althsuller, a Russian who studied 200,000 patents to identify common features in the invention process; TRIZ is a huge conceptual system.

Ilia Bider said that half of the answer is in the question. Fan had put the wrong question by imposing Workflow; flexibility was needed in understanding the structure of the goals. Order is more or less fixed; varying all the orderings would give you too many answers. Fan said he agreed.

Denise Downs' talk on **Analysis and Design for Process Support Systems using Goal-Oriented BPM** was presented in her absence. She had argued that you should define clear goals, vision and values before creating a comprehensive BP model, and then trace each activity in the model to the software design objects that implemented it. Neither the OO approach – which tended to jump on isolated use cases – nor the structured approach – which tended to identify processes and dataflows without looking at sequence constraints or roles. She therefore creates a process flow model, which contains 2 kinds of item: triggers and activities. Triggers can be events or by time; every process flow starts with a trigger. Activities form a chain and identify the precise sequence required; i.e. the process model defines a canonical "best practice" process which is used by staff for guidance. This is clearly a simple case (unlike those raised by earlier speakers) but a perfectly valid one.

Steve Battle (HP Labs, Bristol) and **Deb Cooper** (Contact24 ltd) spoke on **Goal-Oriented Service Management**. We've done what the client asked us to do: why aren't they happy? We got really frustrated as it happened again and again. So we devised a practical approach for our outsourced situation. We want to be perceived as experts, and to learn from every implementation before we face our Board. So Lessons Learnt are important.

We worked with an organization providing outsourced call centre CRM support for the financial services industry. There is voice, as well as manual processes, and web interaction. There's been a big change since Sept. 11th – cutting costs, but also giving better quality not just quantity. In the USA they just have quantity; the UK public is far more demanding and want their questions answered right first time. And clients tell us what they want, not what they need: 2 different things.

So, we first identify BPs and participants with use cases. E.g. queries arrive and are handled by advisors, who forward more specific queries to a well-trained bureau operator, and complex queries to a dedicated advisor. This analysis leads to the design of a massive knowledge base for each class of actor. Traceability is established between client corporate aims, more specific objectives, and then to detailed goals. However the goals depend on the BP model – if it changes, goals have to be revised. New goals can be handled temporarily in contingency by simple (often manual) procedures. If a client asks for anything that doesn't seem to trace back to a strategic goal, we can ask them why they want it, and maybe thus simplify the process model.

Ian Alexander remarked that it had been said 'all good patterns are obvious'. **Ilia Bider** said there was often too much abstraction in the wrong place (academically) but too little in other places (industrially). Ian replied that we all needed to connect research with practice and teaching.

Ilia Bider led the discussion by asking us to work towards a common notion of Goal for BPM. Vision was too unquantifiable to be included. Objectives might also be unmeasurable but were important. Goals had to be measurable. But strategic objectives could said Gil Regev be qualitatively assessed. Ilia Bider asked us to seek the kinds of measures that goals should have. E.g. in a homeostatic system a key goal is to maintain a constant (or constantly growing) market share, etc. Goals could thus be fixed or dynamic (i.e. having a constant rate of change) with respect to an environment. Staying still was like walking up the down escalator: there was always 'friction' so you had to take action to stay still. To adapt to other environments you'd have to change the BP model.

Ilia said that (in a homeostatic or control system) having a goal means the current state is not the goal state, and a corrective (negative feedback) action should be applied. Ian replied that even when the current state was the desired one, the goal (Actual Value = Desired Value, A = d) remained, but no corrective action was momentarily needed. To follow a trajectory, you had a differential, i.e. dA/dt = D. This assumed a continuous environment, but (said Ip-Shing) the environment could suddenly jump.

Goals could be nested, just as processes (and systems) can. Goals change human behaviour by setting measures on our behaviour (said Ip-Shing). These concepts can be applied at all levels in an organization, raising issues of the different languages in the boardroom, in line management, in IT consultancy, and so on.

Every procedure must be measurable against a (functional) goal.

Other goals may not correspond to individual procedures but must be associated with a measure for the organization to follow, e.g. to conduct business ethically, to run the railway safely, etc.

Specification of Distributed Systems Security Policies

Dr Emil Lupu (Imperial College)

Part of the IEE Summer School on Distributed Systems

September 5th 2002.

Issues are still at the border of research and product development: from role-based access control, trust, policy-based authorisation, and conflicts/exceptions. At Imperial the focus is quite practical.

By trust, some people just mean reliability or dependability; others something quite different.

Access control needs to work in very large complex networks, with many machines hosting services. There are millions of objects, tens of thousands of users and many platforms and services. Permissions must therefore be very fine-grained compared to the network – who can invoke what service on what platform; able to change rapidly; and able to cope with heterogeneity. That’s bad enough: but what about laptops, web-enabled cellphones, handheld devices and so on? Visitor policies are needed to ensure security.

The solution is in having defined groups and roles, middleware in CORBA or Java that is now much improved; security monitors, and a policy that spells out what to do when something that looks like an intrusion happens.

Role-Based Access Control (RBAC) means tying permissions to roles instead of individual users. You define roles – like groups, and you can then dynamically assign users to roles without having to redefine those roles each time. The indirection (user – role – permission) affords more flexibility. . Many tools such as Win 2000, Oracle, Tivoli, Sybase all claim to be role-based.

You can also have permissions from junior roles inherited by senior roles through tools that support role hierarchies – but this is a more dubious advantage, argued Lupu: was it true that professors should inherit permissions from lecturers who inherit from research students who inherit the permissions of a generalized department member? It might be right or wrong. Should the prof be able to read all the private files of all research students? The crude rule – senior roles aggregate all access rights of junior roles – needs to be modified by defining private roles (or projects). Hierarchy reduces the number of permissions as claimed, but can increase the number of roles needed. It also raises questions of complexity and efficiency if you have to go around a network to find roles to check permissions for a specific user. The model doesn’t provide a language for specifying policy, and it is basically ad-hoc – every tool has a different approach.

Trust policies started with IBMs Trust Policy Language (TPL) which was a way of using public-key certificates over the internet to control access to resources by user groups; access rights are assigned to groups. TPL is based on XML syntax. Different authorities issue certificates and authorise accesses. Expired certificates have to be revoked. Certificates are valid for a year or so, creating a sizeable overhead of administrative complexity: the key problem in security. The certificate has to name the issuer, subject, type, version, and links to where the certificate syntax and other details are defined. TPL itself allows both positive and negative rules (but you have to be careful with the latter). E.g. you can define that customers need an employee of rank greater than 3 to sign their certificate. Then you can have access rules saying that customers can browse the catalogue and order goods, and so on. But the XML rules are verbose (the customer rule is a page of code!) and unreadable, and no concept of inheritance.

The Oasis consortium is trying to standardize on a different use of XML, with a language for specifying authorization policies (XACML), and a second language (SAML) for marking up security assertions (like “Joe Bloggs has been denied access”). This complies with the idea of separating policy decision-making from policy enforcement. The enforcement point may detect a possible security breach, and inform the decision point which can apply more resources to evaluate the situation. In a blizzard of three-letter acronyms starting with P, policies are handled by being passed around from policy administration to retrieval to decision to enforcement to information and back. Obviously the intended target is the web services market.

There’s a major problem with usability; the stuff is just as verbose as IBM’s XML, and there are no graphical or high-level tools yet before diving into the XML. Pity the poor ex-sergeant-major security officer who has to specify and enforce security! You can clearly create your own higher-level language specification in a database tool and write your own XML exporter; but only the XML format is currently defined – clearing up the current ad-hoc muddle. But Tool support and friendlier notation are clearly needed.

Policy-based authorisation has evolved over a decade. You basically organize a hierarchy of objects (such as computers) into a Domain, and apply a common policy to them all. Again, you can change domain membership without editing the policy – it's just like moving users into or out of a role group. Hierarchy enables you to scale up the application of a policy.

There are 2 basic types of policy: authorisation policies (what you can and can't do) and obligation policies (what you have to do when certain events occur). Negative authorisation is useful especially for revocation of access rights, and can also directly represent organisational prohibitions which are often far more common than permissions. Obligations are typically interpreted by the machine subjects themselves – e.g. when 3 login failures happen, you should disable the affected userID and notify the administrator. Other more specialised types of policy include 'filtering' (weed out confidential parameters) and 'refrain' (hold back from supplying information).

Conflicts can arise when 2 or more policies apply, e.g. when an object is in multiple domains; and because different managers can specify policies, e.g. security and performance policies for the same network. You obviously need a way of detecting conflicts and handling exceptions economically; you don't want to edit a policy every time you find an exception (e.g. students can't reboot servers; but student Smith may do so). A default is that negative rules always override positive ones (so hard luck on Smith). Or you can specify Priorities – but it's hard to stay consistent. A general concept is to evaluate a "distance" between a policy and the object it applies to; near overrides far. Or you can try more recent overrides older. Or more specific (exception) overrides more general (basic rule) – but this doesn't always work. But all of these (modality) conflicts are alas by far the easier type. The hard cases are semantic conflicts. For instance, self-management: a manager may not authorise his own expenses; separation of duties: one person may not initiate and authorize payments. Such meta-policies are very troublesome. You can try to describe them in languages like Prolog or OCL. There is a clear argument for having policies expressed in a declarative language – if all your security policies are encoded in Visual Basic, you can't do much to look for conflicts, gaps, or errors.

Future directions include refinement of trust-based policies; dynamic adaptation; mobile architectures (and pervasive computing, with wearable and invisible devices); and the reverse engineering of existing implementations.

Lupu was cogent and lucid throughout. The group (a dozen) seemed to have been happy with the other sessions they'd attended. There were some jokes about speakers bringing their usual 200 slides for a single session; every aspect of distributed systems seems to be complicated. The atmosphere was studious; the audience mostly male and casually-dressed. Everyone listened closely rather than looking at the neatly-printed sets of slides. There were one or two technical questions at the end of each topic; questions were answered cogently and honestly, pointing up the key issues. It seemed that the Summer School was hitting the spot again.

IEEE Joint International Requirements Engineering Conference (ICRE'02 and RE'02)

September 9 - 13 2002, Essen, Germany.

It was a busy, packed, exciting week, full of stimulating conversation – whether in workshops, in little groups in tutorials, over coffee and meals, walking around the site, or sitting in the sunshine by the water lily pond with the musical splashing of the fountains. Sometimes the question-and-answer dialogue at the end of a paper rose to the same level, as both audience and presenter took wings and soared into the transcendent clarity of original thought.

I went to a 2-day workshop (REFSQ), visited a tutorial (Suzanne Robertson and Neil Maiden's on Creative Thinking; I had to pose as the boss of a bicycle courier firm, complete with yellow reflective sash, but that's another story), absorbed all three keynote talks, heard 11 papers, gave one myself, and looked at several requirements tools. What, ultimately did all that effort buy me? Perhaps it's impossible or too early to say: ideas spring (as Tolkien said) from the well-watered soil and the seed-germ of experience in their own time.

There were inspiring moments:

- Jack Carroll (Virginia Tech) gave a 'global' overview in his Keynote talk of what it means to think with scenarios, based on a lifetime's experience and research.
- Ben Kovitz in his Industrial Talk drew our attention to the crucial need to accept and address ambiguity in requirements, rather than trying to sweep it aside.
- Daniella Damian (Sydney) presented the shattering results of her Grounded Theory observations of attempts to engineer requirements across a continental divide – with people in America and Australia failing to engage with problems from one month to the next, or to trust each other enough to get the job done efficiently, despite the most elaborate high-bandwidth videoconferencing links.
- Martin Feather (JPL) showed that given a tool to evaluate 'cost' (e.g. mass, or power consumption) and another to explore different trade-off decisions, you could come up with a

demonstrably near-optimal project 'plan' (the best combination of features), chosen from many thousands of simulation runs.

- Hermann Kaindl presented his freeware RETH tool, named for 'Requirements Engineering Through Hypertext', showing just how much can be achieved by one man and a compiler nowadays, and introducing some excellent ideas on traceability to a wider audience into the bargain. (He featured in IEEE Software, May/June 2002, pp 70-77, if you didn't see it already.)
- People who managed to get to Sophie Dusire's (Thales) talk about Applying Theory to Reality in Requirements Engineering said that not only was the talk interesting and practical, but that there was an excellent discussion afterwards.
- Matthias Weber and Joachim Weisbrod (DaimlerChrysler) deservedly won a prize for their fine and disarmingly honest paper on Requirements Engineering in Automotive Development - Experiences and Challenges. It gave an insight not only into what was working well but what needed to be addressed to bring about change in long-established industrial practice.

But I also had the feeling that some of the work presented didn't add that much to the world's stock of RE knowledge. Some speakers told us what everyone already knew; others described additions to well-defined paradigms. Still others seemed to be doing something new in modelling or metrics or verification, but it wasn't easy to see how people would apply it in industry. Researchers need to be encouraged to investigate risky and difficult issues, or they'll stick to safely incremental topics.

There are many things that practitioners would like to know in RE. What evidence is there, for instance, for the relative suitability of different approaches in different situations? Can, indeed, we find ways to get hard quantitative evidence when no two projects are alike, and when the obvious experimental guinea pigs are inexperienced students? How should we select analysis methods for different types of problem? How should complex choices be presented to stakeholders? How can we be sure a specification is complete? Can we find effective techniques for distributed projects? The social and political dimensions are barely explored (pace Checkland) but hugely important.

I have a hunch we're actually just at the beginning of a scientific (theory-based) practice of requirements, or dare I say it, of making requirements practice into a genuine (heuristic-based) engineering discipline. We really do have much more useful knowledge than we did ten years ago; it is as yet very poorly disseminated in industry, despite the obviously fine work being reported on by pioneering industrialists like Kovitz, Dusire and Weisbrod, and remarkably little taught in university, again with honourable exceptions. We need to be bold, practical, outward-facing and willing to try new things – which means, willing to get things wrong.

RE'02 was a great place to meet colleagues, to argue, to explore, to start new lines of inquiry, to hear the best and greatest. Let's make it even better next year. Long live RE!

1st International Workshop on Traceability

28 September 2002, Heriot-Watt University, Riccarton, Edinburgh

Andrea Zisman welcomed us to this first workshop, thanking the program committee for their hard work and Telelogic for its sponsorship.

Daniel Gross (University of Toronto) talked about designing systems (using any chosen notation) using 'aspects' to focus on quality requirements (usually global NFRs, eg performance, reusability, reliability, cost): how they are traded-off and refined during design.

In the example, he wrote a frame engine (querying, storing, and retrieving frames) language, Telos, in Prolog, to form a knowledge base. The rest of the work was at the meta-level, reasoning about the design choices to be made.

An Aspect is a new abstraction mechanism that captures design decisions that affect many modules. E.g. the choice of data passsing mechanism affects performance and reliability, so it is an aspect.

Both aspects and code modules are composed during implementation, affecting qualities in various ways. Traces can be used to link NFRs to design decisions and then on to system artefacts; the purpose, of course, is to orient system design to the various system goals, including NFRs – which people often more or less ignore when tracing back to functional requirements.

Intentional Agents can represent design goals; in the I* notation, that's a big circle, which you decompose into tasks and goals with interactions such as 'helps' or 'hurts', effectively describing the reasoning underlying an Aspect of the system (like the data passing mechanism) according to what the Aspect 'wants to do' – in that anthropomorphic sense it is Intentional. The Agents are linked to the various Aspects that they generate (i.e. want). That stuff argues out the design rationale; it is then linked to the relevant design elements (code modules), by what I'd call a justification trace but which they call a reference.

Elena Perez wondered how this might scale up. Gross thought the representation was very compact; also you could have a folding editor that showed relevant detail and hid the rest.

Jeremy Dick and Ian Alexander said that one design choice often forced another. Gross said that you could use dependency links between tasks that the designers had to accomplish, to represent any kind of reasoning about the design.

Antje von Knethen (Fraunhofer Institute) spoke about automatic change support based on a trace model, to enable proper impact analysis before going for a possibly costly change. Documentation Entities talked about goals or requirements; Logical Entities spoke about the corresponding control tasks and environmental variables managed by those tasks. Actors could be linked to both. The relationships between entities include representation, dependency, and refinement. Then you can define constraints, e.g. in an embedded control system, each control task must 'influence' exactly one environmental item (like temperature). Obviously this allows you to carry out helpful validation checks on the installed traceability. A controlled experiment with graduate students showed that guidelines for maintainers using this approach were beneficial for impact analysis. The approach was therefore implemented by extending the commercial StP/UML tool. This was easy, and ensures the approach can scale, but the tool was unable to automate the making of changes or support the whole of the process – e.g. it didn't support use case descriptions (and the use cases appeared as rectangles with no actor symbols!).

Darijus Strašunkas (Norwegian UST, Trondheim) spoke about traceability in collaborative systems development from a lifecycle perspective. Its purpose was to ensure completeness, to propagate changes, and to facilitate mutual understanding and enable meanings to be shared ('semantic interoperability'!).

The essential difficulties included the distance between CP Snow's two worlds – human usage/natural language, vs technical usage/formal methods. An environment was needed to support collaborative development.

Jeremy Dick (Telelogic) spoke about Rich Traceability. He said that we as a community were changing the world, at least because Traceability isn't in any dictionary. We're trying to get people to use it in real projects, not just in software but in systems of all kinds, such as the Joint Strike Fighter – probably the largest system development in the world.

What we say to our customers is that 'information traceability is understanding how high-level objectives are transformed into low-level goals', i.e. we abstract away completely from talk of links. And we tell them 'the benefits include greater confidence that requirements are being met' and that it gives you 'ability to manage change' and 'improve collaboration between customers and suppliers'. This is important – we break down that barrier where people used to post a document to their customer, full of requirements: now the customer helps to write them. And finally it helps to 'track progress and status' and 'to match cost against benefit'.

Rich traceability deals with and/or goal trees, to capture the 'How do you know that this satisfies that requirement?' of a problem. You can do the same for a validation strategy. Goal trees are not new – they're used in van Lamsweerde's KAOS, in safety case notations like Tim Kelly's GSN, in Reveal's satisfaction arguments, and so on.

Plain traceability just has N:M satisfies links, permitting impact analysis – if I change that, I better revisit those. But the bundle of links carries very little information about the satisfaction argument. Adding Assumptions helps; so does making explicit that a node is supported by ANDed or ORed requirements (visually, with '&' and 'or' labels). Adding Arguments and 'xor' gives still more expressive power. These are implemented in DOORS – a simple way is to use attributes, and a more complete way is to use 'establishes' (1:1) and 'contributes to' (N:M) relationships, which can be displayed in a table-like view showing requirements, and/or, satisfaction, and contributing requirements alongside each other. An explorer tool allows arguments to be navigated and examined. This approach captures rationale, exposes it to peer review (which is critical for quality) and gives greater confidence in meeting objectives. Most new DOORS customers in the UK will use rich traceability at least in a simplified way.

I**an Alexander** (Scenario Plus) spoke about moving towards automatic traceability in industrial practice. He described the use of three simple tools (Scenario Plus add-ons for DOORS) in a train control box project: a use case linker, a dictionary term linker, and an exporter to create a hypertext from a use case model.

The effect of these is to speed and simplify the creation of requirements and to build a shared understanding (via the scenarios, and the agreed definition of technical terms) of the system approach. He illustrated how these worked in practice, arguing that having a set of documented stories and terms was a step-change improvement in industry.

The hard remaining task is to find ways of automating the linking of stories to requirements (argued Jeremy Dick): linking scenarios to tests is simple semi-automatically (with Scenario Plus) as testers can select and compose chunks of test cases very quickly using the existing tools. In a well-ordered project starting from scenarios, it is possible to copy-and-link to initialize the functional requirements with 1:1 links; if as in the railway project, the requirements already exist (in part), manual linking is today the only possibility.

Steve Riddle (University of Newcastle) spoke about enabling traceability. Current tools like DOORS and RTM could help but industry was (as previous speakers had mentioned) reluctant to move from paper-based methods. Things have moved on little in the decade since Olly Gotel and Anthony Finkelstein

'identified the crux of the traceability problem as "the inability to locate and access the sources of requirements and [pre-requirements specification] work"; they contended that this was a major

contributor to the other classic (and still current) issues: out-of-date requirements; slow realisation of change; and poor reuse.'

This is indeed 'dispiriting', but it shows that Traceability workshops have something important to achieve. Semi-automation is perhaps what we need – e.g. identifying what needs to be updated, discovering 'implicit' traces buried in documents, and so on, much as Jeremy Dick and Ian Alexander had suggested.

Research needs to focus on technology transfer, to be pragmatic, and to address the issues by staying in touch with industrial sponsors, developing prototype tools to be evaluated by systems engineers.

Stewart Higgins (Philips) said that this presented traceability as a tools problem, but in much of industry the issue was the process in which such tools might be used; it was about people and what they did. Tools alone would always fail.

Patricio Letelier (University of Valencia) spoke on a framework for traceability in UML-based projects. Each project needs its own traceability approach, with no consensus on what information to collect, or what it means. Current tools focus on making a traceability graph between pieces of text. Integration between requirements and software tools isn't ideal; e.g. tools such as ReqPro don't offer a framework for traceability – all there is, is support for use cases and documents.

However a generalized framework was possible, e.g. stakeholder isResponsibleFor artefact; feature tracesTo use case; component isVerifiedBy test case. Generalized artefacts can be very few: traceable specification, model, and test specification. These can then be applied by specialization to a particular project, e.g. a RUP project can have use case model, class model, and so on as instances of model. Work is now in progress to build a module supporting the configuration of traceability for Rational Rose. Obviously this is aimed at software projects.

Alexander Egyed (Teknowledge) spoke on reasoning about trace dependencies in a multi-dimensional space. "I see traces as the simplest and most trivial artefacts of all, mere arrows, things that point at other things."

(Hmm, but what about relationships between requirements? Could be dependency, or constraint, or conflict, etc: would these work transitively, wondered Andrea Zisman and others of us. If transitivity didn't hold, the approach wouldn't work. In any case it only applied to software because everything related to the code.)

There are lots of manual tools and techniques: it's essentially a manual process, and you have to worry about properties like precision, completeness, and being systematic. We'd like it to be automatic! Hardly anybody really puts in traceability.

Traces are transitive and bidirectional, and therefore we can deduce commonality, i.e. if A and B each trace to C, A and B trace to each other. And if A and B trace to Ca and Cb, A and B trace to each other if Ca and Cb overlap.

For instance, if you make a change to requirement A, it will affect Code C and (therefore) may well have an impact on the ability of requirement B to get what it wants. With a trace analyzer, you can follow through any depth of tracing to identify all the trace dependencies between requirements no matter how distant the relationship.

If you have test scenarios, for instance, you can use a trace analyzer to find out all the classes and methods (and even lines of code) it uses and therefore verifies, and then automatically identify trace dependencies between code and model elements (assuming you know the relationship between scenarios and models). You can reason about the connections – via the overlaps in the code! – between dataflows, classes, and use cases, which constitute different dimensions of the problem.

We wanted, said Egyed, certainty about correctness and complete relationships between models and code. We get some way towards this. It is a simple approach that works, provided only that model elements are clearly distinct (which isn't always so).

Elena Perez-Mi•ana (Philips Laboratories) spoke about issues on the composability of requirements specifications for a family of products (TVs) in consumer electronics. These are still managed today through documents. There are incompatible standards for TVs in the USA, Europe and Asia. The configuration of specifications is hard to manage as there is a matrix of applicability against competences. Currently, researchers all use several sorts of model to describe product family requirements, and we concur. A database approach allows different views of a common set of requirements. A Domain Object Model (a class diagram) defines shared concepts with agreed names. A Use Case model defines the functional requirements for the Product Family. Use Cases are much more stable than the User Interface and other details of individual products.

We think use cases (she said) are the most effective way of defining product requirements, but in abstracting up to family level we still want to be able to see individual product details; and to see points of variation between products – both of individual features and of interactions between features.

We want to identify the right amount of natural language for product family specifications, as we think that informality leads to more inconsistencies, but readability is essential. We need a semi-formal language to allow us to express how products vary (as Mike Mannion has described). And we need to express temporal constraints, whether through UML activity diagrams or a semi-formal language.

The meeting ended with plenty of time for discussion. George Spanoudakis introduced the session with a 'biassed' list of open issues:

- Types of relations required – project/enterprise specific? Domain specific? With generic ontology? (e.g. impact traces (satisfies, verifies) vs others (justifies, is defined by)); Across all artefacts?
- Ways to automate the generation of relations: feasible? Ambiguity? Correctness & completeness – what can we tolerate? Quality assurance mechanisms? Can you create traces without automation, and can you trust them with it?
- Process issues: changes needed in human management? enforced while you develop specifications? Granularity of traceability? Or a posteriori, given the artefacts already? Traceability to product family specifications?
- Tool issues: interoperability and heterogeneity (in a distributed environment)?

We argued about much of all this. We didn't like the Gotel/Finkelstein Pre/Post-traceability distinction – it's all the same stuff, we felt.

Jeremy Dick suggested that all true traceability relationships are many-to-many.

Installation of traces is far more difficult on brown-field, legacy system projects; and automation would be valuable, if only it generated trustably correct links.

Andrea Zisman defined Recall as # of detected links / # of possible links; whereas Precision was # of correctly detected links / # of detected links. A precision of 80% is, we felt, like an Internet translation of a foreign document: 20% incorrect can be really distracting, though we may be able to correct wrong links; on the other hand, missing links are a real problem.

Traceability is vital for change management and impact analysis; but it is not sufficient for this, as it says nothing about new requirements. And a simple 'suspect links' flag can indicate only that an item might need to be changed; not that anything that matters has occurred. Wim Dekker suggested that a traceability link is one that cannot be automated! Alexander Egyed said that if you wanted traces from models to be accurate, then you had to have a fine grain. Coarse links would be inaccurate. Stewart Higgins said that inaccurate traces would lead to expense as they'd need to be checked by hand. Alexander Egyed said that existing automated methods could do good and useful work already, saving money.

We concluded that Ariadne, who gave the hero Theseus a reel of thread to find his way back from the Minotaur's labyrinth, was the founder of Traceability.

RE-Papers

Requirements Engineering Research Prototypes – A short survey of tools

Frank Houdek
DaimlerChrysler AG

Presented at ICRE'02/RE'02

Research prototypes are an essential element of ongoing research activities as they demonstrate the applicability of complex and innovative approaches. In one session of this year's International Requirements Engineering Conference (RE 2002) six research prototypes have been presented and discussed.

The **RETH** tool (Requirements Engineering through Hypertext) by **Herman Kaindl** is designed to support requirements engineers while writing specification documents. These documents have to adhere an underlying structure that has similarities to use case descriptions. There are two ways of using the tool: (1) the tool can just be used as an editor where information is edited in any area the author wants to modify or (2) the author uses the build-in process assistant who guides the author step by step. The main feature of the tool, however, is tracing support. Using implicit trace information given by just-the-same-word the tool interferes traces. Here, the tool can distinguish between general terms (e.g. bank) and more specialized terms (e.g. bank account) while trying to trace on the most specialized level.

Unfortunately, the tool isn't freely available.

XlinkIt, presented by **Anthony Finkelstein**, is a commercial tool (www.xlinkit.com) that bases on research performed by University College London. Its main goal is to support consistency between heterogeneous information sources. This is achieved by integrity checks and report capabilities. Using XML, the user can specify consistency rules (first order logic) that can check both data and link. Although the tool is not primarily intended for specification documents, it can be applied to them as well. Thus checks like "does every requirements have at least one test case" or "does every use case description mention at least one actor" can be checked.

The **SeeMe** editor by **Kai-Uwe Loser** is a modelling and presentation tool. It is intended to model socio-technical systems (i.e. processes which involve both technical and human-based activities and their interaction). A major goal of the tool is the ability to express uncertainty. If, for instance, a collection of activities is assumed to be incomplete, this is expressed by means of special symbols. Information can also be presented in various levels of detail. Every element is equipped with a "mouse-hole" symbol. By clicking on this symbol, information that is part of the element can be shown or hidden. Thus even complex processes can be presented easily. There is no “dive-in” mechanism, i.e. by clicking on an element no new window pops up. Instead the sub-elements of the current element are show inside.

Avoiding the “dive-in” concept is also the driver for the **Adora** tool presented by **Christian Seybold**. The tool deals with the observation that dive-in mechanisms intensify the lost-in-detail problem by hiding the context the currently presented information is embedded in. Thus the idea is to present all information in one window. If a user wants to see what’s inside an element, the details are presented on demand. At the first glance such a technique would result in very overloaded and large drawings. To avoid this problem, the Adora tool uses an idea similar to the fisheye lens concept. Information, which has a greater distance to the currently focused elements, is drawn smaller. So, the context is still visible. To avoid distortion, a linear transformation is used which maintains the relative position of the elements among each other. Currently, the tool uses the language Adora, but other hierarchical notations can be implemented easily.

The **NIBA** workflow tool, presented by **Christian Kop**, implements a challenging task, the transition of German natural language text to UML class diagrams models. In German, the average complexity of “simple” sentences is higher than, for instance, in English. This complicates the task of parsing arbitrary input text. The tool provides a front-end that provides support to the various steps from a natural language specification towards an UML class diagram. Under the front-end several proprietary tools are combined, e.g. an editor, a text analyser, and Rational Rose to display the final UML diagram.

The **REM** (REquirements Management) tool by **Amador Durán Toro** is a freely available general-purpose requirements management tool. It was mainly build to support education, as commercial tools are quite expensive (even when considering the special conditions for universities). It provides four information areas. One for user requirements, one for developer (or system) requirements, one for conflicts, and one for change requests. The windows based tool provides a neat front-end. It is highly configurable as it uses XML to store data and style-sheets for display and export. Thus, individual needs with respect to optical appearance (views) can easily be implemented. The tool offers many capabilities a requirements management tool should offer, like automatic history. The tool is freely available at *klendathu.lsi.us.es/REM/*.

Altogether, the RE02 research demo session provided an in-depth insight in ongoing research activities from a tool perspective. My personal favourites amongst the presented tools are REM (as is provides – mainly for educational purposes – a quite powerful tool with some interesting features, e.g. using XML style-sheets for display and export) and NIBA (as it deals with a really ambitious task, i.e. understanding and transformation of natural language text). However, all other tools showed some interesting concepts and features as well.

Requirements Tracking: A Streamlined Approach

James E Archer
Titan Systems

Introduction

It is both a truism and a cliché that software systems should be based on user requirements. Nonetheless, formal requirements tracking methods have not lived up to their promises and those who advocate "agile" methods have largely abandoned requirements tracking claiming they are burdensome and counter productive. What went wrong?

The problem seems to lie in the diverse nature of software requirements. What is needed is a way to categorize requirements and handle each according to its inherent properties.

The primary goals of the Streamlined Approach are to document requirements in ways that are practical, flexible, maintainable, trackable, traceable, and support incremental delivery of software products (Deliverables). The method must be usable on real projects with time and budget constraints. It must scale both up and down. Note: Some of the Items that satisfy these meta-requirements have been annotated below (in parentheses).

The Requirement Approach

The Streamlined Approach is a midrange approach that rejects both a high degree of formality (such as requirements matrices) and informal techniques (such as "user stories"). It is a database centric rather than a document centric approach because only a database approach can provide the desired degree of trackability. Furthermore, the classification of requirements requires a fine-grained approach. The basic requirement unit is the requirement statement, not the document or the use case. Typically, requirements statements are from one to four sentences long.

Requirement Domains

Most requirement methods distinguish between Business Requirements and Software Requirements and this one is no exception. Business Requirements express the reason for writing the software and may of any level of abstraction depending on the project .This provides the flexibility to handle projects regardless of their scope and nature. Software requirements are attached to one of more software Deliverables, which they specify.

Requirements Hierarchies

Business Requirement Hierarchy

In order to assure the method will scale both up and down, two hierarchies are defined, one for Business Requirements and one for Software Requirements. These are tree structures with the entire system represented at the root node and the desired level of decomposition below. The Business Requirements Hierarchy and Software Requirements Hierarchy provide the appropriate levels of abstraction for the corresponding requirement domain.

Similarly there are Requirement Hierarchies for both Typed Business Requirements and Typed Software Requirements. Typed requirements are defined below.

Below each node of the Business Hierarchy are two more levels: Business Functions and the Business Requirements themselves.

A simple example should clarify this:

- Root: Cadastral Automated Request System (CARS)
  - Node: Phase 1
    - Function: Repository
      - Requirement: The system shall provide a central repository for Survey Request data.
      - …
    - Function: Data Entry
      - Requirement: The system shall allow entry of data from multiple locations.
      - …
    - Function: Security
      - Requirement: Authorized reviewers will review all data before it is entered into the central repository.
      - …
  - Node: …

Software Requirement Hierarchy

The Software Requirements Hierarchy has the same general structure as the Business Requirements Hierarchy with the Business Function replaced by the Software Deliverable. The Software Hierarchy is usually deeper and requirements far more numerous then the Business Hierarchy. A truncated example:

- Root: Cadastral Automated Request System
  - Node: Phase 1
    - Deliverable: Monthly Agency Report
      - Software Requirement: The report will list the number of Surveys last month broken down by Agency.

Requirement Linkages and Tracing

The rule for requirement tracing is simple: Each Concrete Software requirement (defined below) must be linked to a single Business requirement (called Concrete) which it satisfies. In other words, there is a one to many relationship between Concrete Business Requirements and Concrete Software Requirements. No attempt is made to use a many to many matrix approach to requirements tracing. In the case that more than one Business requirements applies, one is chosen at the discretion of the analyst. It is sufficient to know that a Software requirement has a justification. This simplifies the model and enhances maintainability.

Requirement Types

The four basic requirements types are Concrete, Indeterminate, Abstract, and Typed.

Software Requirement Types

The key concept of the Streamlined Approach is that of the Concrete Software Requirement. The two primary characteristics are:

- The requirement is linked to a single Software Deliverable which it specifies.
- There exists a well defined completion criterion for determining when the Deliverable satisfies the requirement.

Secondary characteristic are links to:

- A (Concrete) Business Requirement (Traceable)
- A task in the schedule (Trackable)
- A Completion Status (Approved, Active, Tested, Completed, etc.) (Trackable)
- A Software Build (incremental delivery)

Note that if the primary characteristics are satisfied, the secondary ones can always be satisfied.

If a Software requirement satisfies the first primary characteristic but not the second, it is called Indeterminate.

If a Software requirement is not linked to a Software Deliverable but applies to every Software Deliverable below a node in the Software Hierarchy, it is called Abstract.

Finally, if a requirement is relevant to some aspect of the Software (called it's Type) but does not meet any of the above criteria, it is called Typed.

Business Requirement Types

A Business Requirement is called Concrete if it is linked to a Concrete Software Requirement which satisfies it.

A Business Requirement is called Abstract if it applies to all Business Functions below a node in the Business Hierarchy.

Finally, a Typed Business Requirement parallels the definition of a Typed Software Requirement.

Typed Requirements

Requirements should always be related to the non-typed hierarchies whenever possible, But Typed requirements are important because they allow freedom in grouping requirements that do not otherwise fit. When using typed requirements, it is often useful to extend the notion of a requirement to any aspect of interest (e.g. a list of users). This allows the system to be used as a general-purpose repository for system information (practical).

Conclusion

Many other issues could be addressed but are beyond the scope of this paper. These include:

- Attaching documents to requirements for schedules, use cases, graphics, etc (flexible)
- Reporting functions (trackable)
- Data base model
- Prototype system developed in MS Access
- Using the method to support incremental prototyping (flexible)

Examples

Domain	Type	Node	Function / Deliverable / Type	Requirement
Business	Concrete	CARS/Phase1	Repository	The system shall provide a central repository for CARS data.
Business	Abstract	CARS/Phase1	---	The Phase 1 release will be packaged to keep all Indian Request data off local machines.
Business	Typed	CARS	Field Offices	Coos Bay Oregon
Software	Concrete	CARS/Phase 1	Field Office Report	The reports shall list the number of surveys broken out by Field Office.
Software	Indeter-minate	CARS/Phase 1	Field Office Report	The report shall make attractive use of color.
Software	Abstract	CARS/Phase1	---	The phase 1 system will be packaged on diskettes.
Software	Typed	CARS	Project Phases	Phase 3 will be Internet enabled.

Requirements, Myths and Magic

Ian Alexander
Independent Consultant

Saying what we want isn't new; people have wished for a better life since the dawn of time. Western culture has thrown up several myths about asking for and getting what you want; but all of them are rich in warnings which may be salutary reminders. Let's just look at three of them.

One of the Greek Myths of asking and getting is the sad tale of Pandora, whose name means 'all-giving', itself a warning. She was created by Zeus when he discovered that Prometheus had stolen fire from the gods, and was hence able to do many creative and destructive things that had until then been divine privileges. Men 1, Gods 0. Zeus at once ordered Hephaestos, the smith of the gods, to create the image of a beautiful maiden, Pandora. Athene dressed the treacherous thing with lovely clothes. Hermes filled a vessel to accompany it with all the evils and troubles the gods could devise. All the mortals and immortals were astounded by Zeus' ingenious answer to man's attempt at becoming equal in power to the gods: for here was the start of the race of women, "that threatening wile against which men are defenceless"[1]. Men 1, Gods 1. (Political Correctness, 0.)

Worse was to come. Prometheus (whose name means 'Provident') warned his brother, Epimetheus ('Heedless'), not to accept any gift from the gods. Zeus sends Pandora to Epimetheus as a gift, and – true to form – Epimetheus accepts. Pandora opens the vessel and all the evils escape and spread throughout the world, leaving only Hope behind. Gods 2, Men 1.

Pandora's vessel came with a prohibition – it was not to be opened (just as the fruit of the tree of knowledge in Genesis was not to be tasted). We've all seen projects that went hell-or-high-water for new technology without too much cautious work on discovering requirements or evaluating risks, and all the evils – delay, cost overrun, blame, slashed functionality, bugridden software – sprang from Pandora's box and could indeed not be put back again. In the myth, one of the evils – Hope – stays behind in the vessel: it's the sting in the tail, as the project manager continues to produce hopefully optimistic plans that show that all is well while everybody knows the project is sliding helplessly to the right, thus guaranteeing continued anguish while the watching Gods chuckle. One form of this hope is that a Method or Tool (or even a Famous

[1] C. Kerenyi, *The Gods of the Greeks*, 1951; Thames & Hudson paperback edition 1974.

Consultant) can be found to bottle the spirits, to slay the demons with a magic silver bullet, to ward off the forces of evil with an amulet. The unscrupulous will always be on hand to sell Amulet-ware (version 5.1).

Perhaps the best-loved story of specification is Pygmalion. Who? Pygmalion is ancient Greek for Dwarf or Pygmy, i.e. the short and ugly fellow who couldn't get a girlfriend. Being clever and skilful he decides to make one instead, and carves a beautiful woman in the finest white marble from the island of Paros. At once he falls in love with the statue, and longs for the gods to take pity on him and breathe life into it. At last they relent, and she comes to life as the perfect nymph; the only problem is that she isn't particularly keen on the Pygmy himself. Bernard Shaw's play (also called Pygmalion) has an English gentleman breathing elegance and elocution into a market-girl, with the same result as the Greek myth, retold (again) as My Fair Lady. Yet another variant on the theme is Mary Shelley's Frankenstein, where the mad inventor succeeds in playing God, only to find that his long-sought creation becomes a monster. And, yes, we can all think of projects where the requirements seemed to be all right at first, but… Pygmalion is the definitive myth of technology, computing, robotics, artificial intelligence, genetic engineering and all other attempts to specify better systems. Your system may work perfectly – but not deliver the results the users wanted or expected. The enduring popularity of this myth says something about public attitudes over the millennia to technology and technologists.

Finally, and most directly for requirements people, there is the folktale of the Three Wishes. There are any number of versions, notably including Saki's; the genre is mentioned (but not retold) by Gause & Weinberg[2]. (Readers of delicate disposition are advised to skip to the last paragraph now.)

A poor soul wishes desperately for better things. In a tavern he overhears a conversation about the Monkey's Paw and is drawn unsuspectingly in. A strangely-dressed man explains that the thing allows its bearer to obtain whatever he wishes for! Visions of wealth and luxury and eternal youth float before the poor man's eyes. He manages to obtain the amulet and makes his first Wish – a great treasure of 100 gold crowns. The very next day a messenger dressed in royal attire appears at his tiny hovel, announcing that with deepest regret they have to inform him that his son has been killed working for the king, who has sent a present of 100 crowns to mark his feelings.

Horrified, and aghast at what he has done, the poor rich man wishes his son was alive again. The very next day, a sister arrives from the hospital and tells the man that his son is alive, but still terribly mutilated, unable to help himself, and that if he can care for the patient he can come and collect his son. He does so, and he and his wife work hard every day looking after their living-dead boy. Things are so heartbreakingly bad that eventually the sad man wishes silently to himself that his son were dead again, and at once the boy dies. The stranger from the tavern who had been talking about the wonders of the Monkey's Paw appears from nowhere, and asks if the man was happy with his Three Wishes. The man speechlessly returns the hated amulet to the stranger.

All of these myths agree that it is indeed possible to specify systems of great power, and to have them built; but that getting what you really want is another matter entirely. Have you ever tried to frame your Three Wishes, if you could have anything you asked for? It isn't easy to specify your requirements.

Happy Christmas.

Generation Of Requirements From Scenarios

Janos Korn
London School of Economics

Background

At the IEE colloquium 'Scenarios through the systems life cycle', 7the Dec 2000, I presented a brief paper titled 'Scenarios through linguistic modelling'. Later I talked to colleagues whom I had met at the colloquium and consulted some books and journals on the subject of 'requirements'. I found that there was a certain lack of consensus about what was meant by the term 'requirement' in a design context. Also, writers used a variety of methods like data flow diagrams and systems dynamics borrowed from systems science for modelling scenarios. UML diagrams and use cases although more comprehensive, appeared to follow a similar line. A great deal of resources seemed to have gone into software development. These methods have superficial empirical/theoretical foundation, they have no basis in existing branches of knowledge, make no explicit reference to changes of state in time and their use of 'properties' is scarce. There did not seem to be a clear explanation of how requirements could be deduced from scenarios. The general opinion appeared to be that requirements were elicited from a client or a user.

The problem was then to develop a design method which would show how to generate requirements with a methodical representation of scenarios as an integral part. I have published a number of papers dealing with representation by means of linguistic modelling (LM) which does not seem to have the problems briefly referred to above. The objective of this short and fairly informal presentation is to summarise the current state of 'generation of requirements from scenarios'.

[2] *Exploring Requirements*, Dorset House, 1989.

Features of the proposed method

Features of the method of generation of requirements here introduced, are :

1. Representation of a scenario by LM begins with a story or a narrative describing some activities which is then subjected to *linguistic analysis* to convert linguistic complexities into a homogeneous language of one- and two-place, simple sentences which are then sorted into sentences with energetic and informatic interactions. These are represented as a diagram which displays the *topology* of a scenario from which series of predicate logic forms can be deduced. These forms can carry *uncertainties* associated with behaviour of human and/or other kinds of components together with computations as needed. The logic forms represent sentences which express interactions as *skilled power* (with appropriate energy) or *influence* (carrying information) through dynamic verbs like 'to drill' (a hole) or 'to notify' (the public that….). The diagram shows explicitly the outcomes of activities in a scenario as properties when progressions of states *in time* terminate.

2. Requirements are seen to originate from properties which describe the outcomes displayed by an *'initial representation'* of a scenario. These outcomes are considered unsatisfactory and are seen to need specific changes.

3. We introduce the notions of *'dynamic' and 'stative'* requirements. The former is needed to describe a cause of a *'changing property'* of a changing object, the latter are derived from *'quiescent properties'* which are relevant to change and accompany a changing property. *Getting from one place to another* may require a vehicle, in particular a taxi, but the *number of passengers* determines its size.

4. The *first dynamic requirement* calls for the cause of a changing property and points to the selection of one of the *'generalised products'* : artifact or energy/medium for 'physical' or information/medium for 'mental' change of state. Product is needed to generate the interaction or process to induce the change of property in a chosen changing object. The *second dynamic requirement* calls for the delivery of product to the changing object by 'interacting objects' (IO) operating purposively and points to the selection of IO. The *stative requirements* are the consequences of quiescent properties and point to the realisation of specific properties which supplement the changing property.

5. The emerging product and IO are inserted into the initial representation of a scenario leading to a *'subsequent representation'*. This latter appears as a diagram which can be subjected to the same kind of analysis as the initial or any other representation of a scenario by LM. *A uniform approach has emerged*.

6. Properties of any product are described by declarative, simple sentences. Thus, their *effectiveness* can be worked out in a unified way whether a product is an artifact, energy or information.

An example

A. Initial representation of a scenario

Scenario : 'George thought that he was badly served in a restaurant. A decisive man, he immediately picked up the phone and complained sharply to the manager who answered the phone, that he had a long wait for food which felt cold when it finally arrived and its taste was poor. The manager was a sensitive man who preferred to see complaints in writing, with a sense of maintenance of quality of service and food : he expected a waiting time to be less than 10 min, the food to be around 25^{o}C and its taste to score at least 7 out of 10. He considered the situation with a view to contact George'.

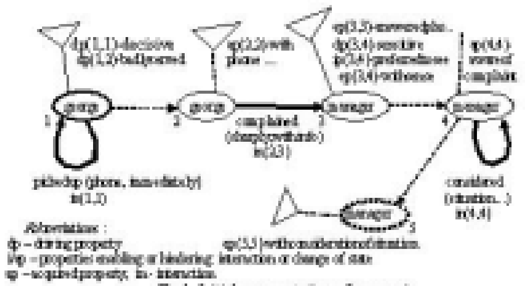

dp(1,1)-decisive
ap(1,2)-badlyserved
ap(2,2)-with phone ...
george
george
manager
manager
manager
complained (sharply, with ...)
in(2,3)
pickedup (phone, immediately)
in(1,2)
considered (situation...)
in(4,4)
ap(5,5)-with consideration of situation
Abbreviations :
dp – driving property
h/ep – properties enabling or hindering interaction or change of state
ap – acquired property; in - interaction

Fig.1. Initial representation of a scenario

After linguistic analysis the scenario is diagrammed in Fig.1. with outcome 'manager with consideration of situation'(ap(5,5)). This may be considered unsatisfactory by 'george'. He can set his objective as 'to complain so as to get compensation from the manager'.

B. Subsequent representation of a scenario

Having considered the initial representation of a scenario, the design process is concerned with development of a subsequent scenario. We use the scheme in Fig.2. as a guide.

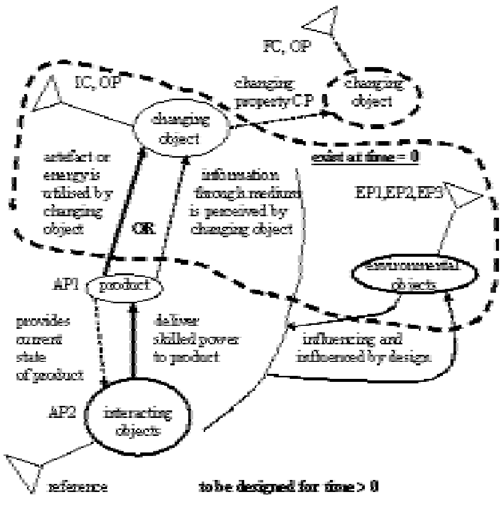

FC, OP
IC, OP
changing property CP
changing object
changing object
exist at time = 0
artefact or energy is utilised by changing object
OR
information through medium is perceived by changing object
EP1,EP2,EP3
environmental objects
AP1
product
provides current state of product
deliver skilled power to product
influencing and influenced by design
AP2
interacting objects
reference
to be designed for time > 0

Fig.2. Design scheme

Quiescent properties are related to : changing property - CP, changing object - OP, environment - EP, alternatives – AP. Initial and final properties/conditions – IC, FC.

Analysis of change (An empirical exercise)

Changing object (a noun) – manager

IC (sentence with a stative verb) – Manager is unaware of complaint

FC (sentence with a stative verb) – Manager is aware of complaint with request for compensation

Changing property (sentence with a dynamic verb) – Manager becomes aware of complaint with request for compensation

Quiescent properties (sentences with stative verbs) –

CP,1 – complaints go directly to manager
OP,1 – manager is a sensitive man
OP,2 – manager prefers printed communication
OP,3 – manager's expectation for waiting time is to be less than 10 min
OP,4 – manager's expectation for food is to be around 25°C
OP,5 – manager's expectation for food is to score for taste at least 7 out of 10
EP3,1 – consumer association encourages customers to ask for reasonable compensation
OP,6 - manager prefers communication to be private

AP1,1 – manager prefers to see complaints in writing
AP2,1 – george has little confidence in anybody
EP3,2 – communications are carried by post

Scheme for use of changing and quiescent properties to generate requirements and to choose product and IO (A theoretical exercise)

1. **First dynamic requirement** (to indicate cause of change):
'Sentence with changing property'- *requires* – 'Sentence of cause with (*generic* initiating object + dynamic verb as infinitive complement)'

2. **Selection of product** (to affect choice of product): 'Sentence of cause with (*generic* initiating object + dynamic verb as infinitive complement) AND (AP1, EP1 and 3)' – *selects* – 'Sentence with (*name* of product + dynamic verb as infinitive complement and its function expressed as adverbial)'

3. **Second dynamic requirement** (to indicate cause of delivery):
'Sentence with (*name* of product + dynamic verb as infinitive complement and its function as adverbial)' - *requires* – 'Sentence describing delivery with (*name* of product + qualified, dynamic verb as infinitive complement'

4. **Selection of IO** (to affect choice of IO): '(Sentence describing delivery with (*name* of product + qualified, dynamic verb as infinitive complement) AND (AP2 and EP2 and 3)' – *selects* – 'Sentence with (*name* of IO + qualified, dynamic verb as infinitive complement)'.

5. **Stative requirements** (to generate particular properties of product and IO): 'Sentences with quiescent properties' – *require* – 'Adverbial and adjectival sentences with (*specific means* + stative/dynamic verb as qualified, infinitive complement)'

Application of the scheme leads to :

Dynamic Requirements

1. 'Manager becomes aware of complaint with request for compensation' *requires* 'Means as encoded medium which is to convey complaint with request for compensation'

2. 'Means as encoded medium which is to convey complaint with request for compensation' AND 'AP1,1 = (Manager preferred to see complaints in writing), EP3,1 = (Consumer association encourages customers to ask for reasonable compensation)' *selects* 'Letter (medium : white paper) with information is to be carried to manager so as to convey the complaint with request for compensation'

3. 'Letter (medium : white paper) with information is to be carried to manager so as to convey the complaint with request for compensation' *requires* 'Letter (medium : white paper) with information is to be carried to manager'

4. 'Letter (medium : white paper) with information is to be carried to manager' AND 'AP2,1 = (George has little confidence in anybody), EP3,2 = (Communications are carried by post)' *selects* 'George is to deliver letter (medium : white paper) with information to a post box' (**dp(1,12)** in Fig.3.)

Stative requirements

5. Adverbial sentences related to product and IO for **ip(7,8)** and **ip(1,10)** in Fig.3.

CP,1 – 'Complaints go directly to manager' *requires* 'Letter (medium : white paper) with information is to be in his intray'
OP,1 – 'Manager is a sensitive man' *requires* 'Letter (medium : white paper) with information is to point out deficiencies in service and food politely'

Adjectival sentences related to product and IO for **dp(1,1), dp(1,10)** and **dp(1,11)** in Fig.3.

OP,2 – 'Manager preferred printed communication' *requires* 'Letter (medium : white paper) with information is to be put in printer' (**dp(1,1)**)
OP,3 – 'Manager's expectation for waiting time was less than 10 min'
OP,4 – 'Manager's expectation for food is to be around 25^{o}C'
OP,5 – 'Manager's expectation for food is to score at least 7 out of 10 for taste'
EP3,1 – 'Consumer association encourages customers to ask for reasonable compensation'
require

'Letter (medium : white paper) with information is to be written as complaint : (**dp(1,10)**)

1. waiting time for food was 20 min,
2. food was lukewarm at 20^{o}C,
3. score for taste of food was 5,
4. a free meal is serve as compensation,

OP,6 – 'Manager preferred communication to be private' *requires* 'Letter (medium : white paper) with information is to be put in an envelope' (**dp(1,11)**)

Concluding points

1. We have outlined a design method embedded in LM which produces requirements leading to possibility of choice of product and IO. The method operates in terms of properties to which a particular product and a system can be fitted. The design procedure begins with changing and quiescent properties the identification of which is an empirical exercise.

2. A logical procedure originating from these properties has been shown. Involvement of a client or a user is essential as the agent with knowledge of a particular scenario.

3. Dynamic and stative requirements have been introduced. The use of the former is usual practice in engineering.

4. 1st and 2nd dynamic requirements expressed in generic terms to admit choice plus AP and EP properties are directed at selecting a product and IO, stative requirements refer to properties of product and IO once selected.

5. Properties of products (artifacts, energy, information) are expressed linguistically in a uniform way. Thus, their effectiveness can be evaluated uniformly (not included here).

6. The operation of the scheme in Fig.3. is governed by an 'objective'. This point has not been considered here. The manager's *decision process* whether to give compensation or not is not shown.

7. Carriers of EP properties bring into the scheme stakeholders and interested and affected parties.

8. The use of *generic* objects, AP and EP properties enables the entry of choice and creativity into the scheme.

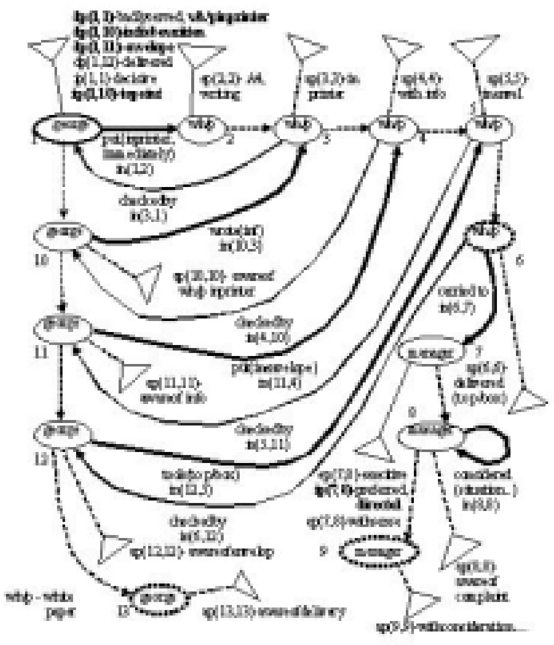


Fig.3. Subsequent representation of scenario

RE-Publications

Book Reviews

Writing Better Requirements
Ian Alexander and Richard Stevens
Addison-Wesley, 2002.
ISBN 0-321-13163-0

This book provides practical and detailed guidelines on how to write user requirements in a style that is understandable by business people and developers. The authors' approach to writing requirements addresses the "gulfs to be bridged" between: development and marketing, users and developers and staff and customers. They provide convincing examples of how even the simplest project has a complex set of stakeholders and how the requirements depend on identifying and involving these stakeholders. And the section on workshops is packed with hints on how to run a requirements workshop and get tangible input from all the stakeholders.

Scenarios are effectively used to identify, structure and link goals and requirements. Each goal is connected to a number of primary and exception scenarios and each scenario is made up of a number of atomic requirements.

The chapter on requirements writing gives advice on how to write good atomic requirements and also discusses common pitfalls like ambiguity, speculation and rambling and how to avoid them. Frequent exercises (along with sample answers) steer the reader to detailed consideration of the everyday problems of a requirements engineer.

I found the discussion of different names for requirements, constraints and goals to be a trifle confusing. But, to be fair, requirements engineering is an emerging discipline and terminology has not yet stabilised.

I recommend this book to anyone who would like a clear non-academic guide to what requirements writing is, or should be, all about and why it matters. The book is suitable for business people and developers. The authors have done what they set out to do; they have bridged the gap.

Suzanne Robertson

Requirements by Collaboration
Workshops for Defining Needs

Ellen Gottesdiener
Addison-Wesley 2002
ISBN 0-201-78606-0

It really isn't every day that someone writes a book that provides a completely fresh take on requirements. Ellen Gottesdiener has managed to do just that.

This readable and practical book is visibly based on a wealth of direct experience of facilitating requirements workshops, and her message is focused exclusively on the power of the workshop approach. This does mean that this book isn't for every project – if you are working on a subsystem under a strict contract, there is little scope for collaborating with stakeholders to work out the requirements together. To her credit, Gottesdiener carefully points out the limits of her approach, and she is admirably well-read.

Part 1 of the book steps through what a requirements workshop is, and what it should deliver; how to make a workshop succeed.

Part 2 looks at what goes on in a workshop, and offers an admirable range of strategies for organizing workshops, types of question to ask, how to facilitate, the logistics of organizing a workshop, what to do if things are difficult, and tools you can use to make things 'flow'. Proven groupwork patterns such as forming-storming-norming-performing are explained and illustrated; lower-level patterns such as 'decide how to decide' are documented in an Appendix.

Part 3 looks at how to design workshops, with case studies on what worked well and what didn't. The final chapter wisely discusses how to make the case for requirements and workshops to management, and how evaluate progress. The sections on the business value of requirements and how to 'surface' problems are excellent.

The book uses the language of Use Cases, and refers to other UML diagrams along the way. It also mentions software from time to time (and software is unfortunately emphasized in the cover blurb), but this is definitely a book that is useful in systems of all types – from civil engineering projects to personal music players.

There are parts of this book that benefit enormously from Gottesdiener's natural enthusiasm, and the simple fact that she is American. She is able to talk in a simple and effective way about making workshops fun; about drawing mandalas and using the 4 principles of the native American medicine wheel; even about using toys as prizes and holding warm-up exercises to get people into collaborative mood.

The helps – navigation diagrams, tables of techniques, further readings, bibliography, glossary, index – are excellent; Gottesdiener has applied her care for participants to her readers. For instance the further reading is never just a list; instead she says '..offers wisdom on..', '..is a landmark article that..', '..is a superb book that..' giving the reader a 3 or 4 line pen-portrait of each recommended reading.

Even engineers who have been running workshops for years will find new insights, conceptual tools and techniques in this lively and welcome contribution. Every requirements engineer should have a copy in a handy place on their bookshelf.

Ian Alexander

Introduction to Requirements Engineering

Ian Bray
Addison-Wesley 2002
ISBN 0-201-76792-9

2002 seems to be the year that Sam Gamgee returns from the War of the Rings to scatter the magic earth from the land of Lothlorien over the gardens of the Shire where we live – at least as judged by the profusion of new and varied books on requirements that have sprouted this year.

This book "is intended for the novice, in particular the undergraduate novice" and it should serve admirably for that purpose. Actually Bray is too modest: this is really quite a general introduction and it may make many an experienced practitioner reflect on what they do, and why. I'll also enter the too-familiar plea for systems not just software engineering; happily this book's message generalises perfectly well to requirements for systems of many types, despite Bray's arguments to the contrary in the Introduction.

Ian Bray has written an engaging, lively, and exceptionally clear book. His skill as a teacher plainly contributes to his ability to write and to discover requirements effectively. He has made an entirely fresh attempt at organising our domain, with some success. Better, he does not merely describe a list of techniques you can apply: the book steps through a credible process – in part 1, from elicitation to analysis to specification to validation; in part 2, through techniques for all of these – and shows how the different approaches contribute to getting the right system. Few other books do this; Sutcliffe's new book does, but in a far more detailed and researchy way. These are very welcome signs that RE is beginning to mature: something coherent can be said about each stage and also about different types of problem.

Bray is fresh and funny on the errors of the past: why did Structured Analysis fail? Why did people call SA approaches methodologies when these weren't even methods? For instance:

"Ironically reflecting the 'Victorian novel' problem that it was introduced to combat, a morass of documentation was produced, much of it modelling the detailed process of a pre-existing system. It is highly questionable whether [such a] *process* .. is a matter for detailed study anyway; the process of the *problem domain*, certainly, and maybe the *function* of a pre-existing system but that is not the same thing.

It might have been seen as a clue to the problem that sometimes much of this documentation was not used. After expending a great deal of effort .. on its production it was simply side-lined."

This is at once witty and serious, light and reflective, though we may wonder whether the novice will appreciate its subtleties.

How can we do better? Bray's recipe is a mixture of techniques for elicitation and analysis and modelling, including more than just a graceful bow to Michael Jackson's Problem Frames. Much of the Analysis chapter explains in detail how to select and apply Problem Frames – in fact forming a detailed and readable introduction to the topic. He's very good on things you probably thought you knew, like Decision Tables and how to optimise them –- the reader is invited to try to make the simplified table smaller, which is possible (I did it) but which (surprise, surprise) makes it harder to understand. And like Kovitz, who is also one of Bray's heroes, he's good on making requirements readable:

"[Avoid] Duckspeak – meaningless padding and tautologies, e.g. 'The lift request validation function will validate lift requests'."

Unfortunately the book also contains a few mistakes. Tacit knowledge does not just mean stuff 'considered too obvious to mention'; it's to do with what 'we know but cannot tell', such as skill, as Polanyi says. Reviews do not need to walk through documents – it is often better to walk through the change requests instead. Several diagrams risk confusing "the novice" with wrong labels, e.g. Figure 4.40 swaps the reality and machine boxes. It isn't true that RE happens at the start of software projects; it is strongly connected with acceptance and verification, not to mention tracing requirements during partitioning into separate hardware and software subsystems. Design constraints are far from rare as claimed – at least in railway, telecommunications, and aerospace, to name just a few domains; indeed, non-functional requirements often make up well over half the specifications. It seems peculiar to classify safety, security, reliability and durability as 'performance' when people typically use that term to cover speed and capacity requirements.

The lack of system focus is revealed by the absence of a discussion of traceability, which to industrial ears is the crucial thing that enables us to get our requirements met. Indeed, atomic requirements are dismissed as those

"that may be expressed in one sentence and which are at the lowest level of abstraction short of detailing the physical details of the interaction… Perhaps the terms are best avoided."

But no, we trace back to individual 'atoms' to ensure we have met them and can verify (e.g. test) them. We know we don't have an atom when we can see there are two things to be tested. It matters very much when we want to ensure that our subcontractors have indeed done every single thing we needed.

Happily, Bray more than makes up for this with well-crafted and often witty explanation and discussion. The organisation of the book, the glossary and other helps are all excellent. The book's value on undergraduate courses is enhanced by exercises at the end of each chapter in Part 1 ('The Topics'), though there are none in Part 2, and no answers seem to be available. For instance, the reader is invited to complete a process model diagram; to match ten descriptions to ten RE terms; to classify 25 statements as problem domain, commercial constraints, design constraints, functions, or performance; to devise a problem frame for a small problem.

There are also good examples, which are both used throughout the text and then listed out to a reasonable degree of non-repetitive completeness in Part 3. In the space of fifty pages this describes four systems with elicitation plan, elicitation notes, requirements, and specifications (incidentally, this is a characteristically clear and workable naming convention). These show students (and practitioners, perhaps) how the products of each activity help to make the requirements clear. The range of examples is interesting – yacht racing results; the dreaded lift controller; a drilling machine's data file translator; and a Petri net diagramming tool. Happily, at least two of these are certainly system problems.

Teachers of university courses will find this a serious (if unusually readable) contender for the role of introductory text. There aren't many existing rivals; Sutcliffe's excellent book is intentionally far more advanced; Kotonya and Sommerville has a strong slant towards viewpoints and conflict resolution (scarcely mentioned by Bray); Alexander & Stevens isn't intended as a university text and doesn't cover analysis, though it is introductory and has detailed exercises (with answers); the Robertsons' is actually quite suitable but is designed for industry.

More experienced practitioners may also find it useful as a light introduction to techniques that came into service since their graduate days. It's a very welcome addition to the literature.

Ian Alexander

Tell Me A Story

Roger C. Schank
Northwestern University Press, 1990
ISBN 0-810-11313-9

In this pioneering, enjoyable and insightful book, Schank leads the reader to start from scratch when thinking about what we mean by knowledge, memory, and intelligence.

His thesis is essentially that being human means telling stories:

"…all we know is embodied in stories."

We remember stories, and perhaps they are the key to how we organize our memories, which is to say our minds. We know what to do in a situation because we have a wired-in script that we have (maybe painfully) learnt – and we behave with skill in that situation because we can follow that story efficiently. We can intelligently plan and reason about alternative plans, because we can tell stories with happy or unhappy endings for each variation.

I hope this is enough context to make it quite evident that Schank is totally relevant to thinking about how to capture and organize requirements. His work on Scripts set the tone of much of the Artificial Intelligence research in the 1980's: what was then called "knowledge elicitation" has of course transmuted into "requirements elicitation" today. Schank consistently argued that people ran their lives by stories, such as the famous restaurant script – you go in, hang up your coat, get shown to a table, sit down, read the menu and so on. This was taken terribly literally as a precise set of rules at the time, which (of course) didn't work; but Schank's work has also always emphasized the vague and associative nature of human thought, such as the way we connect one story with another, or events and details with a story. This should have warned people not to be too rule-bound and literal.

Schank writes fluently, cogently, and with a sense of fun as well as of exploration -- he dares to break the rules (and across the cover of the book is a strapline that says "rethinking theory"). The rethinking has to reach back far into the past, indeed to Aristotle, whose view of reasoning was that it had to be theoretical and hence rule-based, *episteme* – and for the last two millennia, the theory of knowledge has been Aristotelian "epistemology". A competing view even in ancient times was that you could reason from instances or cases, *casuistry*, which like rhetoric and sophistry was not originally a pejorative term. Only since the 1980's has case-based reasoning come to be honoured again, and rulebases have come to be seen as hopelessly limited except in very tightly-delimited domains. It's almost funny that it has taken 2000 years to get back and fix mistakes made by the Ancient Greeks, but that's the way it is. For me, this is extremely important – it is a revolution in thought, and it goes a long way to explaining why traditional arguments always break down. Another (closely-related) part of the repair and replacement of outworn arguments is Polanyi's demolition of the idea that knowledge is conscious – much of the way that skill works is plainly tacit.

Given the contributions of Polanyi and Schank, requirements engineers – indeed, system thinkers and developers in general -- need to come to grips with the fact that people, even if highly skilled and even 'expert', cannot give us precise rule-like requirements. If we try, we will get bad and wrong requirements. Instead, we urgently need to start thinking and working in stories – scenarios, use cases, scripts, user stories, whatever. We can use these as hooks to retrieve connected ideas and build a shared understanding of what people need, by talking about what they actually do. Then we'll have requirements that are somewhat closer to reality.

The book looks at where stories come from and why we tell them; how we understand and index stories; how stories shape memory; "story skeletons", i.e. the structure of stories; the stories of different cultures; and the role of stories in intelligence.

There is a very short list of references, but Schank sprinkles the text freely with quotations and jokes, examples and stories from all over – if you don't know what Chutzpah is, you soon will from *Tell Me A Story*.

Everyone interested in writing better requirements should at once read this book. Academics will find many stimulating ideas for things that ought to be researched; practitioners will be able to reflect in an easy-to-read book on their daily work; students will get an insight into one of the leading minds in the field of human knowledge.

Ian Alexander

RE-Sources

For a full listing of books, mailing lists, web pages and tools that have appeared in this section in previous newsletters, see the RQ archive at the RESG website:
http://www.resg.org.uk

The requirement management place
http://www.rmplace.org

A good general resource for RE issues. Includes Alan Davis' Requirements Bibliography.

CREWS web site:
http://sunsite.informatik.rwth-aachen.de/CREWS/

An interesting collection of 72 papers (!) and a description of an ESPRIT project on co-operative requirements engineering with scenarios.

Requirements Engineering, Student Newsletter:
http://www.cc.gatech.edu/computing/SW_Eng/resnews.html

IFIP Working Group 2.9 (Software Requirements Engineering):
http://www.cis.gsu.edu/~wrobinso/ifip2_9/

Requirements Engineering Journal (REJ):
http://rej.co.umist.ac.uk/

RE resource centre at UTS (Australia):
http://research.it.uts.edu.au/re/

Volere:
http://www.volere.co.uk

Reduced rates are available to all RESG members when subscribing to the REJ.

Mailing lists

RE-online (formerly SRE):
http://www-staff.it.uts.edu.au/~didar/RE-online.html

The RE-online mailing list aims to act as a forum for exchange of ideas among the requirements engineering researchers and practitioners. To subscribe to RE-online mailing list, send e-mail to majordomo@it.uts.edu.au with the following as the first and only line in the body of the message:

subscribe RE-online <your email address>

LINKAlert:
http://link.springer.de/alert

A free mailing service for the table of contents of the *International Journal on Software Tools for Technology Transfer*.

RE-Creations

To contribute to RQ please send contributions to Pete Sawyer (sawyer@comp.lancs.ac.uk). Submissions must be in electronic form, preferably as plain ASCII text or rtf.

Deadline for next issue: 1st December 2002.

RE-Actors

The committee of RESG

Patron: Prof. Michael Jackson, Independent Consultant. E-Mail: jackson@acm.org.

Chair: Prof. Bashar Nuseibeh, Computing Department, Faculty of Maths and Computing, The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK. E-Mail: B.A.Nuseibeh@open.ac.uk.

Vice-Chair: Dr Alessandra Russo, Department of Computing, Imperial College, 180 Queen's Gate, London SW7 2BZ, UK. E-Mail: ar3@doc.ic.ac.uk

Treasurer: Dr. Neil Maiden, Centre for HCI Design, City University, Northampton Square, London EC1V OHB, UK. E-Mail: N.A.M.Maiden@city.ac.uk.

Secretary: Dr. Wolfgang Emmerich, University College London, Department of Computer Science, Gower Street, London WC1E 6BT, UK. E-Mail: W.Emmerich@cs.ucl.ac.uk.

Membership secretary: Steve Armstrong, Computing Department, Faculty of Maths and Computing, The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK. E-Mail: S.Armstrong@open.ac.uk.

Newsletter editor: Dr Peter Sawyer, Lancaster University, Computing Department, Lancaster, LA1 4YR, UK. E-Mail: sawyer@comp.lancs.ac.uk.

Newsletter reporter: Ian Alexander, 17A Rothschild Road, Chiswick, London W4 5HS. E-Mail: iany@easynet.co.uk.

Publicity officer: Juan Ramil, Computing Department, Faculty of Maths and Computing, The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK. E-Mail: J.F.Ramil@open.ac.uk.

Co-Publicity officer: Sebastian Uchitel, Department of Computing, Imperial College, 180 Queen's Gate, London SW7 2BZ, UK. E-Mail: su2@doc.ic.ac.uk

Regional officer & Chair for the North of England: Dr Kathy Maitland, University of Central England, Perry Bar Campus, Birmingham, B42 2SU. E-Mail: Kathleen.Maitland@uce.ac.uk.

Industrial liaison:

David Bush, National Air Traffic Services, UK. E-Mail: David.Bush@nats.co.uk.

Dr Sofia Guerra, Adelard, Drysdale Building, 10 Northampton Square, London, EC1V 0HB, UK. E-Mail: aslg@adelard.com.

Dr Efi Raili, Praxis Critical Systems, 20 Manvers Street, Bath BA1 1PX. E-Mail: efi@praxis-cs.co.uk.

Suzanne Robertson, Atlantic Systems Guild Ltd. 11 St. Mary's Terrace, London W2 1SU, E-Mail: suzanne@systemsguild.com

RE-Funds

Minutes of the 8th Annual General Meeting

Held at University College London
Wednesday 10th July 2002.

Chair: Bashar Nuseibeh (Open University)
Minutes: Wolfgang Emmerich (UCL)
Present: approx 100 members

Agenda:

1. Minutes of Previous Meeting
2. Chair's Report
3. Publicity Report
4. Treasurer's Report
5. Election of Executive Committee Members
6. Any Other Business

1. Minutes of Previous Meeting

The minutes of the 7th AGM in 2001, which had been published in RQ, were agreed as a correct record of the proceedings.

2. Chair's Report

Bashar Nuseibeh started his report by highlighting a number of events that were organized in the past year. The events were:

- Requirements for Mobile Systems, 21 November 2001, UCL, London
- Key Challenges in Safety Requirements Engineering, 27 February 2002,
- Praxis Critical Systems, Bath
- Recycling Requirements for Systems and Product Families, 24 April
- 2002, Imperial College, London
- Scenarios Work! 10th July 2002, UCL, London

Moreover, RESG had co-sponsored a number of events and that RESG members could therefore attend at a reduced admission fee. These included:

- Managing Evolving Requirements, 11 December 2001, The Royal Society, London
- Mastering the Requirements Process, February 18-20, 2002, London
- Requirements Modelling, Feb. 21-22, 2002, London
- RE'02 (www.re02.org) - 9-13 September 2002, Essen, Germany

Bashar Nuseibeh then thanked Pete Sawyer (Editor) and Ian Alexander (Reporter) for editing three issues of Requirements Engineering Quarterly (RQ24-RQ26). There had been only three issues because only 10 months had passed since the last AGM. RQ production and distribution has in the last year been moved entirely to Lancaster (from U Hertfortshire). Bashar encouraged members to write articles and send them to Pete Sawyer in Lancaster. He highlighted that there would be no formal reviewing process but that Pete Sawyer might work with authors if refinement of articles was required.

Bashar concluded his report by highlighting an industrial liasion team that has been established during the last year. The aim of the team is to ensure that RESG continues to address the needs of practising requirements engineers. The membership of that team included Efi Raili, (Praxis), David Bush (NATS), Suzanne Roberston (Atlantic Systems Guild) and Michael Jackson (Independent) and is open to welcome new members.

3. Publicity Report

Juan Ramil (Open University) reported on recent efforts to improve communication with the RESG membership. He reported that the RESG Web Site (http://www.resg.org.uk) had been transferred and was now hosted by the Open University.

Juan highlighted that the RESG mailing list that is being used to inform members and other practitioners about requirements engineering events currently has about 700 members, from both in the UK and overseas.

Juan welcomed suggestions from the RESG membership for improving the communication further and invited comments to be sent to info@resg.org.uk

4. Treasurer's Report

Neil Maiden (City University) in his role as Treasurer of the RESG reported about the current financial situation, which can be seen from the Profit and Loss statement below:

Balance in May 2001:	19,513.30
Receipts	
Group subscriptions	1860.00
Gold account interest received	917.07
Gross receipts from events	1832.28
Gross other receipts	270.65
RECEIPTS TOTAL	4880.00
Payments	
Printing and stationery	444.84
Postage and telephone	221.79
Gross event payments	1047.14
Gross other payments	33.10
PAYMENTS TOTAL	1,746.87
Balance in April 2002	22,646.43

Neil commented that the financial situation of the RESG was healthy and highlighted that it was possible to organize to continue to run events that were not necessarily making a profit.

5. Election of Executive Committee Members

The chair proposed the following individuals, who agreed to stand for the RESG executive committee:

Patron: Michael Jackson (Independent)

Chair: Bashar Nuseibeh (The Open University)

Vice Chair: Alessandra Russo (Imperial College)

Secretary: Wolfgang Emmerich (UCL)

Treasurer: Neil Maiden (City University)

Membership Secretary: Stephen Armstrong (The Open University)

Publicity Officer: Juan Ramil (The Open University)

Co-Publicity Officer: Sebastian Uchitel (Imperial College)

Newsletter Editor: Peter Sawyer (Lancaster University)

Reporter: Ian Alexander (Independent)

Regional Officer: Kathy Maitland (UCE at Birmingham)

Industrial Liaison Team: David Bush (NATS), Sofia Guerra (Adelard), Elena Perez-Minana (Philips), Efi Raili (Praxis), Suzanne Robertson (Atlantic Systems Guild)

The election was proposed and seconded. The motion was passed unanimously.

6. Any other Business

There being no other business, Bashar Nuseibeh closed the meeting.