# CS12 and ESUG 15, Lugano, August 27th - 31st, 2007

I stayed with friends in the Lugano area and had a great time swimming in the lake and exploring the high mountain valleys and cascades when I was not at the conference.

When Suzanne arrived on Monday, it chanced that most of us were not near the registration desk and the student volunteers who were manning it had not met her before, so led her through the usual intro: “Are you registered? What’s your name? ...” For the first time in a long time, one of her favourite joke catch-phrases, “Do you know who I am?” gained literal significance.

We were treated to a meal at the top of Monte Generosa mid-week. Lugano lake is at 271 metres, the top of the mountain at some 1700 metres. Fortunately we only had to walk the final 100m of the difference (not quite enough to walk off the meal’s many calories :-).

## Style

In the text below, ‘I’ or ‘my’ refers to Niall Ross; speakers are referred to by name or in the third person. A question asked in or after a talk is prefaced by ‘Q.’ (I identify the questioner when I could see who they were). A question not prefaced by ‘Q.’ is a rhetorical question asked by the speaker (or is just my way of summarising their meaning).

## Author’s Disclaimer and Acknowledgements

This report was written by Niall Ross of eXtremeMetaProgrammers Ltd. (nfr AT bigwig DOT net). It gives my personal view. No view of any other person or organisation with which I am connected is expressed or implied.

There was much activity in the Camp Smalltalk room, only some of which I learnt enough about to summarise (with possible errors) below. Inevitably, my notes treat my project in much more detail than others.

Likewise, the talk descriptions were typed while I was trying to keep up with and understand what the speakers were saying, so may contain errors of fact or clarity. I apologise for any inaccuracies, and to any participants whose names or affiliations I failed to note down. If anyone spots errors or omissions, email me and corrections may be made.

My thanks to:

- Katerina Barone-Adesi, Thorsten Seitz, Adriaan Van Os, Michael Prasse and John O’Keefe for working with me in the Custom Refactoring Browser project, and to Katerina (again) for taking notes of some Academic Track projects I missed
- Michele Lanza, his University colleagues and all the student volunteers, for a smooth-running conference.
- the speakers whose work gave me something to report, the ESUG organisers and the sponsors: see their logos on http://www.esug.org/conferences/15thinternationalsmalltalkjointconference2007

## Summary of Projects and Talks

I give the Camp Smalltalk 12 projects summary, then the ESUG and STIC activities reports (including the awards presentations and ceremony, and the Summer of Code report). Next I summarise the conference talks, sorted into various categories:

- Applications and Experience Reports
- Application Frameworks: Seaside, GLORP, Cairo
- Vendor Reports
- Compilation Research
- Java Connectivity
- Development Processes and Frameworks
- Utilities
- Miscellaneous

followed by the Research Track. I close with Other Discussions and my Conclusions. Talk slides are on http://csl.ensm-douai.fr/Esug2007Media/.

### Camp Smalltalk 12

Camp Smalltalk 12 ran for Saturday and Sunday before the conference, and during the conference breaks, afternoons and some evenings of the five conference days.

#### The Custom Refactorings and Rewrite Editor Usability Project

(See also my presentation in the Project Summaries section.) Katerina Barone-Adesi and I wrote a test for ‘extract with holes’: the ability to select text for ‘extract to self’ and then select within it those parts that were not to be extracted but instead made additional parameters of the method. Thorsten Seitz then joined us and began implementing a solution. After the CS, work will continue to add this behaviour to ‘extract to component’, etc.

While Thorsten progressed the new refactoring, I paired with John O’Keefe to fix some tests that were failing in VASmalltalk 7.5.2 on Linux. Line-end conventions and hard-coded values in RBConfigurableFormatter were the cause; we sorted them out. Meanwhile, Adriaan van OS made the ‘Remove Parameter’ refactoring visible in VASmalltalk. I also worked with John to rationalise our MethodWrappers implementation. (Longer-term, John will explore using VASmalltalk method-spy features to reimplement method wrappers using the same API.)

Michael Prasse works on a large VW system in which several sub-bundles implementing basic utilities are contained within more than one super-bundle. When he does ‘Find Class’ on a class in one of these sub-bundles, the Refactoring Browser is very slow because the full expansion checking code for the sub-bundle is run for each of its containing super-bundles. Michael implemented a variant in which the redundant expansion checks for a repeated sub-bundle were not done after its first container but instead reused in the others. He also built a tree widget for the class pane in the VW Refactoring Browser.

I also paired with Travis to see his ideas for Cincom’s VW RB and review what from our project could usefully be incorporated into the VW base.

See also my talk in the ‘Utilities’ section below.

#### Sport

Bruce Badger’s group ported Sport to Visual Smalltalk / Visual Smalltalk Enterprise (I learnt that some users still call it VS and see the E in VSE as an unneeded addition to the name :-). See Bruce’ talk for more details.

#### Seaside Applications

Martin McClure led porting the Seaside-based Gjaller application to GemStone (Gjaller is an issue tracker). Dale Heinrichs and Lucas Renglii helped me begin porting Pier to VW. Work was also done adding to Seaside’s port-enabling test suite.

#### Exubery

The Exubery group worked hard debugging a crash which they found was very close to the start of memory and eventually managed to solve.

## ESUG and STIC Activities Reports

### ESUG Activities Overview, Michele Lanza, Stephane Ducasse

Michele thanked all the people who had helped him, starting with the ESUG Board, especially Stephane, Noury and Serge. Marco d’Ambros organised the social event. Katerina Barone-Adesi led the student volunteers. Romain Robbes helped with travel information (as did I). Mauro Prevostini found local sponsors. Marcus Denker did the DVD. Christian Caggiano designed the logo. The general support of Richard Wettel, and Marisa Clementz and Christina Zanetti, was much appreciated.

Stephane then welcomed us all. He thanked Michele for the excellent organisation. He thanked the Prague sponsors and the sponsors this year in Lugano: Cincom, Instantiations and Gemstone, Georg Heeg, National Spaarfonds, Lifeware, metaProg, gai&partner and JPMorganChase.

Total ESUG attendance at Lugano was 109: 94 who pre-registered and 15 further participants who registered at the desk or attended through local sponsor packages. Stephane asked how many were attending their very first ESUG? Quite a few hands were raised. He urged us to use the student volunteers as the first point of contact for any questions about either the conference or the local area, and also urged us to help the students get in touch with the Smalltalk community by showing them our projects.

Anyone who wants to be one of the local organisers for ESUG in 2008 should tell him. Spain and Romania have never had a conference and they are likely candidates for 2008 on current plans, so he would be happy to here from anyone located such that they could assist local organisation there. (ESUG could go to Belgium in 2008 but has been there recently and often, and it is likely that ESUG 2009 will be in Lille.) Anyone wondering what helping with local organisation requires can read the description on page 3 of the ESUG 2005 report (http://www.esug.org/data/NiallsReport).

ESUG sponsors Smalltalk in various ways:

- ESUG can sponsor presentations of Smalltalk, i.e. pay travel expenses, etc. For example, Marcus manned the Squeak booth at RMLL.
- Via the Summer of Code, ESUG sponsors students to do projects.
- ESUG sponsors free Seaside hosting (handled by netstyle.ch).
- ESUG offers material for giving Smalltalk lectures. If you get a Smalltalk article printed in a magazine, ESUG will give you 100 euro (3 articles were sponsored in 2007).
- ESUG helps students who move to Smalltalk groups.

Attending ESUG is how you sponsor all this. Registering late for ESUG is a way of sponsoring more. :-)

**Promoting Smalltalk, Smalltalk Industry Council**
Georg Heeg presented the Smalltalk Industry Council. He showed the new STIC logo, created by Vassili Bykov. STIC has four purposes of which the first and second, to create Smalltalk awareness and to promote Smalltalk, are the focus for this year (the others are to attract skilled people into Smalltalk and to help form cross-dialect standards). STIC is an association under North Carolina law (which is strict; German is the worst and Georg loves French; the French law on associations is much more relaxed).

Q. Interact with SqueakFoundation? Yes we will talk to them.

They work with Smalltalk user groups but STIC is concentrating more on attracting strangers to Smalltalk.

In 2008, Smalltalk Solutions will be in Reno, Nevada at the Grand Sierra Resort (www.grandsierraresort.com) in mid-June. See the slide to get motivated by pictures of the Nikki beach and the bar (there was also a picture of the lecture hall but that was only a quarter of the pane; let's concentrate on the essentials :-).

James Robertson is the main webmaster of STIC, assisted by Andreas. Hand Martin-Mostner, Frank Ralf and others will help them. In future, the STIC website could host a Smalltalk product network and success story network. STIC is a mainly volunteer organisation: a secretary is sought and a new treasurer will be needed in September. At Smalltalk Solutions 2007, Cincom brought out OS8, Instantiations brought out VA7.5 and GemStone brought out GLASS; for the first time in over 10 years, all three major vendors are strong at the same time.

Q(Tim) What has happened since 2006? Georg has been in charge since May 2007. Not much had happened between 2006 and May 2007. (STIC lost its old URL during that period.)

Q(Tim) Should someone from Squeak be on the board? A volunteer with time, energy and finance from Squeak would be welcomed. Suzanne stressed that people who will do things will be welcomed.

Q(Joseph) Let’s see concrete links. David Pennington has hosted the TotallyObjects smalltalk newsgroups on his own for 8 years. Let’s link to him. (There was agreement that this made sense.)

Q(Jaroslaw) How will you promote Smalltalk? We’ll get the website running first (stic.st). Send your ideas to stic_management AT stic DOT st. Seaside is important so STIC will embrace the representatives of Seaside.

(Stephane took over from Georg.) We don’t know how to talk to managers. Send ideas to Stephane (ESUG mailing lists are being changed to eliminate spam). ESUG can support you to do something, e.g. present at a conference about Smalltalk (email them: if they say it is OK, they will pay the budget, so it is very simple to do). They will pay 100 euro for a published article on Smalltalk. If a university or public body want a Smalltalk teacher, they will provide one free. They will support Smalltalk research papers at non-Smalltalk conferences: 200 Euros per paper (only one per person and 3 per institution per year; send them the PDF plus conference acceptance). They will sponsor booths at conferences. They will support (i.e. with money) open-source projects; ask them and they will consider and decide.

Seaside ambassadors are wanted. They will support your talks. Lucas went to Poland last year to do that. Then they just paid his costs but now they will pay that and a bonus (enough for yet another IPOD).

ESUG missed Google’s summer of code so did their own Summer of code for Smalltalk. In 2006 they had 5 projects. In 2007 they had 2 projects. Projects must be in Squeak or in VW at the moment.

**ESUG Board**
The ESUG board is re-elected every two years. This year, Roel retires (due to the time commitments of his new job). Stephane, Noury, Marcus and Serge are already on the board. Michele Lanza was proposed as new member. All were elected on a show of hands.

**Smalltalk Awards Ceremony, Noury Bouraqadi, Joseph Pelrine**
(Happily, a rumour that the wine for the awards ceremony had not arrived proved groundless.) There were 6 entries at Kothen in 2004, 9 at Brussels in 2005, 11 in Prague and 15 this year. All Smalltalk code (and related code, e.g. a Smalltalk VM) is eligible, whether used commercially or for research, and whether written by academics, by students or by commercial programmers, provided it is separable from its background system. Prepare your software for next year!

The entrants made two minute presentations:

- Pier is Lucas Renglii’s open-source content-management system. It is meta-described, using Magritte to describe relationships and fields. Content can mix ordinary wiki input with Seaside components, etc. It is based on Seaside but is not tied to Seaside; it can use Morphic instead or whatever renderer is available. Pier is implemented in Squeak.

  [Niall: I have ported Pier to VisualWorks; it is now in the Cincom OR.]

- DakarTest is Carsten and Damien’s work: it provides additional features and a better UI on top of SUnit and SUnitToo in the VW Refactoring Browser. It provides coverage of what code a test ran, using which it tracks when code changes require rerunning a test.
- RBPolymetricView: Moose only does post-mortem analysis. Adrian Kuhn has added its tools to the RB (“in VW; non-VW people vote for me and I’ll port it”) to show code structure while browsing.
- The SmallProjectObservatory (SPO) is a web application that looks at your Store database and shows what is there via visualisations: who changed what and when, what are the dependencies, etc.
- PetroVR runs on VSE plus Gemstone (there is also a relational framework; see Leandro’s talk below) provides discrete-event simulation for planning 20 year projects to the petroleum industry.
- EyeSee is a programmatic diagram-drawing engine. You can script diagrams well in 3-4 lines of code. You can customise every detail and export the results as Cairo, use them in Seaside or whatever.
- Itrex is written in Dolphin by Tim Mackinnon. “Dolphin apps look so good, they do not look like Smalltalk at all.” Itrex supports XP projects, providing story cards, diagrams, etc., and cross-relating the data. Tim is a former ThoughtWorks consultant: he wrote this app to replace the spreadsheets used in some XP coaching.
- Rob Vens’ MijnGeld is a personal finance manager. It is open-source so you can join the project.
- Bots Inc. (Squeak): see Stephane’s talk. It should run on the OLPC.
- EasyMorphicGUI: see Noury’s talk. It aims to connect visual GUI programming to programmatic GUI programming.

Joseph Pelrine presented the prizes. (Joseph felt that the level of bribery that had been offered him was far below what he had hoped for. :-)

- 1st prize (500 euros): The SmallProjectObservatory
- 2nd prize (300 euros): EyeSee
- 3rd prize (200 euros): Pier

Joseph thanked all the entrants for an impressive array of applications.

**Smalltalk Summer of Code**
Stephane introduced the summer of code and stressed the importance of the mentor being either on hand or available to answer emails promptly. The students made quick presentations of their work.

I missed one presentation, which was about working on a new compiler / decompiler, with SmaCC compatibility.

Damien Cassou worked on Monticello 2, mentored by Stephane Ducasse. He restarted Monticello development, enhancing its visibility and writing documentation. He produced a GUI based on OmniBrowser to ease its portability. [Niall: GemStone, amongst others, will be pleased.] He aimed to make the actual repository MC1-like. He refactored and cleaned a lot of the code. He posted a lot on blog.summer.squeak.org. He put links to useful old blog posts of Avi and others, UML diagrams, and other documentation on wiki.squeak.org/squeak/5642.

The model is nearly complete and has many tests. The GUI is open to new features. Damien still has to reach main MC1 features. Then he will use the new design to test new features (e.g. non-package-based features).

Juraj Kubelka worked on OmniBrowser Traits integration, mentored by Stephane. He is now working on refactoring tools, mapping hierarchy-oriented code to Traits. He is working on a tree widget. Put other requests on the ob-dev mailing list. Find the work on wiresong.ca.

Squeak HTML/CSS was worked on by Jerome Chauvea. The details are published on the blog. His work passed the acid test of web construction.

Oleg Korsak worked on extending Pier syntax, supervised by Keith Hodges. Benjamin was supervised by Ralph Johnston. Another student was held up by lack of mentoring; Stephane stressed that projects must ensure mentoring will happen.

Q(Bryce) these projects have ended? Yes, summertalk ended last week. The projects are not all finished; specifically, further work will continue on the three presented here which are of interest to and integrated into the Squeak community.

### Squeak by Example book

The ‘Squeak by Example’ book (Andrew P. Black, Stéphane Ducasse, Oscar Nierstrasz and Damien Pollet with Damien Cassou and Marcus Denker) teaches Squeak and is full of examples you can select, execute and explore. To avoid it becoming out of date too quickly, most of the examples are SUnit tests. A Squeak application reads the book’s LateK and convert the examples to runnable Smalltalk tests.

The book is open-source: they would really like people to contribute missing chapters. It is available now free in PDF. It will be on Lulu.com (print-on-demand) and available in all the other usual ways. Visit http://www.iam.unibe.ch/~scg/SBE/index.html.

## Applications and Experience Reports

### Exploratory Modelling, Andreas Tonne, Georg Heeg

Georg Heeg ran a very successful project working with the SAP company. They realised that the approach was what they always do, so Andreas invented the term ‘exploratory modelling’ for it.

The Standish report shows that between 1995 and 2005, 50% of projects were ‘challenged’ (fail in layman’s terms) and many were cancelled. In the same period at Georg Heeg, 0% were cancelled and 5% were ‘challenged’; the customer challenged them to do yet more and they did. :-) Andreas listed some successful projects (see slide). These projects tend to be very long lived and very resistant to being replaced, e.g. in Java. Why?

Georg Heeg captured requirements in good models (‘deep models’) that could be iterated and added to, and that were robust to change: when the way the system was used changed, adding new features did not force serious model change.

Why is good modelling so easy in Smalltalk? You can map domain concepts to code easily through powerful tools, lack of type specs, etc., but no customer cares about this or even understands it. A model abstracts something (in the real world, or it could abstract another model). You must map things to the model, omitting some things and make pragmatic choices so that the model is useful. Models are targeted at one or more receivers. (Andreas showed three human anatomical models, one focused on circulation, one on the skeleton, one on proportions, to exemplify this.)

Almost everything a programmer does - coding, testing, whatever - involves creating a model. The programmer’s intent affects the model or the view of it that is wanted, as does the receiver’s (i.e. the customer’s) interest. The customer knows their domain (we assume :-). This is the modelling scenario.

To most customer’s, modelling means UML or UML. UML is a good thing carried too far (Niall: I would say a mediocre thing carried too far). The customer cannot really understand UML models (as has been recognised by e.g. Microsoft who are now going strongly towards domain-specific languages). Smalltalk is a generic domain-specific language: concepts can be expressed directly in the program and shown immediately. In Java, changing one aspect impacts types, etc., so it is like throwing a stone into a lake: ripples go everywhere. Smalltalk is much better at keeping changes local. The result is a customer excitement level; more quickly than in other languages, the model starts saying what the customer wants.

Andreas then spoke about the SAP project (see Rolf Ehret’s talk for more details). Invoices are typically received by one person and paid by another in a company. Thus it is easy for them to be paid twice through accidental duplication. It is not an easy task to detect these duplicates.

The clerk says they compare two invoices; if they look ‘fishy’ (too similar), they try to reclaim. Andreas described a typical conversation with such a clerk about what fishy means and how they compare. They cannot compare every possible pair of invoices, it would take too long, so they just look at the interesting and potentially duplicated ones. But how do they know what ones are potentially duplicated? A long discussion would end, “I use my experience”. So how to model experience? Does this clerk know what he is doing but not know how to express it, or does he actually use guesses to supplement fuzzy rules? Whichever it is, he cannot offer a fixed spec. Eventually he says, “Let me show you an example.”

Georg Heeg cannot build a useful model without continual reference to the domain expert. Requirements are expressed as needs and examples. As soon as you fix the unknown, the customer moves on. There is usually a language gap in terms of what the developer and customer say: developers take a formal approach, customers a domain-aware, informal one. (Andreas showed two pictures of a lady in a hat, one old-style and one cubist-style, as a metaphor for the customer and developer way of looking at things.) So iteration must develop a common set of concepts. Customers find UML very poor for doing this (Niall: I would say it is both too formal and not expressive enough).

If the customer trusts that you know what their problem is then they will stay with you through bumps in the road; if not, they will panic on every possible occasion. So we need a modelling process that we can show (prove) to the customer is doing something useful. It needs to be fast: no, “Good point, let me go away and show you something on that next week.” And it must be right, not just look good.

Exploratory modelling is Andreas name for using Smalltalk to do this. Whitebox UML is too technical for customers. Blackbox prototypes are not visible enough to them. Smalltalk lets us have our cake and eat it. The xM cycle uses Smalltalk to implement the model and document the results iteratively.

The concepts and class names should be the same, likewise for the visible-to-customer methods. Make the model executable by adding an experimentation environment, probably starting with workspaces and developing what suits the customer. You experiment with the customer by executing examples / use cases. Get immediate feedback and iterate.

Create model documentation: the implementation expresses the model exactly. The customer may want more (UML diagrams, a document with other diagrams, whatever) and you provide that.

This is *not* prototyping: the implementation must express the model.

You need a dynamic language for this that is non-technical, barrier-free, scripting style, meta-programmable, concept-oriented and interactive. You could use Ruby but Smalltalk has far better tools, etc.

The duplicate invoice analyser took three iteration cycles over 7 days and acted on 170,000 real invoices. The first cycle just built their initial model understanding: its 12 classes read invoices, compared everything to everything and was hypersuspicious. Working with clerk experience and looking at examples gave them a second cycle with much more detailed rules and then a third cycle. They started finding real duplicates that the customer did not know of, enough Andreas suspects to pay on their own for that 7 days of work. The final model had customer-specific rules that could be reused.

It is hard to explain to a customer who does not know Smalltalk why Smalltalk is so good. They coined xM as a process modelling name to apply to modelling problems that are hard, expensive and risky - even 'unsolvable'. The customer ends up with a working executable model and a document which is a design for implementation in any language. Thus it is a good way to introduce Smalltalk that evades the 'we implement in X' problem. They can keep the model to evolve it further and if some part is quickly changing, they may implement that part in Smalltalk. Thus xM uses Smalltalk to give the customer well-verified quality models and gives the customer reassurance early in the project.

Q(Rob Vens). Naked Objects is a development environment where you only model domain objects; to a Smalltalk programmer it just looks like you have souped-up inspectors. There is also domain-driven design (book by Eric Evans, Andreas recommends it) and domain-driven development (Andreas saw this as a much discussed rather broad concept so not ideal for using to express the xM idea).

Q(Rob) Robust prototyping is a term used in Smalltalk, where we do not throw away the prototype? Customer experts warned Andreas that the term 'prototype' means dirty and throw-away to the customer, so should be avoided. But xM does throw it away? Yes (maybe), but xM keeps the domain concepts and implementation in synch so it's not just a prototype.

Q(Tim Mackinnon) How does this relate to XP? It is a small cycle so you can inject it into any lifecycle. Andreas feels that pure test-driven XP can produce too simple implementations that are not robust to serious change so he recommends this modelling as a way to get the framework for XP implementation. (Tim) Could you say this is the way to write story cards? Object-Business Analysis was an old useful technique; this is a powerful way of doing it. Yes, it could be a way of preparing XP cards, etc.

Q (Rob) In 2002, he presented on a large project where xM happened. The system was then implemented in Java but even today some developers are still using and evolving the Smalltalk model to understand the domain and plan work.

Q.(Georg) Concept and 'notion for concept' are two different things. xM is a concept, what we all knew to do with Smalltalk before we heard it is a notion for a concept.

**Capture Accurate Solution Requirements with Exploratory Modelling, Rolf Ehret, SAP**
Ralf has been a development architect at SAP for 9 years. He has much enjoyed being at this conference in beautiful Lugano.

In software development, building to the wrong requirements is common, expensive and undesirable. Andreas gave an excellent talk on exploratory modelling. As John commented, exploratory modelling is what Smalltalkers mostly do.

SAP develops business software for their 41,000 customers. Recently, SAP has done two projects with Georg Heeg and Cincom. One concerned detecting invoices that had been paid twice (or more). Another was studying how to choose the right carrier for your shipping requirements.

Ralf works in the business process renovation team of SAP. Their target is not *innovation* (SAP research does that) but near term (1-2 years) *renovation* of existing business processes. They learn and brainstorm existing processes that have problems and seek pilot customers to use prototypes. The duplicate invoice project had several customers, the carrier choice project was done with three customers.

Two years ago they decided they needed to build prototypes to do what they now call exploratory modelling to explore solutions. They already had done work on the duplicate analyser and knew their solution was not ideal. At end-2005 they got one week of training from Georg Heeg in Smalltalk and did a two week project with people from Heeg. There are three levels of OO: knowing there are objects, using objects and designing objects. When he met Heeg he learned a fourth level, that everything is an object; he himself is an object. That was a new idea. :-)

They needed to get right what the model actually did rather than how it was written up. ‘Must conform to all existing and upcoming internet requirements’: what does that requirement mean in a document? What does it do for you? Neither their own development language, nor Java, were suitable to do this kind of rapid iteration of a model to get it right.

Recovering duplicate invoice payments is a multi-billion dollar business. They built their own system using SAPs free text search engine, rules engine, etc., and had a solution in one month. The customers they showed it to were happy because it was the first time they had any system at all but Ralf’s team were not so happy because they did not feel they had solved the problem too well. They did not feel they understood the problem too well. Practically, their system found vast numbers of false positives so that its value was limited.

He opened VisualWorks and showed their first UI: not one they would deliver to an end customer but one written in one and a half hours (a lunch-break) and it showed everything they needed to discuss this problem. It was an experimental playground. He showed tools that appeared (not very much) later that eliminated 400 false positives and showed a real duplicate: same vendor name, similar reference number, same date, same amount.

They realised they had asked the wrong questions. They had asked, “How do you identify duplicate invoices?”, and the clerks could not explain. With this tool and analysis, they began to understand that every rule they found was pointing to a gap in the customer’s process. He then showed the real UI: seriously more attractive and better structured. It showed lots of data and the actual scanned images of invoices. (The display showed a faked example as he was not allowed to show any real data.)

Iterating and showing it to clerks gave them a much better domain understanding, a much faster and more accurate prototype. Ralf had never had a project that hit its timelines. This one did not either - but it was the first time it finished sooner, not later :-). They allocated 10 days and after 7 days asked each other, “OK that’s looking great, what shall we do next?”

Next they looked at the carrier selection application. Tylan worked with them (very well; thanks, Tylan) on this. SAP has seven or eight rule engines and clearly carrier selection looked like a rule-oriented problem but after evaluating seven engines they failed as the customer asked for all technology features, which no single engine had, because the requirement-collection process ended up being driven by their own technology, not really by the customer.

Hence they built a rule engine prototype in Smalltalk. Their first engine version was working after a week and they were ready to visit the first customer with it on their laptop after two weeks. He showed the UI at that time; a window to edit rules, another to process rules on data. The complex rule mathematics of rules in generic engines disappeared in the Smalltalk prototype’s interface through simplifications oriented to the domain.

So having built the engine in two weeks they then spent two months integrating to SAP via web services, which was horrible. So they and Georg Heeg built Cincom Smalltalk VisualWorks SAP Connect (legally SAP cannot let its name start with SAP :-) and it takes a fraction of the time to connect any Smalltalk model.

The business customers ended up getting what they wanted, even though they could not say what they wanted at starting.

The cycle in both cases is:

- exploratory research and modelling of the customers requirements
- discover and test heuristics
- rule systems
- enable fast customer feedback
- executable case studies

The team also asked themselves some questions.

- Does Smalltalk work for us? Yes (not yes but just pure yes)
- Does Smalltalk environment hinder in any way? No, far from.
- Do we still see overall benefit when you reimplement in SAP? Yes: for a few days input we avoided implementing a large wrong solution and then reimplementing it over a long time.

Q. How is the final document output to the developers used? The document provided UML. The end-product developers were not always be able to read Smalltalk easily but they used the prototype to see what their product was supposed to do. (Andreas) SAP knows the distinction between using a running app/prototype to guide implementation and reimplementing it in another language. Accidentally implementing a Smalltalk interpreter in ALA (SAPs language) while porting the prototype could be slow.

**Honourable Squires, Uwe Leibold, Torsten Happ, AMD, Taylan Kraus-Wippermann consultant from Georg Heeg**
AMD make microchips, probably including the ones in your computer. Taylan showed a picture of AMD's site in Dresden, started in 1996 and much bigger by 2007. In this period they have transitted very successfully from 130nm through 90nm to 65nm.

Some 500 Equipment Interfaces handle the Factory Automation Systems production steps, communicating with each other (via events on a bus) and with the equipment each handles. The EI is a Smalltalk application. There are some 50 types of machine, each needing a different customisation of the basic application.

The FCS controls the production scenario, telling the appropriate EIs to do the steps (the Manufacturing Execution System gives it jobs to do). The EIs collect data and report it back to tune processes, to monitor, etc.

The application is called CEI Baseline, consisting of a framework for the interface, config, event management and etc., the development tools and the standard information for a standard machine.

Different forces pull the app in different directions. You want to expand functionality continually but you also want to minimise restarts; any stop to production costs greatly. You want to handle errors automatically but you also want to minimise failure. This maps to developer concerns: test-driven is good but you never get 100% code coverage. It is hard to ensure behaviour is correct when the number of configurable influencing factors explodes exponentially. Hence tests need to be continuously improved. Doing that ensures that your errors are rarer, but when they occur, they will be more complex to analyse. The production environment is not the right place to analyse such errors so they need to reproduce them in their development environment.

Their Remote Service Tool lets them monitor running applications. A replay tool lets them rerun the situation. A Log-Viewer saves them from reading huge ill-structured text logfiles. It also opens and closes zipped files, and provides shared viewing to remote analysts.

Q. Synchronised time between applications? Viewer is per application. Events are time-stamped.

The RST listens on the message bus so sees the events. It also has a limited remote inspector (read-only, so production can be inspected without risk of its being corrupted).

The replay tool is a 'flight recorder for the Equipment Interface'. The idea is that if you feed the same information in at a later time the application should behave the same, which is not in fact true. Recording all data traffic with external systems might give deterministic behaviour but would generate huge quantities of data and it might take a long time to reach the same error state from a known initial state. How should you simulate internal processes of the device being managed?

The CEI connects to the MES, SAP, and many others. They intercept the interfaces (MQ, SECS) to record the events (important that implementation had minimal impact on the app). They write snapshots of the application state to avoid having to record huge data volumes over long times.

Q. The state includes blocks? The data is protocol data so this does not arise.

To simulate, they have to feed the data back by imitating the external interfaces: they have simulators for TCP/IP protocols (virtual sockets, CORBA-IIOP, HTTP and SECS) and for MQ series communication.

Q. Do this for 500 machines at once? Usually, one machine errors; we just have to handle that one.

The Replay tool's simulation component has a dispatcher to decide which message should go to which simulator. Messages can sometimes get out of order between the original and the simulator. The recorded events are merged by timestamp and the simulator can apply time delays to its dispatching to try and keep the order. Messages 1 then 2 may write answers 2* and then 1* in the recorded message stream. In the simulator, messages 1 and 2 are easily reconciled but the dispatcher could easily send answer 1* back as soon as it sees 1. From the recorded logs, the simulator sees what delay it must impose to reply 2* before 1*. This time-based triggering is deduced from giving prerequisites to messages: some come after others.

For this to work, messages must be reconciled. They have various strategies: sequence ids, timestamps and (within a limited window span) out-of-order messages. The user can use the tool to explore stretching and squeezing the rate of replaying messages relative to the original, assigning breakpoints, etc. They are still studying how to handle these out-of-order messages; it is not a solved problem. There are other challenges: keeping all recorded messages in memory has a huge footprint.

Currently, snapshotting is still in development, recoding and replay with a limited number of interfaces is possible, and the simulation control UI is ready to use. They find that acceleration can only be done by factors of 10 to 50. You might want to run faster, to reach the error state you are studying faster, but some of the interfaces break. To enable recording, CORBA needed 7 code changes, MQ needed 10 and HTTP OpenTalk needed none.

They would like to be able to edit production data.

Q. Timing your replays; you cannot compress unlimitedly? How much time do you have to write the application snapshot? Wafer processing has some millisecond processes during which you can exploit the EI's time. A machine may also be reserved ('locked') for known incoming production, during which the EI may be available for a minute or maybe as much as 15 minutes. A real image snapshot would break some interfaces, e.g. SECS has a heartbeat.

Q. How many messages? One per millisecond.

Q. Copy image and then snapshot; common in Seaside? He wondered if it could be done with VW image. The image is 40Mb.

Q(Niall). Feedback error scenarios you've recorded to test-driven development test style? Not thought of yet. (We then discussed offline.)

Q. Speed-up problems? Zero delay breaks some interfaces. Millisecond delays are OK for all of them.

Q. Your app runs 24 x 7; any maintenance periods? You have to find the right moment when machines are off otherwise (such times are very tight).

Q. Numbers of errors? he's not in that team so cannot say.

**Managing Business Process with Smalltalk, Janko Misvek, Eranova**
Business and IT (still :-/) tend to see things differently. Business Process Modelling Language is an approach to narrowing this gap. It uses a graphical flow-like notation like UML activity diagrams (this was published in 2004 by BMPI who are now called OMG, like UML).

The idea is that these models are executable. Some people (e.g. quality control people) just model without executing. Human-centric people model workflow. Integration-focused groups try to automate processes.

Janko started by supporting ISO9000 systems. He got interested in executing them with a view to providing computer support and measurement.

He looked at a pipeline example. The process models the measuring of data about the pipeline's state; as it is government-regulated, the attention to the measurements and their assessment is highly formal. They used the Visio program to draw the models, with a plugin (from Swiss company) to prepare the process for execution. This is done by a 'process engineer'.

The process execution engine is written in Smalltalk. It is a web application. The end-user does not see that it is tied to the process, simply that their web app tells them what task they are in, who is doing it, the status, etc., which they can use in their day-to-day management and which also keeps them ISO9000-compliant.

The product is called BiArt. It started as management of ISO9000 documents and now archives eDocuments (a new Slovenian law makes these equal in status to paper ones), presenting a web app on the archive and the current document set.

All this is built on Aida/WEB. Behind the Swazoo web-server, Aida security and other utilities handle the output from Aida's presentation framework, which sits on top of the Smalltalk application’s own presentation and model layer.

The BPM model is stored as XML, converted to objects in Smalltalk and run by the BPM engine which uses events to communicate with monitoring and other services.

Q. Client integrates your web pages with their ISO9000 data into their internal management? No, they use it as a web service used by their existing internal systems.

Q. Basic BPM-type models tend to be very constrained; is this easy to extend? Their engine and model is easy to extend with properties.

Q. Typical time needed for on-site modelling customer processes? It can be quite fast provided you understand what you are modelling.

**SqueakBot: a pedagogical platform for educational robotics, Julien Bourdon (Planete Science), Severin Lemaignan (Planete Science), Serge Stinkwitch (University of Caen)**
Julien has been a student volunteer at ESUG for several years; today he appears as a speaker. Planete Science is a non-profit French organisation promoting technical activities for young people. Each year, some 50,000 children are introduced to concepts by 1000 people like Julian in summer camps that last for 2 weeks with 6 hours of science per day.

He teaches the idea of robots detecting their environment and then acting on it. The programming language of such a project must be easy to learn, in the children’s native language, and *fun* to use. (Q. Age? From 7 up to 13-15. Usually, the children do not have a technical background.) Actually building a robot needs much tedious hardware work, so they build from ready-to-use electronic parts that can be programmed easily. There must be a creative element; the children should not just assemble a kit.

They used LOGO but it only works on Windows [Kat: it is on some other platforms], has a syntax to learn and is not Object-Oriented. Hence they developed Squeak/EToys to use.

Super Module Electronique de Commande (SMEC) controls one or two DC motors and 5 binary inputs. These can be interconnected via a I2C bus.

ASPIC controls 4 analog inputs, 4 servomotors (angle, speed, etc.) using USB interconnection. He showed pictures and offered demos later.

So now we have the bits for our robot but a robot needs a brain. EToys is one of the funniest (i.e. fun-est: most fun not most laughable :-) parts of Squeak. You drag and drop to program. It is very intuitive so ideal for children. You can start using Smalltalk syntax at any point.

Plugins are written in C using FFI with classes representing the bus interface and the electronic devices. The smalltalk talks to the devices via these plugins.

He showed a mars-rover-style robot built at a summer camp. Another group built a walker (reminiscent of Star Wars robots) which didn't work (walking is difficult) but the kids learnt a lot. A third example was a caterpillar; a video of it is available on the web). An intruder-detector robot took pictures of anyone approaching close to it; the kids also used EToys to calibrate the sensor.

EToys gives fast results that keep the children motivated; imagine if they coded for two weeks and just saw 'segmentation fault'. This is part of a larger project called 'Boite a Bots'.

They need to make all this work on Mac and Linux. They also want to test this work in a larger audience, e.g. in schools. They plan to do this and hope next year to will have a success story to tell. Links on slide: http://www.squeaksource.com/SqueakBot.html and two French sites.

Q. Can you buy the hardware? Yes. It is all in French and some things are unfinished. (*All* in French? EToys is available in English, German and Japanese.)

Q. Any use of Lego Mindstorms? No, we make the children start from scratch. It teaches the kids more. (Lego Mindstorms demo was offered offline.)

Q. You do it all? We also train trainers so it can be used more.

**From One Tree to a Forest, Alfred Wullschleger, Swiss National Bank**
Fifteen years ago, Alfred wrote his first tree structure. The tree(s) have grown a lot since then. He showed the Swiss National Bank application with a tree of the data in the left pane and a node-specific window for editing a given selected node in the centre right.

He then created an example tree for the ESUG conference. The conference root node has some participant nodes. He added another participant and developed that node with a 'where he was sleeping' (in the barn :-), what talk he was giving, etc. He added titles to define the node and offered help on nodes (sleeping in the barn explained what that would be like :-). The user, not the program, creates the nodes, configures their 'types' i.e. titles, etc., and adds the help, etc. Various renderers present the tree data as formatted text or whatever.

In 1993 they added help trees to the basic trees (because on Windows then you had to compile strings to use windows help so their users would have had to ask developers to compile their help strings). Alfred later switched from his previous company (OVID) to SNB. (Claus then also worked on the system.) The major change was temporalisation circa 2001. He showed three ESUG conferences with time ordering. This was important because banks are always changing their names and merging and whatever. (In OVID this was not an issue.)

Next they added delta trees; trees to apply changes to trees. He morphed a participant via a delta. He added ‘middle name’ (type String) to the ‘first name, last name’ participant node structure in this delta. After saving the delta (Gemstone provides their persistence; he is saving to Gemstone), he then (showed us that no validation was relevant at this stage - see below - and then) transformed the tree by applying it in the delta transformer (very fast as it’s a very simple change): all the participants now have middle names. (Standard layout; the UI layout can be configured).

Q(Georg) So this is a schema migration triggered by the user? Yes. Typically the end-user does not do this casually. Normally they agree changes, test, then transform 10,000 trees. They recommend end-users not to transform in production before going through tests.

The tree algebra lets them combine and reuse elements. They define tree homomorphisms for various reasons, e.g. to map their structure from the most user-friendly to one easily mapped to an external relational database.

In 2007 they added validation trees. He showed a tree that checks that all participants have accommodation. Two nodes (‘is hotel element’, ‘is barn element’) are ORed in the containing node to warn if a participant seems to have nowhere to stay. Now we no longer get told there is no validation information. Instead we see errors, e.g. a time error on participant 4 ‘the presentation is not within the presenter of the conference’ because he was presenting in October which is too late, and warnings e.g. a warning that participant 3 has no accommodation (just a warning because maybe they live in Lugano and don’t need accommodation).

Q(Christian) Can you make accommodation optional to suppress the warning? Adding a title ‘lives in Lugano’ would remove it. Suppose they don’t want to do that. They and their customers have recently had detailed discussion about whether warnings should be hideable or not.

It took them 3 weeks from their customers request to add validation trees.

Lastly, he showed an actual production tree of data coming in from the banks. This data must be validated in many ways. He selected one validation and displayed it in the tool. He generated a test tree for the validation with generated random data. He showed running the validations and having their results put on a stack. Users (not them as developers) understand what these validations mean. (On the random data, some passed, some failed; one would use this to verify running and then move on to testing on real data).

A major advantage of Smalltalk is here shown: in 1992 he coded trees and still in 2007 he is able to use his old code. In Java he would be adding new types and that would force revision but not in Smalltalk (move of company and dialect from VSE to VW forced nominal rewrite but it was the same really).

It’s important that the configurer does not override trees that are in production. A checkin-checkout mechanism controls this and this same mechanism applies to all trees.

Q(Georg) When Georg was young, there was discussion about relational databases, network databases and trees. People said trees are excellent as long as there are no network requirements. You have none, or you solve (how)? They have network requirements and they solve them by references which are like URLs. It is not a heavy network.

Q. Validation trees; you can have different semantics at different times? Yes indeed; the validations are temporalised (just reusing the feature) and the validation requirements do indeed vary. Messages from banks always have reference dates. Validation trees have to provide similarly a date related to when rules applied. They do this by adding fixed dates but they would like to provide dynamic date intervals.

Q. Is the validation linear or exponential in terms of the rule complexity? Linear in the size of the tree for their individually-fairly-simple rules.

**Impromptu demo of CMSbox: the Netstyle Content Management System, Adrian Lienhard, Netstyle**
By popular request, Adrian demonstrated Netstyle’s content management system. It has been developed for 1.5 years by Netstyle in cooperation with a guy who does user interface design; he gave some of the concepts and Netstyle had the ability to implement them. The new company that will exploit it is called CMSbox. Systems like typebook 3 are really complex to use because there are hundreds of options and it can be hard to figure out how to change the content of the web page. CMSbox has two views, the page and the different view of the editor where you edit that page. They bring this together.

He logged in (clicked bottom button, it scrolled up). You click on text and you edit it in situ. You can drag an icon, and drop it in a new place.

Q(Bruce) when is it saved; immediately? When you press the publish button. You can also save without publishing, defer publish. The menubar gives you all the icons (text link, etc.) of the various widgets: you select and add to the page. You can link within the page, to others, to external. He selected pictures and enlarged them, shrank them. This is not just extending the size, it is scaling the picture to keep the quality. Pictures can be styled (with boundaries, etc.).

The toolbar is part of webpage; there are no plugins.

He created a form, selected text boxes and set style, then made a dropdown with 4 options and added them.

Q. Where did you specify the identity of that form? That is just normal Seaside. The default effect if you submit is that an email is sent.

He showed an undo.

Q. Version control and undo? We have sessions and the undo is within a session. Published pages are versioned; you can see the page of e.g. 20th of last month, etc. You can go to an old page and see it, copy and paste from it. He showed the calendar to select by date the page-version of that date. Immediate persistence is achieved simply by saving the image; they could talk to Gemstone.

Q. Browser compatibility? They work on that. Of course you can sometimes get an issue just as with other web apps.

Searching your page and site with the usual abilities is there. You can automatically generate PDF.

Q. Search-engine friendly, e.g. google? They have done work to offer nice URLs with single h1 tags (search engines can deprioritise if you have more than one h1 tag). They support nice XHTMLs.

This is the Seaside-based CMS for stuff to be served by Seaside.

Q. You can have seasonal and other dynamic mixing strategies. Is this supported? Not supported; you'd have to add it with a bit of Smalltalk programming.

Q. Backend to other CMS systems? Not yet but to do (RSS and so on).

Their business model is to provide the whole solution and the hosting (for prices, see their prices page), not a system modified for a specific user, but they could talk about supplying a fork of their code to a user who wanted it to kick-start the Seaside front-end of a specific system.

Q(Georg) Multilingual web presence? They support this. The underlying mechanism is to have a different tree for each language. The content provider, having created pages in the primary language, copies and translates them. (Georg mentioned the WebTCM system. He will email Adrian the WebTCM reference and I will email its write-up to him.)

There is an English language version. It will be released at the start of 2008, not before. They are determined to grow at a sane rate in Switzerland next quarter to have the right base for going international.

Q. Mix with outside stuff e.g. other Seaside hosting for a widget in a page? This is not yet done but is on their plan. (Discussion with Monty.)

There is a video to use for demo purposes and they would make a sandbox available to demo-ers under suitable conditions (email Adrian).

## Application Frameworks

### Seaside, Lucas Renglii, www.lucas-renglii.ch

At the last ESUG in Prague, Lucas gave two lectures: 10 Steps to Mastering Seaside. Then the old `rendererClass` was still the default. Since Seaside 2.7, the old renderer class is deprecated and the new is the default. (This potentially broke old legacy code so Seaside 2.6 remained as a little-changing version for some time.) This replaces e.g.

```
    html anchorWithCallback: [self doIt] text: ‘do it’.
```
with
```
    html anchor callback: [self doIt]; with: ‘do it’.
```

They also made the canvas use the same canvas API for <head> as for <body>, e.g. `html meta name: ‘generator’ content: ‘Pier’.` The developer toolbar now warns when you call deprecated code.

Many people complained that serving static files was cumbersome in Seaside but people wanted the programmatic control of it so e.g.

```
html stylesheet url: PRFileLibrary / #mainCss
```

but of course they still recommend you use Apache or similar to serve large static files.

Avi was already busy with DabbleDB and only had two major commits into Seaside 2.7. Michel Bany ensured the portability to VisualWorks and the rest was done by Lucas et al. Seaside 2.7 became the standard.

Next, they redesigned the website; the new one was released in 2007. It runs in Pier which runs on Seaside. This made it very easy to integrate examples directly on the wiki pages, hosted by www.seasidehosting.st. Morello (web designer), in many iterations over two months, created the new Seaside logo and the new page look. The news component aggregates some well-known seaside blogs.

Q(Bruce) Interesting you picked almost exactly the same colour scheme as OpenSkills uses, which is good.

The example applications show both the effects and syntax-highlighted source code. Pier can be told that something is Smalltalk source code so that it automatically annotates it by div so it can be styled by CSS.

He showed a map constructed from google analytics that showed from where in the world people visit Seaside.st. (Finland may just indicate that Lucas went on holiday there soon after release and frequently checked it was still running.)

A slide showed a long list of who contributes. There is also a silent community of people subscribed to the seaside mailing list. The numbers doubled from 400 to 700+ in the last year.

That is today. Tomorrow is Seaside 2.8 (some 2.8 versions have already been released). He showed benchmarks of Seaside’s evolution. Kent Beck claimed that having between 0 and 1 percent of comments was not a problem but Lucas does not agree. Old Seaside was ill commented and some mailing list emails were longer than all existing Seaside comments. He showed Seaside class comments by version: 2.5 had 29%, 2.6 had 41%, 2.7 had 39% and 2.8 had 99%. Method comments have also greatly increased (but not to 99%). Seaside 2.8 jumped unit tests from 44 (in 2.6 and 2.7) to 131.

Memory: he loaded the same complex application into 2.5, 2.6, 2.7, 2.8, clicked through it 10 times along the same path and measured the backtracking and state memory footprint. 2.5 had 170kb, 2.6 had 193kb, 2.7 had 204kb and 2.8 had 48kb *with no loss of functionality.* You can let your sessions live 4 times longer, serve 4 times as many hits or just buy a quarter as many machines.

Rendering is a similar story. Benchmarking how long to write the last byte to the socket is in 2.5 90ms, in 2.6 90ms, in 2.7 94ms and in 2.8 51ms.

How were these improvements obtained. Rendering was just not as efficiently coded as it could be. Memory has a more efficient backtracking algorithm that avoids unnecessary stuff. Some poor configurations were also removed.

In 2.8 the old renderer is not there any more. They added development tool plugins. Especially when porting to VW, they found that many plugins were not cross-dialect and other plugins were not backportable (and sometimes were implemented in bad ways) so they built a new plugin framework. You can use plugins to extend the halos too: the class name in a halo is now a plugin, etc.

Backtracking of state. In old versions:

```
initialize
  super initialize.
  self registerObjectForBacktracking: model.
```

backtracked a component for the whole lifecycle, even if it was not shown any more. Now they control it better:

```
states
  ^Array with: model
```

In old Seaside, URLs were sometimes strings, not always first class. In 2.8, the URL class is used everywhere possible. Streaming is also improved.

Seaside 2.8 beta is released today, one year after starting 2.7 and half a year after releasing 2.7. Beta means it is now feature stable. The remaining bugs will be fixed and the final release made as soon as possible.

Seaside was developed in Squeak where it runs very well. There is also the Squeak Kernel Image (note new logo on slide to indicate that Kernel is minimal) which will run Seaside and Pier; use it if you have memory footprint concerns. Gemstone runs it (see GLASS talk) and Cincom VisualWorks now officially support it (good ports from Michel Bany have been available for years but now it is official). GNU Smalltalk will have it soon. VisualAge expect to support it in their next release. Dolphin support Seaside 2.6; they would be happy to hear from people willing to port 2.8 to Dolphin.

Seaside commercial people are too busy to come to conferences but he has two success stories: Dabble DB (Avi Bryant and Andrew Catton et al) and netstyle.ch.

Christoph Wysseier, CEO of Netstyle, then presented. Netstyle are one of the first and most successful Seaside companies, thanks to Lucas. He showed an art content management site and an insurance broker site. Swiss juniorwebaward is a school web app building contest site; this is a Seaside and AJAX site. They've also done a workflow management system for a Swiss site and a business card creation system.

From his experience he very strong recommends use of Seaside. He has used it to create a wide range of apps for very different customers. They get higher quality at lesser expense than rival frameworks. Their clients are very happy. (End of Chris' presentation.)

Seaside 2.8 is already being used in production. There are a few bugfixes needed as known. What will Seaside 2.9 do? The architecture at the moment is the application code sitting on the large block of Seaside with a little Javascript sitting on the web server. For 2.9 they will split Seaside into several packages: a Seaside Application Server will support Platform, Core, Tests, Examples, Javascript and other packages.

They are looking for website maintainers, documentation writers and code content contributors.

The T-shirts can be bought from the online shop seaside.spreadshirt.ent

Lucas Streit wrote a flash generator for Seaside. Lucas demoed subclassing WACounter to FlashCounter. `html flash: [:flash | ...]`. He opened the browser and showed flash behaviour (zooming in and so on) with all the usual Seaside behaviour. This will be developed further.

Q. Licence situation? MIT licence with names Avi, Philippe, Lucas and one or two others. Do what you want except remove the copyright notice.

Q(Bruce) How has use of continuations evolved in Seaside? Old Seaside used continuations a lot in the render loop. They have realised these are not necessary; they are still used but they are no longer stored. However the remaining continuation use is essential to the functionality. If you remove them it remains useful but you would lose `call:` and `answer:`.

### Advanced Object-Relational Mapping with GLORP, Alan Knight, Cincom

"Unlike my previous talk, I am now discussing stuff I *do* know (a great deal) about." Alan used a motivating example, the Ruby-on-Rails Active Record pattern, to explain what GLORP can do, both generally and in a Seaside context. GLORP can read database schema, create mappings dynamically, etc.

Ruby-on-Rails is opinionated software. It emphasises certain patterns and conventions: you can go fast but only on the rails. It is a reaction against heavy systems like J2EE. It fits greenfields projects best since for example it insists you organise your DB schema in a given way. It has tricks that Smalltalk programmers will recognise. Its two big pieces/patterns are web presentation and database handling, the latter being called ActiveRecord.

GLORP is very different from ActiveRecord. Ruby-on-Rails uses naming conventions, tied to English language forms, and very little and trivial metadata. GLORP uses metadata declaratively. RoR uses code generation; GLORP does not. GLORP has a single broker that manages the cache of objects and which is very un-dependent on other frameworks such as the web. RoR has no brokers or caching and no identity (ask for an object twice and you get two objects; going from parent to child back to parent gets you a new parent object). RoR's session and transaction are very global.

GLORP has no (necessary) metadata on domain classes. Its typical use case has an existing, maybe hard to change, schema and (different) domain class model. ActiveRecord puts such metadata as it has on the domain classes and the database has to fit via the conventions. RoR tries to deduce foreign keys but the DB knows what are FKs and GLORP just reads that.

Alan would like to get the benefits without the pain. He wants to have a quick set-up convention which he can then modify incrementally.

Before we can automate we have to read the DB schema. DBs have their own standard (i.e. different on each DB) way of holding their schema. This data is in tables and GLORP can read these tables so it is very simple to have 'meta' mappings to an object's DatabaseField. The only hard mapping is `isPrimaryKey` which is a boolean in Smalltalk but is determined by, 'Are there constraints of type primary key referring to this field in the appropriate DB table?' in the database:

```
(aDescriptor newMapping: DirectMapping)
  from: #isPrimaryKey
  to: [:each | each primaryKeyConstraints notEmpty]
```
(where `each` is the key we're referring to and `primaryKeyConstraints` needed a full-page slide to show its 4-part composite subselect).

A lot of things that look like collections in Smalltalk turn into subselects when mapped by GLORP to SQL.

```
read: Customer where:
  [:each | (each orders select:
    [:order | order amount > 1000]) sqlCount > 5]
```
(where `sqlCount` is the only difference from ordinary Smalltalk).

This reading of schema works for Oracle, PostgreSQL and MySQL at this moment (others being added) and needed no changes to the domain model.

GLORP metadata lives in descriptor systems in methods of two types: one kind defines what a class looks like, the other what a table looks like. So for the schema stuff, he can look for a class whose names and structure looks like it matches.

RoR use an inflector that knows how you pluralise and singularise words (so it will map a class called Person to a table called PEOPLE). The inflector is just a big ball of 40 regular expressions so Alan just ported it to Smalltalk (one Regex that Vassili's did not support he had to work around). His utility supports customising the table name to a more user-friendly class name. It generates a package with a class for each database table and an empty descriptor system.

Incremental set-up uses a deferral mechanism. Mappings can be tweaked:

```
self mappingNamed: #bankCode do:
  [:mapping | mapping type: Integer]
```

(and you can similarly change e.g. the regex-found default names). This is post-hock: it runs the block after generation.

Rails Migrations help you migrate from one version of your schema to another, running upgrades sequentially to go from e.g. version 35 to version 42. Rails supports some data transformations.

Alan uses subclasses (but you can do it otherwise) to map schemas from old to new (see e.g. the StoreGlorp descriptor class hierarchy). He also wrote a utility to update schemas as a proof of concept - he has not yet added data mappings.

Alan looked at Ramon Leon's MagritteGlorp (generate Glorp from Magritte) but instead generated Magritte from Glorp, which means you don't have to write any meta-data at all. This code is in the Cincom open repository in package GlorpActiveRecordMagritteSupport.

Q(Niall) Others will want to use and develop this both in Seaside and also in data/schema migration work? Yes. Look for *GlorpActiveRecord* in the public repository, and use blessings and comments to show the status of any increments you publish to help Alan integrate them back into the main branch as and when appropriate.

Q. He used GLORP recently on database of bug reports from Mozilla. He just wanted the comment report field but he found a large footprint for loading all the table columns into objects. Can this be avoided? Use the `retrieve:` method to have GLORP retrieve values only for those columns you choose.

Q(Georg) There were ideas to use Gemstone as the backend to Store. Can GLORP migrate an OS database to Gemstone? I and others participated in the follow-up discussion. James Foster has already done some work to let GLORP fake an RDB appearance so that Smalltalk utilities expecting that could be quickly ported. Migration could certainly be done. Arguably the main requirement is a business case, or someone motivated to just code it.

**Post-talk demo: GLORP schema-reading, Alan Knight, Cincom**
Look for the GLORPActiveRecordDemo package and trawl through all GLORPActiveRecord* pundles in the Cincom OR for more. Do

```
ActiveRecordDemoWorkbook openExample
```

The pattern sets up descriptors for subclasses of the defaultRootClass (by default, ActiveRecord) with very little code. The login reads the metadata from the database, e.g. Oracle, then constructs a fairly direct mapping. The reading is (not very) lazy because descriptors validate and the validate will almost immediately kick off more reading and descriptor system creation.

Oracle has a standard human resources database demo. You tell the Inflector whether to use plurals but it will still search for the singular if the plural is not found in the table names. The example uses the tables in the HR schema. The result is HRPackage.

```
anActiveRecordDescriptorSystem
  createPackageName: ‘HRPackage’
  forTablesWhere: [:each | each schema = ‘HR’]
```

This creates descriptors. It does not write and compile `descriptorFor*` methods. It will accept overrides: if a matching `descriptorFor*` method is already there, it uses it. It would be easy to add behaviour that walked these descriptors and generate their creation methods using `storeOn:` as a kick-start to conventional use of Glorp on an existing schema.

The MetaDataDescriptorSystem shows how meta-reading is done. There is also a MetaDataWorkbook menu item. To apply this to a new database, it might be best just to copy the MySQL implementation and fix anything that did not work. A diagram of the MySQL info schema is on the web somewhere; google for .../MYSQL_INFORMATION_SCHEMA.html.

**Ballooning with Cairo, Travis Griggs, Cincom**
(Travis passed round two sticks, one of them mine, of the PDF of his Smalltalk Solutions slides. which were almost the same as his ESUG ones, to help those at the back see the detail.)

In 2003, Travis presented the ExtraEmphasis utility. He wanted better fonts for Linux, so went via XFT and FontConfig to Cairo, who told him to use Pango. Travis wants graphics programming in Smalltalk to be fun again.

Cairo is an open-source equivalent of the Mac graphics stuff. There are many graphics library backends, many platforms are supported and there are many language bindings. There are also multiple users: mobile phones, Firefox, Adobe’s Apollo, etc., and Sophie was playing with it for backend rendering.

He hopes in time to ship the libraries with VisualWorks; see slide for how to get it today. Cairo is good about keeping backward compatibility of APIs and this is good as DLLCC when updating tends to overwrite your hand-tuning. Enums are represented as Constants subclasses, for example `CapStyle butt == CAIRO_LINE_CAP_BUTT`. This is where Ruby et al stop, obviously interfacing to C. Travis wanted, e.g. `moveTo: aPoint` and `moveToX:y:`. To keep Cairo happy (it knows nothing of keywords) sometimes things are not as Smalltalk-style as they might otherwise be.

dllccDouble is used a lot as almost everything in Cairo is a double; they don’t use floats much.

The first thing you do in Cairo is create a surface, in formats a1 a8 rgb24 rgb32. You can do this wholly in Cairo (it manages the bytes):

```
ImageSurface format: CairoFormat argb32 extent 100@100.
```

or by creating a ByteArray in fixed space and telling Cairo to use that. Pixmap surfaces are longlasting but you should recall the cairoSurface (to synchronise: pick up bounds and etc.) every time.

```
aWindowOrPixmap cairoSurface.
```

(Brief excursion to `DumbJack dungBeetle`; at 01:00 today, he found a problem generating PDF surfaces; there is a problem there!)

When you have a surface, you then create a context (often abbreviated as cr or aCR, because for some reason they do not call them contexts although they admit that's what they are), obtained by `aSurface context`. This is like a GraphicsContext but it has more state, its shape. You can map any GraphicsContext to the equivalent cairo context.

Cairo has some 'verbs' which you can apply to a stroke of a path, cleared at the end of the stroke unless you explicitly preserve: paint, paintAppha, clip, fill. He showed a VW and Cairo slide, showing how Cairo's infinitely thin lines avoid fill overlap in thick boundaries of filled shapes. Unlike VW, you put in your shape and *then* invoke a verb to actually display it; forget this and you can go mad wondering why it did not draw.

The source of a context is a pattern, (`aCR source`) which is a solid colour, a gradient (linear or radial, and it can reflect at boundaries), etc. Travis has promised to do a contour gradient for Cairo if noone does it first. Just as we all caught font-itis when fonts first appeared, so Cairo can have the same effect on you when you first see all the things you can do with it.

You need at least two beziers to define an arc so they have helper methods. You can exploit the fact that they will drag the path to the start of the arc. Drawing a rectangular path is not the same as drawing a rectangle; the former gets two close caps at the joined location.

Paths are built via connect the dots: moves, lines, curves (all bezier quadratics - the wikipedia article has a great animation and explains well) and closes. Paths can be disjoint and are infinitely thin, not 1 pixel wide. Paths must be closed to be filled and must be done explicitly: a path not declared to be closed can return to its start point but it will be unfillable. Paths are pen-down moves: `lineTo:` or `relativeLineTo:`.

His slides are rendered by getting the path of the font and then stroking it which is not your typical font package approach.

Enumerating over paths is tricky as the various elements are so unlike: moves, lines, curves and closes. They wrote a case-like method as the easiest way to express an enumeration. He showed a few lines of code to render the Norwegian flag colours fitted to the Cincom logo.

Filling and Stroking in VW would take a top left and bottom right where the fill omits the latter point as VW lines are finitely thick/thin whereas Cairo's are infinitely thin and your rectangle gets filled from the very top left point to the very bottom right point.

By defining subshapes backwards (anti-clockwise as against outer fill shape clockwise) you can have complex cutouts drawn via a single shape.

Cairo is a different way of drawing in which you can apply affine matrix transforms for everything ("use the matrix"). This is good for doing e.g. translate to centre, rotate, translate back in a single operation.

Rotating and scaling in place has to be done in the right order; naive coding can just make the shape disappear. Applying a transform just before stroking can give a subtle calligraphy effect.

You can temporarily create another surface and retarget your contexts to it. This is a form of grouping and/or double buffering: draw to your group then display it when done.

Cairo has a 'toy' text API, which they prevent people using too much by refusing to answer questions on it and saying pango, Pango, PANGO to anyone who asks; he 'got it' eventually. He wrote a minimal Pango interface for some slides; raw Cairo was too tedious. This is faster but because it is so powerful you will soon use up the extra speed so as regards the final result it is not necessarily faster but a lot more attractive (Joachim's train simulations look much better but run at the same speed). Travis likes using the new API so does not want to provide a compatibility layer from the old VisualWorks methods but maybe someone will.

He demoed the new (non-Cairo) Refactoring Browser's test tools and progress bar. Yesterday he did a Cairo version: it has cool radar circling, waves across the bar, throbbing when failure, etc.

The first thing a newbie to Cairo is told to do is build a clock so he showed his clock (usual demo hiccough: he had to switch to Motif and then had to make it display on screen). The clock could be made to rotate in various axes, become an orrery (i.e. a clock orbiting in a clock). Michael did two clocks; a chronometer where the hand never moves, instead the numbers rotate round, fading away when 180- degrees from the hand, and a Dr Who clock (not very useful for telling the time; I suggested you were meant to believe you had moved to another time). He also showed the screencast of the stock graph example from his blog: glowing line, years as bookend curves, selection brightens selected area and fades the rest, etc.

Q. Any printing suggestions? Just postscript or whatever.

Q(Georg) Any relationship to Widgetry framework? No. Widgetry and Wrapper both use graphics contexts so you use it with either.

Q(Georg) Cairo is written in? C.

**Application Frameworks: an Experience Report, Arden Thomas, Cincom**

Suzanne introduced Arden as the new product manager for Cincom Smalltalk. James Robertson, who proposed this change, is now the product marketing manager and will be growing yet further his blog, screencast and other high-visibility-for-Smalltalk activities.

Like the last speaker (John O'Keefe), Arden started in Smalltalk in the 80s, and worked for IBM for a while (not such a long while :-). He also worked at ParcPlace. In 1999, he went to work for a Hedge fund in Connecticut and in fixing things he needed an application framework. The chance to build a framework is wonderful opportunity but if you ask your manager how much time you can spend building it and they look at their watch, that is a bad sign. In effect, he was not authorised to build one but he was authorised to buy one. So he had to think about what makes a good framework.

A framework should make things easier, simpler and clearer. It should also not make it difficult to go beyond the box in which the framework makes things easier, simpler and clearer. In the early '90s, Tim Howard wrote a book that explained DomainInterface. Steve Abel (who, with Adele Goldberg, built LearningWorks) built the ValueInterface (inspired by the slamdunk architecture). These frameworks both had a single domain: the ValueModel has a single domain object. They both hide things: ValueModel hides additional instance variables in the builder so that the ApplicationModel is not cluttered by them.

ValueInterface extends the ValueModel idea to ApplicationModel. This makes it easy to connect widgets to the domain and to react to domain changes. It also makes it easier to reuse applications. (`nameChanged` will be run when you change aspect name - issue when refactoring?) He likes ValueInterface for some of the same reasons that he likes Smalltalk. It is a simple concept consistently applied that is more than it seems.

Locality of reference - putting similar things near each other - is a value of a good framework. Arden prefers to put his initialization methods in an initialize method rather than in many lazy accessors because the former gives locality of reference. [Niall: I think the same for the same reason and also for locality of time. Only when locality of initialisation time is *not* the correct thing to aim at for domain reasons, as opposed to a side-effect of poor coding, am I happy with lazy accessors.]

A ValueModel is a model from which you can get information via a simple interface (`value`, `value:`) with dependency notification. Arden talked through examples of how the ValueInterface, ApplicationModel and domain object relationships allowed localising changes for various reasons in basic VisualWorks code and in Tim Howard's framework. Arguably these are examples of the rule: "You can solve any computing problem by adding a layer of indirection." :-)

ValueInterface pulls up the `value`, `value:` interface to your application, allowing much more pluggability between the layers. The `value`, `value:`

calls can be to blocks that recompute, to buffers, to many things besides mere data holders, and can be to other value holders and so on. ValueInterface makes it easy to connect widgets to the domain, to react to domain changes and to reuse parts of the application.

Steve Abel was surprised that this framework was not used more widely than in Smalltalk.To someone who does not know Smalltalk, starting by explaining meta-class regression would not be the way to explain it (or not if you wanted people to think Smalltalk was simple). Arden suspects that ValueInterface explainers made the same mistake, trying to explain the fundamentals of the pattern first instead of showing what it could do first.

In the hedge fund, Arden extended the framework for the application, allowing mathematical expressions to be put in the aspects, prefixes (`my name`, `my phone` modifying aspects `name`, `phone`) and deferred messages (c.f. similar pattern in Widgetry). He opened VW and showed some UI and code examples.

Q. Do you have to wrap each item in a collection with a ValueInterface if you return a collection? There are two approaches. You could make a List your domain but it can be more useful to make what is selected in a list your domain. He showed a SelectionInList example:

```
initialize
  self databaseModel list: TradingSecurities all.
  self broker: self selectedRow.
```

where the broker domain object is tied to the selection in the list.

He also discussed `changed: #aspectChanged` and its counterpart `update:with:from:`. Implementing the method `aspectChanged` in your domain means no specific `update:with:from:` is needed. You can provide hooks such as `MyApp in: AnotherApp` to connect all these.

The `void` object is like `nil` except that it disregards messages it cannot understand. [Niall: this is the message-eating pattern of which there is much debate whether it is good or bad. It can make certain types of code much neater but by deferring when you see an error it can make spotting and fixing errors harder. People have a variety of views.] Arden commented that you should not overuse this pattern. We discussed this: Arden said one of his rare uses was when he had to allow for access via some external interfaces that might provide nil so void kept his code clean.

Q. Adding these ideas to Widgetry? He has starting doing so. He observed that you can generate a lot of Announcement classes to handle changes and he thinks he can shrink these again.

## Vendor Reports

### What's new in Cincom Smalltalk, Alan Knight, Cincom

Alan's most important job is soccer referee but in his spare time he does Smalltalk. Alan's official title is now 'lead technical architect' so he gets slightly involved in many things he does not wholly understand as opposed to working on a few things he understands very well.

ObjectStudio 8 is shipping Friday. (Long ago, a much earlier version of ObjectStudio was what Jack Sutherland was working on when he originated Scrum.) OS8 has the VW class libraries but looks like OS to OS users. It has the OS GUI - windows native widgets - but the much faster VW VM (OS's was pretty naive) and all the VW libraries and utilities.

OS was probably the most divergent from VW of all the Smalltalk dialects so the integration had many technical challenges.

There were semantic changes, e.g. `at:put:` returns self not parameter in OS; Alan found 30 places just in the Glorp port where this matters. In OS8 `at:put:` returns self for OS code and parameter for VW code in the same image. OS classes live in a separate namespace and use a different compiler which just does sophisticated handling of code and then hands over to the VW compiler. File handling also differed.

OS8 uses the VW RB and debugger because those were simply better but it retains the OS inspectors where the users preferred them. VW Web, server, networking and database libraries are now available to OS users (and Cincom now only has one set of these utilities to maintain, which they are happy about. :-)

VW users also have gains. VW windows-only users now have native widgets if they want it. Vista-certification is pending (getting certified is an interesting exercise). OS also gives them COM and Active X support code, and legacy database connects to APPC and EHLLAPI.

Virtual machine work: Mac work is the most active area (because it most needed it, as Mac owners can testify). VW7.5 supports Intel Mac OS X. Alan uses Macs a lot and has few problems but he does not do much heavy graphics stuff; he is aware that those who do have issues. The Apple VM has had two native OS threads, one for UI/OS calls, one for running Smalltalk code. This has stability implications - can get race conditions - and the general issue that Apple is not built with the expectation of this approach. They have written a single threaded VM which is much more stable but slower at the moment. However they expect it to become much faster as they develop it further.

They will make the VM into a DLL (i.e. (Q. from Bruce) also as an SO on UNIX - and it may well run there first) and are now looking at the issues. They have to decide whether it runs when it is not being called, etc.

The 64-bit VM has been around for a while but needs development to exploit it e.g. via a much larger space for identity hash. (ST-80 has 14 bits so identity-matching containers that grow larger start performing poorly.)

Generally, the hashing code is being reworked to improve the distribution of hashes. Collections of strings were hashing via first, last and some middle chars; this could act badly for some cases so string hashes now check all characters. General hashing has dropped the bucket scheme for a better-performing single-level hashing. A hash analysis tool is in the public repository; by all means use it to check hashing in your own apps.

Sockets and THAPI are being much speeded up by solving old problems.

Eliot Miranda left in December and John Sarkela has taken over, working with Peter Hatch and Andres Valloud. They are addressing process issues, e.g. the VM code is now in regular subversion check-in instead of sccs with the VM checked in once every release (“whether she needed it or no” :-). The VM’s regression test suite is being formalised for all platforms.

Widgetry, the UI framework they have been working on for (quite :-) some time, is now stabilised at release 1.0. The (strongly-presented) prior view that all VW users would (have to) migrate to it is now wholly changed. They are resuming development of the old Wrapper UI, which has not been comprehensively updated for some time, and this is how they will address their customers’ UI needs. The effort of rewriting a new GUI framework was more than expected and extra resources are not available. Existing customers were reluctant to migrate for valid reasons. Re-examination of Wrapper has persuaded Cincom that underneath its framework-obscuring GUI-builder UI, Wrapper is evolvable. Gradual evolution of the existing is more the Smalltalk and XP way than big-bang replacement.

Widgetry 1.0 lacks some things you might like: it has no migration support and no UIBuilder. For documentation, there are web resources, Sames blog, Jim’s blog, etc. Eager adopters may use it and those who have UI requirements that Wrapper cannot meet. Widgetry 1.0 will be a supported product but users should not anticipate Cincom devoting serious resource to moving it beyond its 1.0 state in the next 18 months.

(Various questions) They had 7-8 people full-time on Widgetry for a while to port VW’s IDE UI and could not sustain that. Porting the workspace (that’s a simple UI!): doIt goes to the compiler which is hard-coded to expect a ParagraphEditor. A number of things like this emerged. They had to keep old and new working (there’s some stuff from the UI framework before Wrapper). Business-case-wise, the new GUI framework is less important than the web opportunities.

Seaside is a major new initiative. Michael Lucas-Smith is now leading the Seaside work. They see that as a major way of reaching new customers for Smalltalk. J2EE, .NET and web services are heavy and there is now a lot of reaction against them. Lighter frameworks like RoR gain traction because of this. Seaside does better than RoR. Michel Bany has been maintaining the VW Seaside port. They now want to make it supported and integrate it with VW facilities (less use of Squeak compatibility stuff).

- They want to make it easy to choose how to load and use (just Seaside, not ‘Seaside for VWWebTookit or Seaside for Swazoo?’ type questions). Seaside will now use OpenTalk-HTTP. OpenTalk is VW’s general distributed programming library.

- They will keep strongly in synch with the Squeak version: tools to support automatic code mapping between the two will be provided.
- They will provide relational database support (see Alan’s GLORP talk)

They are also improving network support. Streaming large content (HTTP serving and mail messages - SMTP, POP, IMAP) is being improved, as are handling mail messages in general.

Q (James Foster) Fast CGI? Alan sees it as obsolete. (James) Reverse SSL; do we have to handle the reverse proxying? No, it works OK. You can use Apache as front-end.

The have a MySQL connect preview. Alan is using GLORP to parallel (and much surpass in scalability) RoR’s ActiveRecord. (e.g. RoR uses a naming convention for foreign-key to database mapping, but a database already knows this mapping and GLORP reads it from the database).

Q. Available in other dialects? Some of the recent code Alan is presenting is in Cincom OR (licensing to be reviewed). GLORP is available in several dialects. [Niall: I’m a fan of GLORP. I’ve learned by experience that some ports talking to some databases see more use than others. If your planned dialect-database pair is not one of the common ones, talk to current or most recent users in your dialect and/or talking to your database; it may repay you in lessened set up and configuration time, and robustness of use.]

RuntimePackager is being improved to make it easier to switch between headless and headful. Some stuff, e.g. debugger stack traces, is being moved into the base, so RTP is less needed. The base image is being made smaller (for example, now that they have the stack trace in the base, the old debugger is no longer in the base).

Scripting support: they want to run scripts from the command line or files without launching the whole base image. Thus they want the base more tolerant of headlessness. Some string-handling operations are easier in scripting languages than in Smalltalk; they will try to make VW less verbose in these. See the package ScriptingSupport in the Cincom OR.

Q(Paolo) I will demo scripting stuff tomorrow and others have also done work; interested? Yes; we don’t want VW XML format to be the syntax.

(Alan skipped several slides for time.)

Shadow Compilation: atomic loading is in 7.5. but switched off. It exploits namespaces to compile in a sandbox and then maps to the actual location only when (if) it all does compile. There are interesting issues. A VW subclass of the compiler will rely on a class (inst?) var being populated which this breaks at the moment. Overrides are also not yet handled.

Q. Applies to pundle loads / pundle updates? Yes, a pundle loads atomically, but prereqs are loaded separately.

Q. Atomic publishing? This is the case now. The questioner thought he saw the opposite that very morning when he lost his connection while publishing a bundle; several packages were published although others and the overall bundle did not. Alan will check and fix it as needed.

Q. Graphics work? Heavyweight solution is e.g. Cairo; Travis doing work but it is experimental at the moment. Some lightweight work is being done.

**VASmalltalk, John O'Keefe, Instantiations**
John left IBM after 39 years just to stay in Smalltalk. He started in the 1980s, doing extreme modelling - see Andreas' talk - without knowing it. Instantiations has an interesting history. It was formed by developers from Tektronix many years ago, then bought by Digitalk, from whom some spun off to form ObjectShare which swallowed Digitalk from which some left to form the second Instantiations.

VASmalltalk 7.5.2 has Windows Vista (32 and 64 bit) support. It supports SuSE, Red Hat and Ubuntu with 32 bit and 64 bit. (It does not have 64 bit support as such but it tolerates 64 bit.) It integrates the Refactoring Browser and the RB Envy extensions, and SUnit with SUnit Browser.

They tolerate and take advantage of Windows Vista but are not certified and do not seek it (he's been through that process before). User Account Control: Vista has really tightened up security - in the windows sense. :-) Users run as standard or administrator (who runs as a standard user). You have to identify upfront what security resources your application requires. This is done in manifest files (usually; you can also set properties). They ship these manifests with the executables, some of which require privileges. Installing on Program Files mean standard users cannot modify files there so the image refuses to run, with unhelpful error messages. Future releases will prompt about this; for now, you have to know that you must copy your images elsewhere or not install under program files.

Windows Aero gives you new colour schemes, themes, widgetry and animation. John is not wild about it but it looks nice and was easy to use so they use it. Windows Vista themes are not enabled by default but can be turned on in the manifests (mostly) or by explicit invocation if you want custom controls. They supply some manifests with commented-out sections that you uncomment to get it. He showed a window with lots of widget types in classic windows, then in Vista without manifest and then in Vista with manifest (see his slides to compare the look in each).

VA also offers widget add-on kits for custom widgets, e.g. WidgetKit. These are *not* theme aware. WindowBuilderPro *is* theme-aware.

Windows help support now relies wholly on compiled html and deprecates .hlp; Instantiations have replaced their few remaining .hlp files.

SuSE and Ubuntu were mostly a platform validation effort; very few changes were required.

Windows large address support lets applications use 3Gb of address space.

They support Oracle 10 LOBs and BFiles being stored outside the database (which holds pointers to them), and microsecond-precision timestamps.

They have integrated the Refactoring Browser (thanks, Niall) and the extension of it to call Refactoring Browser features from Envy Browsers (thanks to Joseph and Alan, and to Niall for maintaining it) and also the checkpoint stuff that Adrian did. The also have SUnit and SUnitBrowser. The ENVY/QA code quality suite is also in the product. It is extensible so you can add new tools (alas, documentation on how to do this is poor, but he has found it easy to add new tools).

Q. Envy/QA has problems with `abt*` methods, so he has to make it omit them. These are the generated methods. (Another point) Their generation changed from 6 to 7, giving spurious differences in comparisons. John offered to discuss offline to make the default settings what users want.

Code Coverage tools is fully integrated with the development browsers. Code formatting can be customised; there are three different formatters you can use, the basic, the Refactoring Browser and the customisable one; they aim to coalesce these.

His near term goal (in next version if possible but not guaranteed) is Seaside. The WebConnect and web services frameworks are heavy and it is great that the Smalltalk community is coalescing on Seaside. He had hoped to show the Seaside counter running on their 2.8 port today; alas, he cannot but they are very close to having it running. Right now it runs on their Http server. They are feeding information back to Sport when they find compatibility issues with this Seaside port. However the main issue is that the context model in VA is not at all like VA and this is the major task. They would rather make VA support continuations without modifying the VM but they may have to.

Their web services support is rudimentary (workspaces where you copy and edit templates) and they will work on tools to improve this, gathering together in one place all you have to do to create a web service.

Over the years VA has tried several IDE approaches, e.g. TrailBlazer (which is like Dolphin’s idea-space), VAAssist and so on. They aim to consolidate and rationalise all this, plus the QA and other tools above, to if not one then certainly fewer browsers.

Installation and uninstallation is being simplified and made consistent across platforms. Support for DB2 V9 and Oracle 11g may be added and they are looking at GLORP for object persistence. ObjectExtender is their old such framework but it was bloatware (very large and has had no updates for 5 years).

Their documentation is HTML based and looking old. They will replace it with PDF (but adobe reader does not run on 64bit today) or CHM. They will maintain the cross-book search and invocation-from-image features.

They have had 6500 downloads. They have 2200 active users and 225 customer companies. They have been very successful in converting the IBM installed base to Instantiations (which was a bit of a surprise to IBM). Via an IBM contract, John also supports IBM VisualAge Smalltalk on mainframes, which IBM still sells, so if you need it on 390, talk to him.

Q.(Thorsten) How many developers? More than just him and less than Cincom :-). It is about half- a-dozen. They are almost all in Raleigh NC except for Eric and sometimes a consulting opportunity causes one of them to be pulled off development.

**Free GemStone / Seaside in Linux, Monty, GemStone**
In a ten-minute slot, he explained how small apps (including commercial apps) can run for free. He ran through the Seaside-implemented web applications that guide you through starting GemStone and coding in it. For details, see James Foster's talk (also my 2007 Smalltalk Solutions report).

**GLASS: Gemstone Linux Apache Seaside Smalltalk, James Foster, Gemstone**
Gemstone is the second oldest Smalltalk dialect but it has not had much of a client interface till now. GLASS provides the Seaside coding web interface. (Martin McClure is also the volunteer graphical designer; the GLASS logo is his.) James started by reviewing Gemstone's current state, then focused in on GLASS.

The US congress seems to think there's no problem they cannot solve and decided to solve the latest energy crisis by changing when the sun rises and sets. Gemstone provide a TimeZone patch to help their clients deal with this, which has been like Y2K for some. You can now have a great many flexible timezones. Another blessing :-/) from the U.S. congress was the Sarbanes-Oxley act. They have made the tranlogs easier to audit.

Q(Bruce) Captured in syslog? No, they have their own logging in the database and this tool runs over it.

They have done more 64bit stuff, including: multi-threaded stone, polymorphic lookup caches, some float operations are now primitives, there are ten writelock queues to let you get another if your first attempt failed. They have done locale work for international customers. RcQueue adds are faster (millisecond-based), size can be preconfigured and reduced conflict equality indexes are available.

There is better handling of LostOT (if nothing done, eventually thrown out with exception thrown). Sorting in collections is done better. #and: and #or: have been optimised. They now offer level B support (you get supported but don't call us in the middle of the night) for Intel Mac libraries.

James was a customer before he worked for Gemstone and it long had a reputation like luxury cars; if you have to ask how much it costs you can't afford it. At Smalltalk Solutions, they announced a *free* edition of Gemstone. The GemStone Web edition is a no-cost licence (even for commercial use; this is not a trial or non-profit-only licence).You can have a 4Gb image (i.e. repository) size with 1Gb shared page cache. (Changed since Toronto) they no longer limit you to 2 VMs; discussions with early users, Avi in particular, showed them that some web transactions are long (credit card authorisations can take 5 seconds and you cannot block all other users while that is happening). This licence is to run on 64bit Linux on 64-bit Intel / AMD / etc. (but not, of course, PPC) i.e. AMD64 instruction set, i.e. x64 instruction set. It must use only one CPU (maybe should be corrected to 'one core') on one host.

One of Gemstone's major values is transparent object synchronisation. GemBuilder for Smalltalk is explicitly disabled in this free licence. Alfred commented how surprised he was at the idea of working without having this and James was most pleased that a paying customer (Swiss national Bank) could not imagine transferring to the free licence; their market segmentation is clearly OK. :-)

Other Seaside-supporting Smalltalk dialects are single-user non-persistent so an application running on them must add that. Gemstone's value is that it is multi-user and persistent; it can run across multiple hardware and so on. Seaside is a server application so the lack of a UI is not a drawback for GLASS. Persistence approaches in other Smalltalk dialects raise issues. If an image quits you lose data and it is only in one image. Identity is an issue with binary file-outs. Relational databases are extra work, maybe not too much with GLORP but you must code it. Multi-user can be one image serving multiple clients: Avi has a huge number of small databases each accessed by few people so as long as he can route all requests to the right image he's OK; fine for his market segment but not scalable to all.

Gemstone can have multiple VMs each having its own OS process but each with full access to the database. So you have close to linear scaling (close to; there will always be bottlenecks but their work is to remove and streamline them continuously). Customers run 1500VMs on 200 hosts doing 1000 commits per second. They have tested 3000 VMs with 1 terabyte of data (16 billion objects). Customers sometimes ask them to prove it will work with 5-10 terabytes of data and they reply 'send us the hardware to test.' (A question verified these are US billions, not UK ones.)

They have ported Swazoo's Hyper HTTP server and FastCGI server, Squeak's Monticello code store (for File and HTTP types), Seaside 2.6 and Seaside 2.8 and SqueakSource 2.6. They provide Squeak-based tools based on the OmniBrowser framework plus some of its tools.

GemSource is at seaside.gemstone.com/ss. This is a zen image (runs on the same server as runs www.gemstone.com). They have 28 members and 12 projects. They read with some amusement the squeaksource discussions about 'what happens if I lose code'; if your squeaksource db is gemstone you do *not* lose code (server may be put down occasionally but it comes back up, replays any transactions and the code is all there).

Most ports from the official squeaksource need an export process that is slightly involved. GemStone has the ‘advantage’ of having no native source control system they needed to stay with so they simply ported Monticello. Thus they get their Seaside direct from the source (and anything else they want to port). While others try to get persistence, Dale had to struggle with keeping some Seaside objects transient.

Namespaces: different users in their multi-user image may want different code. Gemstone has long been able to namespace classes and globals. Now, each VM can have its own method namespace. They’ve added to the lookup SessionMethods for classes so a method sent to Array looks in the SessionMethods for Array, then Array methods, then the SessionMethods for SequenceableCollection, then SequenceableCollection and so on. Thus each user has the illusion that they are alone in the image and can change their code. This doubles the method lookup but only on the first call, after which it is cached. James would like to develop this further.

They now support _ as the assignment statement so you can import code from Squeak, change in Gemstone and export again and not have to worry about noise in your diffs. They *emphatically* do not encourage using it; it is just there to reduce noise (and yes they could have had other solutions for that). It was noted that Squeak now *discourage* it; maybe they’ll drop it.

A class can now have the attribute DbTransient. When you commit, all the instvars of such an object become nil i.e. they remain local to the session. Use this for Semaphores and suchlike objects you need to be transient but referenced from persistent ones (a typical pattern is to wrap such objects).

You do not want to rely solely on your UI layer to validate your objects (c.f. Leandro’s talk). Similarly when you rollback domain objects, you may not want to rollback your Seaside UI objects. Thus you can put domain objects in one bucket, Seaside in another and have distinct rollback. This is not really nested transactions but just to solve this issue: if the user enters a date of birth in the future, they do not commit the domain object but they do commit the continuation; you want those menubar links to keep working.

Whereas many Seaside applications use a Smalltalk web server in the same image, large-scale applications - i.e. the kind Gemstone will scale well for - will usually want to use a separate web server. You can use e.g. Apache to support ReverseProxy (and/or FastCGI, noting that they have heard today that FastCGI is probably not needed), serve static pages, provide anti-web-hacker security, load-balancing, SSL (https) and fail-over.

Porting: export your Squeak application to the Monticello repository. Load packages into Gemstone and fix the compile errors you will see (Gemstone does not have {}) so you modify the code in Squeak, export to Monticello again, load again. Then you will have initialization errors, usually DNUs referencing missing classes or methods, so load them (or workaround them); many of these will be platform specific. Now compare your image to what is in the repository. They should be the same; if they are not, something did not load. Now look at Undeclareds: the key is the global and the value is the set of classes and methods that reference it; fix those. Now run your SUnit tests and finally check the application's behaviour by hand.

Q. Why not add {} which is useful, instead of _ which is not? Alas, Gemstone has another use for {} and they offer similar semantics. As they were porting, they just found a lot more underscores than curly braces. It was also a lot easier to make the change. However Dale commented they had not finalised this. We know that since version 2.7 there are no underscores left in Seaside so it may be removed again.

Q. Which version should we use? They have ported 2.6 and 2.8. Judging by Lucas' talk, you should use 2.8. An early customer with a 2.7 app ported it to 2.8 to move it to Gemstone and is happy with 2.8's behaviour. Dale was on vacation (in the 'Seaside' resort community in Oregon) when he ported Seaside 2.8 and it was an improvement, cleaner and leaner. Some stuff he did for 2.6 he realised was no longer needed.

Porting has been discussed in detail but it has indeed been fairly easy. Dale ported Pier to the state of loading cleanly almost over a coffee break.

Q(Tim) Automate builds? Yes, automatic build tools could be written.

They offer a shared Gemstone server on the web, with Sandbox etc. All this works today but has just been set up and various bits have teething issues being solved. James closed by stressing that Gemstone does not imagine they know Seaside better than the audience. Their aim is to provide tools to let Seasiders go on doing their good work.

He demoed running Linux Gemstone on VMWare on a Windows machine. There is a webpage you can visit to smooth the non-trivial installation of Gemstone. Seaside is pre-installed. A Squeak image is provided with some tools, call the UI to set where the server, the database, etc., are. He opened the GS/S Monticello browser which was looking at code in Gemstone. He opened a SystemBrowser G/S[1] to browse some of the ported Monticello code, itself in Gemstone. These Squeak tools give you an environment with a user interface. He ran 100 factorial in Gemstone from a GemStone/S Workspace he opened in Squeak. This does NOT do object marshalling between GS and Squeak (c.f. Alfred's question), That GBS product exists for VA and VW and is very sophisticated.

OmniBrowser separates the user interface from the domain so the Squeak UI can talk to the Gemstone domain. OmniBrowser UIs with their panes showing lists of things get populated from and talk to the Gemstone domain objects. He showed classes from Gemstone's base in the OmniBrowser UI. You can browse Gemstone code in one window and Squeak in another (and when porting that can be useful).

Gemstone is a very solid Smalltalk product. Seaside is ported into Gemstone and Seaside is very solid in Smalltalk. These tools are more recent and rough ports, useful but a month or two away from availability except in beta.

**Impromptu OS8 Demo, Georg Heeg**

What Georg demoed became product that day. He installed via the regular windows installer; they do it that way because Vista does not allow (by refusing to certify) any other installation. Open source DLLs must be signed by an authority. Now that folders within ‘Program Files’ are not writable any more (see John O’Keefe’s talk), you choose where to install it.

This was Georg’s favourite Smalltalk project in all his 25 years because it was so synergetic. Seaside.String>>+ was the only incompatibility (Georg thinks Seaside’s implementation of String>>+ is lousy). He showed how in VisualWorks the ObjectStudio look is wholly available. Grab a window and swirl it around to see if it is an ObjectStudio window or a VW window: emulated widgets wipe-out while swirling, OS native widgets do not.

He did `Date today inspect` with a halt (OS text menus do not include ‘Debug’). Compilation does a tree transformation turning `method` to `os_method` (see the os-substitutiontransform shared variable for a list of all 110 of them). Default implementation in Object just maps back to the original call. The other part of the transform is the namespace ObjectStudio which imports as many VW classes as possible but each is chosen by name; it does not simply import all classes in a namespace.

What can OS offer VW. It has a native windows GUI and a designer (palette of widgets), and a modelling tool.

Q. Will we see OS packages in Store? Those parts that implemented this were developed in Store in a Cincom internal repository. The distribution is via Smalltalk archives which present the loading order in the same way as Store. A Smalltalk archive is a tar file of parcels, ordered correctly for loading, which a Smalltalk utility reads directly; see the SmalltalkArchive bundle. It would have been easier to do it all in Store but this helped existing OS clients defer Store learning.

Q(Martin) Tried running ObjectStudio with Gemstone? Georg has not tried it yet; he wondered if GemStone had? The Gemstone guys had not (they only realised this week that the possibility existed) but thought it should work. Georg invited them to try it (he gave his first Gemstone demo in ten years in Vienna last week with no Gemstone-knowledgable guy there :-/) but Gemstone was not in his demo image because of the Vista signed DLLs issue mentioned above, so they continued offline.

Q(Tim) What’s the future of OS? OS was developed concentrating on database integration and native widgets, then sold to Easel then sold to Cincom. Many improvements were made in that time but not to stability. Cincom invested in stability. Until this OS8 initiative, they just bug-fixed it. OS8 did compatibility. Georg foresees completeness for the next step (adapt components that have not yet been adapted). The next step could be the UI; the look is Windows 95 timeframe. OS is seen as a core product.

(Alan) The OS team has had a large chunk of infrastructure removed by consolidation; they now have less to maintain and more ability to develop.

**Paolo Bonzini, GNU Smalltalk**
In the last two versions they have worked on support for scripting. The GNU Smalltalk has an image but it is not as central as in other Smalltalk's. You can start from a fresh image at any moment. The image is built up from the base rather than being stripped as in other Smalltalks. (It takes one second to rebuild the basic image). On a command console, interpreting what he typed, he showed using collection protocol to manipulate stream contents. A generator takes a block and returns a reference: he showed using these in `anySatisfy:` with a `yield` call creating a continuation used in the next iteration of the loop.

He showed putting the stream, not its contents, directly to files. He ran some SUnit tests. He showed unicode manipulation done directly and after loading iconv into gnu, after which he saw strings displayed more usefully. He compared the sizes of two same-no-of-chars strings in unicode and utf8 (second twice first, of course). He then ran the II8.IconvTest SUnit tests.

You load volumes to connect files on your system to gnu for handling. After making them visible, he loaded NetClients, MySQL bindings and GLORP (usual demo hiccough: had to remake at one point). He then ran the large number of GLORP tests.

GNU has a small graphical interface but it is not fully developed for lack of time. He wants to port the OmniBase framework to get access to some good tools. He showed the simple UI: class browser, transcript, normal Smalltalk debugger. Finish (i.e. Resume) in the debugger uses a clever continuation to run the rest of a method at full speed.

GNU 3.0 has a better script writer (written in Lugano last semester) and better import from other dialects.

## Compilation Research

**Reflectivity: sub-method reflection, Marcus Denker, Philippe Marschall, David Rothlisberger, Nik Haldiman, Stefan Reichhart**
(Adrian Lienhard and Lucas Renggli helped, as did Eric Tanter, Stephane Ducasse and Oscar Nierstrasz.) Smalltalk has reflection which is good for development environments. Methods are objects but their contents are not. It would be nice to have sub-method-reflection. Some tools that do this exist [Niall: e.g. the Zork-Analysis package in VW, SmallTyper in VA] but in most reflection frameworks the method is an irreducible object. They made a compiler that can generate a reflective method. When you try to run it, the runtime complains and generates the compiled method. At any time, you throw away the compiled method to regenerate the reflective method.

Marcus demoed: he inspected a class dictionary containing reflective methods, then he ran the code and reinspected to show that all methods run had been converted to compiled methods.

Code rewriting can be done during this conversion from reflective to compiled method. For example, all assertions could be removed as the compiled method was generated. They can also add annotations to code knowing that these will be ignored on conversion and so at runtime, or else rewritten by a variant converter-to-compiled-method defined to convert those annotations to acceptable Smalltalk code. They can bind elements to meta-objects with activation conditions, etc. (In Java, this idea has been explored but it is very ugly to implement since it works by manipulating bytecode. In their system, it is very neat since they know much more than the bytecode level does, so the code they generate can be very efficient.)

He demoed the bouncing atoms app. He made it beep every time an atom bounced. That was rather too often so he made just one atom beep when it bounced (turning it a different colour so we could see it was working OK).

```
beeperLink := (AtomMorph>>boundIn:) sends...
```

Q. Can you apply this to any method? A primitive method can map the alternative code into a reflective method, then convert to the primitive.

Q(Niall) This allows dynamic rewriting of annotations; can it handle dynamic collection of type data for variables within a method, as Zork-Analysis does? Marcus said it could but I did not understand how since conversion of the whole reflective method to a compiled method occurs before any part of it is run, so the points for collection were no longer there. It seemed to me that the reflective method substructure would have to provide an evaluation framework, c.f. Michel Tilman’s Abstract Grammar, so collecting type data by evaluating the method via that framework, instead of converting and running. We lacked the time to resolve this.

Q. Can you have several links on a single parse node within a reflective? Yes, but only one can be the replace link used by the method converter.

**Squeak VM Performance, Bryce Kampfjes**
The Squeak VM is not hugely performant but it is simple and robust. It does 9.5 clocks per bytecode, so if you remove instructions but do not fix what is holding up their execution, that does not much help performance. It takes 314 clocks per send. Branch mispredicts takes 10-30 clocks. There are L1 and L2 caches and RAM access takes 100 clocks. You can write one word every 4 clocks on a good CPU, slower on others so if you try to write faster you end up queuing and that’s a hit. Modern CPUs do out-of-order executions so you only pay for your worst delay.

They have a method cache and recycle contexts (flyweight pattern). They inline `interpret()` and do indirect jumps at the end of every bytecode.

Context recycling lets you avoid creating and GCing a context every time you need one. In Squeak, contexts are always normal objects (unlike some dialects which reify them only when needed).

Q. When are they initialised? When they are created; you can’t have uninitialized contexts or your GC will get unhappy.

Uncaptured contexts are recycled. They are held on a linked list and a counter; a sorted collection would get two pointers to the same thing and then crash. (The Appel argument does not apply to Squeak because Squeak does not use a copying collector.) Anything that can access a context prevents context recycling, i.e. normal stack accessing but also blocks.

Method caches provided a 30% gain for the benchFib benchmark and will give much more in most cases. [Niall: see Eliot's PIC talk at ESUG 2000]

The interpreter loop is wholly inlined to one vast C function, thus exposing all of it to C's optimising compiler. An indirect jump, not a switch statement, jumps at the end of each bytecode. In tinyBenchmarks, you spend 95% of your time in this interpreter loop of which 78% is spent compiling bytecode: this is where to optimise (branch mispredict time is the next, low-grade, rival to this). Waiting for memory or on branch mispredict means computing power is being unused.

Squeak's GC is generational compacting, and mark and sweep ('a funky combination'). A remembered set is a form of write barrier for pointers from old space to new space, since old objects can point at new objects and you want to compact newspace without looking at oldspace. You can have pointers and byte storage in the same space. This GC is simple and does not waste much RAM. It is fast enough for the interpreter.

The GC can have four branch mispredicts in certain cases, so we could use card marking. A copying new space collector could make it quicker to free objects (most objects die young) as you could just ignore the dead objects, not free them. Giving SmallInteger tags to 0 and pointer to 1 could be an improvement. However changing the GC would change the image format.

Q. Why? In an addition, you type-check both args, then remove both 1 tags, then add the ints, then check and re-add the 1 tag to the result. Pointers don't care what their tag is but by the laws of arithmetic, SmallIntegers do care. If the SmallInteger tag was zero, we could just add them.

Duplicating bytecodes was suggested as an optimisation. If you have one copy of a bytecode in a method you have many branches at the end of it. If you have multiple bytecodes, each bytecode is easier to predict. This made sense when branch mispredicts were more common than on modern CPUs, which have been optimised to the point where this optimisation offers much less gain. Only 20% of relevant time is spent in branch mispredicts.

Q. Why is this 'change image format if change GC' a problem? It is an organisational problem. Four Squeak teams handle various aspects and they do not release on the same schedule.

**Exupery, Bryce Kampfjes**
Exupery is an MIT-licence Squeak program that other dialects can use if they wish. It is a native code compiler that is trying to compile well. Writing a compiler is mostly a software engineering problem, not a domain problem. Most issues come down to bugs, reliability, etc. Exupery aims to be 2-4 times as fast as Self. This requires a heavy duty optimiser which will be slow. Self's inline compiling was very sensitive to pauses and Exupery

has to get around that somehow.

Compilers are complex so he is writing it in Smalltalk, a simple language, to get round that. It is not trying to replace the interpreter.

Smalltalk is slow for a number of reasons.

- We send frequently.
- Our methods are too small to optimise. This is a real killer. An optimiser likes to find loops and then move things outside them.
- We do a lot of tagging and detagging: before you do anything with a value you need to check that it is the kind of value you are expecting. Just because you are doing a + doesn't mean it is a SmallInteger.
- We have to do range checking: we avoid all C's crashes but we have to do all that checking.
- The write barrier is also an overhead. GC has two impacts: its own performance and all the bookkeeping GC needs to do its job. (GC is faster than malloc though not as fast as a perfect hand-written solution.)

To make Smalltalk fast you need:

- dynamic inlining: remove your common sends and expose loops to the optimiser
- a decent optimiser that removes redundancy

Bryce' analysis is that we can reduce Exupery from 2 bytecodes per clock (which is also what VW gets) to one bytecode per clock (C's range).

Dynamic inlining is conceptually very simple: I am calling this a lot from this implementor so inline it. Self had a counter at the front of each method but Bryce has begun with a Smalltalk-only approach. However dynamic inlining makes debugging much harder (since what the same operation actually runs can vary dynamically) unless it can be controlled easily: you need to get the code cache in the same state.

To beat Self's optimiser we need what Self did not have. We can combine type checks and range checks and move them out of loops. If `at:` is to be a single instruction then it cannot do type-checking, have a write-barrier, etc. Moving stuff out of loops can remove redundant checks. These moved operations can affect control flow, which adds issues but we know what these issues are, so can handle them.

Optimisers are slow; worst-case algorithms are $N^2$ or $N^3$ which is not what you want if your compilation time is a concern, but the ordinary range is NlogN which is mostly acceptable - jet liner software would not want it.

Compilers are complex so you don't want to add complexity from the language. If you get a segmentation fault, it is nice to know that it is your fault, not your language's fault (i.e. your generated code's fault, not the fault of the compiler you used to generate that code). By compiling in the background, it is easier to run the compiler inside the image without hitting

"I'm compiling but I need to compile something to run the compiler." Generally, Bryce cannot afford *not* to use Smalltalk: life is too short.

Self stopped execution while compiling, a great approach if you can compile fast enough. If you can compile even faster, you could compile everything before you execute it (c.f. Mimic: a research interpreter from IBM; they published some nice papers) and so solve a lot of problems by storing stuff in the compiled code you will soon throw away. Or you can drive the compiler manually as Lisp does. Bryce is compiling slowly but in the background; it's a lot easier to decide the details of this when you have a real system to play with.

Exupery is a large hobby project being done in small blocks of time, an hour here, two days there, five hours later, some long pauses. He uses test and incremental development: Compiler bugs are a pain, especially real ones. (Real ones are almost the only ones in Smalltalk; everyone in this domain knows the unreal ones that are raised in other languages that are in fact the user's code's fault - but can you show that?) Another issue is planning: this is a domain with a rich literature and many research papers but researchers sometime publish those results that make them look good, so you must plan to monitor, choose, investigate, verify.

During development you normally only debug new code and avoid regression (reappearance of the same bug). If one of your test-driven development tests breaks, you are promptly doing serious low-level debugging. Some time must be spent maintaining and refactoring your tests. He may spend a few hours thinking about and writing a test; a week later he runs it and starts adding its feature to the system, often coding in the debugger.

Acceptance and System tests are less useful. He kicks off the compiler and lets it run till it crashes (typically 15 minutes at the moment). Performance tests are also needful.

Dynamic compilation bugs depend on what is in the code cache, maybe 1500 methods of decompiled code. To debug, he starts by getting the code cache log and replaying it; does it happen again. Some bugs are intermittent - a GC happened at an unfortunate time. Gradually you focus in on one or two methods. Since such bugs often happen only in odd-ball conditions you *must* control what is in the code cache, which Smalltalk makes easy.

The current system has no dynamic method inlining nor a proper optimiser. It does inline primitives. It is a basic compiler with a colouring register allocator. It can now run in the background thread (needs a little more debugging). It runs on Linux and on Macintosh (since yesterday, and only with gcc optimisations switched on; a gcc bug in Mac is suspected).

He is still thinking about how to avoid the cost of slow compile times. You could stick the compiled code in a database and distribute to many machines, only one having to have compiled it. It is all just objects so how do you keep them and ship them around? How should he allow frameworks to modify the compiler? A general compiler cannot by default do many things that a specific framework using a compiler could safely have it do.

Q(Bruce) Are you happy for people to use only bits of it? Yes; two projects already are doing that. We coordinate by discussion. At present, the two projects say, “We want to see it improved so just keep pushing forward and don’t worry about coordination with us”, which makes it easy.

## Java Connectivity

### Calling Java - The JNIPort Framework, Joachim Geidel, blueCarat Consulting

Joachim has ported Chris Uppal’s JNI framework (www.metagnostic.org) from Dolphin to VisualWorks. We have a wonderful language so why call Java? It keeps IT managers happy that you can answer ‘Yes’ to ‘Can you use JMS?’ ‘Can you use ...’. Secondly there are many packages written in Java, some of which are worth using, others of which you may need to use and most of which you have not the time to rewrite. Having this ability lets Joachim use Smalltalk more often than otherwise.

Joachim looked around, found Chris Uppal’s framework and decided porting it was the best solution for his case. JNIPort lets you use any Java class with any Java VM in Smalltalk. It is free (Chris’ licence says do what you like but don’t claim that you own it or that you wrote it, and don’t sue me). Joachim ported it in his spare time in 2006/7. To call some Java, do

```
jvm := JVM current.
class := jvm findClass: #’java.lang.System’.
class currentTimeMillis_null
```

A helper DLL and some Java callback classes sit over the base Java along with the Java Native Interface code. The Smalltalk side of the callback handling, along with basic wrappers, make up the next layer and the static and dynamic (ghost) wrappers sit on top of that.

JNI is an invocation interface, using the Java VM (a library, not an executable) function table and the JNIEnv function table. You must remember that, for Java, local references are valid in a single thread while global are valid in all threads; misusing crashed your app.

```
jniEnv := JNI library new
               createFirstJNIEnv: JavaVMInitArgs new.
math := jniEnv
          FindClass_name: ‘java/lang/Math/’
          onException: [...].“this line optional”
absID := jniEnv
          GetStaticMethodID_class: math
          name: ‘abs’ ‘sig’ ‘(D)D’“path notn, not dot”
          onException: [...].“this line optional”
arguments :=
  JNIValueArray fromArray: #(-321.2d) types: #jdouble.
result := jniEnv
          CallStaticDoubleMethodA_class: math
          method: absID args: arguments.
          onException: [...].“this line optional”
env JavaVM destroyJava.
```

He really likes that last method name. :-) He showed the model of the function table pointers (see slide diagram). The JavaVM table has 5 functions, the JNIEnv has more than 200.

Q. Do you need a separate invocation method for each possible type of argument? Yes (aren’t typed languages wonderful :-/).

Using this low-level API is not much fun. So then we move up a layer to ‘The Twilight Zone’. Here references are automatically encapsulated within Smalltalk objects. We can send messages to Java statics and instances, not directly to the JNIEnv. There is automatic garbage collection and we can use reflection (Java reflection capabilities) to ask e.g. what messages it understands. Now we can use dot notation:

```
jvm := JVM current.
zipfileClass := jvm
                 findClass: #’java.util.zip.ZipFile’.
args := JNIValueArray new: 1.
args objectAt: 1 put:(‘file.zip’ asJavaString: jvm).
zipfile := zipfileClass
  callConstructorSignature: ‘(Ljava/lang/String:)V’
  withArguments: args.
size := zipfile callintMethod: ‘size’.
entries := zipfile
  callObjectMethod: ‘entries’
  signature: ‘()Ljava/util/Enumeration;’.
```

Q. Why do I have to pass the Java VM? We are constructing, and getting back a reference to, a java.lang.string it so we must know the external Java VM we will use to build it.

There are many predefined wrapper classes, but many objects are instances of JavaLangObject and we use reflection to find out what it is. Instances of JavaLangClass are then built from the reflection API. His slide of the predefined Wrapper classes shows a few grey abstract classes and many yellow boxes for Java classes, plus a JavaPrimitiveInstances class for the Java primitive non-objects.

Working in this layer is still not ideal, so the topmost layer is Ghost Classes, where we just send messages directly, as in other Smalltalk. Hence we need wrapper classes for each Java class we will use. These Ghost classes are generated automatically when the first reference to an instance of that class appears in your image. When a ghost class no longer has an instance, it vanishes automatically. (Later discussion indicated this is not quite the situation in all cases.)

Q. Performance hit of not keeping the classes? No performance problem in the uses he had so far. You need to do this dynamic (re)generation if you want to be sure of up-to-date compatibility.

Ghost methods are generated from the reflection API. Context-specific info is embedded into CompiledCode as literals (be aware your decompiler will not like this so your debugger may also show it strangely). In

```
jvm := JVM current
zipfileClass := jvm
                findClass: #'java.util.zip.ZipFile'.
zipfile := zipfileClass new_String: 'MyZipFile.zip'.
size := zipfile size_null.
entries := zipfile entries_null.
entries asEnumeration do:
  [:each | Transcript cr; print: each].
```

the methods with underscores are automatically generated (you can configure the string that is appended to the generated methods to denote arguments). The Ghost class is a Smalltalk class that wraps the static part of the Java class. JavaLangClass by reflection gets the rest of the class. These Ghost classes have a hierarchy parallel to Class and Metaclass in Smalltalk. If you are not confused by this meta-hierarchy then you are one of those unusual people who like to study meta-circular class definitions. If you are, be assured you do not need to worry about it. :-)

He reimplemented all the Dolphin JNI tools. The Settings tool is where you specify the path to the JVM and the runtime command-line settings, plus many configuration options. It is a good idea to use an absolute path to the JVM *and* for the class loading path: different versions of a Java runtime on your computer mean the Java VM can get confused about which path to load from. Once Java 1.6 decides to load a Java 1.5 class you are doomed.

The Class Wrapper browser lets you browse classes and methods and the decompiled code of a ghost method (will show you the embedded method ID argument, bypassing dynamic lookup of the method id so calling is much faster). The JVM id is also embedded because it is the JVM that can throw an exception (which JNI will re-raise in Smalltalk). The Inspectors will show you e.g. Java strings.

He demoed calling the Java logging framework from a VW workspace.

```
...
logger severe_String: 'Arggh'.
...
logger severe_String: 'Better now'.
...
```

Doing this in a workspace gets you a warning that some messages are new (ghost methods not yet generated); you just proceed and the methods are there when called. He ran and opened the logfile to show it had worked. He then invoked Swing, making the Java VM load a lot of Java classes and JNIPort generate a lot of Java classes. The window appeared and he then ran some more code to show a JPEG file (usual demo hiccough; JPEG file in the wrong directory).

He then demoed callbacks from Java to Smalltalk. He opened a Swing window that showed the Smalltalk class hierarchy obtained from Smalltalk. Calls to Java are fast. Callbacks to Smalltalk are slower so use them minimally. If you passed a Smalltalk code block to Java, you would get into trouble if Java then ran the callback in a different thread, as it can do. So JNIPort has a HelperDLL. Java calls back into the HelperDLL which queues the request and then passes it back to Smalltalk in the same thread as the call. You must write some wrapper code in Java to use this HelperDLL; it is straightforward for an experienced programmer but he recommends using it sparingly as needed. One use is Java Messaging Service; if you want to receive messages from Java asynchronously you need callbacks.

He plans to allow generating static wrapper classes instead of generating on the fly. This is for performance, on his laptop, it currently takes 8 seconds to attach a Java VM. Mac and Linux should be easy to add. Disguising Java packages as Smalltalk namespaces should also be doable. Now that Dolphin is no longer being developed, so the need to keep in synch with new Dolphin versions of JNI is unlikely, perhaps we should unify JavaConnect and JNIPort. See slides for code and doc webrefs.

Q(Andreas) method call speed benchmarks? The stable way to attach a JavaVM is to generate all the ghosts at start-up. The other way (option in settings tool) is to generate classes when a message is first sent to an instance. Startup is then less than a second but it sometimes crashes strangely. Sending messages to Ghosts should be far faster than sending to static classes since the latter cannot embed the IDs. (Andreas: Cincom has experimented with embedding all the information into a DLL and obtained microsecond times.) A document management user of JNI who were using web services have already seen a huge speed-up so at the moment it is fast enough for users, Yes, it could be optimised.

Q. Visibility? Ghost classes do not appear in the RB, etc., because those classes do not know their subclasses, will be GCed when no longer used, etc., all of which the standard VW tools dislike.

**Cava := Eclipse asSmalltalkPlugin, Joachin Brichau and Coen de Roover, Universities of Louvain and of Brussels**
Joachim wants to reason about Java applications using SOUL and IntensIVE. Coen and Joachim are researchers studying code. They can do a great deal in Smalltalk using SOUL, Moose, etc. However they often have industrial partners who use Java code. Most of the time they import Java code into Smalltalk via parsers. This is done via an adaption of Frost called Irish (http://prog.vub.ac.be/~jfabry/irish/).

However, Java is really not Smalltalk. You have to implement symbolic resolution, call graph and control-flow analysis, etc., plus when your tools change, the code of the original is unchanged, and you must keep up with the way Java changes (this was a big problem for Frost). All this is too tedious but they were not going to move their tools to Java/Eclipse.

However they can reuse Eclipse facilities such as resolve name and call graph static analysis. So they built Cava (Code reasoning for Java). JavaConnect is very similar to JNIPort; it is a different implementation but a similar approach. Using this, they can call the Eclipse interface to get Java parse trees. Then they can use their tools to reason over these.

Using the Seaside web class browser, you can start a JavaVM loading an application and browse its code. In SOUL you do

```
?project isJavaProject
```

and you can see a very raw dump of all the code written unformatted from the Eclipse AST nodes (that's what Eclipse does; not so impressive). They can open Smalltalk inspectors on Java objects. They can install additional Smalltalk methods into the Java class wrappers (as can JNIPort; maybe they do it better) so you can provide better display methods. This Smalltalk mixed-in to Java could be done via Traits for better separation.

So queries might be 'All invents published on the bus should not be modified thereafter' or 'Which classes should be enums' (enums are available in Java 5), which could be done via:

```
?cu isCompilationUnit,
?cu hasclassDeclaration: ?class,
?class classDeclarationHasBodyDeclarations: ?body,
findAll(?field,and(?field isChildOf: ?body,
          ?field isPublicStaticFinalFieldDeclaration,
          ?field hasNumericType),
       ?fields),
?fields hasLength: ?l,
[?l > 2]
```

(as their examples show, SOUL syntax is now very Smalltalk-like), or 'Refactor all for-loops to use enhanced Java 5 style' so that:

```
for(Iterator i = c.Iterator(); i.hasNext();)
  {String s = (String) i.next(); ...}
```

becomes

```
for (String s : c)
  {...}
```

Every AST node in Eclipse has meta-information which you can use to generate appropriate Smalltalk to navigate the tree of nodes (e.g. methods `hasNumericType`, `isPublicStaticFinalFieldDeclaration` and `hasclassDeclaration:` above). He showed the full query which was several lines of SOUL. There are utilities to provide static points-to and call-graph analyses for Java programs. They can use these to simplify such query expressions. For example, finding accessor methods needs two quite complex SOUL queries. They want to let people write just template methods that they convert to queries via tools. For Java programs, they make the template like java with logic variables. Find accessors:

```
if jtClassDeclaration(?c) {
  class ?c {
  private ?type ?field
  public ?type ?name(){ return ?field }
  }
}
```

Find concurrent collection modification:

```
jtStatement(?s) {
  while(?iterator.hasNext()) {
  ?collection.add(?element);
  }
},
jtExpression(?iterator){?collection.iterator()}
```

Cava will be in the Cincom OR soon. It is currently focused on SOUL and IntensiVE. Joachim is working on extensions to use Moose as well. They would be glad to integrate other code analysis tools if anyone offers them. See www.info.ucl.ac.be/~brichau/javaconnect.html.

Q. Difference between JNIPort and JavaConnect? The approach is the same. They subclassed to have JavaMetaclass and JavaClass to get similar tool behaviour. They have integrated Java packages to Smalltalk namespaces which JNIPort has not yet but they do not yet have callbacks which JNIPort does have. They have discussed and should merge.

He browsed it in the Refactoring Browser, then started Eclipse (took a few seconds) and opened a large project from the drop-down list. The first query you run takes a while as they are cashing all the ASTs (he ran a simple `?cu isCompilationUnit` query - it took quite a few seconds to find 130,000 - to effect this); thereafter it is faster.

## Development Processes and Frameworks

### Working Smarter not Harder: Development Tools, Processes and Automation, Angela Wilson, Northwater Capital Management

Angela has been working at Northwater in Toronto, Canada for 5 years. At Northwater, 90 people manage $9 billion of assets with clients in the Netherlands, the US, etc. There are 4 smalltalk developers working in VA and Gemstone. Bob Nemec is the creator and lead developer of much of what she will present today. Her earlier talk (Smalltalk Solutions 2005, Florida) was tool-focused. This talk is more about their processes and how they have evolved over the years. (Be aware that what works for their small team may not work for you.)

They have a custom Northwater menu to open their tools: login window to their app, the instance migration tool, etc. They have customised their Refactoring Browser and many other tools.

They run the exact same code in VA and in Gemstone, all version-controlled in Envy. A string-comparison tool keeps the image in synch with Gemstone. It used to be done manually and was much performance-tuned. Now it is done automatically and noone cares about performance.

In Gemstone, you can have multiple schemas at the same time. You will want to map instances in classes of the old schema to instances of the new schema. Gemstone provides a basic utility for this and their tool gives it a UI and means to find which collections of instances should be migrated (they rarely use `allInstances` migration). Some years ago, they took a few weeks to migrate 30 million objects, which was one tenth of their database. The tool lets you execute on the database you’re logged-in to or lets you generate a script which you then apply to other test databases and finally to production.

Q. Lazy migration? No; Northwater always migrates actively. (She heard about JPMorgan’s approach just yesterday.)

They use WindowBuilder and have tools to let them inspect their attributes and accessors (i.e. aspects), to assign types, etc. They can use this to check that objects are in a valid state. She showed an example of a visit to a hedge fund, navigating to ‘who visited whom’, etc. Edit policies, themselves with UI for editing them, let you define how developers see things and how users see things. This lets them offer a tool quickly for later development into a custom tool. A range of buttons let you launch code and WindowBuilder views of components.

They use TotallyObjects’ socket package to know when users have a runtime error. Users are sometimes slow to call about errors and sometimes Northwater calls them first: “What did you do here?”. Some users want to be notified of specific system changes. Sometimes their optimistic locking policies see conflicts and if they see that happening regularly, that is a prompt to think about revising that part of the system.

Their GemStone monitor is a large complex window. They monitor long transactions (sometimes someone opens a dialog box and then goes to lunch) and use of old systems. All users have an asynchronous abort every two minutes [Niall: this is a common approach]. They also have some buttons that the users can configure for themselves. All users see all windows and permissions control what they can do (they used to hide windows but that was confusing).

Q(Bruce) This works by statmon in background? Although statmon is running in the background, this data is not from statmon but by querying the gem.

Q(Bruce) Inhouse or via library? Inhouse. (Bruce) You make it sound so easy? (Gemstone people) It *is* easy.

Their nightly automated process was one of the first they developed. Originally, it collected quotes, copied them and assigned them to tasks. Now it does much more, backing up the database, verifying a random selection of objects that were updated during the day (the accessor specs plus any rules e.g. bidirectional links having both ends set), user-requested business checks, etc. This also checks sessions to see if users are in a strange state (did someone leave for the weekend and forget they had a transaction uncompleted?).

They keep reviewing their approach. They sometimes try something for two weeks and see if it works or not. They have an in-house-developed issue management system.

Northwater has to comply with regulatory standards so *every* change to the system has to be documented and justified. It seems like a lot of red tape, but they were able to implement it so that it improves their process and reduces walkbacks. Some changes are non-business (fixing typos, spotting better ways to code something) and they run regression testing on these and if it passes, the change is OK. Otherwise, every code change is associated with an issue and on release the change report captures that relationship.

Their old process was a one-week cycle. On Monday, they decided what was to be changed. The person tasked with a given change would figure out the migrations, package changes in a set of files, run tests, and effect the change. However this meant that they would get no feedback till Friday, when the release was already being packaged. A small problem meant that the next release was a follow-up. A large issue meant delaying release. (A good point was that the release cycle was short.)

Their new process is a two week cycle but they start new every day. If Angela fixes an issue on Monday of the second week, the business users must test it on Tuesday. If they do not like it, it can be pulled out and given another couple of days work before code freeze on Wednesday for Thursday and Friday's release testing and packaging, after which production is updated over the weekend.

The rule is that whoever changes a class must write the migration script for it. All test databases are linux servers. Cron jobs update them and emails arrive every morning telling them whether each built and how long it took. It is better to see any issues in this mid-cycle than during release packaging. This is one of several heartbeat emails they expect to see every morning.

Code export and VA runtime image build are triggered by a windows scheduled task. The image is built using the VA packager. They also package the GBS client.

They have three different types of test. Small SUnit tests check specific functions. Large SUnit in-memory tests check the image but not the database. Because the same code is in Gemstone and VA, they can run tests on both. TestMentor tests on a copy of production may not catch everything they would like to test as it is constrained to actual production data. She would like to test on a clean database loaded with test data.

Q(Georg) You have two apps, one for VA and one for ObjectStudio. How do your processes differ and is OS8's ability to use Gemstone of interest? The OS system is focused on trading and they do not like to have issues after release so the effective release package is two weeks beforehand.

Q(Mike) Geographic distribution? The server is inhouse in Toronto. People login from various locations e.g. New York. They could launch the server locally to them but do not at present.

Q(Joseph) Does anyone work on Sunday and complicate your release process? Do you have to work every second Sunday? You must check noone is logged in plus everyone can login from home and technical support does that on Sundays to update.

Q. Smalltalk gives you competitive edge? Yes and the management do notice that we are managing $9 billion with just 90 people.

Q. Do large features need more than two weeks? Yes. We break them into workable pieces that fit the cycle or they can have code that is regression tested but not in a release.

Q. Getting your business users to test in the cycle was easy? No. They are happy to do it for high-risk, high-change features. Otherwise, it tends to be the Northwater business analysts, not the customers, who do it.

**Expressive Testing and Code for Free, Tim Mackinnon, Iterex**
Tim was in Smalltalk at Carleton University years ago then was (forced into) Java and C#, etc. He is glad to see SUnit still there and the inspiration of all the JUnit, NUnit, etc.

(He gave a brief demo of Dolphin's model-view-presenter abilities by displaying the font size with a slider to expand and contract it till he got the ideal size for audience.) TestRunner UIs in all Smalltalks tend to give minimal indications of what failed; you go to the debugger to see what is happening. During test-driven development that's OK; you are probably working in the debugger anyway. (At university, his professor said, "If you don't find any errors in your code - you should be very worried!")

His tool takes the assertion that failed and extracts some info from it, e.g. *Constraint Error: Set('Alice' 'Bob') should include: 'bob'* lets you see immediately the trivial error you made.

TestResults do not contain the exceptions that raised failures. He started expressing his expectations as Constraint objects, e.g.

```
(Equal to: 5) verifyWith: 8
```

As he produced a hierarchy of constraint objects he found he wanted not just to check the values he got but make assertions about their distribution e.g. that they were not all equal to one another. He noticed that whenever he found an error he would rewrite it as a constraint.

In Java he made much use of MockObjects. (He is aware that in Smalltalk MethodWrappers can be used for this purpose.) In test-driven development, he creates proxies that fake up parts of the system he is not working on yet but which the area he is working on needs to call. He realized his constraints interact with these.

```
nameParser := mockery
                    createStrictMock: #SUnitNameParser.
...
documentor
  process: ...
  using: nameParser ...
  expecting: [nameParser
                 parse: (Kind of: String)& 'tc'
                 answerWith: #('...)].
```

In Java you have no blocks so the style is more upfront declaration and less readable. The parse statement uses `Kind of:` and `& aConstraintExpr` so the tool can use that in presenting if it fails.

These test constraints express the mock protocol so he can generate the protocol when he starts work on that part of the system.

In future he plans to gather useful constraints (e.g. Sequence, Different, etc.). He will look at non-simplistic code generation: could one infer missing or conflicting test cases from your constraints?

The code is in Dolphin but is fairly generic; proxy objects can be done differently in different dialects.

**Code Measures, Tim Mackinnon, Iterex**
Tim gave a ten-minute talk on code complexity. Keith Braithwaite proposes a complexity measure that takes the log of the cyclic complexity of non-test methods. Keith claims, based on analyses of Java code, that things below 2 are suspicious and things above 2 are good. SUnit has value 2. Swazoo's packages are mostly well above 2 except for SwazooPlatform. A Dolphin utility that Tim knew contained poor code scored well below 2.

Now that Dolphin 6 is halted, Tim encouraged other vendors to replicate its nice IDE features in their IDEs.

**Extreme Validation, Leandro Caniglia, Caesar Systems**
Leandro subclasses TestCase to distinguish the abstract classes ConcreteTestCase and Validator. A Validator will have methods `validateThis`, `validateThat`. A `test...` asserts that some value is correct. A `validate...` asserts that some aspect is correct (by 'aspect' understand instVar).

```
self aspect: #someAspect.
self aspectVerifiesInvariant ifFalse:
  [self failBecause: ‘It is not OK’].
```

Example: a class NormalProbabilityDistribution should never have standard deviation of zero. ProbabilityDistributionValidator has validates:

```
validateSd
  self aspect: #sd
  self valueIsPositive ifFalse:
    [self failBecause: ‘StandardDev must be positive’].

validateMean
  self aspect: #mean.
  self valueIsDefined ifFalse:
    [self failBecause: ‘ProbDist must have mean’].
```

This subframework of SUnit has everything that SUnit has. However you can write general validators on your classes:

```
Variable>>validate
  self aspect: #name; validateNotBlank.
  self aspect: ‘quantity; valueIsDefined.
```

and then call them in your validates:

```
includeValidationOf: value
  self isDefined: value.
  value validate
    allValidationFailures do: [:each | each resignal].
```

You can also define general validation checks e.g. `namesDoNotCollide`.

Whereas SUnit verifies specific classes and methods, i.e. code, validators validate objects. The validation happens during the application as a result of user actions as much as of specific test running. He is not claiming that validators are better than test cases. He is just explaining how they are similar and how they are different. Tests check his code is correct. Validators check his objects are valid. Validators are activated by requests from the application and so can be left running (and so will report to) end-users in real use as well as programmers during development, if you wish.

Every time you run a test you start from scratch recreating a set of objects to test. You validators run on any objects your image contains, however constructed. Running validators is much faster than running tests because you are just checking state, not creating an object structure to test. Leandro also thinks it needs much more skill to write tests than to write validators.

Q. What message do you show the user given that `isValid` just returns a boolean? Where do you validate: in the UI or model layers? They validate in the model layer: whenever the model changes, *all* its validators are run.

He looked at the example of a relational database providing objects to the Smalltalk image, perhaps via some intermediate ‘table’ objects which are then reified to application objects. Both the table objects and the reified ones can be checked, the latter at a higher semantic level of course. Validators detect bugs earlier; an object can remain broken for some time before an application reveals that something is wrong whereas the validator will detect the model-layer change that caused it.

He showed the petroleum application, inspected a reservoir object and ran all its validators. Next he browsed a well, entered invalid data and the validation framework promptly reported this. In this application they only validate the object that the user’s current pane is viewing by default; a screen lets all other objects be validated.

Q(Christian) Link back to input field? No; would be nice to have. (I mentioned an example of how this could be done, described in my ‘Value of Smalltalk’ keynote at Smalltalk Solutions 2005.)

Q(Tim) Are you forced to have observers of every aspect? No, the object validates when it is told to do so, which mostly means when the user orders it, but their app is also coded to validate every time the UI is refreshed.

Q(Bruce) is this code available and if so under what licence? This is Andres Valloud’s idea originally. The framework is available in the Cincom OR and he’s writing a book in which one of the chapters is about this.

**SPORT BoF, Bruce Badger, OpenSkills**

Sport is a Smalltalk Portability library and API. From the point of view of any one dialect, it is a library. From a generic point of view, it is an API. Using Sport, Bruce can develop applications in VisualWorks and then run them in Gemstone with no code changes whatsoever. Sport works in VisualWorks, GemStone, Squeak, Dolphin, Visual Smalltalk (VSE), etc.

How to port Sport: port a Sport-using utility of your choice (Bruce recommends Hyper or PostgreSQL), attempt to run it in your Sport-free dialect, and fix the missing things.

Ideally, Sport should shrink as dialects standardise on it. Inconsistency between dialects will cause it to grow, which at least will make such inconsistency visible. Extending Sport's API is fine; all the existing Sport-using utilities will still work. You need only ensure that your additions get ported to all dialects. Refactoring *does* break code, so why refactor. Well, some things in Sport are horrible because they expose existing nastiness; we could have better socket structures, etc.

- Some people do not want their use of Sport to break.
- Some people will use two things that depend on Sport, so must have any revised API released in both at the same time (or must upgrade at the same time)

The Sport version naming scheme uses letters: A B C D. If you email, use the letter, e.g. SportA or Sport(A).

Sport solves things that are candidates for the ANSI standardisation process. Let's use Sport to motivate the ANSI standardisation process.

Sport is being documented: see www.openskills.org.

Q. Do we really need an ANSI standard? Yes: one of the perceived issues with Smalltalk is that its dialects are not compatible.

Q(James Foster) Why are the API versions separate from those of the implementation? Because Sport is an implementation in a given dialect but an API overall. If I change how Sport methods are implemented in a given dialect (e.g. for efficiency or because a new dialect version has new features to exploit), or add or correct methods in that dialect for a given API version, that means one thing. If I change Sport methods' signatures then I must do so in all dialects and that means that I am changing the API.

ANSI exceptions are in ANSI so they are a prime candidate for removing from Sport *when* all dialects are the same but not before.

Q(Andreas) Microsoft library versions have a proper name and then ext1, ext2, etc. which Andreas does not find especially helpful; he wants clean separation of concerns in version naming. Andreas also asked about namespaces, but very few dialects support them. Gemstone and VW are the only ones with namespaces but perhaps two examples are enough to devise a standard. Joseph suggested that a standard Namespace concept was agreed 7 years ago in ANSI; has anyone used it? And if not, does that mean it is not a problem. (John O'Keefe who was also at that meeting seven years ago thought that in fact there was no consensus on namespaces.)

Q(Niall) If, as appeared from some examples, Sport will rely on class-based version names, how do we version extensions (e.g. extending a Collection subclass with a Sport method)? There was discussion and agreement to continue offline.

Q(Mike) People moved away from ANSI for business reasons. Kapital is an example: it has several (too many :-/) DateTime implementations. Bruce suggested Sport be an underlying DateTime, not the perfect abstraction; a better abstraction could use it as a basis. Or its API can become an ANSI standard adhered to by all DateTime implementations.

Q(Joseph) The ANSI standard was put together in 1998 by various people, none of whom are doing Smalltalk now. (The last, Blair McGlashlan, was very reluctantly obliged to halt further development of Dolphin Smalltalk a very short while ago). We can feel competent to restart as suits us.

Q. Sport touches Sockets, DateTime and other areas; should we break it into independent subpackages? Bruce sees ANSI as a process to arrive at a consensus.

Q(Georg) STIC must work towards Smalltalk standardisation. In 2007-8 (i.e. till Smalltalk Solutions in June next year), STIC will make broadening Smalltalk usage their priority. Thereafter, STIC could get involved.

Bruce recommended comp.lang.smalltalk as a place for Sport discussion. Use Sport if it helps you; don’t if it doesn’t. Let Bruce, and others, know how things went for you. (Gemstone uses Sport to port Squeak so e.g. Dale could present on Gemstone’s experience of using Sport as an enabler.)

## Utilities

### Toothpick (SUnit) Joseph Pelrine

The LoggingFormatter lets you log your tests (or whatever) as simple text, XML etc., formatted however you like. LoggingPolicy decides whether you log each event or log composites subject to conditions, etc. One of Joseph’s clients has a listener on a socket and when something weird happens they ping the file and reset the loggers without affecting the application. This is on VW, VA, Dolphin and Squeak (but Squeak does not yet have the new external loggers).

Joseph demoed some scripts:

```
logger := TranscriptLogger new policy: ..; format: ...
LoggingMultiplexer current
  addLogger: logger;
  startAllLogins.
...
```

then he configured it to respond to an event, send the event and see the log written to the Transcript.

You can download a log viewer or log to IRC by outputting the log to a little Java program that writes to IRC. This could be used to note automated builds. In production use you’d want a dedicated socket. Get all this from his website (www.metaprog.com).

Toothpick has full documentation, unlike much else he has seen.

**Custom Refactoring, Niall Ross, eXtremeMetaProgrammers**
The custom refactoring project (http://customrefactor.sourceforge.net) adds refactorings to the Refactoring Browser and usability features to the rewrite tool. In VW7, this is add-on work. In VASmalltalk, VAST and VW3 (both Envy and non-Envy) we are the prime maintainers of the RB and its Envy extensions.

Where to get our releases:

- In VW7, go to the Cincom Open repository and find the ‘Tools-Refactoring Browser’ bundle. Our versions have ‘released’ blessing and version names ‘VW... CS... RC...’, showing the VW version they run on, the Camp Smalltalk that the work was done at (or after) and the Release Candidate number. (The latest is VW7.5 CS11 RC3 released.)
- Our VW7 tests are in the RefactoringBrowserTests bundle or in parcels at our sourceforge site; match the ‘VW... CS... RC...’ version.

  ‘Released’ blessing-level versions have no *real* load warnings. (Shape changes to Refactoring subclasses cause wholly spurious warnings in VW7.5 and earlier. RBTestMethods has bad code meant to be found by code critic, so will warn.) Except in pathological situations, your open base-release RB will still work after loading our version (of course, closing RBs before loading variants is good practice in general).

  ***Note:*** the ‘Refactoring Browser’, ‘RefactoringBrowser’ and ‘Refactoring Browser Tests’ bundles in the Cincom OR hold old versions of the RB code which Cincom later renamed. Ignore these unless you are using an old version of VW7; see these bundles’ blessing comments for version data.

- In VASmalltalk, VAST and VW3/Envy (and VW3 non-Envy) go to http://customrefactor.sourceforge.net and download the appropriate zipped .dat (or parcels) to get both our versions and base RB versions. (VASmalltalk releases now include our current stable RB version.)

Documentation: our bundles and config maps have detailed comments. Summaries are on our web pages. Discussions of implementation approaches are in the Camp Smalltalk sections of my ESUG reports for 2001 and after: see http://www.esug.org/data/ReportsFromNiallRoss. Our old wiki pages vanished when the UIUC Camp Smalltalk wiki was spammed but have now been put back.

In the past, we have added refactorings (DynamicRename, SplitCascade, Cascade, RenameInstanceVariableAndAccessors, etc.), rewrite features (rewrite expressions take multiple search expressions and wildcards in method names, refactoring changes inspector now editable), usability features (menu for creating rewrite expressions, better error and warning reporting, etc.), base RB improvements.

What we did this CS: Hernan Wilkinson emailed me code to address a VA performance issue: class rename slow in very large systems; after discussion with John Brant, a final fix was added to our next release. John O’Keefe paired with me to solve some tests failing in Linux. Thorsten Seitz, Katerina-Barone-Adesi, Michael Prasse and Adriaan van Os worked with me on new refactorings. The results are:

- Improved Refactoring: AddParameter now lets you reorder where the new parameter is added in the method and name it.
- New Rewrite: ExtractWithParameters lets you extract code leaving behind some sections to be parameters to the new method.
- BugFixes: in VA, both our tests and the RBConfigurer now behave better on Linux. In VW, renaming Smalltalk.MyClass then undoing the rename caused a walkback in the base release 7.5 RB (our 7.5Base CS11 RC2 is in the Cincom OR).
- Performance fixes: in VA, RenameClass was slow in very large systems due to accidental duplication of checks when the RB was adjusted to Envy. MethodWrappers had a slight infelicity, now corrected. In VW, 'Find Class' in several packages when each package is in several bundles was very slow due to redoing the package checks when the same package was re-encountered in each bundle.
- Making Things Visible: in VA, RemoveParameter was not in the menu.
- UI: in VW, code to switch the 4-pane browser between viewing the whole system and viewing a specific subset of pundles has been replicated to the Cincom OR as ‘Internal Release’ in package RBCustomBrowserUI. A whisker-like code tool that shows all code of several selected methods in a single pane is largely complete in VW and designed to port easily to VA.

Future plans: our work this CS has to be packaged, integration-tested and released. Tim Mackinnon may port our custom refactorings to Dolphin. Travis Griggs will review what might be integrated with base VW. Gwenaël Casaccio may port to Squeak. ExtractWithParameter needs generalising to ExtractToComponent.

**Moose, Tudor Girbe**
The new website for moose is moose.unibe.ch. Moose is moving beyond software to general analysis tools. Moose has a repository to store models, a FAMIX metamodel for analysing software systems. Meta is the engine in which you describe meta-models. They have a Generic UI and specific UIs Mondrian (see his talk last year) and EyeSee (see this year’s demo).

You can get data in by parsing your code and importing or you can write your data in MSE (a very simple Moose format). On top of this are a huge set of tools: Chronia, Cook (visualisation) DynaMoose (dynamic analysis), Hapax (analyse how you name your variables) Softwarenaut, SmallDude. And there are more: BugsLife (analyse bugs to system), Subversion (map from it to Smalltalk via client). And there are more and more and more ...

He then showed some of the visualisations. CodeCity is a 3-d visualisation of a system’s code, whose shapes recall a city’s towers, streets and blocks.

The map colours code based on properties (e.g. where is code that calls SQL in my system).

Actual uses: you can map code ownership and code commits to developers. He showed some initial graphs and then evolved them by enforcing some ordering display lines to make it more readable. Behind visualisation there is clustering - what is the right order of presenting data to answer useful questions - and data collection. He showed browsing from class diagrams to seeing their code in tooltips. He then opened Mondrian's Easel and showed typing a model and seeing its look immediately. Thus you can explore data *and* ways of representing it.

Recently, Moose acquired a new Smalltalk importer (much faster) and improved dynamic analysis and improved meta-descriptions. They are working on a charting engine in Widgetry and are using Cairo.

Moose is a collective effort Stephane Ducasse, Adrian Kuhn and him; the prior team included Michele Lanza, Sandar Tchelar and a huge list of collaborators. At this ESUG, they founded the Moose foundation.

**Code City, Richard Wettel, University of Lugano**
He is researching how to display the structure of complex software visually. The metaphor is to present classes as buildings and packages as districts in a city representing a whole system. The colour and size of the buildings can be assigned to represent various code-related measures such as how many methods a class has, who uses a given facade, etc. This work is Moose-related. He showed flying around the city. He has implemented a simple generic query engine to locate specific classes or methods that meet some condition ad go to them, change their colour, etc.

He uses Jun to render all this.

Q(Christian) Layout is wholly automatic or rearrangable by the user? Wholly automatic.

**Swazoo, Janko Misvek, Eranova**
Swazoo (Smalltalk Web Application ZOO) is a smalltalk web server. At the 2000 Camp Smalltalk, it merged some Smalltalk web projects (hydrogen, AIDA/Web) and was worked on by Joseph Bocancas, Claus Gittinger and some who went to the dark side (but Ken Trace came back again :-). Their aim was to be on all dialects and thanks to Sport (thank you, Bruce) this is now finally becoming true.

Swazoo sits on Sport and provides one or more HTTPServers talking to one or more Sites (i.e. web sites) in a many-many relation, all of which can be on one IP and one port (or on many if you prefer). One user is AIDA/Web, which has Swazoo integrated within it (so new releases have usually already been tested in a production environment). Swazoo 2.0 has just been released. It is on VA and Gemstone and will be on VW and Smalltalk/X soon.

Swazoo 2 has a new request framework (by Bruce) and streaming: input streaming of large file uploads and output streaming when the server changes (reverse AJAX/Comet) and when serving large files (streaming is a popular topic today). It has become more stable and better optimised so can mange 3Mb/sec file upload, which is probably still 10 times slower than Apache but fast enough for many uses.

Use your Seaside or whatever on Swazoo. see www.swazoo.org.

Q. Compare Commanche and Swazoo in performance? He has only studied the comparison with Apache.

**SLAPS, Bruce Badger, OpenSkills**
OpenSkills (www.openskills.org) is a global non-profit association of individuals.

The Lightweight Directory Access Protocol is lightweight relative to X.400 DAP, just as an elephant is lightweight relative to a blue whale. It is a tree structure (c.f. Alfred's talk) that runs a protocol. In LDAP, you can say, 'let there be a new concept', just as in SQL you can say, 'let there be a new column', or, 'a new table'.

In LDAP, data is held in a tree, the Directory Information Tree. Why use it? Because it is used for authentication (prove who you are), authorisation (now I know who you are, what may you do). Any email client will use LDAP to connect to a public mailing list. In more sophisticated situations, people use Kerberos which lets you authenticate once and get a ticket you can then carry around.

Why write an LDAP server? Well it looked like it would just be another wire protocol. OpenSkills wanted member authentication, authorisation, and to make member address books available. Using an external LDAP server is non-trivial: you have to define a schema, have an LDIF export format to map your stuff to it and then you must keep in synch with it.

Bruce showed some LDIF. You could write it but imagine writing the code that will look at an object model and writing the LDIF to map its changes into LDIF; it would be like writing an OO-SQL layer.

An LDAP server listens on TCP/IP port. The client connects and the conversation may go like BindRequest (hello, may I use your service), BindResponse (OK), SearchRequest (tell me about ...), SearchResultEntry (here is info about ...), SearchResultDone (and that's all I know about ...), UnbindRequest (thanks, I'm done). Which would be easy enough except that it uses ASN.1. Abstract Syntax Notation, version 1 ("I hope there is never a 2!") is about how you specify and implement wire protocols (how you convert information to and from octets: 'octet' is a formal definition of 'byte'). If you look at it quickly, it looks OK. Actually it is horrible.

ASN.1 is like Chinese: there is a single written form but when they speak it is utterly different. You can encode a single ASN.1 expression on the wire in many different ways. LDAP uses Basic Encoding Rules (many other dialects: Canonical Encoding Rules, Distinguished Encoding Rules, etc.). BER uses Type Length Data but sometimes it says, ‘don’t know the length, use another approach’. Thus there is no sure way to jump around the spec: you must parse sequentially and completely.

Bruce opened his VW image and ran to a breakpoint he had placed within `testLDAPBindDecoding` (make it run, make it right, make it fast - he is at make it run here). It created a BDU (Binding Data Unit) and got a catalogue: a dictionary binding schema mapping numbers to their types in the schema. You have to use bitmasks all over the octet to read it. You consult the catalogue for these tag classes to see what the next lot of data is encoding. “You thought that was going to be the class of thing we’ll create. No, it is just the class of the identifier that will tell us what we will try to create”. He stepped through these stages to get the universal identifier which will be primitive (basic ASN thing) or compound and will have a tag that tells us what it is. At present he does not parse ASN.1; he hand-crafts class-side methods for the specific bits of ASN.1 he must handle. (SLAPS probably will parse in future but for now he’s still learning.)

Finally we reached the target class which takes the whole spec as input to parsing the octet stream. Inside a huge `self printString` we saw `3`, the actual bit of information that made it through all this wrapping!

As he worked on this, Bruce’ incredulity kept growing. However in the end, SLAPS will give Smalltalk great flexibility. We will get requests into a tree and project that tree to and from our rich object model. For example, the OpenSkills skills tree is a tree and projecting data from it will be much easier than generating LDIF for every view we might wanted to offer. Even more, existing LDAP structures - posix account, address book, DNS configuration, SMTP server configuration - will now be controllable by us.

When it is working and available, you will use it (sort of) like an HTTP server: start server, send request, get response. It will probably be fast enough for long-tail applications and if it is not fast enough for something we use it to replicate to an open LDAP server, just as our HTTP can sit behind Apache if the app is being hit a lot.

Q(Christian) Why so complex? ASN.1 came from X25 and bellheads.

OpenSkills worked on Hyper. Now, running an HTTP server in GemStone seems sensible. Likewise LDAP use may develop. Kerberos could be looked at. SLAPS is in Smalltalk so it will be far easier to learn.

Q. Licence? Bruce has not yet decided.

Q. As client? All ASN.1 is symmetric so SLAPS should be fine as a Smalltalk client.

Q(Bryce) Squeak cryptography has worked on this, also VW? Bruce did not know of the Squeak and wants to put this on any dialect so could not just reuse the VW implementation, plus the VW one is just a client.

### Miscellaneous ten-minutes presentations

Two late afternoon sessions allowed a range of projects to make 10 minute presentations. Those not included in other sections are here.

**The CellStore/XML Project, Jan Vrany**
This is an XML-native database management system. The various features are layered so that if, for example, you only need single-threaded persistence you can remove the transaction and access control layers.

Currently, they have an XML-compliant API, and they can store 500Mb+ of XML documents. Email vranj@fel.cvut.cz if you have any questions.

**Krestianstvo.org, Nikolay Suslov**
Nikolay uses Squeak and Croquet. Krestianstvo is a Croquet SDK-based learning environment. Sophie's XUL logic says how to fill Croquet spaces with content. Nikolay's concern is not modelling virtual reality but adding Croquet displays to real environments. He showed a recent Moscow gallery exhibition, with Croquet environments appearing on gallery walls, then moving to other walls and etc. Another project is a collaborative curved-space explorer. They used Jeff Weak's algorithm to render various curved spaces. He demoed an art gallery portal.

**Three Issues, Georg Heeg**
Our new logo [|] is illegal in most Smalltalk dialects. Georg has made a parcel that makes it legal by overriding a single Parser method to make a one-line change that makes it not error when done or printed. But if he then says 'Format Selection' it still fails because there are 4 parsers in VW, one for real work, one for formatting, one for setting breakpoints and one, that he likes best, for colour highlighting.

A second example converts Smalltalk into non-keyword syntax i.e. horrible Java-style (presumably to show how horrible it is :-).

He did `Date today inspect` in VW, then `Date today os_inspect` and after confirming 'Do you want to start OS in this image?' he saw the OS inspector. `Date today inspect` in an OS-launcher-launched workspace displays the OS inspector and `Date today vw_inspect` in the same workspace shows the VW inspector.

He created a new class with one method, subclassing ObjectStudio.Object, and typed in a TransformedSource window some seriously unSmalltalky, lisp-style factorial code. Then he hit transform and it was in Smalltalk.

**Collection Behaviour, Adrian Kuhn**
When modelling a domain, we tend to create specific collection objects, e.g. a BookList containing Books and similar. Adrian has extended Smalltalk to support this idea. Classes have an instance-side and a class-side. Adrian has added a group-side.

He tried to do `1 group` and had the usual demo hiccough from which he recovered to display an object IntegerGroup which applied to any collection of integers. He added sum to it; now we can sum over collections of integers but collection classes such as Array will still say they do not understand sum because only collections of integers understand it.

The class of such groups is CollectiveBehavior. 20% of Moose’ domain functionality can be held in this behaviour. His problem now is how to extend Store so he can store this code in a Store database.

### Research Track on Dynamic Languages: Smalltalk

#### TypePlug - Practical Pluggable Types, Nik Haldiman, Marcus Denker, Oscar Nierstrasz, University of Bern

Marcus presented as Nik cannot be here today. He started by claiming that static typing was good: uncompilable programs are rejected, parameter names match their types as programs change, etc. Then he stated that static typing is evil for the reasons we all know: cool programs (e.g. those using reflection) can’t be written, only trivial errors are found, etc. This of course is what we believe but we would not reject the information that static typing would give us if it had no cost. Can we have our cake and eat it?

Gilad Bracha proposed pluggable types: optional type annotations that do not change the semantics, simply acting as a Smalllint++ that tells us more. Exotic type systems handle many problems that current implemented ones do not but they are too exotic, too hard, ever to be implemented in a static system. Pluggable types would let us experiment with how to build them.

Alas, one type annotation in an untyped base image buys you nothing. We must somehow type the base system first. There is a sub-method reflection framework in Squeak (see Marcus’ talk on it). Using it, Nik did three case studies: non-nil types (this var should never be nil), class-based types, and confined types (i.e. this instvar is private: never pass it outside the class).

Marcus demoed the non-nil type system.

```
Object subclass: #DemoLine
  ...
  typedInstanceVariables: ‘... endPoint <:nonNil:> ...’
  ...
```

Their browser navigator has an additional pane that shows types. Selecting a setter method that does not guarantee the type shows a warning. Annotating the method return with nonNil type makes the warning vanish.

OK that was a few methods; how to do the whole Squeak image? They looked only at the span of annotated items, used much type inference and made explicit type casts for the rest. They also allow external annotations, since they need to annotate the base libraries but have no wish to maintain a branch of the base, so they add types that are saved as declarative Smalltalk code in a separate package.

Q(Niall) If the base code changes, does your system detect this? Yes.

Where next? The first thing they need is better type inference. They want better algorithms. They want to integrate with RoelTyper's heuristic inference. The second is to be able to check reflective changes.

Q(Christian) External annotations: can we not use them everywhere rather than annotate the code anywhere? That could be done. The external annotations are keyed to the type system that defines them.

Q(Niall) Do you plan to use dynamic type collection (utilities to do this exist; you would need to add merging of data from many images)? It is worth looking at (he would also like dynamic checking of annotated types).

Q. How can you tell if your type system actually works? There was discussion. I felt the answer was that they could only tell in same way that we tell that Smalltalk is good, by using it and seeing what benefits accrue.

**Feature-driven Browsing, David Rothlisberger, Orla Greevey and Oscar Nierstrasz, Software Composition Group, University of Berne**
David is a student in the software composition group. In Pier, actions such as login, add page, copy page, etc. are all features. A feature is something a user can trigger. If a user finds a bug in e.g. copy page. the maintainer must find where that feature is implemented, so fixing the bug requires searching to see what code supports that feature. He proposes representing features as a tree of method invocations. The idea is to collect the called nodes dynamically while exercising the feature, then display this information in the IDE.

Features could be selected by observing user actions or by executing test cases. They have implemented a feature browser in which the user selects a set of features. Called methods are represented by boxes, colour-coded to indicate which features are used in every feature of the set, which are unique to a single feature and which have various partial degrees of coverage. Other browsers are

- an adapted class browser that only shows the methods the feature uses
- a called-method tree display

He demoed, loading Pier and running some test cases for it. Some of the tests failed. He looked at a failing assertion, then selected its test case class and invoked his feature browser on all its five tests. We saw the diagrams of the methods called in each of the five tests, colour coded for coverage between the tests. Only one test failed so he looked first at methods unique to that test. He opened its method tree browser (also colour-coded) and thus examined them and found the bug. (He could also have opened a standard refactoring browser in which only methods called by that test were visible.)

They experimented with fixing bugs using standard and feature-oriented browsers for a group of 12 students. The feature-browsing group found the bugs 56% faster and fixed them 33% faster. Of course, this was an artificial experiment with deliberately-introduced bugs.

In discussion, it was noted that often common methods fail in one test and not another because different data is presented, or different state is held, in the two cases, but the colour view of what is unique and what is common is useful, as is being able to view an RB of just what methods a test called.

Q(Georg) in VW, DakarTesting also has the feature of mapping the test to the methods that it called.

**Bridging the Gap between Visual Morphic Programming and Smalltalk Code, Noury Bouraqadi,**
Morphic makes all GUI elements into objects you can handle, move, resize, send messages, etc. He showed us doing menu New > New Morph giving a catalogue of morphs he can drag to the screen. Selecting halos lets you change a morph or write a script for it in EToys. He created a script (`ellipse fowardBy: 5. ellipse turnBy: 5`) all by drag-drop; he did not write a line of Squeak.

This visual way of writing EToys scripts does not scale up to complex applications. The set of expressions is limited and it is harder to reuse code within the visual environment. By contrast, real Squeak applications are built by programmatic GUI construction; the GUI toolset is not especially rich, just because the programmatic interface is so powerful.

They want to build real applications whose GUIs are built wholly by direct manipulation of morphs, while the model layer is built in the standard Smalltalk way. They want GUI code reuse. They want the GUI code naturally stored with the rest of the project.

They have created Easy Morphic GUI (EMG). A mediator connects a model layer built by standard Smalltalk coding and a GUI built in morphs.

He showed building a Counter and giving it a GUI via the EMGGuiMorph subclass CounterGUI (EMGGuiMorph is both a mediator and a morph). The CounterGUI is then the container for other morphs which complete the application GUI. CounterGUI is created programmatically with an instvar counter that holds its model. He then opened it, using its halos to set size of the plain rectangle that first appeared.

He then created two button morphs for `+` and `-` and dragged them to the CounterGUI morph, where they jumped to the LHS because that was the layout policy. He changed the policy to free layout and placed them where he wished. He also added a morph displaying a number 0. He set the buttons to send `increase`, `decrease` to the CounterGUI morph (via menu and dialog). He set the number to get the result and the UI now worked. He showed the mediator methods `increase`, `decrease` now in the CounterGUI class.

Q. How does the mediator know to update its display when a button is clicked. It is a dependent of the counter and gets update changes.

Reuse has two approaches. One is embedding, e.g. the drag-drop we just saw. The other is to clone all source morphs to a destination morph, preserving the link to the source and updating when the source changes.

He demoed this by building a circular counter. He showed the very simple CircularCounterGUI class he started from, then cloned statically (dynamic clone also supported) from CounterGUI to get the same interface. He then changed the counter GUI and saw no change of the CircularCounterGUI. He then reset the displayed CircularCounterGUI to have a dynamic clone of CounterGUI. He then changed the latter and saved it, at which point the CircularCounterGUI changed to match (usual demo hiccough; re-layout did not stay within the outer morph as layout policies were not the same).

They use a null object pattern to handle missing morphs, which is a common situation early in development. UIs is serialised and stored as string in class method on the mediator.

EMG is very new and needs further work. They want to integrate with morphic wrappers, integrate better with morphic and EToys, and show links between morphs visually.

They are now using it on larger examples e.g. simulating robots. It seems to meet their needs. See csl.ensm-douai.fr/EasyMorphicGUI.

Q. How do you visualise references between morphs? EMG can have morphs reference each other circularly, etc.

Q(Tim) Morphs seem to have intimate knowledge of your mediators (for example, the demoed views knew about `increase` and `decrease`). By contrast, Dolphin's Model-View-Presenter pattern keeps them separate and so allows much reuse of mediators; will you leverage that? There was discussion with more discussion offline. Tim also mentioned that Dolphin serializes with symbols included so that refactoring will work (e.g. if you renamed the `increase` method).

Q. EToys integration? In April he forgot his watch at a conference so built an alarm model. He showed this, then used EMG to make the colon blink when the alarm is running.

**Redeveloping with Traits: the Nile Stream trait-base library, Damien Cassou, Stephane Ducasse, Roel Wuyts, LISTIC University of Savoie, IMEC, Leuven and Universite Libre Brussels**
A trait is a group of reusable methods. A trait has some methods it implements and some it requires. They provide a form of multiple inheritance and handle conflicts when same-name methods are inherited. Traits are implemented in Smalltalk and also are or will be in Fortress, Scala, Python and Perl 6.

TMagnitude implements > >= <= ... from < and =. They can be used by several classes, Date, Number, etc.

Traits have never been used to design a framework from scratch. They want to research issues of good trait granularity, etc., so therefore created the Nile Stream library. Streams are naturally modelled via multiple inheritance. Writing `nextPut:`, `nextPutAll:`. Reading `next`, `next:`, `peek`. Positioning `position:`, `reset`, `setToEnd`. There are existing implementations, some Trait-based. In the Squeak single-inheritance implementation, various methods are provided at top-level, then disabled at lower levels, then re-enabled in ReadWriteStream, etc.

Nathaniel Scharli refactored Streams using Traits in 2003. This was a pure refactoring. The new implementation, Nile, is based on a core of traits: the Reading, Writing and Positioning Traits. Then there is the collection-based trait and one class for each original Squeak class. The also added file-based streams, random number generator stream, a SharedQueue stream, etc.

TGettableStream is used by 22 classes. TCharacterWriting requires 1 method and provides 8. The whole has 18% fewer methods, 15% less byte code and no unwanted behaviour. Nile is always as fast as the Squeak implementation and sometimes faster: traits do not compromise performance.

Issue: traits require a lot of accessors that may not be needed by a class-based implementation. You may also require two setters / getters, one the class' one with initialization or lazy effects, another the trait's simple one. There are more entities: 11 traits and classes compared to 4 (larger and uglier) classes (Stream, PositionableStream, ReadStream, WriteStream) in the original Squeak.

Thus they feel they have really improved the design but want to do further work. Nile has hundreds of tests and can be used today.

Q. Why need extra accessors? There is a paragraph on that in the paper.

Q(Andreas). Right balance between trait and class; has Nile gone too far putting all the behaviour on the traits and using classes only for composition? They chose the traits to reflect ANSI Smalltalk which defines Streams by protocols, not classes. (Andreas) the same issue occurs in typed languages, e.g. Java has heavy use of interfaces in their Streams; have you compared with them? Not yet.

Q(Thorsten) Observation: if we had real multiple inheritance as in Eiffel, perhaps just by a mechanism for pushing the state to other classes, this would solve the same problem.

Q. `peek` could be provided via `self next; position: -1` ? Yes.

**Transactional Memory for Smalltalk, Lucas Renglii, Oscar Nierstrasz, Software Composition Group, University of Berne**
Smalltalk gives you `Semaphore forMutualExclusion` (all dialects) and `RecursionLock new` (some of them) and `Mutex` (Squeak only; lets processes wait without blocking, etc.). Using these can be complex. Misusing them can lead to deadlocks or starvation, or to priority inversion

when a low-priority process is in critical section.

```
tree := BTree new.
lock := Semaphore forMutualExclusion.

lock critical: [tree at: #a put: 1].“writing”
lock critical: [tree at: #a].“reading”
```

Transactional means making a block atomic, i.e. write-access runs in isolation and read-access can just be done ordinarily.

```
tree := BTree new.
[tree at: #a put: 1] atomic.“runs in isolation”
tree at: #a.
```

They implemented this with lazy code transformations, method annotations and context dependent code execution. Their runtime engine decides when to execute in transactional or non-transactional mode.

CompiledMethod subclass: AtomicMethod (uses framework of Marcus Denker). AtomicMethods are created lazily as needed. As soon as we enter the transactional context we must stay inside it to completion. Therefore they prepend all relevant message sends, so that, for example,

```
BTree>>at: aKey put: anObject
  | leaf |
  leaf := root leafForKey: ...
```

transforms, with all the messages it sends, to:

```
BTree>>__atomic__at: a Key put: anObject
  | leaf |
  leaf := root __atomic__leafForKey: ...
```

State access must also be transformed:

```
  leaf := root __atomic__leafForKey: ...
```

becomes

```
  leaf := (self atomicInstVarAt: 1)
             __atomic__leafForKey: ...
```

This works for over 90% of cases but some code must be protected from this. Their own infrastructure code must not be changed (to avoid endless recursion). Exceptions must be passed out and execution contexts used by exception handlers are not transformed. There are many primitives that improve the speed by directly accessing the state and the atomic must call the replacement code or their code. Variable-size object accesses must be treated as message sends to be tracked.

Every system process may have one active transaction. If activated, it tracks all changes to objects touched inside the atomic block. A hash map maps the previous objects to the changed ones for instvars (by far the most common case). All the changes are mapped to the working copy during the atomic block; on clean exit, these are mapped back to (i.e. overwrite) the originals.

He showed some benchmarks: method invocation is equally fast for normal methods and half as fast for special byte code methods. Instvar read and write is massively slowed (by 20 times for named instvar read, 19 times for named instvar write, 18 times for indexed instvar read, 17 times for indexed instvar write). He then showed N concurrent edit operations in Pier. Standard Pier uses a global lock, which works fine for up to 20 processes and slows for more. The atomic case was a consistent 200ms slower than the global lock. With 2 cpus it should have half the angle of increase so should overtake the global case (in theory, but no Smalltalks exploit multiple cores today).

He sees this being used in concurrent applications, in atomic source code loading and in context-oriented programming.

Q(Georg) This was demoed 21 years in Gemstone; what is different? Travis answered: Gemstone does it in the VM; this shows how it can all be done in the image.

Q(Christian) Why will two CPUs speed things? During the transaction everything works on copies and only the commit phase needs to be atomic. This is usually a much shorter time than the whole atomic code.

Q.(Thorsten) Your read appears trivial but it is in fact a retry? Yes, limited times. It could be smarter, checking whether objects have been touched.

Q.(Alan) what happens in atomic commit; you block everyone? Yes.

Q. Nested transactions? Not supported; all is handled in the outer transaction.

Q. Primitives? Squeak has so many that it is not possible to check completeness generally. We are sufficiently complete for Pier but another use would uncover more work to do.

## Research Track on Dynamic Languages: Python, PyPy, HALO, AmbientTalk

### Context-oriented Programming: Beyond Layers, Martin v Lowis, Marcus Denker, Oscar Nierstrans

(This was a Python talk.) Martin started by explaining COP before explaining how they have gone beyond it. Programs have contexts:

- environment state data
- what user is accessing the system: e.g. an administrator may have access to different facilities from others, which can be implemented by layered methods, and ordinary users may each access their own user-specific data, which layered methods are not suitable for implementing
- mode of execution: e.g. rendering may depend on the output device.

Method layers add to Object-Oriented Programming the idea of a group of classes and methods making a layer whose contents are all used together in a dynamic execution scope. A class can have some of its slots and some of its methods in a layer, others outside it. Layers can be explicitly activated by a code block that turns them on or off.

An example is a web client whose web server's behaviour can vary depending on the user agent header. An automated web browser may need to act as a user agent. If the client consists of multiple modules each using different software layers to access the underlying HTTP libraries then you cannot pass the user-agent info. If the client is multi-threaded, a global is not a solution. This happens in Python where the HTTP library has just these limitations.

A `with:` statement (enter/leave behaviour, c.f. method wrappers) can let a method layer send the user-agent string in the HTTP request even though the called method does not pass it explicitly. A new class can inherit from a class and from a layer to augment the base class with the layer features. Its methods have the same name as their supers with extra parameter `context`. He talked through how 'before' behaviour could add the user-agent header or (via 'instead' behaviour) suppress any other user-agent headers added.

The above is what layered methods are in Python today. Martin has added implicit activation: layers have conditions and when the environment satisfies them, they are activated (in contexts where they have already been listed as available layers).

This research looked at what layers are used for today to find patterns. They assumed that caller and callee run in the same context: a layer will apply to both or neither. He studied three examples. Two web apps (Django, Roundup) needed to handle the 'current request' notion. They were passing parameters representing this context through many layers. Another application used dynamic variables (variables that were specific to a given session/thread) accessed via `with:` calls attached to appropriate methods; the effect is to hold the latest value set in a thread-specific way. He looked at method wrappers in Squeak and implemented dynamic variables similarly in Python.

He concluded that dynamic variables solved cases that method layers do not handle well. Dynamic variables use thread-local storage and are accessed by doing a stack walk.

Q(Christian) The combined semantics is hard to follow; can we avoid that by simply modelling the problem better and ending the need to supplement with layers? There was discussion; Julian thought not.

Q(Andreas) Often Georg Heeg find themselves mapping (or advising on mapping) refactoring single-user applications into multi-user applications. They sometimes use thread-specific data to do this conversion. There is an analogy.

**PyPy: How not to implement VMs for Dynamic Languages, Armin Rigo**

Armin is interested in how to implement complicated dynamic languages in the context of limited resources. Smalltalk can usually be implemented in a small core VM since mostly it is written in itself and there are relatively few primitives. By contrast, Python needs a large VM containing e.g. dispatch tables and much else in C code. Python itself contributes relatively little to the ability to run Python.

So if you are not a huge company who can afford to spend a lot implementing a variant VM to study something, what can you do.

PyPy’s architecture has a PythonInterpreter written in a very high level language (not having to address memory management and suchlike), which is then compiled down to C. Using their framework, they can generate from this a pypy-c standard-looking Python compiler in C (and they can generate a Prolog, JavaScript or Scheme compiler similarly).

He demoed the approach for an interpreter of a tiny language. The interpreter loop was very simple, just iterating round bytecodes. He showed its code, then started the framework’s translation of it. The interpreter has no type annotations so the framework performs type inference on it. Python does not fix even which fields are allowed to exist on instances so there is a lot to infer and no reflective features to exploit in the high-level language, which is effectively a restricted subset of Python (so would run on any Python compiler).

Next you have two paths. You can specialize for an object-oriented environment (CLI, JVM or JavaScript) or for a low-level environment (C, LLVM). The output from this is handed to your choice of various backends which create the code for your chosen environment. Helpers deal with specific issues: for example, C is unfriendly to advanced garbage collection, so a helper pushes references into functions as necessary.

So far he has simply turned an interpreter into a more efficient interpreter. Any interpreter can be converted into a just-in-time compiler. Both walk the bytecode to produce machine code but the JIT eliminates the loop by treating the bytecode as a constant and propagating it inside the function. You need to write 10 or 20 hints but that is OK in the context of generating a 20kloc Python VM. For example, the interpreter treats SmallIntegers like any other class; your hint may make them tagged in the compiler. The original interpreter is also retained in the VM since JITs are good but you don’t want to just-in-time compile everything.

He had the usual demo hiccough while trying to make the generated VM execute 2 + 3. [Niall: in Smalltalk, the convention is to execute 3 + 4 in new VMs; Smalltalk, always ahead. :-)]

Q(Stephane) what is the impact of the language you want to implement? They have validated this approach for Prolog, a language unlike Python. It is much easier to produce a JIT compiler than a normal one because a JIT can get feedback from the environment at runtime. This stopping and waiting for runtime info is the same approach as the Java Hotspot compiler’s polymorphic inline cache usage.

This is an approach which was chosen for large VM creation but could be applied to small VMs like Smalltalk.

Q. If you started all over again but with Smalltalk instead of Python, what would you change? He would use the same general approach. They wrote the type inference and backend in parallel and might distribute tasks between them differently in a new case.

Details are on http://codespeak.net/pypy. They develop in Sprints, programming camps; there may be one in Berne or in Germany soon.

**Forward-chaining as an implementation strategy for the history-based pointcut language HALO, Charlotte Herzeel, Kris Gybels, Pascale Constanza, Coen de Roover, Theo d'Hondt, Vrije Universiteit Brussels**
Charlotte works on HALO: History-based Aspects using LOgic. Aspects aim to allow you to decompose your program in several ways, not just one, so that concerns that would otherwise be scattered in the main functional decomposition can be united. In practice this means before, after, around type behaviour addition. She talked through a Java-oriented example of adding concerns to account crediting and debiting operations.

She showed some CLOS pointcut code (to create user, set username, etc.) re-expressed as logic facts. These facts can then be chained. HALO allows temporal relations (most recent, all past, etc.) so that, for example, the sequence login -> buy -> checkout -> logout is valid whereas invalid sequences will not match. These temporal predicates are higher-order predicates: you put the particular temporal chains in as their arguments. Other predicates can define things like discounts for specific promotions, etc., and so be unified with users that match the discounts' conditions.

She showed the architecture: a base of predicates and a fact-base of specifics are acted on by the inferencing engine.

She then looked at backward chaining versus forward. Backward would launch a specific rule (e.g. discount? user? rate?) and try to unify for the terms. She ran the rule from the slide (her slides were in Squeak) and showed the query trying many invalid cases before finding a valid one. Forward chaining starts from a known fact (e.g. the user has logged in) and seeks forward to relevant predicates. The effect is that forward chaining is stateful (i.e. it caches) because all the states it accepts at each stage are valid whereas backward is not performant and cannot cache because it never knows if any of the data of any intermediate stage is valid in itself.

They use forward chaining and conserve memory by at each stage caching only known valid states. She showed some benchmarks for an eCommerce application written in HALO, comparing all facts generated to the much smaller number that had to be kept in memory. She then demoed this app and showed the various HALO tools.

Q. This is based on CLOS; could you do it in Squeak? Yes; all you need is to be able to wrap 'events' (i.e. methods) such as instance creation.

**Linguistic Symbiosis between Actors and Threads, Tom van Cutsem, Stijn Mostinckx, Wolfgang de Meuter, Vrije University Brussels**
AmbientTalk is their OO domain-specific language, implemented in Java, to program mobile networks. It is an event-based language. How can you combine this with standard OO languages, e.g. Java. He first showed the problem. If you simply embed AmbientTalk calls and callbacks at various points your OO code then you may get race conditions, violating the event model's expectation that events will be executed serially without shared state between event processes.

The E programming language is an object-oriented scripting language. Actors have message queues and internal event-processing loops that takes events from the queue and sends them to objects (standard OO-style objects) within the actor. Local objects are those hosted by the actor; remote ones are hosted by other actors. Only asynch messages can be sent to remote objects (by adding them to its actor's message queue). In this system, actors cannot deadlock each other.

Java is both a symbiont language and the underlying implementation language. Thus it is easy to invoke Java objects as native objects in AmbiantTalk (using Inter-language Reflection) via various methods.

Mapping AmbiantTalk to Java is easy. If an AmbientObject wants to add something to a Java collection, that happens in a thread. But Java shared state means that a synchronised collection can nevertheless block. If there is a callback, e.g. ordered collection will call compareTo on another previously-added AmbiantTalk object, this callback can be done synchronously so raises no problems.

They wanted to have AmbiantTalk and Java unit tests. AmbiantTalkTest implements the interface for the test so then AmbiantTalk and Java tests can all be in the same test suite and run. However the Java test runner cannot just enter the AmbiantTalk actor so they provide a wrapper which queues the call in the message queue. Thus Java thinks the call happens synchronously but the Actor performs it asynchronously.

The documentation of event-driven systems in Java, e.g. ActionListener, says 'make the event terminate as soon as possible' :-/. In their symbiosis, as before, a wrapper takes the task of adding the event to the actor's event queue and immediately returns as far as the external Java invoker is concerned.

Thus actors become threads when they call to Java and wrappers protect Java calls to actors from waiting on the asynch.

They have a PhD student trying to implement the self/squeak morphic interface in AmbiantTalk. See http://prog.vub.ac.be/amop.

Q(Mike) What are effect of bugs in misprogrammed concurrent Java code? Yes, bugs can spread but at least they cannot corrupt AT actor.

Q. Is it worth the effort? The student implementing Morphic took a day to get drag-drop of morphs.

Q. Scalability; what happens as you add more actors? Some papers by other groups indicate major speed ups, similar to Erlang with their millions of concurrent processes, whereas JVM dies on 10,000 threads.

Q. Is the motivation of the symbiosis to reuse existing Java code? Well, as a small research lab of two Ph.Ds, they cannot rewrite all existing libraries in AmbiantTalk even if they wanted to.

### Talks I missed

All these were in the Research Track.

**Changeboxes, Tudor Girbe**
Change boxes are immutable objects that encapsulate changes and behave in some ways like classboxes. They have studied merging changeboxes and applying changeboxes to changeboxes.

**Change-oriented Software Engineering, Peter Ebraert**
They modelled changes types (parameter changes, behavioural entity changes, etc.) and the relationships between changes. Thence they extended the star browser to capture/record and browse change hierarchies.

**Object-Flow Analysis - Taking an Object-centric view on Dynamic Analysis, Adrian Lienhard, Stephane Ducasse, Tudor Girba**
They capture all references to an object and track the transfer of these references. They have studied various ways of representing object flows.

## Other Discussions

Bruce remarked that it was great ESUG and thanked the organisers, which others warmly seconded.

The Moose association was formed (administered under Swiss law).

There was a lot of offline discussion of the future of Widgetry and Wrapper. Some who have used Widgetry a lot, e.g. Christian, are impressed with what it can do and plan to continue using it. For my part, I regretfully understand the concerns of existing VW customers, and the general logic that evolving an existing framework is much more the Smalltalk and XP style than big-bang replacement solutions.

- Had there been the opportunity, I would have greatly enjoyed the challenging task of helping to build the refactoring utility that would have converted Wrapper UIs to Widgetry ones. (Both the enjoyment and the challenge would have enhanced by the fact that the utility would have had to perform to a very high standard.)
- Christian will work on a wrapper to let Widgetry widgets be used within Wrapper UIs (since the conference he has published to the OR). He will also start a Widgetry user group.

Tim Mackinnon (a former ThoughtWorks consultant) was *very* impressed with CMSbox, as were others. This application, like Dabble DB but in different ways, shows what Seaside can do.

I demoed my multi-threaded testing utility and model-layer / UI-layer testing pattern to interested parties (c.f. my Smalltalk Solutions 2007 talk). I also ported Pier to VW, helped by Lucas and by Dale, who now has a lot of experience porting from Squeak.

ClearStream, a VA user, does work for DeutcheBourse (the Germany Stock exchange).

## Conclusions

My ninth ESUG: great fun in the most scenic location since Bled at least.

- Seaside 2.8 combines great features with great performance, footprint, etc. Cmsbox and DabbleDB are advertisements for what you can do with Seaside.
- The latest GLORP work makes the exploratory modelling paradigm very available for RDB work, leading immediately to implementation: complex schema mappings could be quickly modelled in GLORP and then effected, with performance tweaks within GLORP as needed.
- Existing Smalltalk apps seem buoyant.

* End of Document *