CS7 and ESUG, Bled, August 23th - August 29th, 2003

Getting to Bled might be a little harder than to prior ESUGs' locations but when we got there it was worth it. Southampton centre was what you would expect from somewhere that was well bombed in WWII and rebuilt in the uninspired post-war style. One could probably say the same of Essen. Douai was somewhat better but still just a town location. Not since Gent have we had such a stunning location for ESUG; a lake, an island, a church, a castle, surrounded by woods and mountains. The weather (hot in the day, stunning thunderstorms at night) also did its bit to make the event special.

In fact, getting there was very easy, though it did not seem that it would be beforehand. A cheap flight from Stansted to Klagenfurt got me within 50km and Janko gave me, and Andy and Jean Berry, a splendidly scenic drive to Bled over the Karavanken range. Mike Routenberg drove me and Michael van der Gulik back again, and got me to my check-in gate a full ten minutes before it closed :-). (Both refused to take any money so I made a donation to ESUG's funds in lieu.) To complete the convenience of this ESUG, the hotel found it had double-booked my shared apartment so gave Adriaan and I separate rooms in a better adjacent hotel at the same price.

Some had harder journeys. Joseph Pelrine was taken aback when the taxi driver at Ljubljana airport said the fare would be, 'ten thousand dollars'. At length, Joseph mastered the accent enough to realize he was actually asking for 10,000 Slovenian Tolas (the root, Thaler is the same word from which Dollar comes). He rather wished he hadn't as it became very clear during the drive that the taxi's transmission was shot.

As Bled itself was well worth a tour and puritanical devotion to Camp Smalltalk had previously prevented my doing so, I skipped Wednesday afternoon's tour of Ljubljana (thus doing my bit towards ESUG's finances by helping reduce numbers to a single busload) and explored the castle (the cliff path up to which is not for those who dislike heights and suggested it would have been hard to attack) with its museum, and the church under it. The museum included a coinage exhibit. The display's heavy emphasis on the happy times when Slovenia was independent and could mint its own currency, and how the unhappy times when it was not were symbolised by its not having its own coins, suggested a lack of enthusiasm for the euro.

Total attendance was 92, less than the 150+ who came in 2001, but I conjecture that this has no Smalltalk-specific implication and merely reflects slightly the not-yet-over economic downturn and mainly that Bled was a somewhat intimidatingly unknown location for some.

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@bigwig.net). It presents 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, less than half 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 well contain errors of fact or clarity. I apologise for any inaccuracies, and to any participants who are covered by the abbreviation ‘et al.’ because I failed to note down their names. If anyone spots an omission or error, please email me and corrections may be made. A single programme track meant that I caught most talks, formal and informal. I apologise to the few whose talks are covered by a brief summary or ‘missed it; see their slides’.

My thanks to Adriaan van Os for helping administer my CS project, to Michael Prasse for working in it, to John Brant for giving us good advice, to Joseph Pelrine, Daniel Vainsencher, Fransisco Garau and Bernard Pieber et al. for useful discussions, and above all to the speakers whose work gave me something to report. Thanks also to the conference sponsors, Cincom, eXept Software, Eranova, OdaTeam and Comtron, to Janko for local organisation, and to ESUG.

Summary of Projects and Talks

I begin with some Camp Smalltalk 7 project summaries, followed by the talks, sorted into various categories:

- Thinking about Smalltalk: History and Future
- Using Smalltalk: Applications and Experience Reports
- Writing Smalltalk: Tools
- Writing Smalltalk: Testing and Process
- Extending Smalltalk: Languages, Metaclasses and Aspects

and then the Academic Track talks, which were already classified as

- Program Analysis
- Languages and Reflection
- Environments
- ESUG Academic Activities

I close with Other Discussions and my Conclusions. Talk slides are available from http://wiki.eranova.si/esug/Talk+Notes+And+Slides.

ESUG Programme, Stephane Ducasse and Roel Wuyts

Five international keynote speakers (most conferences only give you three), a stunning location, a hotel with both indoor and outdoor swimming pools (and a lake), and it was even arranged that the weather would be blazing hot over the weekend, saving the rain for the first day of the conference; clearly, Janko, the local organiser, has arranged an exceptional ESUG locale and Stephane congratulated him on doing a superb job.

Roel thanked ESUG's sponsors: Cincom, eXept Software, Eranova, OdaTeam and Comtron. This year, ESUG had a new format:

- talks morning and early afternoon
- informal sessions from mid-afternoon on

The idea was that speakers kept strictly to time and then remained available for the rest of the day so that people who had detailed questions could arrange a talkussions session with them. People also used the sessions to schedule demos, unplanned talks and Camp Smalltalk work. ESUG encouraged this because there is much good stuff in the community that is not published through people not having time or missing deadline.

Camp Smalltalk 7: Project Summaries

Camp Smalltalk 7 ran over the preceding weekend and in the breaks, afternoons and evenings of the five conference days. There was a great deal of activity in the room at all times, Camp Smalltalk projects, new project ideas, people showing other people problems and solutions in code, etc.

The Custom Refactorings and Rewrite Editor Usability Project

I spent most of my CS time on this. Adriaan van Os arrived first, set up the project and got started. I arrived on Saturday afternoon and worked with him and Michael Prasse. John Brant explained things and gave us advice and utilities from his extensive collection of both.

The list of tasks we considered is on the Camp Smalltalk Wiki at http://wiki.cs.uiuc.edu/CampSmalltalk/Custom+Refactorings+and+Rewrite+Editor+Usability+at+CS7. The main task we addressed was making John Brant's dynamic refactoring work accessible from the RB and generalising it to allow user input. In our VW7 load (which passes all tests and should be available as a release candidate by mid-September), any refactoring subclass of ChangeMethodName (i.e. add / remove parameter, inline and rename) will now tell you how many senders as well as implementors there are when it warns you that there is more than one implementor.

The rename (only, at present) will then offer you a third choice: as well as proceeding or aborting, you may choose to rename dynamically. If you do, a screen of all the call sites, in red, will be shown; selecting one shows its source with the call highlighted. Running tests and/or exercising the system turns the sites black as it discovers which implementor(s) call them and so whether to rename. The user can also select from a menu to make the refactoring rename that site, or all sites in that method or all sites in that class. The tool then checks for polymorphic calls, broadening the rename if necessary, and effects it. As with the main RB, the result is presented as a list of changes comparing old source to new, which the user may browse (and, in our load, edit) before choosing to effect all or some.

Work on implementing this in VA was also done and will be completed off-line. As dynamic renaming depends on method wrappers, porting it to other dialects depends on those dialects having a method wrappers implementation available. In time we will generalise the dynamic part to all the ChangeMethodName refactorings.

Other work included clearing the (very short) bug list, doing a small refactor of the prior VW release candidate, and converting from an Error to a Warning the Exception that prevented you renaming a method if any of the implementors had a method of the same name in its super or subclass hierarchy (it is still an error if the class itself already defines that name).

I gave a 20 minute presentation of what the project had done, concentrating on the right-click menu added to the Rewrite Editor; it is described below. Go to http://customrefactor.sourceforge.net for the latest information on the project. For even more detail, see our pages on the Camp Smalltalk wiki (reachable from SourceForge).

SUnit
Paolo's logging approach will replace sunitChanged: calls, in 3.2, making it easier to connect SUnit to your developments of, or replacements for, the TestRunner UI. Joseph showed me the code; it seemed a good solution.

Stephane has coded a solution to the competing singleton resource problem (e.g. some tests need to be logged in to one database, others to another database, and your system can only be logged into one database at a time); I did a little work on another solution which I prefer because it integrates seamlessly with the resources platform instead of being an alternative.

Bernard tests code using an asynchronously-called utility. SUnit fails to trap errors in the utility. Michael Prasse and I, with Bernard and Christian, wrote a method-wrapper that a test can use to trap errors at the top of the forked thread and add the test to the test result errors collection in response.

This was done in VA. I will port it to VW and upload both as a custom refactoring add-on. (It will not be in the general SUnit TestResources examples package, because Joseph feels, and I agree, that SUnit examples should not be dialect-specific, and this is to the degree that it relies on method wrappers, for which not all dialects have implementations today.)

More detail is in two SUnit-related talks Joseph Pelrine gave; see below.

Star Browser Project
Roel and I discussed its use on test suites. Roel made drag-drop a service in Squeak and knows how to make it effectively such in VW (overcoming some obstacles in how VW's drag-drop works). See Roel's talk below.

Monticello, Gardner, Squeak Map et al.
Daniel Vainsencher wants to make SqueakMap also see other repositories (so someone looking to see whether a utility had been written could find whatever examples exist in any dialect). Presenting the Cincom Open Repository RSS feed from within SqueakMap solves this problem for

VisualWorks. I suggested TotallyObjects site as one possible source for VisualAge goodies and promised to discuss it with David Pennington. Daniel and Joseph worked on code for this. Unfortunately, I missed Daniel’s presentation on this (got the time wrong) so cannot say more.

Thinking about Smalltalk: History and Future

Puncturing the Balloon. Re-inventing and Selling Smalltalk in the 21st Century, Joseph Pelrine, MetaProg GmbH

Kent Beck defines a keynote speech as one in which you can tell people things they don’t want to hear. Kent’s Smalltalk Solutions 1999 keynote quote,"Smalltalk is dead", when taken out of context, has done Smalltalk more harm than any other. Joseph talked about the things that frustrate him in hopes that we can find solutions. He began by showing a hilarious video of two Japanese table tennis players who seemed superbly skilled, martial arts style, but whenever they moved out of their black background you saw people dressed in black moving ball and players to fake the stunning stunts.

The only constant is change; the rate of change is not constant. Moore’s law is not a law, it is an observation. Moore initially said, “will double every 12 months”, then revised it to “every 24 months” and ‘18 months’ is just an average he never said. It is now flattening out. Meanwhile, our tools are good for solving 30-year-old problems. He walked through how our tools have changed over the last 30 years. (Then we could get full source listings for operating systems; now, with linux, etc., this is back. :-)

Smalltalk has more vendors than any other language. These vendors often use Smalltalk to sell another product that profits them. We are spending too much time building things that already exist in another language just so we can say, “Smalltalk has that”; this is a questionable use of resources. ParcPlace (in its various incarnations) did more to kill Smalltalk than any other company (including Sun).

- VA is a vehicle to promote IBM services WebSphere and VA platform (just a checkbox: ‘WebSphere also supports Smalltalk’)
- If VW stopped being an IDE and became a platform; it would be his choice for enterprise Smalltalk. A VW-as-BEA-style platform would be a solid player. A key element is supporting suitable integration with Java (JNI/ActiveX-like interface) and other marketed technologies
- Dolphin is fun. It is the small business and Smalltalk aficionado choice, well polished, and his choice for these groups for windows (VW would be his second choice but it has so many gotchas).
- Squeak is the true legitimate successor to the original Xerox Parc research project. It is very interesting but has no clear architecture or vision. It is full of lots of really cool stuff that pushes other stuff aside. He doubts it will ever be suitable for serious commercial development. Perhaps it should remain an experimental playground.

- Smalltalk/X is an interesting evolution of what VW could and should have done to solve internal design legacy problems. Smalltalk/X has a big library of commercially-valuable functionality all implemented in a common style (the advantage of small group). The Smalltalk/X base classes would be his choice for a Smalltalk foundation library.
- Smalltalk-MT: you really have to know how to program windows to use it. It seems to have lost steam of late and is mainly being used as a consulting vehicle by its vendors.

Then Joseph reviewed the third-party market. For Configuration Management we have:

- Envy: the living dead
- Store: the emperor’s new clothes of CM
- no other serious candidates

As regards other third-party tools and vendors, Eric Clayberg is the only person who ever made money as a third party provider in Smalltalk and today Instantiations (like many not in Smalltalk, to be sure) finds that times are hard. Eclipse is open-source and while Instantiations have very good technology for writing plug-ins for it, Eric is competing with a horde of monkeys who sometimes write that line from Shakespeare first.

Some weeks ago, Joseph challenged a bunch of good marketers to say how they would make Smalltalk succeed. The rest of the talk is the outcome of that five hour session. Corporate behaviour, design and culture all matter.

Behaviour is how a company looks to the outside. On comp.lang.smalltalk, the first thing we all say to a newbie is, ‘Which dialect are you using?’ (and it often prompts inter-dialect flame-wars). Many threads vary between “We’re the one true OO language, we don’t need X” and “Other languages steal our stuff, they got X from us”. Smalltalk user groups seem quiet.

Joseph reviewed user communities at conferences and mailing lists. The first thing a newbie sees is the Squeak table; if you are not a known squeaker they’ll let you sit there but.... Next we see the VisualWorks group: Eliot at the epicentre surrounded by torch-holders for the one true Smalltalk, with its own mailing list and irc channel (open for anyone but they don’t tell anyone about it). Dolphin has the most open, friendly, non-flamewar newsgroup he has ever seen but at conferences they’re either not there or in the bar that serves the best beer dreaming up the next cool demo.

Paraphrasing Sir Winston Churchill, “Smalltalkers are friends separated by a common language”. Joseph manages SUnit so he has to make it work on all dialects (so he’s against all vendors :-)). Interchange has many standards (SIF, Chunk, PePe, etc.), *none* of which work on all formats. When SUnit had to move code across all dialects there was no tool so he wrote Rosetta (more on Rosetta in talkussions). The ANSI standard is autistic: no UI, I/O, sockets, namespaces or modules, and only minimal read-only reflection. We’ve talked of another standard but he doubts it will happen from the vendors; their resources are too limited. We, as smalltalkers, must do it (see Common Base Smalltalk Camp Smalltalk project in last year’s report).

“Smalltalkers just doIt”. Do they still, now? Some people are still waiting for S#. Someone else earned $500,000 last year using S# because he just did it: he got it from Dave Simmons, asked questions on the newsgroup, called Dave about hard stuff, and just did it. Too many of us wait for the shrink-wrapped product. Dave was good because he presented an extreme alternative. Some of S#’s syntactic sugar is irrelevant to Smalltalkers and it has undocumented features (e.g. use { instead of [and the semantics of the block inside change to be more Java-like instead of Smalltalk-like.)

We have an aversion to new technologies. Indeed, we do not use the stuff that is already out there (change does not mean reinventing wheels).

Next he looked at Corporate Design, which is the easily identifiable way to recognise the company. (Look at Java v. Starbuck’s logo; which one of these is a coffee company?) In 1981, the balloon was meant to represent Smalltalk “escaping the remote island of academia, entering the jetstream and reaching the main user community”. In 2003, his marketers saw the balloon as full of hot air, old fashioned, unprofessional, with an 80’s colour scheme that yet does not work in B&W; it looked childish and cute. It symbolised nothing; it was not a logo, it was an illustration. (BTW their comments on the Java logo were, to put it mildly, even more unflattering.)

Marketers like logos that give a feeling of ‘We’ (we belong to an elite group, we can trust these guys, we feel comfortable recognizing something familiar). Smalltalk needs a logo that invokes trust, looks modern and crisp (Paul Rand asks, “Would my 15-year-old son be happy to have it on his tee-shirt”), and has clear, precise symbolism. Paul Grant did the IBM logo in the 60’s. Slight changes to it have been made a few times since then. The IBM logo implies seriousness. By contrast, Joseph showed the 80’s-colour-scheme coloured-stripes apple logo; Apple changed it to the current monochrome logo and it looks better.

“An apple is just as healthy as a vitamin tablet but its marketing isn’t as good” (Claudia Chiecchi). Joseph showed some poor old Smalltalk ads (including the ParcPlace Digitalk add about the merger being “good for everyone”). Java is a household word; how do we make Smalltalk that? We need a logo, a new image, a target group and the ‘we’ feeling.

The last part of this talk brought it all together. Joseph showed a video of the New Zealand All Black team’s pre-match warm up: a Maori war chant. You felt that most opponents would just give up on witnessing it. We know Smalltalk can win. How do we convey that feeling as well as the All Blacks convey it to their opponents? Joseph showed the ‘Knights of the Square Bracket’ logo (which is also the Camp Smalltalk logo). It is crisp, it suggests power, and the ‘strangeness’ of the symbols (to non-Smalltalkers) is part of that power.

(You can get the ‘Knights of the Square Bracket’ screen-saver from the MetaProg page: http://www.metaprog.com/.)

Smalltalk Today and Tomorrow, Smalltalk and .Net, Georg Heeg
(Jim Robertson sends his apologies. Georg's family were delighted to take a holiday in Bled, so Georg was delighted to stand in for him.)

In 1987 Georg Heeg started in Dortmund and is still there, since 1996 also in Zurich and since 1999 in Koethen/Anhalt. In his entire 16 years he did 198 deutchmarks of Java work (< £100); everything else was and is in Smalltalk. The company mission is officially to make sophisticated projects for customers and secretly to spread Smalltalk in central Europe.

He now teaches an introduction to Smalltalk course in Anhalt University. (Perhaps a future ESUG conference will be in Anhalt.) It used to be that mathematics screened out the students who were not up to the course; now Smalltalk does part of that job. Long ago (1972) the dynabook was to have the size and weight of a book and be usable by non-computer-geeks. Eight years later Smalltalk-80 appeared. Georg knew Simula-67 before 1980 so Smalltalk is not the foundation of OO as it is sometimes called. However Smalltalk had the first browser. Georg founded his company before ParcPlace (and has run it rather better, I note !!!). The first Smalltalk projects were The Analyst (done for the CIA and so secret that Peter Deutch honestly told him there were no Smalltalk projects; Georg didn't believe him, pursued some leads and got told to travel to the south, call this number, you will be met, ... :-)). Another early application was ProfiSee, displaying a simulation of e.g. an aquarium.

At the same time, a company in Switzerland began using Smalltalk to simulate their production process. In the early 90s, several large projects (JPMorgan, and long other list) were started. In 1996 Java appeared and simultaneously the Smalltalk projects reached production so

- (almost) no new projects were started
- existing projects users were happy so they carried on

In 1999, Cincom bought VW. Cincom was founded in 1968, is privately owned, has long-term vision. Today, new things are happening. New people show up (some young people in hall) and new companies appear, new Smalltalk books appear (some in German). Several companies in the Anhalt area told the university to teach Smalltalk even though these companies do not use Smalltalk.

What's special about Smalltalk is that *everything* is an object. The theory is that Smalltalk is intended to be a modelling language not a programming language. It can represent every well-formulated theory concept as a system of classes (e.g. a prior speaker had the idea of making instructions change a simulated processor's state and could then express that idea naturally in Smalltalk). A corollary is that any well-formulated computer science theory concept can be represented in Smalltalk.

Outside this room there is a fear of Smalltalk: 'It won't be there in a few years', 'It's too slow', 'It's not our company strategy'. If you refute these arguments, those who offer them search for others. Fear is in the heart, not the mind. It is not about the arguments. As the Germans say, "What the farmer does not know, he does not eat.” Smalltalk looks different.

The fear of Smalltalk is justified because Smalltalk demands intellectual quality. Understanding a concept is hard. Developing your own concepts is harder. Further, all of our thinking is incomplete; you need the inner freedom to see that what you did is wrong and change it. If you can hide yourself behind a C program, noone will ever see your concept is wrong because the concept is not visible. However once you overcome this fear, you don’t want to do anything else. Smalltalk is said to be only good for prototyping, for XP, for redesign. Actually, it is ideal for these things.

What about Smalltalk 2004. We are now in a recession. In Germany the economy actually shrank by 0.1%; no growth!! The reaction of large companies is to outsource to low-salary countries (India, Belarus, Ukraine). In the 60’s, textile companies where Georg lived did this. The know-how got out of their control and now those companies don’t exist. When you outsource, you also outsource the control. The other alternative is to do programming more efficiently.

Georg then turned to .Net. Bill Gates says .Net will enable the potential of the internet. Microsoft marketing says .Net is *the* platform for XML-Web services. Technically, .Net is a development environment for Windows. It is also an OO execution platform (we say ‘VM’, they say ‘Common Language Runtime’). It is a class library. It is an interface for web services.

It has the goal of unifying programming. There are 26 .Net languages, including S# and #Smalltalk. In.Net you should be able to subclass one language’s class in another language. Its other goals are to replace native programming and all the COM, OLE and other stuff.

Georg showed that the frameworks of VW and .Net look very similar in an architecture diagram; the same elements in the same layers. In Las Vegas, there are rolling walkways carrying you swiftly and comfortably towards the Caesar’s Palace hotel, but try getting *out* of that hotel. Java tries to dominate the world by ignoring everything else. .Net tries to dominate the world by integrating everything else and making it hard to use .Net specifics from outside. However you should be able to access .Net from Smalltalk because Smalltalk can represent any well-formulated theory.

Integration with .Net is compellingly necessary to GH and Cincom as Windows is an important platform for VW.

- S# shows one way of doing it. It has high Windows acceptance but lower Smalltalk acceptance (no platform independence, lose existing code base).
- Using .Net bytecodes is another way: it was tried with Java and had problems (Java bytecodes inadequate).
- Web services are unsuited to integration of local components (printing a word document using web services is a sledgehammer to crack a nut)
- GH solution is DotNetConnect: Smalltalk and .Net co-exist side-by-side. (Q. Who paid for DotNetConnect? Microsoft? No, Cincom paid.)

In DotNetConnect, Smalltalk can be the client, .Net the server. You have remote garbage collection. No adaption of .Net components (mscorlib, xml) is necessary (mscorlib is huge; they only take some of it). The communication channel takes calls from the VW connection stub and maps it to the registry in .Net (standard remote comms architecture). The .Net statics and constructors are mapped to class-side methods while the rest is mapped to instance side methods in Smalltalk. Georg showed the generated stub method for addDays: (simple) and its wrapper method (more complex; see slides for details). Encoding is done in Unicode strings.

They generate assemblies, method signatures and other reified objects into Smalltalk from .Net. As we do not use interfaces from Smalltalk, a Smalltalk-side call will always return the real object, never an interface object that needs casting, unlike a .Net-side call.

They tried to use the COM layer to connect but this needed the source code of the assembly so they junked that. Now they use the official but undocumented feature (widely used by MS itself to integrate existing components so likely to persist) that if you compile the same C++ component as managed and unmanaged, it will magically be synchronised.

DotNetConnect is available in 2 bundles (one dev, one runtime) and 3 DLLs. A demo with DLLs is available now. A release including full communication layer is due February 2004. A similar interface to Objective C is available, so your VW app can be designed to talk to .Net on windows but (via Objective C) to equivalent Mac platform utilities on Mac. They will provide automatic generation of COM interfaces and Java RMI (later; not in February version).

Lastly, Georg showed Smalltalk on portable devices: Siemens Simpad SLC (A4 screen size), skeye.pad SL and Tatung WebPad (A5 screens), and pocket PC (tiny screen showing quite readable class browser but it is too small to program on effectively). VisualWorks on Windows CE (on X86, StrongARM, etc., processors) is a complete implementation of VW giving you instant mobile compatibility for your VW apps. They are checking the market for porting it to a phone. To use it, load a Unicode-adapted DLL&CConnect parcel. You must have complete unicode support at the image/VW level. You must also load a parcel of special WindowsCE classes, and an adapted platform-recognition system. (Stephane: Look at JNI in Squeak's recognition system; it's much better than the older code.)

Q. Handwriting recognition? We have not done it but Smalltalk is an excellent environment for doing it.

An example application: the idea for an application for Hoeft &Wessel (who make ticket machines for DeutcheBank) was floated on June 16th at 17:19. They delivered an as-is stocktaking-for-inventory app on 17th June and demoed it at the client on 18th June.

Adele Goldberg said that, “This indeed looks like the DynaBook in its original conception.” (adding that Smalltalk has still not figured out how to let non-programmer-users model as directly as the original project wished.)

Q. Any .Net areas not accessible from DotNetConnect? Today we cannot have .Net draw into a VisualWorks window.

Q. Where is the mobile market going? Each different device tends to have a specific niche. The Tatung (large, radio, 1.5 hours battery) is meant to live in a factory and be carried around to record inventory, etc. A smaller device might live in a doctor's pocket carrying patient files. Generally, the smaller the device the larger the market. Ideally, you prefer to talk to someone manufacturing for a specific niche because then you get to talk to programmers quickly, so Georg is focused on the larger devices.

Q. Performance? A StrongARM 200Mhz has similar performance to a 50Mhz 86 (between 386 and 486) i.e. 50% of a Dorado or 500 times slower than a modern high-end PC. A Tatung (300Mhz Geode) is fine; same as 300Mhz Pentium 5 (so same speed as some laptops in the hall today).

Q. Who is Smalltalk DotNetConnect aimed at? Smalltalkers who must work on Windows. It is an enabler you must have so you are not washed away. (If any .Net programmers pick it up, that is fine of course.)

Using Smalltalk: Applications and Experience Reports

smallCharts development and design, Christian Haider, smalltalked visuals

Christian built a product for himself, found a client who wanted it so developed it further, then founded a company to market it.

The problem: a weekly magazine had hundreds of small charts to display in every issue and they did this by copying data into their special format by hand. The task was so tedious they could not even get students to do it. It was also very error prone; every issue had several. Their process was that the editor got the idea and pushed the work onto the graphics department.

With smallCharts, the editor draws their idea, makes the smallChart and does page layout. Version 1.0 had a single window with the chart and various choices for font, scaling (just linear at first but after the stock market crash they needed logarithmic to keep up :-)), line-smoothing, etc. The editor creates a chart and saves it (as postscript). Thus it was a very simple tool: one user, no database. However it had some challenges. Line smoothing is tricky. He created and implemented an algorithm, then later found that map-makers smoothing coastal outlines use the same algorithm. Output format was encapsulated postscript but not all programs that should use it actually do. Freehand cannot read eps though it says it can! Christian had to do many strange things to cope with Freehand's limitations.

Over several slides, Christian built up a picture of nested and adjacent boxes that showed the entire system, starting with the VisualWorks base layer. He had problems with VW fonts because the font subsystem (PSFontDeply5i) is quite old. He had to match the final print appearance to the screen appearance. (On the other hand it was very nice that he could use umlauts, diacritics, etc. in class names, namespace names, etc., thus allowing one-one mapping to domain layer names.) He is bound to windows because the data provider has only a windows client. He started development in VW5i3 which could produce a single windows executable, very convenient for delivery. Above the VW layer, he needed a DELocale, postscript handling and TIFF handling. This is the platform for SmallChart.

In version 1.0, SmallChart had a file reader which read time series data into a ChartTable. This fed a Chart that handled Eps for printing and ImView (painted on a bit map) as the screen preview. The ChartLineMaker created and managed smoothed chartlines, deferring to ChartScale objects for scaling. The GUI had a single ApplicationModel, the ChartTool, that had ChartViews and Pickers (i.e. pick choices for font, etc.) and Editors (user-supplied values). And that was the core of smallCharts 1.0. The ImView charts just referenced the Eps charts because the final print appearance is what matters. It worked but, because both were on the same level, it was challenging to extend. It was very easy for the editor to change layout and choices till they were happy with it, seeing the effect right away.

The second customer had more issues. They needed to resize, they wanted many more editable texts in more fonts, they had new calculations (e.g. moving averages), they wanted multi-coloured text strings. They wanted automatic chart generation for charts whose data changed regularly (weekly, monthly) and style sharing for groups of charts. They also wanted updatable charts: prepare chart style slowly in afternoon, then refresh chart data quickly in evening just before deadline. They also wanted spot colours (printing concept: B&W plus just one extra colour). Christian demoed smallChart 2.0, in which all this was added.

In-place editing was tricky with the current VW framework. The magic factor for font matching (see reference in my Smalltalk Solutions 2003 report of Travis' talk) is 1.14:

```
self textSize / Screen default defaultFontPolicy
defaultFont pixelSize * 1.14
```

Upgrading to VW7 was no problem. He added various objects and restructured into more packages. There is now a Painter package, Data Reading package, etc. Picker was renamed Choice (handles all user choices), subclass ChartChoice. He was careful to keep all old functions working while restructuring (but not through being compelled by XP tests; see below). The client had their own fonts and was very particular about using them so Christian had to let the tool manage customer-specific fonts.

He showed the structure diagram for version 2.0. Formats (static) related to Charts (dynamic) describe what is wanted, the result being rendered in the Painter (subclasses EpsPainter, ViewPainter).

He had to deliver runtime so packaged from pundles. Christian is very happy with Store. He was careful to have fat-free runtimes, moving all unused methods to development and letting the packager tell him if it were wanted again. Runtimes were small; for 1.0: 311Kb, for 2.0: 1030Kb. He showed the package list for both, showing which packages were

- development
- base runtime
- customer-specific runtime
- testing

He is careful to package without warnings.

His process was to develop only solutions to concrete customer demands (as in XP). He delivered early: his first package was delivered when the first format was done, and he got very useful feedback, so he recommends delivering early. He did a great deal of refactoring, including refactoring which packages things lived in. He now uses checklists when doing anything complicated, writing down the steps he did, so that later he can see what he missed and also automate it if it is routine and frequent. He built one development tool, an eps viewer.

Whenever there is an issue, he always writes a test to recreate the problem, even if he must then eyeball its result. Testing is invaluable for two things:

- incoming data may not be as advertised so you must have test cases to verify the assumptions your code is about to make about it
- modules with very small interfaces (e.g. his scales) he tests carefully

However most of the program was about producing charts that ‘look right’. It is hard to write a non-brittle test for that (bit-map compare is very brittle).

Problem areas:

- Format determination: a recognition function tries to detect the format of incoming data and pass it to the right handler. Adding a new format may therefore influence what happens globally, by changing who receives given incoming data, forcing him to check all rival detects.
- GUI events: he does the most up-to-date event handling but it is still hard to track each new GUI element, so he often has to do double-refreshes, etc. Redisplaying is likewise a problem.
- The model has too much double-dispatch (legacy of 1.0 to 2.0 change) but he’s confident he can clean that up.
- Should format constants be methods or constants? (Christian did not have time to explain this involved technicality fully.)
- VW fonts; matching the View font to the Eps font is hard. They do not even have the same names!! You just have to know which pairs match.

He has a to-do list: COM connection to the datastream, web graphics, an Eps parser, feature-based structures (allowing customer-specific feature-editing to be moved out of core), etc.

Seaside, Adrian Lienhard and Lucas Renglii, netstyle.ch and University of Berne.
They work in Squeak (Seaside has also been ported to VW and Dolphin).

They founded netstyle.ch in 2000 just to offer web presences but now find they are building web applications all the time. They use Apache to talk to the web client with appropriate URLs rewritten to static files (as Seaside / Commanche is slower serving these and is not needed). Apache talks to Commanche / Swazoo which talks to Seaside (because Seaside doesn't handle all the socket stuff). The application talks to Seaside.

They have an application for a specialised Swiss health insurance company with agencies. The first version was deployed last week. It is an extranet for handling proposals, taking client data, selecting relevant products, etc. Important for the client was printing PDF files from the application with a given corporate look. They generate the PDF dynamically so they print it from within Squeak (ported the framework from VW and extended it a little). They could have exported XML and processed it externally but that was slower; the intra-Squeak route was fast.

They also have a web application for configuring the Squeak server and for connecting to Monticello to upgrade the server while it is running. They also have a web test interface to run their tests against the production database. Lastly they have a simple web Smalltalk browser so they can change code on the server if they need to do so in haste, or just evaluate Smalltalk expressions to learn more about the server's current state. He inspected a collection of insurer categories in the application; the web browser handled the inspect quite well considering. (They can screenshot the image and have the webapp translate clicks on the bitmap to clicks on the browser; not much use but you can look at a debugger that way).

ROE is Avi Bryant's generic Smalltalk Relational Database access framework. You generate SQL queries by sending messages to relations. The slides have code examples; the protocol seemed an effective way of letting smalltalkers write SQL as if it were Smalltalk. The generated expressions are evaluated on the database, not in the image (important as the database has several Gigs of data which must not be dragged into the image during evaluation). It works by subclassing Collection to Relation, subclasses Concrete and Transform, thus using standard collection protocol. Visitors walk the representation generated to create the SQL. They do not optimise queries because POSTGRES does an excellent job of optimising queries and their generated ones run as fast as optimised ones (they have verified this for several cases by hand). They also handle transactions and rollback. Avi is still working on ROE; he feels he can make it even easier.

Using SmartCards to login to GemStone, Alfred Wullschleger, Swiss National Bank
SmartCards are being introduced to Alfred's bank so he investigated using them with Smalltalk. His project (OASE) handles high-security financial statistics for Swiss banks and companies. GemStone uses username and password as usual and these are sent over the network (some risk) and chosen by users (high risk; users choose trivial passwords or else write them down everywhere). Thus SmartCards offer significant security gains.

GemStone has a 1024 byte size maximum password limit and none on username. Password expiration can be set to one trial. Hence they can use a complicated once-only password, called the EPW. To combine the SC with GemStone they use a logon server which talks to both the client application and the GemStone database. The logon server runs on the same machine as GemStone, since the communication between them is unencrypted and must be secure. It handles all client login requests for all GemStone servers on that machine. It is implemented as a headless VW3.0 application so it runs anywhere as a TCP/IP server.

The protocol begins with the client requesting an EPW from the Logon Server, who verifies the user with GS. LS generates EPW and sets it in GS for one use. LS encrypts EPW and sends to client. Client decrypts EPW and makes normal GemStone login with username and EPW. The LS can run in SmartCard RSA mode (user has SC) and in Password Diffie-Hellman mode (user has no SC). They allow the latter as user may have forgotten their SC. This mode is disabled and must be enabled for that user by administrators after appropriate checks.

For DH, they use blowfish block cipher (because VW7 implements it; they just ported it back to VW3 which was easy). Each client session must have a session key, generated by Diffie-Hellman. He showed the formula for DH. Let Zp be the numbers from 1 to p (p a large prime > 256 bits). Let b be a base chosen in this set. Each party selects a secret random number r1, r2 greater than b. Now b to the power of r1 or r2 mod p makes their public key. Each party calculates their key as the other’s public key to the power of their random number and this gives the same number on both sides. As soon as the LS detects that a client has opened a socket, the LS creates a DH half-key and signs it with its own key (or else anyone could pretend to be a server) and returns it with the connection-accepted message. The client can now generate its own DH half-key for the session.

In SC-RSA mode, the Client just sends the logon request in clear text. The LS requests the user’s RSA details from GS (in clear; they’re on same machine) and generates the EPW from it. They send this in clear to the GS and encrypted to the client, after which LS communication with the client is terminated. The encrypted EPW can only be decrypted by the client using the SmartCard’s private key (unextractable from SmartCard so secure). The EPW is decrypted and the client logs on with it.

In PW-DH mode, we cannot verify the validity of the client so we require the initial Client-LS login to supply a password as well as the username and server name, all encrypted. The LS requests the user’s DH details from GS and decrypts them using SHA-1 hashed password. They verify the username in the result after which the process is as above but encrypting the EPW with the session key not the public key. Alfred remarked that SHA-hashing does not help if the password is trivial; one must demand non-trivial passwords from the user in this mode. DH access will be erased whenever the user again uses their SmartCard and also quickly aged.

RSA uses two secret primes p and q which multiply to produce a 1024 bit number n. Let e be the greatest common denominator of p-1 * q -1 (typically e - 65537 or 17 or 3). The public key is n and e.

They found back-porting to VW3 very easy (they added some methods to file-outs to remove namespaces). They have not moved forward yet as GemStone code management was not supported in VW7. They use 256 bit keys (blowfish needs 128 - 448) in CBC mode (each block XORed with preceding to avoid cases where block is all zeros and so obvious to decrypt). All requests and responses use XML, with integer formats in all communications that use encryption to avoid escape sequences (e.g. &#x0027). Their XML protocol is very simple as it is only used to transfer data in the above protocol.

They could improve the protocol by making the user sign the LS half-key they send (using the corresponding X509 certificate). This would defend the server from denial-of-service attacks in which the server send a key that the fake user cannot decrypt but the fake user has made the server do work.

OASE data can be requested from an XML bulk server (for clients who are using web services instead of a VW environment). Like the Logon server, the bulk server is a headless VW3 application. The user must logon to GS but cannot supply an EPW so they will use an XML-ticket, obtained from the LS and presented to the BS which will then supply data from GS. The protocol is very similar until, instead of an EPW, a ticket of the timestamp, username and servername is sent (signed, encrypted) to the client and (in clear) to the BS (timestamps must be within 10 second window). They plan also to introduce tickets for batch processing. Job gets ticket, holds it at secure place in their program, and presents it when they need the service.

SmallWiki, Smalltalk Wiki towards CMS, Lukas Renglii
If you are wondering why they built another wiki then you have never tried to modify one of the existing wikis. Last ESUG, they collected ideas about a new wiki that would be easily extensible, not just with new designs but with new components, web servers and storage strategies, etc. They also wanted test cases to make it easy to port to other environments. They use SmaCC to parse wiki source and Swazoo to serve web pages. The whole talk was done in SmallWiki, with some UIs hidden and a larger font setting.

Lucas edited a page to show the syntax, which is largely the syntax of SWiki; asterisk-delimited links, etc. The default renderer gives you the SmallWiki look but if you want to adapt it to your company, you plugin your renderer. SmallWiki has folders as well as pages; use folders to organise your pages and subfolders. Administrators have an interface to reorganise folders. You can have more objects than just web pages; actions, repeated components on a page, etc.

Lucas demonstrated the template editor. You select a predefined look from a list or edit one to make your own template. (The interface seemed very usable for a web browser; nice menus, etc.) He then demonstrated how they ran tests in the wiki; a simple ‘create and run TestSuite, return result’ code fragment was written on the page and executed; the page displayed the result. The code was executed whenever that page was rendered.

The basic design is a WikiItem that can be visited (e.g. to render itself), subclassed to Structure, DocumentComponent, Template and Action. Structures map directly to URLs; each URL has its own structure. Structure subclasses to Folder (can contain Structures), Resource (any file that you want to include into a Page; image, PDF, SIF, etc.) and Page. All Structures have a predecessor (their old version), thus everything you store in structures gets versioned automatically.

SmaCC parses the text. Lucas suspects they are the first Wiki to use a parser as they could find no examples in the open source literature. Having a parse tree makes things much simpler. After parsing, the document is pulled into its page and you're done. Links in the wiki are automatically maintained; moving things around will not break a link (can't do anything about external links of course).

There are several extensions:

- Search Editor: this exploits the parser, e.g. you can search for strings in headers only, which is hard in Wikis without parse trees.
- Keyword Index
- Link Collection; lets you access your links from anywhere, escaping browser limits
- Importer: move your existing wiki into SmallWiki

He demoed in VW by building an RSS news feed template component suitable for use in many places on many pages. The BottomFeeder feed was the basis. He subclassed to TemplateRSS and wrote

```
accept: aVisitor
  aVisitor acceptTemplateRSS: aTemplate
```

then extended the Visitor to handle the new extension. The visitor has a stream on which he put the code he wants; all very straightforward. The new template appeared in the menu and raised the debugger when called, showing that he also needed to override the for: message. Then (after the usual demo hiccough: needed to kick wiki to pick-up changed code) it worked. He then added the SmallWiki newsfeed.

He closed by saying that he did not want SOAP and PHP to rule the world and showed a witty recasting of the 'Knights of the Square Bracket' logo as a mouth about to eat the Java logo. He mentioned other projects that were using SmallWiki code: Daniel Vainsencher's Garden project, and the Gardner, both in Squeak. Info at http://kilana.unibe.ch:9090/smallwiki.

Smalltalk to teach Computer Architectures, German Fabregat, University Jaume I of Castellón

German thought Smalltalk was only for demos not real applications. Then he learned Smalltalk on a visit to another university. Now he can write a new simulator from scratch in a week!! Three hundred students use it each year in his course. He teaches them the basics of assembly programming, i/o, exception management, CPU internals. Then they look at control unit design, instruction level parallelism, superscalars and performance issues. All these studies benefit from practical work. As they are not a very rich university, they must do practical work in a simulator.

The ideal laboratory environment should have homogeneous simulators so the students don't have to learn new simulators as they progress to learning more advanced topics. They must be platform-independent (5 labs and 500 students give quite a timetabling problem). They must be easy to modify, upgrade, customise and, above all, use. Complex things (e.g. VHDL) are not really required; all you need is the simulators.

A general circuit simulator had classes CPURegister, CPUCombinatorial and could be built with value holders, avoiding the need to build a specific simulation engine as such. They built it in 2 months with no prior knowledge of Smalltalk. (Knowing C well, he usually took 4 months to build another simulator on his existing C codebase.) Students view the state and perform the simulation clock by clock to see what is happening.

The next version was used to teach Control Unit design at the University of Bretange Occidental in Brest. This showed the circuit visually with state of the various components. He used it to design and build two processors.

The final development was a new simulator to be used by 600 students over two years. It took him one week to build from scratch. The instruction set was not fully known during development (six teachers had different requirements) so he knew the system would change during the first weeks of use. Therefore he changed the design completely, making each Instruction subclass responsible for its execution and for updating the processor. New instructions (e.g. to simulate indirect register addressing) were added in a couple of hours during use. They also found it easy to write widgets for input devices and connect them to the system.

So why Smalltalk? Building rich user interfaces is fast and easy. All the simulators have the same look and feel. Applications are easy to build and above all to modify. Applications are platform-independent. Thus complex systems are easy to grow from simple systems. Applications can be built focusing on the important aspects, not worrying about e.g. the simulators' data types and instruction coding, which could be looked at when needed. Tools (e.g. parsers) can be added and integrated with applications.

Smalltalk is well-established in his own research team. His other university colleagues like the simulator but when planning to build things they prefer Java. The people who teach computer programming think Python is the new panacea 'because the students have no compilation problems'.

Lastly, he demoed the simulator, loading an instruction set and simulating an LED being driven and its processor being simulated, interrupting it, etc.

(From Qs) Smalltalk is being used commercially for processor design. The first JIT for Smalltalk by Peter Deutch was preceding by his writing an emulator for the 68000 on Dorado.

Q. Any influence on students? They know the simulators are written in VW but he does not know whether it influences their language preferences.

The image is available at http://mermaja.act.uji.es/otros/pb/pbintro.html (and the code but it's in Spanish :-)).

Writing Smalltalk: Tools

The Secret Life of Tools, Vassili Bykov

Vassili, with John Brant, deserves much of the credit for the transformation that has come over the VW toolset in recent years. Tools are hard to talk about (which is why Vassili has always talked about something else at prior conferences). Vassili sees himself as a Smalltalker and a VisualWorks Engineer first, as a Vendor from Cincom second.

Several people in Cincom still use parcplace.com as their email domain despite their contempt for 'DarkPlace'. Way back when, Xerox Parc management moved five research groups - the 'loser' groups, the ones management couldn't figure out how to make money from (something they were not good of doing) - to a less prestigious site. Adele noticed that the five groups' names spelt 'PLACE' as an acronym, thence 'ParcPlace'.

Many people in the hall did not know who Bill Lyons was; Vassili was glad of that; "Time heals".

Tools organise the user's experience of the system. This aspect of tools was not designed in early VW. Vassili became a Store user because the store browser was easier to use than the three-button browser in VW5i. Three years ago, Vassili began work on tools, focused on fixing that user experience 'eek' factor. A second focus was to integrate all the tools used in the community: RB, PDP, Trippy, ... A third focus was to fix all the worst things (parcel browser, modular dialogs, ...).

Some people think working on tools is God's own job. Vassili claimed that his blonde hair was actually black when he started and is now en route to grey. :-)

- The tool-maker is at the top of the foodchain; he must integrate everyone else's work as it changes underneath him.
- The 80/20 effect means you spend most of your time on the hard and boring 20%.
- The patchwork quilt effect: all these tools were built by different people at different times and you must integrate the accidental divergences.

VW is based on ST80. ST80's vision was of a world of Smalltalk. The real world is of islands being integrated. The RB must be made aware of Store; until then, he must maintain the old GUI to maintain the old and not very good Store browsers. Opentalk wants good tools. The GUI builder must be upgraded. This will take time since there are only a few people working on it and while Eliot talks of cloning Vassili and Jim talks of those 8 hours he wastes sleeping every night, Vassili objects to both these solutions.

Vassili described his reaction to seeing bubblegum alley: ‘strangely fascinating and disgusting at the same time’. When he first saw Windows XP, he was reminded of bubblegum alley.

Vassili showed the st80 v2 (first release) browser. It looked more like Squeak than anything else. Vassili has resurrected it in his Hobbes (Highly Objectified Blue Book Equivalent Smalltalk) project. Smalltalk implementations are still measured against how fast st80 ran on the ($100,000 of 1970s money) Dorado. The first commercial VMs ran at a few percent of a Dorado. Now the best are two orders of magnitude faster than Dorado. Hobbes is as fast as a Dorado. Bob did a VM (taking over two years elapsed but say a year actually, in his spare time) by implementing the blue book in C. Vassili thought it looked like fun but would never program in C in his spare time for a year, so wrote it in Smalltalk in 3.5 days (reusing GC and Bltblt from VW). His performance was comparable to Dorado, was faster than Bob’s for basic object access and was better for more complex operations. This illustrates Lars answer to the ‘specialised hardware’ question. Dorado is optimised for Smalltalk bytecodes but Hobbes beats Dorado more and more as operations get more complex.

His objects are identical with VW objects so when an object disappears in Hobbes it would disappear in VW, so he could reuse the VW garbage collector. He has active instructions, reminiscent of German’s final design mentioned yesterday. His Smalltalk system is his model in MVC terms so he can bring up multiple synchronised browser views of it (useful for pair programming). Debugging his VM was “too easy” (Eliot Miranda) and as penance Eliot told Vassili to implement a file system. The Blue Book described a file system with a single primitive to actually reach the file system and this was harder than implementing the VW (i.e. it took him 4 days, not 3.5). Thus he can see real source held on disk. If the file is large and you select a method at the end of the file, it takes a moment to read whole first time. Glen was reminded of that behaviour in Smalltalk 80 when he saw it again in Hobbes.

Why did he do this. It was fun. He also felt guilty ripping old Smalltalk 80 stuff out of VisualWorks so he preserves it in Hobbes. It also lets him study VM implementation in great detail. Vassili then demoed various things.

Trippy is an example of Bilbo Baggins’ warning about roads. He started out meaning to add simple extras to the inspector and a few weeks later there was Trippy. Trippy has forward and back buttons and offers multiple views of an object. You can select and view multiple elements. You can also drag-drop elements from one view to another when it makes sense (e.g. moving elements between collections). You can edit the object’s methods in the inspector (if you don’t like printString for this object, just select methods, override printOn:, select view to see if you like the result). You can inspect a window controller, thence view, and see both the preview and the component hierarchy of a window (selecting methods makes that tree view show the classes of the components in the hierarchy).

You can add settings to the new settings tool. The VisualWorksSettings class-side has methods to define settings; and you can add more. Vassili showed a simple edit to add a setting for text tool character length. If you make a mistake, the debugger will show you and you just fix and carry on.

The above is in VW7.1. In VW7.2, there is a search dialog with auto-completion, fuzzy match, etc. There is a class creation dialog which generates accessors, subclassResponsibilities and initialize as requested and appropriate for the class and superclass.

Vassili is using Modular Dialogs. The incremental search dialog use the incremental search module (also used by 'Published Items') which uses the list module. He had to solve the issue that users had different ideas about e.g. what icon to display, what iconBlock: to invoke. Vassili has a pattern (<relay: selector> pragma) for handling this and noted that we might like it or hate it. Stephane inclined to the latter view, given the dangers of pragmas growing into a parallel ill-analysed language within Smalltalk.

Lastly, he had a question without an answer. The more features you add the more complex your framework code becomes; more classes, more methods to search before you the programmer can change it, leading to a loss of what Vassili calls the 'Habitability' of code. How can tools be both powerful and yet easy to learn and modify?

Q(Roel) Deadline for Goodies for 7.2? Release date is November so October should be OK.

Q(Joseph and Stephane) Issues of adding pragmas. We do need some concept of where it is and is not legitimate to add these. Pragmas are good as annotations but not as new frameworks. (Andrew Black) In Traits they took care to make the code do what you thought it would if you just thought of it as Smalltalk code. This should also be the goal of any pragma framework. Vassili showed his framework to get feedback. It will be reviewed. It is not seen as the progenitor of a general framework.

Q(Andrew Black) st80 VM in VW is excellent for teaching. We should teach VM design with awareness of its history.

Q(Niall) The RB can be made not just Store-aware but aware of multiple simultaneous CM systems. Only a minor refactor (which I have done much of) is needed to support multiple pluggable CM views for browsing and refactoring. (Vassili and I discussed this off-line and browsed my code.)

Q. Making people aware of these tools? We could make videos of tool use. We could use UI tests. Assuming these are written anyway as part of XP, give the user a simple framework that can select designated UI tests and invoke a 'next' button on them to run till the UI updates. 'Next' would either watch commands that drive windows (but then user must use these in test) or capture UI events, so each 'Next' shows a UI change.

Vassili is using Pollock for his new work (thus helping lay to rest a suggestion that Pollock be renamed Godot. :-))

Custom Refactoring and Rewrite Editor Usability, Niall Ross and Adriaan van Os, eXtremeMetaProgrammers and SOOPS
The Refactoring Browser has much more power than most of us use. The Custom Refactoring and Rewrite Editor Usability camp smalltalk project aims to expose the RB's power by adding new refactorings, adding UI features to the rewrite framework, and providing examples of how to write your own refactorings, either in the Rewrite UI or called from code.

I began by showing a new refactoring, the 'Rename Variable and Accessors' refactoring we added to the standard 'Rename Variable.' (One aim of this talk was to get feedback; e.g. do people want this to rename lazy accessors, not just default ones?).

Next I showed the new menu in the Rewrite tool.

- Right-click on a selected a Smalltalk code expression to select from a list of increasingly general rewrite meta-code expressions that will match it. (The menu will also let you expand the selection to show what the tool thinks is the currently selected node, if it is not all selected already, to help understand the Rewriters' view of Smalltalk code.)
- Request the OR separator (in the search pane, only) to match multiple alternate patterns, all rewritten to the same pattern in the replace pane.
- Paste Method Source and Match All menus to speed up code writing. (N.B. entire method source will, of course, only match if 'All method' box is selected; that was my demo hiccough :-)).
- Save options stop rewrite text vanishing every time you change the navigator's selection.

We've extended the meta-character list at the RHS of the pane and added a new meta-character for wildcard-matching in meta-code. We use underscore as our wildcard symbol, which is a convenient choice in VW and VA. Any issues with this in other dialects? Any suggested alternatives?

We provide example custom refactorings called from code, not from the UI. I walked through (or, more accurately, ran through - we only had a twenty-minute slot for all this) the ExSearchSymbol(s) example of the pattern that stores selected symbols in some classes class-side dictionaries and uses them in method calls scattered throughout a large system. The refactoring detects any method calls whose symbol parameters are not in any of the permitted dictionaries.

I mentioned the add-ons we also provide as they arise in the project, e.g. Michael Prasse' code colouring in VA (that works on all UNIX platforms, with a little help from Thomas Koschate).

Lastly I demoed our work in CS7; dynamic refactorings. John Brant has a dynamic method renaming utility. We have integrated this with the static method renaming refactoring so that the latter's warning message now tells you how many senders, as well as implementors, you have, and offers you a third choice of dynamically renaming. We also added a menu to let the user force renaming at selected call sites instead of waiting until tests, or exercising the system, should discover it.

Our long-term goals include offering dynamic type recovery as a code tool in the RB and integrated with the refactorings. Michel Tilman's Analysis Browser (which I have ported to VW7 and put in the Cincom Open Repository) offers an excellent starting point for VisualWorks. SmallTyper might serve the same role in VisualAge.

Joseph Pelrine has ported some of our code to Dolphin. Otherwise our loads are for VA and VW to date. We would like to run on all platforms that support the RB; this depends on getting volunteers skilled in those platforms. Our strengths are in VW and VA today, so we code in them.

I walked round our pages at http://customrefactor.sourceforge.net/. Go there for downloads (or pointers to development downloads in e.g. the Cincom Open Repository) and brief explanations. Longer explanations and discussions (some of them old and outdated) are on our Camp Smalltalk Wiki, reachable from the sourceforge pages.

You are our users. We need feedback. What would help you get more from the RB? Please raise enhancements and bugs on Source Forge (also you can write wiki pages and put links to them in ours, if you have a lot to say).

UML-X, Ernest Micklei, PhilemonWorks
Ernest, like most Smalltalkers, spends most of his time coding but was once asked by his manager to produced a design. He therefore wanted to keep UML and code better synchronised. In a tool, you both specify your model and you also build a diagram, a view of your design. Ernest decided to separate these two, moving his design spec closer to his code. He wanted a textual form readable by tools and writable by him and his code.

It is impossible to extract a design from code; you must have the original abstraction. Implementation diagrams are uninteresting. It is important to keep design information where the code is, e.g. in class methods. Standard UML formats (petal, XMI, UXF, etc.) are targeted at tool exchange, are proprietary, are horrible to try and write, are not human-readable, store diagram and specification munged together, are not suitable.

XML can be read and can be written. He went for a simple structure of no XML namespaces and just a few constructs for package, class and association (no generalisation relationship; he just gives class a superclass attribute). Package = e.g. Envy application, class list is classes defined in that application, so he does not record that but generates it from code.

A diagram is just a view on a design. He writes a UML-X file (by text editor or else generate from code via UMLClassBuilder and then edit to fit design). He then creates diagrams using SUMO (Simple UML Modeller, a UML diagramming tool he wrote in VA, available free, can only edit diagram, not definition spec), exported as gifs for inclusion in e.g. html documents. All classes can be given a umlDefinitionString method that defines the class in this XML format. SUMO shows the model elements as a hierarchic structure in the left pane and lets you create a diagram of them in the main pane. The magic F5 button lays out your new design in your old layout as best it can. He showed a case of the tool making good layout guesses and another where it was not so good.

Ernest has created an Object Model Archive of reusable models and other things which can be found at http://www.philemon.com/.

System Change Notifications, Roel Wuyts
This is not a presentation of something finished or even half-finished. It is part of the refactoring of the Squeak Kernel. A lot of tools need to know that a class was changed or a method was compiled. A system of change notifications is needed. Roel is building one so we can clean up change sending and also clean up the browsers which at present use a mix of keeping lists, getting changes and polling. He plans to port this to Visual Works and Joseph will port to Dolphin (but Joseph feels he can simplify it first, which is why Roel knows it's going to change).

SystemChangeNotifier is the centre. It has protocol noMoreNotifications: to deregister. Various notify* methods let an object request notification of system changes of any object of a given kind by receiving events. Roel showed various tests that demonstrated the system. AbstractEvent subclasses to AddedEvent, ChangedEvent, CommentedEvent, DoItEvent, RecategorizedEvent, RemovedEvent, RenamedEvent, ResuperedEvent. The idea is to make events as orthogonal as possible to the class of the item generating them, e.g. a class, protocol or method can raise an AddEvent. In principle, all event types could cross-product with all raising object types. In practice, some cases do not occur; one does not Resuper a method.

Roel allows very fine-grained events (e.g. make this browser only receive events that mean a class was added so it need not ask, "Was this an add event? "Was this a class event?"). It can also be used coarsely, to notify all class events or all add events or all events whatever and let the recipients either check the events' details to see whether to react or else just launch an update on itself to re-read and reflect the new system.

The change set can be a client of this system. However if you put a self halt in trapping the event and then exit it, your change set might lose the change. Roel therefore traps all errors raised while processing the events and passes them after all clients have received (i.e. a handler calls `valueWithArguments: an Array startingFrom: anIndex +1` and then passes the event). There was some discussion of whether he could instead force the change set to be always the first dependent.

On the Refactoring of VisualWorks, SUnit and Store, Petr Stepanek,
I think Petr's travel plans went awry; the talk had to be cancelled.

Writing Smalltalk: Testing and Process

First SUnit BOF on Getting Started, Joseph Pelrine, MetaProg

This talk was about SUnit in general. A later talk dealt with Test Resources.

Joseph started with a quick summary for anyone who did not know it, then went on to talk about common confusions and problems in SUnit. Setting `MyTestCase>>isLogging: true` will cause your test case failures (but not errors) to be written to the transcript. To test code that you want to raise an error, use `should: aBlock raise: anExceptionalEvent`.

Being ‘test infected’ means being addicted to that green colour. ‘The sun does not set on bad code’. For many tests, it is convenient to have example instances. Should the test set these up or should there should be class side example instance test methods. (Joseph and Kent think the latter; I think the former, while conceding that an example instance can be an addition or alternative to a class comment.)

If we were starting SUnit from scratch, Joseph might rename test method as test case, test case as test scenario (actually he said test suite as test scenario but I assume that was a slip of the tongue).

Joseph likes to deliver SUnit as a run-time assertion framework. When working with a client who used a changing ill-documented C interface, he used this to test the latest interface every night as a cron job. He also has a tweak that lets you run SUnit / Smalllint on code and bring up not the debugger but the Smalllint results browser.

What should you test? Having a TestCase class per Model class can tie your tests to your solution instead of to your problem. The alternative is to have one or more tests per (interesting) use case. Joseph thinks both styles are acceptable but notes that you will have more refactoring if your tests are tied to your code structure.

Joseph urged people to just start testing. A poor test in an inadequate test suite is much better than none, will help you write your code quicker and will teach you how to write better tests and fill out more complete suites. As in XP generally, so in testing, start writing and expect to refactor your tests as well as your code from time to time. (Stephane Ducasse: Tests are like Solar Energy. They’re cheap and they protect the future.)

Test ordering was discussed. Joseph favours unordered tests (could be held in Sets). I favour predictability; I don’t want to see my tests pass or fail at random because the order is being silently changed on me. An explicit ‘test in all order permutations’ in the UI would be my preferred solution.

Second SUnit BOF on Test Resources, Joseph Pelrine, MetaProg

A basic SUnit rule is that *all* the data and infrastructure for a test must be set up from scratch at the start of a test and torn down at its end. Thus you ensure that your tests do not pollute each other and start from a well known clean state. However some artefacts’ state is invariant over tests but costly to initialize or finalize. TestResources handle these cases. The

TestResource framework is open and flexible; so open and flexible that it can be hard to see how to get started with it.

TestResource is subclassed to create specific resources. It can also be subclassed to create abstract subclasses for patterns that vary the generic behaviour. A framework call of `MyTestResource current` gets the singleton instance that our resource class manages. Joseph walked through the code which lazily initialized a current instance whose initialization called #setUp on it. Similarly, at the end of a run, the framework calls #reset on the test resource class and this calls #tearDown on the instance.

We can let the SUnit system handle the resources or do it ourselves; the standard approach is to let SUnit handle it. Simply define a class method `MyTestCase class>>resources` that returns a collection of those TestResource classes your test needs. That is the *only* thing you have to do.

Because the SUnit test package is testing SUnit itself, its tests can be very confusing when considered as examples. SimpleTestResourceTest is a good example of this. As an *example*, all it needs is its class-side resources method. As a *tester* of the SimpleTestResource, it has an instvar that hold an instance of SimpleTestResource, which merely confuses you if you are looking at it to learn how to use a test resource. SUnit 3.2 will have a package of SUnit examples including test resources and will have comments warning against using SUnit’s own tests as examples.

Resources are handled in TestSuite>>run (while presenting, Joseph realised that in 3.1, a failure to initialize one test resource could leave an earlier one un-tornDown; this will be fixed in 3.2). Here is where the resource instances are created, initialized (which calls setUp) and then asked isAvailable. Joseph demoed (with the usual demo hiccough that saw him acting Hal saying ‘Dave, we have a problem’ because his demo laptop lacked the software to play the sound).

Things you can do with resources. There are databases you can login to very slowly but the connection is good when you are in. When you get an error, it is handy to have the resource wait a bit before tearing down and logging you out, since you may fix the error in a few seconds. Joseph’s `TimedReleaseTestResource` forks off a process on reset that waits a bit before tearing down. If current is called again, it finds and kills the process and carries on.

To avoid tearing down the actual resource if it was already there, there is `LazyResource` with instVar `wasAlreadyAvailable`. This only calls setUp if the resource is not already there, setting an appropriate instVar value. It only calls tearDown if the resource was not already there.

The ForeverResource (aka the KillItYourself resource) overrides reset to not tearDown (you must kill the resource later yourself of course).

If something needs different state for different tests it is not well suited to being a resource. Stephane Ducasse has written a test case class wrapper that does some stuff to handle such situations but in general you should prepare the state in the setUp for the tests and accept the performance cost. For the case where distinct tests have resources for e.g. which database they should login to, and the system only allows being logged into one database at a time, and you then find yourself wanting to run suites of tests needing different databases, I am working on a different solution to Stephane’s (I call it `CompetingResource`) that integrates seamlessly with current resource handling, so that there is no need to refactor the tests.

SUnit 3.2 is expected for October, probably first in VW and in other dialects very soon afterwards; look on http://sunit.sourceforge.net. If you have any questions or problems with SUnit, please put enhancement requests or bug reports on SourceForge. It is easy, the code to do what you want may well exist and be sent to you, and if not, Joseph may do it.

Q. S# SUnit? AOSUnit already has stuff (test logging output in XML) that he hopes will make SUnit 4. It can be added to an existing SUnit as a subclass of TestCase from which you then subclass you own tests.

I raised some points I had discussed with Bernard earlier about persistent state and TestResources. It may be argued that TestResources automate that setting up and tearing down of resources which the user did by hand at an earlier stage of developing the first test needing that resource. Thus a TestResource instance does not represent the resource it manages as such, but rather the task of setting up and tearing down that resource. Therefore it is properly thrown away at the end of each test suite run (that’s each `run`, not each `run:`) and re-acquired at the start of the next run. Therefore, any persistent state associated with a resource (e.g. a database session) will usually be held in a registry and the corresponding TestResource’ setUp will re-acquire the object in the registry on each run using some key held in a hardcoded method of the TestResource class. If the test also needs the session object, ensure the hardcoded key method is a class-side method and provide the resource class to the test class via a method, e.g.

```
DatabaseTestCase class>>resources
  Array with: self databaseResource
```

instead of

```
DatabaseTestCase class>>resources
  Array with: DatabaseResource
```

so that you can then request the key in the TestCase’s setUp, e.g.

```
sessionKey := self class databaseResource sessionKey.
```

in a way that will not be compromised by the resource collection changing due to subclassing and refactoring of the test case.

Bernard has tests for an application, part of which exercises a framework that replies via asynchronous callbacks. Debugging the tests is OK but running them does not trap the debugger as the handler is on the calling method, not the forked thread. Michael Prasse and I wrote a method-wrapping resource that traps any error at the top of the fork and stuffs the test into the testResult errors collection to solve this.

If model layer tests are written to set up data for the test, then invoke that data on the model layer, then make assertions about the resulting state, then it can be very easy to subclass each model layer test case to a UI layer test case. The latter need only override setUp and tearDown to launch and tearDown the UI and then override a single method to drive the test data into the UI instead of the model. Then all the inherited tests can be rerun as UI-driving tests. The Custom Refactoring project's dynamic method renaming tests are an example of this pattern.

I discussed off-line with Joseph the fact that while `TestSuite>>run` and `TestCase>>debug` both check test resources, `TestCase>>run` does not. I believe this to be an error, hidden if you use raw TestRunner to run all your tests because it never runs individual TestCases but exposed whenever a subclass of TestRunner or an alternative UI allows this. We agreed that I will raise this issue on SourceForge and Joseph will address it.

Virtual Pair Programming, Kirk Blackburn, Stan Benda, Qwest
Kirk is in Denver and the rest of his team is in Minnesota, connected by a corporate WAN. (Using Envy on a WAN remotely is slow because Envy is very chatty.) Kirk wants to feel connected to the rest of the team. Thus he has explored technologies that let people share a common desktop.

They do a great deal of virtual pair programming. They use VNC (http://www.uk.research.att.com/vnc/download.html). It is cross-platform and does not require windows. Voice communication is essential for virtual pairing, as is a headset. Kirk has also investigated doing without telephone using internet voice messaging.

In VNC, do not do full screen polling (costly in bandwidth and VW widgets are not native so VNC cannot receive events from them). VNC with headsets is very acceptable when the Smalltalk image is close to the Envy repository. He has a server in Minneapolis and connects to that; an image in Denver connected to a repository in Minneapolis is not usable.

VNC displays two mice and two keyboards. This can be disconcerting, so you must have a conversational style for who has control and how control is handed over especially with two or three other clients. VNC shares your entire desktop, raising issues of privacy you must manage. (Q. Use server and VNC to it so only server is shared? Yes, we often do that.) On Solaris, have the set user id bit set and run as root to avoid problems.

There are other approaches. Using VNC corporately means getting management to download some program they've never heard of. Managers probably have NetMeeting on their desktop already, making it a very useful alternative. The explicit giving or taking of control is a different style to those used to VNC. The ability to share only one application is good (especially re managers :-)). NetMeeting has a (recent? Niall has not seen it?) fault of not rendering VW windows correctly (some lines were being lost). Kirk's workaround is to start a remote server and share the Remote Desktop Connection.

(Kirk said that RDC was a pretty good product but mentioned some issues. His experience with the RDC Experience Tab is to turn all the settings off. Reducing the size of your remote desktop seems to help your performance. Event handling is occasionally very slow; it seems to be related to a VPN issue, and can be got rid of by selecting another area, or hanging up and reconnecting. Citrix is an alternative which has worked OK for me.)

Sharing machines when pair programming in the same office is now often done. It is better ergonomically. You can ‘mute and munch’ while your pair is working. Kirk finds he feels less tired after a day of virtual pairing than of crowding round someone else’ desk. Virtual pairs don’t chit-chat as much; they stay more on task. He often programs for five hours at a time. He suspects this focus is also intellectually valuable (the two brain issue).

Voice over IP is cluncky and has delays but was economically valuable. Nevertheless they reverted to using telephones; net voice wasn’t adequate.

Some research in 2001 looked at an NCSU course (Dr. Edward Gehringer) teaching OO, Smalltalk and Java. All the students that session chose to do the course programming tasks in Java so we have no data on Smalltalk alas. They had some students co-located while others paired remotely, using a VNC modified to capture data on who drove, etc., and then pairs were switched. They found no productivity differences. However Kirk was saddened to see that all of Gehringer hypotheses were set up to check the null hypotheses; Gehringer was not *expecting* to see any differences.

Otherwise there is little direct research. Virtual pair programming is just a subset of collaborative cognition, of course, and David Lieb’s distributed cognition research is of interest. Kirk finds VPP fun and productive.

Q. Knew, and had met, pairs before VPP? Yes, and that as important

Q (Andrew Black, I think) VNC on a Mac? (He had problems a year ago.) Someone in Qwest uses VNC on Mac. (Andrew was using VNC on MacOS 10; it may work on other OS and maybe now on 10 with a newer driver.)

Q How many hours daily? 2-5 hours per day. You have to take breaks periodically and it is important to keep talking, keep a conversation going.

Q Would video conference add anything? Research says videocams are really not helpful; frame rates too slow. A way to gesture visually would be useful. (Roel: HP labs are working on that re conferencing systems.)

Micro Object Testing, Andy Berry,
In previous ESUGs, Andy has never shown any Smalltalk code. Since he writes it on a daily basis, he thought he should remedy this. (His code comes with a legal disclaimer; use it at your own risk. :-))

MOT is part of the move to agile developing. Andy finds that many managers are scared by eXtreme Programming. The way round that is to call it agile software development. Andy has been using MOT for years, was recently asked to explain his approach to testing (in the last two projects he was involved in, there were no formal testing methods used). Andy wrote a paper on what he did and it needed a title so he chose ‘Micro Object Testing’. MOT is a minimal overhead testing method. Why Micro? Because it takes little effort, needs a small amount of test code. You develop a bit, test a bit and deliver a bit, repeatedly.

Andy next talked about Models and Views. Andy always separates model and view layer (he has worked in environments where there is no such separation, e.g. Visual Basic programmers who write Smalltalk as if it were Basic). Views handle the messy things needed to deal with those horrible users. You must be able to change the view of your model. The view must know its model (in instVar). The model must not (as directly) know its view. Adaptors link views to models.

The first MOT principle is that you builds example objects for all classes that hold data, returned by `MyClass class>>exampleObject`. The simplest test just opens a view on the example object. Andy puts this test in the model `MyClass class>>basicTest`. Tests must run standalone and must display results. Run tests frequently, every few minutes or whenever you complete a testable code chunk. Test domain objects, not views. Build tests from the bottom up and test infrastructure code before application code. Keep the tests as simple as possible and reuse as much test code as you can. (Recently, Andy was able to reuse infrastructure tests from earlier parsers while building a complicated language parser.)

Provide lines of code to rerun tests and rerun tests that affect any code you change whenever you change code. Correct errors whenever you find them (and if you simply cannot at that moment, write them in a ‘tasks’ workspace that you keep visible, nagging you till you fix it).

Tests form an essential knowledge base about your application. If someone asks you what class X does, tell them to look at the code in class X’ class-side tests protocol; basicTest opens a view on one, databaseTest loads one, etc. (Needs you to keep objects to a reasonable size; Andy always splits up objects when they grow past 30 or so methods). By being meticulous about his tests, Andy finds that his applications usually work first time.

Q. When doing a databaseLoad test, how do you abstract the database? Andy would normally put it in a separate ReaderObject.

What does it cost? Andy spends 20% of his time developing tests but having code work first time makes it worth it. Andy advises everyone just to start adding exampleObjects to their classes and take it from there.

Full details are in his paper on www.tof.co.uk/mot. The paper has practical examples and confessions of mistakes he made along the way.

Q(Niall) SUnit? MOT is simple and code-based so Andy has no concept of test classes. Andy believes that tests belong in the objects being tested. Therefore he differs from, for example, myself, who thinks that they belong in test case classes in distinct (application-package-pre-req'ing) packages. He also ties tests to the application code structure whereas I would rather tie them to use cases (c.f. discussion in the SUnit BOF above.)

Q. Roel has a student working on making test outcomes visible. Roel will put him in touch with Andy re examples of making test outcomes visible.

Q. Christian uses example objects most often when he is unsure about his domain objects, when it helps him get clearer ideas about domain objects.

Q Kirk: you could use this pedagogically to teach people objects because it is very simple.

Post-talk remark (Adriaan); you could tie this and SUnit together by having a test that ran all tests in Andy's protocol, checking for which views appeared, so it could shut them on tearDown (or not, according to setting).

XPSwiki: An Open Source Tool supporting the XP Process, Sandro Pinna, Paolo Lorrai, Giovanni Corriga
I missed this talk but arranged to get a description of it. They have an XP swiki launcher that fires it up (lets you choose port, etc.). The pages are stored as XML. A template lets you describe your project, identify your team members and their skills. (The usual wiki functions let you use these templates as starters for links to extensive other pages).

Iterations page holds releases, and their stories. One 'point' is one effort day and this is how they record cost. A release has a points capacity. Stories can be assigned to iterations by simple menu selection. Stories have point cost, risk estimation, task breakdown. Historical data is presented in tables at the foot of pages so you can compare estimates to actuals. You move values from points left to points done to indicate progress on tasks. Each project has its own swiki (so can have its own security, admin, whatever).

Test details are not integrated with SUnit today; they plan to add that. Adding and removing an iteration is done by typing a name, not selecting.

Then we looked at the code. There is a class for each entity. The classes have instVars and methods for their data. A page is subclass of XPSWiki page. subclass of NuSWikiPage. The view part of the page has addresses actions and templates. Each page has a user-story edit template and a user story show. Edits have a swiki syntax `(.. response: .. result: ..)` that reminded me of servlets though I gather it not the same.

XPSwikiComanche is the bridge to the page server. For downloads, visit http://www.agilexp.org/xpswiki.

Wikis are a good forum for informal documentation of issues (we use a VW wiki in my project for this in my current work). The above is work in progress. I suggested they develop their templates to have more integration with code, with CM and with SUnit. More commercial projects run in VW than in Squeak; porting to SmallWiki would make it easy to maintain a common template and codebase across Squeak and VW.

Framework for Evaluating Distributed Object Interfaces, Jan Lukes
Distributed systems always have problems with performance. Jan wanted to visualize and profile how distribution affects performance, thence to understand the issues and see how to handle them. He intends to analyse and compare the following distributed systems.

- Distributed Smalltalk (DST) provides CORBA in Smalltalk. You either generate IDL from your Smalltalk code or use the implicit invocation interface (I3), an alternative marshaller using Smalltalk's meta-protocol (I3 must have Smalltalk at the far end as well to work).
- OpenTalk is a general Smalltalk-Smalltalk protocol for VW.
- GemStone is an object database which talks to Smalltalk images and has its own Smalltalk.

DST IDL registers objects for their whole lifecycle using a naming service. An object-object connection abstracts the real connection provided by the ORB serialization the object and sending it over the network for reconstruction by the far-end ORB.

I3 can pass objects by value or by reference. Classes are passed by name only (no methods or definitions passed) and what data is passed is controlled at the class, instance and instVar level.

OpenTalk runs directly above TCP and UDB. It supports several data and broadcasting protocols.

GemStone is a large shared object repository that has forwarders and also can replicate objects between GemStone Smalltalk and e.g. VW.

Jan's framework's UI lets you choose which nodes to start. Starting registers the machine with the framework. A hierarchic task list lets you choose which tests to use to exercise the system. You add your machine name to the list of free nodes, start a VM and interface on that node, choose a test to run and observe the results.

Various services are tested: registration (for protection from GC), naming, etc. Data transfer of simple, collection and complex data types is also tested. The third axis of test is parallelism: single client and server, many clients, many servers. The fourth is the direction of data transfer: from client to server as argument, from server to client as result, or both. The final axis is whether you have no ORBs (to get the performance of ordinary message sends), one ORB (simulate distributed sending in a single image) and two ORBs (real distribution). (Another axis was language; some Smalltalk to C++ comparisons were done.)

Jan showed the results as graphs (see his slides). The speed of naming service registration was unaffected by object numbers, as was simple data type sending. By contrast, one-way message sending speed rose for low numbers, then slowed as numbers grew past a critical value. Collections tapered off in rate and volume as numbers grew, etc. (see his slides).

A large volume of results can be created from this framework and need to be stored clearly (directory structure and naming convention) to be useful later. The framework attempts to present results with sensible scalings and layouts, etc. All tests were done on VW5i4 and 7.1 on various machines.

Their next task is to add OpenTalk and GemStone tests. Later they will look at other programming languages and at comparison of ORBs from different vendors. They will also investigate the effects of optimisations.

Q(Niall) Are you comparing GemStone's ORB to other ORBs or the speed of GemStone to Smalltalk transfer to that of other Smalltalk-Smalltalk transfers? He has not yet started GemStone so that is to decide. I indicated that the latter would be more interesting to more people.

Q. Any optimisation of VM settings? None except that each test was preceded by a garbage collection to ensure a common start state.

Extending Smalltalk: Languages, Platforms and Metaclasses

Building Robust Embedded Software, Lars Bak (OOVM ApS)

Lars was one of the implementors of StrongTalk (Animorphic, SUN). He has recently left SUN and talked about what he has done since.

Embedded systems are built the way we built desktop apps a decade ago: C code and libraries and RTOS kernel. The programming language is unsafe, there is no serviceability (if it doesn't work, try rebooting). Productivity is low.

In the toy industry, you may have 6-8 weeks only to build your software; they often start from scratch. Telephones also want rapid deployment. Embedded Java does not support incremental recompilation (v. hard in statically typed language). Its bytecodes are not designed for speed or compactness. Its virtual machine is complicated to implement (Lars knows, he's implemented several). The VM is not enough; you must always have libraries conforming to some spec defined by SUN or partners. The standard offerings for these are far too big; you may use 1Mb of your phone's limited space to get Java running. The result is that much of the industry has rejected Java for embedded systems and are still using C. Thus Lars goal is not to convince Java programmers to switch to Smalltalk. He's trying to persuade C programmers to switch.

Scenario: Bang and Olufson use firewire to connect their speakers to their stereo (this is a real scenario being looked at today; firewire lets you daisy chain your components and is cheaper than gold-plated wire). Speaker goes wrong, stereo is connected to web; customer calls B&O support and asks for it to be fixed. It would be nice to be able to do so.

Lars has spent the last 18 years working on VMs. In mid-90's he was caught by the Java wave and he reimplemented Animorphic Strongtalk as

HotSpot which was bought by SUN. (Last year there were 150 million downloads of the latest HotSpot VM.) Lars has also implemented a VM for cellphones that was 10 times faster than its predecessor. Meanwhile Lars has left SUN and is now doing OOVM.

OOVM is a different Smalltalk system:

- Reflection is in the development environment only. They can build up reflective changes on the device and then apply them at the end of the process, thus getting a consistent update.
- New ‘atomic test and store’ statement. The system can be interrupted between any bytecode so an atomic bytecode was needed to implement semaphores, synchronisation, etc. in the Smalltalk layer (implementing synchronisation in the Smalltalk layer was something Lars was keen to do as one of Java’s errors was implementing synchronisation in the VM when expert customers want to do synchronisation in their own way but that is hard when the VM has its own mechanism).
- namespaces
- typed LIFO blocks (see below)

The OOVM embedded platform is a tiny fully dynamic virtual machine executing platform-independent smalltalk bytecodes. It needs no RTOS or user C code. The IDE connects to the running program, supports true incremental recompilation, etc. As soon as you have burned your VM into the device, you can upload libraries, code, etc., to it. The programming environment is written in Smalltalk, but not yet itself; they deliberately chose another (ST/X) to ensure it was separable from the device’ Smalltalk. It talks to the non-reflection-aware embedded VM using a simple reflective protocol through a reflective interface on the device.

The programming environment is a web server written in Smalltalk (but former C programmers can use vi if they want).

They’ve kept it simple and clean. An object has a one-word header pointing to its class; no hash codes. The bytecode set is smaller and more efficient than the blue book one; there are only 20. The VM is mostly written in C++ (typed in, not generated like Squeak) because Lars is experienced using C++ to write VMs. It will run anywhere C does. The garbage collector will be real-time on release, is not quite so today. The scheduler controls what to do when any resource limit is reached and is fully configurable.

The Smalltalk is standard Smalltalk with a few twists. Namespaces let them group classes. They dropped pool variables (because they disliked them) and class instance variables (because they did not need them and only added features they needed). (Commented) raw code looks like e.g.

```
Mutex “class def” = Object “super class”
  | owner | “instVar for Mutex”
  do: [block] “method for Mutex” =
    “ordinary smalltalk syntax in methods”
```

The atomic test and store allows some neat implementations (see slides for examples). No reflection means only the programming environment can create classes and do performs. Their NetWorking libraries include TCP/IP and Firewire. They have SmallIntegers (30bit) and (not too) LargeIntegers with 32 bits only (signed only) because writing device drivers on a 32 bit machine must use 32 bit arithmetic and their customers want to be sure of that. Integers automatically convert from large to small and vice versa as needed but are (of course) not mutable.

They have slightly changed the Array hierarchy to make things clear for their customers (UpdatableIndexableCollection, FixedByteArray, etc.) but I did not manage to grasp precisely why. There is no Bag.

Namespaces let them modularise code. Each class can be a namespace. There is no import and export; you simply provide a simple class name, a partial path name or a complete path name and the system finds the class that has that name and matches such namespace path details as you provided. Thus you need only provide full names when needed.

Performance is the achilles' heel of Smalltalk. Smalltalk's inline basic control structures, flatten code (e.g. in the collection hierarchy), have an interpretation overhead (because they reify the stack and so on when needed) and method invocation is slow. Strongtalk solved much of this but at the cost of making the VM much larger. They needed new solutions.

They use typed LIFO Blocks that give 5 -10 performance gains, e.g.

```
collect: [collect] do: [block]
  self do: [:e | block value: (collect value: e)]]
```

where [collect] and [block] type the arguments as blocks. What they do not have is a block that can be stored and then return back to its original defining context.

Methods that have the same implementation share the same bytecodes (saves 10% of space). Having only 20 simple bytecodes, they can have other bytecodes for speed. If `load nil store 1` is a common sequence, create a single bytecode for it. They compute bytecode-pair frequency histograms and define these ‘super’ bytecodes for the commonest pairs.

A problem with standard Smalltalk is that left to right evaluation may need to get the receiver not from the top of the stack. Reversing the evaluation order lets the receiver be always on the top of the stack and can be done without changing any behaviour to the user.

The system runs happily in 128Kb (includes TCP/IP stack) which is smaller than all the OS+Java systems. It saves a lot of power because it uses less memory. Currently it runs on 200Mz and 50Mz StrongARMs in Intrinsyc Cerfcube and ICE Lynx, and on the IA32/Linux and StrongARM/Embedded Linux hosted systems.

Benchmarks: Lars compared OOVM to Java KVM, Java HotSpot, ST/X (JIT off) and Squeak for the DetaBlue, Richards and Stanford integer benchmarks. The goal is for it to be twice as fast as the fastest interpreted JVM (which also means being faster than the fastest current interpreted Smalltalk VM). Currently they meet this for all but HotSpot where they are faster but not twice as fast yet; they will be!! They also do well on method-invocation-heavy benchmarks.

The result is a fast small embedded system on which an application's development and maintenance times are drastically smaller than those of today. A version will be downloadable by year-end, free for non-commercial use.

Q.When marketing, do you mention the S word? No, they just say, “We have this very simple powerful language”. Smalltalk can evoke memories of ‘clever but slow’ (also it is important that programmers see they can go on using vi to edit code if that's what they want).

Q(Roel) he's talked to some embedded people who were fanatical about C syntax? Lars is opposed to making systems that are different look the same so deliberately avoided any C-like syntax. He's met the ‘We use C’ people and found that when you start talking about the problems they have in C then they quickly lose that and become more open.

Q. Specific CPU? Plenty of start-ups have tried it. At release, they look good but they can never keep up with the general speed of release of faster CPUs; next cycle they're hosed. You also cannot do general optimisations.

Q(Andrew Black) Real-time garbage collector? They will have a small Eden for new objects which get promoted to the next stage when full. A Smalltalk garbage collector will run on this next stage. They will also put their core at the end of the heap so GC will never need to process it.

Q. Where are you? In Denmark, 4 strong (and will not expand until the core technology is stable; 10 alpha-programmers in a room just get in each others' way).

Q. Use this for desktop applications? Yes, if you want.

Q. Why disable JIT for benchmarks? To compare against interpreted systems. They have the slowdown of being an interpreted system but they have a gain from breaking down the interface between OS and application.

Lars then brought up his web IDE: evaluator page, hierarchy page, class page, method page, edit (class) page and debugger page. Other buttons and widgets let you see what device you were talking to, upload classes to it, see when the GC last ran, space details, etc. The class hierarchy is shown as a tree structure you can filter in various standard ways (class name, implementors of, ...). There are no menus; you do everything with buttons. (At this point there occurred the usual demo glitch when he suddenly received an email while demoing.) Lars pinged the device and changed the ping-handling code to output a print; the device ping response changed while running.

You can get the code from the device into the IDE and prettyprint it so you can look at devices you have not programmed. The debugger does not yet have breakpoints. It will use XML to browse object graphs. Simplicity is their aim in the IDE, which is why they did not use Eclipse.

Their simple 20 bytecode set has a very uniform style (always operand argument) so you can always pre-fetch the argument. Java repeated Smalltalk's error of having many complex bytecodes that made it harder to write interpreters. Optimising the remaining bytecodes is usually done on their own code base, assumed to show typical bytecode-pair frequency. It could of course be done for a specific customer application if requested.

Because they do not have perform and stored blocks, static analysis is easier, so you can reduce the footprint on the device.

Q(Joseph) You can assign blocks to temps and parameters but not to instVars. I can live without 90% of pluggable behaviour but not without SortedCollections. How do I get that behaviour? Lars suggested a lightweight class; classes are very cheap in this system.

The Illusion of Cooperating Objects, Stefan Urbanek, Slovakia University
Stefan's problem is the same as Roel's in the StarBrowser's talk. He wants to integrate tools. The Smalltalk environment is his inspiration. He would like to create the illusion that real-world applications (mailer, spreadsheet, charting) are objects and that languages can be used to communicate between them. The StepTalk framework is intended to provide interfaces downward to the applications and this illusion upwards to the user.

StepTalk is written in ObjectiveC. The StepTalk framework talks to the ObjectiveC runtime via engines. Each application needs an interface that advertises application objects and then scripts can be executed on those objects. Application objects are identified by name, e.g. #currentMailbox. The environment interface translates between user-readable selectors used in scripts and the actual selectors. Environments are provided by either the Framework building an environment from an application description or an application can provide an environment customised for a particular task. To execute a script, StepTalk uses the environment to resolve the script's object names, classes and methods to actual invocations. The applications must be in ObjectiveC.

Currently the only engine is for Smalltalk. A bytecode interpreter executes compiled smalltalk code over a bridge to the ObjectiveC runtime. The arguments receiver and selector are taken from the stack, converted to a C method signature and argument values, and invoked.

He has implemented a script executor and a shell tool for user-interactive chatting to system. He will create a Session tool. Methods can be scripts (in any supported language). Details are at http://steptalk.host.sk/ and related.

Q. Relationship to FScript? Fscript uses only Smalltalk. His framework lets you plugin different languages. Otherwise, quite similar.

Q. Define classes in script? No. This is not a tool for writing complex applications, just for gluing them together and calling them from Smalltalk.

Inside AOStA, an Adaptively Optimising Smalltalk Architecture, Paolo Bonzini
(As this was described in detail in Eliot's talk at Toronto, written up in my Smalltalk Solutions 2003 report, I will be brief below except where Paolo is covering different aspects from Eliot.)

AOStA differs from self in that it is written in Smalltalk and compiles to optimised bytecodes, not native code. An AOStA-enabled VM has two areas for JITted methods. The optimised code works just as VW does now. The unoptimised methods must count each send and every backward branch (these latter are for optimised ifTrue, to:do: etc., where the optimisation would otherwise hide info).

Many VMs already have PICs. (I've described Polymorphic Inline Caches in detail in my Southampton 2000 ESUG and Toronto 2003 StS reports.)

You can recognise methods whose receiver class is constant and replace them with constant sends. This is not very interesting on its own but becomes so when you specialize methods. If an argument is 99% of the time a string, you could specialize the method and run the string-optimised one whenever the argument is a string. You can also inline and hoist the argument checker outside loops for performance. Another optimisation is to use additional bytecodes to eliminate bound-checking when accessing indexed instance variables where you know it is safe, Eliminate checks for e.g. being SmallInteger, etc.

The optimiser has a front end (done) that converts bytecode to SSA, a middle end (in progress) that actually does the optimisation and a back-end (not yet started) that converts special selectors to the extended bytecode format. The SSA node hierarchy is similar to the Smalltalk compiler's but lower level (has gotos !!). SSANodes have statement nodes and value nodes (that can be converted to statement nodes by the EffectNode decorator). A VoidNode eliminates dead code. ConditionalNode (subclass of StatementNode) and MessageSendNode (subclass of ValueNode) also holds profiling information from the VM. The SSANode hierarchy is very similar to the RB parse node hierarchy: both have methods to replace one node with another and both make much use of visitors. The optimisation pass is implemented by a subclass of SSANodeVisitor just as RB searching and rewriting are implemented by subclasses of RBProgramNodeVisitor.

Some predefined visitors order the basic blocks in different ways; picking the correct one speeds the optimisation greatly. Dominance information (i.e. which basic blocks must execute before others) is worked out while converting to SSA form and is used by the visitors to traverse breadth-first, depth-first or basic first as appropriate. Constant propagation (replace TempNodes whose values are in KnownConstant with ConstantNodes) is done early because it also exposes dead assignments. AssignmentNodes can be optimised if the RHS is a ConstantNode, or must be one of a limited set. Paolo's slides show code examples.

The unoptimised code runs for a while (at much the same speed as ordinary VW). Then you run the analysis and continue running. The optimiser would only be run to take a few percent of the CPU time at most.

Like Eliot, Paolo found writing an optimiser in Smalltalk great fun. AOStA is easy to support in a VM, especially one that already supports PICs. The potential benefit is high; the implementors' lack of free time to progress it is the main limit (collaborators welcome). Paolo demoed (with the usual demo hiccough) how a complex boolean expression that always assigned the same value was optimised to calling the expressions in the boolean then assigning the value.

Another 2-3000 lines of code will see this working without source-level debugging, which will let them collect performance figures.

Class Boxes, a Minimal Module Concept supporting Class Extension, Alexandre Bergel, Stephane Ducasse, Roel Wuyts
We want to reuse and extend applications. Smalltalk (and c.f. CLOS) support class extensions in packages; we can add or (sometimes) redefine methods to existing classes. In Smalltalk, class extensions are global (seen by all) and any application can modify any class. He wants to have class extension be controlled by the packaging / module system.

A classbox is a unit of scoping (i.e. a namespace). It can define classes, import class definitions from other classboxes and define methods on any class in the scope. An example is two class boxes, WWWClassBox and URLClassBox, who both add an asURL method to class String (imported from a pre-req classbox). A fourth classbox can see either method or none depending on whether it imports from WWWClassBox, URLClassBox or from their pre-req. Other examples showed how this interacted with inheritance; the class box pre-req graph is conceptually flattened into a single classbox which has only one implementation of each method so methods in prereq classboxes that call redefined methods get the redefined implementation. In effect, imports take precedence over inheritance; look for the method in each import pre-req and if not found go to superclass and again look for it in each import prereq for that class, etc.

To make this perform they had to change the VM method lookup, after which the system ran 60% slower. (And how did it run before? Don't ask.)

Q. How does this differ from selector namespaces (e.g. in S#). Selector namespaces interact differently with inheritance. If I leave the selector namespace while calling submethods, then I leave the namespace and when these methods in turn call methods that have been redefined in my namespace, they will not call the redefined method but the original method.

Q. How will this integrate with Traits? A little thought suggests there will be no problem. Ask him again when he has given it much thought.

Q(Niall) Use Noury's mixin's instead of changing the VM? Not tried, he will think about it.

He then demoed. The system runs in Squeak. He created a classbox, imported class Morph and added a subclass of Morph with a couple of methods to his classbox. (There is no browser yet so this was all done in Smalltalk text.) He then imported it into another classbox and redefined a method then showed that instances of his class had behaviour appropriate to the classbox in which they were created.

Q Why import class rather than all packages. We also support just importing all classbox but the explicit class import has the advantage that you handle conflicts when they occur. With packages whose class and method contents change, conflicts can arise under the covers so explicit class imports are useful, especially while developing the classbox system.

They will use the current system to see what is needed (perhaps selector namespaces are enough?) and to explore optimisation further.

Extreme Late Binding, Ian Piumarta
Ian maintains the Unix VM for the Squeak community. However he is not using Smalltalk today 'in anger' to build the systems he is researching. His talk will explain why that is. He is working on a dynamic architecture for building dynamic languages and systems (it could be an application in its own right but the aim is for it to be a platform).

There are a wide variety of systems emerging that are

- autonomous (e.g. self-repairing)
- dedicated: well-matched to a specific domain
- highly-reactive and/or dynamic

Ian is also interested in building a VM shell, a platform to build VMs, a virtual virtual machine. The shell should produce a VM adapted to the domain requirements. He calls this Yet aNother VM.

At one extreme, there is the Ants system of small encapsulated agents, but the creator of the Ants system said the major obstacle to building such systems was the programming language. Object has as many definitions as there are languages, garbage collection also varies widely, etc. A second set of challenges is primitives. You need synchronisation and saturated arithmetic semantics. You may import them from outside, the pragmatic but unreliable choice. Comparing them raises the issue of whether there is a compiler on your platform at all and if so does it do what you want (plus it means delay). Then you have to worry about the basic execution machinery and all the fancy additions to it you might make.

Conventionally, the source is interpreted via some syntax and semantics and pragmatics by a compiler to produce a binary application that a runtime understands. The source and application are under programmer control but they cannot change the compiler or the runtime or the syntax, semantics and pragmatics. Smalltalk make changing them less impossible than in most programming languages, where these things are hermetically sealed, but it's still hard. (E.g. Lars adding atomic test and set in OOVM).

Saying it another way, a standard programming language is like a walnut; you are stuck outside the shell and all the above was designed by a committee (that you never met). Conventional programmers spend vast amounts of time nibbling on the shell because they know there is some excellent stuff inside; either they never get inside or they get their friendly VM hacker to drill a very rough hole.

Thus most programming languages are disastrously early-bound, have insufficient meta-data (a huge amount of information is generated by the compiler and then instantly thrown away), have needlessly-closed-off access to internals and no reification of the implementation. Ian thinks that Smalltalk has an excellent idea about late binding. So let's apply it to *everything* (including language implementation). Make programs and native code first class objects. Leave the entire implementation open.

What is 'everything'? (Ian reminded us of the 'Make your theory as simple as possible but no simpler' quote.) 'Everything' is the usual Smalltalk stuff plus the platform's native code, the calling interface and the connections to libc, posix, windows or whatever. Now everything can be changed except the hardware (and with the right mixture of FPGAs and luck that's not impossible :-)). His universal dynamic compiler (YNVM) is not a VM but looks like one. It has dynamic compilation (of course), and offers simple persistent meta-data with visible bindings to meta constructors as well as accessors. From this, the application inherits lots of good things, all of them changeable on the fly.

Ian showed us some structure diagrams. The parser talks to the tree compiler, the tree compiler talks to the stack compiler, the stack compiler talks to .. etc. The dynamic assembler generates binary instruction emitters.

Ian demoed his dynamic C compiler written using this. He showed a C program to convert centigrade to fahrenheit by building a function at run time from a reverse-polish expression. He compiled and ran it. Then we looked at how it worked. Tell it what platform, loop over C string finding operators and compile them. As shown, it is horrendously unportable (though has been widely ported of course). Now lets do it on a Virtual Processor Unit that abstracts over the actual hardware; instantiate objects, send them messages, then tell objects to do their stuff. Program pushes operators on stack as objects and returns VPU which you then tell to do its stuff. This program will run on any machine. Ian now has a programming language equivalent to C (with some obvious improvements) which has an arbitrary concrete syntax, allows on the fly changes and will run anywhere.

At this stage we still have to write a compiler so lets reify that and bind it late too. Ian grabbed the getTimeOfDay function for his computer, used a C function to convert it to a string, and the sleep function, grabbed memory to store an integer, and wrote a loop that printed the time of day every second. All functionality used by this program was obtained dynamically. He showed another fifty-line program that opened a window to record events; it imported all the Xlib functionality it needed dynamically.

He then ran a program to convert reverse polish to prefix notation. The program constructs a tree that represents a block that does the conversion. This is compiled and then returned to the calling program which can use it as if it was a statically-compiled function. His dynamic C compiler reifies all the syntax and semantics understood by the compiler. Having explained this, he then rewrote the conversion program to a more sensible form for the new power he had (result was scheme-like).

Everything, including function application, assignment, etc., is bound to a symbol and can be changed by the programmer. Ian wrote a small smalltalk-like classes and methods system. The essential operation was assignment. He built an abstract class array, extended it with concrete arrays for primitives, etc. He then compared the fibbonacci function written in C and in Ian's system; the static took 430ms, the dynamic took 900ms. Getting all this for only a factor 2 slowdown is very encouraging. Ian showed the dispatch mechanism; it uses an inline cache.

Smalltalk relevance (immediate):

- Slang looks like Smalltalk, executes like Smallltalk but is in fact C code. He can convert slang to native code on-the-fly. (Ralph Johnson did a system like this in the early 90s.) Ian showed it building SmallInteger primitives.
- Reduced smalltalk systems such as SqueakScript or Slate could use Ian's work.

Q. Really hard issue in VM building is detecting overflow? Ian's trick method isIntegerValue takes the result (a signed 32 bit number), left shifts by one, xors it with itself and sees if the result is negative; if it is, you overflowed. (Obviously, there is more work to do.)

Many programmers get frightened when presented with a too general system. To evolve a complex system from a simple system you need stable intermediate forms and the resulting complex system will be hierarchic. (Parable of the watchmakers; Hora and Tempus build watches of 1000 parts and both keep getting interrupted by phone calls. Hora builds them from subassemblies of 10 parts making assemblies of 100 parts, Tempus builds them all at once. Whenever they are interrupted, the current assembly has to be started again. Hora grows rich, Tempus goes bankrupt.)

Hence you need to freeze out the variability in you system to create these intermediate forms. Choose what you mean by a class, by an assignment, etc., and the usability then goes up (for a domain suiting those choices).

The symbolic space matches the semantic space; everything the compiler understands is accessible from the symbol space. YNVM will boot on bare hardware. On three year old hardware, a dynamic function that builds 1Mb of native code takes one second. The system is wholly described in 10kloc and occupies 250Kb with absolutely everything compiled and linked.

The walnut has now been turned into a bowl of custard that we can stir into whatever we want. Presented with so much generality, some people just get confused so you must convert it to fixed forms by pouring your custard into a specific-shaped bowl, applying semantic sugar and getting a crust, You now have creme brulee and can break the crust whenever you need.

Q. Past work? Ian noted similarities between his work, Ocode, PCPL, etc.

Q. Commercialisation? He had some interest from Alcatel but, like other companies, they have no money today. Ian has build a JDK-compliant system. It was 3000 lines of code and is twice as fast as the best purely-interpreted Java (thus meeting Lars requirement in a sense but they are not a pure interpreter and not working in a resource-bound environment). Many telecoms companies have respectable amounts of space on their devices and are looking for ad-hoc solutions; for them, this may be of interest. Ian is now building a truly reflective introspective Java. A student has worked on this for two years (‘hello world’ took a fortnight).

(Ian computer’s response tag is ‘morituri te salutant’. I don’t know if this relates specifically to the reliability of YNVM or not. :-))

Smalltalk in a .Net World (How to write a Smalltalk Compiler without writing a VM), John Brant, The Refactory
John and Don started building a Smalltalk compiler for .Net last October. They called it #Smalltalk (because it is pure Smalltalk syntax and #Smalltalk is a valid symbol in Smalltalk; do not confuse with S#, which may be a valid symbol in S# :-)). It is open source and is written in itself (except the large integer DLL). It runs under Windows, and under Linux and Mac if you use Mono.

Normally you build the virtual machine first when you build a Smalltalk, then a compiler with parser, and so on. Their virtual machine was built for them already (.Net) and the Refactoring Browser has a parser which is a big step towards having a compiler. Currently they still use VW as their primary development environment but are no longer bound to it.

Everyone has an opinion of .Net and most people in this room would have ‘sucks’ in that opinion. It has a common VM and byte code instruction set so language isn’t supposed to matter; of course, it does. They are doing this for both business and personal reasons. Business-wise, Microsoft backing means there will be lots of code, some of it good, in .Net and they want access to the good stuff. They also want to use Smalltalk on applications that ‘require .Net’ (in the past he has had to argue to use Dolphin because some clients see it as ‘not .Net’). Personally, he wants to build a Smalltalk compiler and wants to get quickly past VM stuff that does not interest him to the compiler stuff that does interests him. He also wants to learn .Net.

Smalltalk is dynamically typed. .Net is statically type. Smalltalk has tagged primitive integer types, etc., whereas .Net types are primitives *or* objects; you cannot combine. .Net exceptions are not resumable; Smalltalk's can be. In Smalltalk, all methods are virtual; .Net allows static methods. In .Net, you can only add classes and methods at run-time (there is a rumour that they are looking at enabling modifying). .Net has structs and enums.

In Smalltalk, the root supertype must understand every message and send doesNotUnderstand if necessary. John simply inserts a root class above all their Smalltalk classes that understands every message and calls DNU on it. It is not that hard to support DNU.

While nil is a special object that understands DNU, null throws a null pointer exception when you send it a message. For classes, it is no problem; just initialize all instVars to nil. For methods, it is harder. You could assign every temp to nil but most temps are written before read so this would be dead code. Hence they just look for every temp that is read before written and assign them to nil at the head of the method.

The hardest problem of implementing Smalltalk on a statically typed system that does not provide tagged integers is SmallIntegers (and the other immediate types). Smalltalk compilers do this by stealing the bottom two bits to tag immediate types; its only an object if those bits are (John said 10 but Georg assured him it was 00. I thought it was four bits, one tag and three to represent eight immediate types; meant to ask after but forgot). .Net cannot do this so they have to use real integer objects, which are 8 - 10 times slower in their implementation than in normal VW.

Blocks are handled by creating a class (e.g. MonadicBlock) for each block. Pushing a block on the stack equates to pushing a new instance of that class on the stack with an overridden (as in 'subclass', not as in 'VW override') value method. When the block has variables, parameter values are easy (they never change) so can just be copied but temporaries must be handled by active variables, variables that are pointed at by other variables, so the temp and the block variable both point at the active variable and both change when the active variable does (like what VW does in some places).

There is no support in .Net for non-local return. You can only simulate it with exceptions and the result is 100 times slower than in VW (luckily, there are not very many of them). The exceptions have a tag object (just an integer) so a handler can see if this return is for it or for someone further up the stack. (The tag is per activation of the method, not just per method, so it can handle recursive calls. A fork to another thread will cause the raised exception not to be handled. You could create a stack of 4 billion returns and get clashes but John thinks that a rare case.)

Some actions are primitives (not representable: identityHash, + , etc.). VW primitive tags only allow one per method. John has a method

```
Compiler primitive: aOneArgBlock
```

which you can call anywhere in a method. For example

```
identityHash
  ^Compiler primitive:
    [:codeGenerator |
    codeGenerator call:
      (System.Object
        getMethod: ‘GetHashCode’);
        constructMethod ...
```

Thus users can add their own primitives.

All implementations of Smalltalk have messages that are (almost) never sent (ifTrue:, etc.) because they are optimised in the compiler. John decided to use macros to transform code into Compiler primitive: []. Their macros are RB rewrite rules that e.g. search for all ifTrue: sends and rewrite them to the equivalent Compiler primitive: [] form. This also allows other optimisations, e.g. ifNil: -> isNil ifTrue:. This raises the issue of how the debugger handles it. Existing nodes just have their source interval copied so the compiler can highlight the right interval. New nodes are just ignored. Macros can apply to the whole system, per class hierarchy, etc.; the system is configurable.

John wanted to integrate with .Net. There are 2500 classes in .Net and he wants to use them as if they were Smalltalk classes, in code that looks like Smalltalk. He showed Smalltalk code that created HashAlgorithms and computed a Hash using .Net classes while using Smalltalk FileStreams to open and close a file. It runs 90 times faster than the same code in VW because it uses .Net to do what it does best (computing hash algorithms).

When Smalltalk and .Net classes have the same name, you must refer to the .Net class via System.ClassName. Some variables must be typed (e.g. System.Int32) to get values from .Net; many base classes use typed instance variables (e.g. their OrderedCollection is essentially a .Net ArrayList). For untyped instVars holding .Net objects they have a GenericObjectWrapper that traps DNU and sends the message to the .Net object. It is a .Net standard that you should not give an accessor the same name as a field because some languages cannot handle it, but you can do it.

.Net has no keyword methods. They make the first keyword of any call to a .Net method be the .Net method name; the rest can be anything. So when calling .Net methods you should have a standard of using simple second and subsequent keywords (e.g. with:with:with:... seems sensible) to avoid unnecessary complexity. Overloaded .Net methods are sorted right to left and generated as type-checking if-loops. For example,

```
System console write: anObject
  (anObject isKindOf: SmallInteger)
    ifTrue: ...
    ifFalse: (anObject isKindOf Character
      ifTrue: ...
      ifFalse: ...
```

John plans to do type inferencing, when this will go away again. Meanwhile, he feels that double-dispatch is not faster than isKindOf: in this architecture. He did not want to make the programmer provide type information, so as to make using .Net code as like Smalltalk as possible.

Events use add_EventName:, remove_EventName:. This is not as seamless as John wants since delegates are statically typed and so

```
Button new add_Click:
          ([:obj :args | ..] asDelegate: EventHandler)
```

must have a compiler trick to tell statically typed delegate what kind of block it is. Exported properties / methods are done by annotations.

The initial version was a proof of concept. It took a month or so to get working but did not feel right as they had to insert a block of text to compile with .Net ilasm at every call to a .Net method. One of Ralph Johnson's students made a DLL to connect VW to .Net and with that they asked .Net for the method to call to make it seamless (now VW has a DotNetConnect which sounds like it will be much better). Then they overrode qualified name in VW to look in .Net for unfound dotted names. At least, .Net has a namespace for creating your program; System.Reflection.Emit generates your .exe. They kept iterating on version 2 until it could compile itself; they achieved this in March. Whenever they did this, they compiled it in itself three times to make sure they had no problems, then ran their SUnit tests.

John show some benchmarks. For all but integers and sets they were faster than Squeak and in the Dolphin ballpark (Strings slower). The SmallInteger problem affects all their figures.

Visit http://www.refactory.com/Software/SharpSmalltalk for their work, http://www.rileywhite.com/software/assemblyconnect.html for the student's DLL, and http://www.go-mono.com for .Net on Mac, etc.

Q. Use VisualStudio as IDE? Can do but John won't be spending the money to arrange it. Maybe ask Microsoft but who should he ask? (People gave John contact names.)

Q. How much VW code will run on this? Except for primitives it should run any of it (e.g. they have run all the ANSI SUnit tests).

John demoed a Swazoo server running in #Smalltalk on .Net. He used it to browse his local Access database. He has also ported Glorp (John looks for stuff that is open source and has tests to port).

Q. General views of .Net? John's biggest gripes are the integers and the block returns. By building something and showing Microsoft the result, they may be more influenced to change. One of Ralph Johnson's students is exploring tagged integers in Rotor (the research arm of Microsoft) and there is a Smalltalker on that team so influencing them is not impossible.

ESUG: Academic Track

Nine papers were submitted, of which six were accepted by the programme committee. All six were also accepted for an Elsevier special edition on Computer Languages. Roel encouraged people to submit papers for next year and offered help to anyone unfamiliar with how to write a scientific paper (provided they sent him an email sufficiently long in advance).

Program Analysis

MudPie: Layers in the Ball of Mud, Daniel Vainsencher

The problem is packages with cyclic dependencies which are therefore hard to move to other images, dialects, CM systems, etc. People write utilities, load other utilities, develop, and then discover (or more often, someone else wanting to use their work discovers) that they have cyclic dependencies between packages: class name or class extension references.

Daniel met this in a project to decompose Squeak. Much functionality has been added that is now hard to remove and/or restructure. Utility classes such as SystemDictionary knows about sockets. UI construction code knows about Menus and Halos. Etc., etc.

The first task is to resolve the image into cycle closures, which he called ‘strong components’: component groups without cyclic interdependencies. Daniel showed the Refactoring Browser components in Squeak, with many cyclic dependencies. He resolved this into ‘strong components’, producing a simpler graph. Within each component, you can then refactor to remove the cycles. By looking at the relative size (how many individual links there are pointing from one package to another) of the package dependencies, you can see which one is wrong and to be removed. In Squeak, Parser has only three links to ParserExtensions whereas ParserExtensions has many links to Parser; even without the clue of the name, this would suggest to you which dependency to remove.

You can do this refactoring in XP style. Write a test that asserts that the wrongly-dependent component is not dependent and then refactor till it passes. Keep the test in the system to detect when others accidentally add dependencies again. (Example code for such tests is in his slides.)

Currently, his code detects static dependencies. Type inference might let you see dynamic dependencies but the ease and usefulness is unclear. Generating the dependency graphs must be done anew each test run. It is quick (Daniel can analyse the whole Squeak image in one minute on an old machine) but perhaps they should be maintained in the image. He sees no problem in making it real-time as an always-on check. Integrating dependency information with the Browser would help the user. Daniel demoed something he has just implemented, showing the components sorted into a tree structure by dependency.

Q (Joseph) Static dependency types omit explicit references to Globals, which may fit your current definition of a package but you should plan to extend to it? If package depends on globals being initialized by another package then there is a dependency. A. Squeak does not have an integral definition of package so Daniel used a ‘package info’ utility that indeed did not include global definition. He recognised the problem.

Q. (Stephane) Compare design model provided by system creator with actual observed dependency? Could do.

Daniel has looked at doing the analysis in what-if mode to guide work: “Suppose I could solve this cycle; what would it look like then.” (Joseph has often wanted to do such what-if checks and encouraged pursuing this.)

Q. Use for porting pre-sort? That is one application.

Language-Independent Detection of Object-Oriented Design Patterns, Johan Fabry and Tom Mens
Johan apologised for the fact that ‘language independent’ includes Java. They have developed SOUL (Smalltalk Object Unification Language; see my earlier ESUG reports) to handle other OO languages because they have many research results on Smalltalk and would like to compare them.

Declarative meta-programming in SOUL lets you write a prolog-like construct to detect patterns, and can embed Smalltalk code in the logic query to assist defining the pattern. The inference engine uses a library of logic goals, e.g. for finding things like ‘allClasses’. The Library talks to the meta-level which might implement

```
allClasses
  ^System allClasses
```
and this talks to the Smalltalk image.

To map this to Java, they used the Frost parser to parse Java code. The Java code is stored as parse trees and they have a Java code browser (in the Star Browser; could code in Java if you wanted to but that’s *not* their goal :-)). With this, they implemented a Meta-Level Interface for Java that talks to the Java repository (and understands interfaces). Thus they are language independent for class and method signature queries.

To be language independent for method bodies, they transform the body parse trees to logic representations. However the parse trees of Java are much more diverse than Smalltalk (if-then-else, while, for, etc.). They decided to keep the parse trees as close to Java as possible instead of transforming them to Smalltalk parse trees. The required extensions to the system can be confined to the parse tree traversal layer.

They compared HotDraw and the RB in Smalltalk to Drawlets and JRefactory in Java, searching for two simple patterns: double-dispatch and getters. For double-dispatch detection, they found they needed three language independent rules and three dialect-specific rules. They also searched for Visitor; this only needed language independent rules. They also looked for Template Method.

Their conclusions are that you can reason over OO programs in a language-independent way. They found that the set of dialect-specific rules grew more slowly than the language-independent ruleset, suggesting that in time the dialect-specific ruleset will become complete.

They want to detect which patterns are truly OO-language-independent and which are really language-dependent (or at least, not independent as current phrased). They have just made it generate code from the language independent form. They do no type inferencing yet; they may in time.

They found no false positives. False negatives are harder to detect; they checked with great care and believe there were none.

Q. Uses? You can understand a large system better by searching for e.g. visitor or template pattern uses and browsing them.

Q. On your slide, the RB has 14 visitor patterns but JRefactory has 174; why? 174 is the number of participants in visitor, not number of patterns; Java forces a much larger numbers of objects to participate in the pattern.

Q. Speed? Some checks took an hour but can be much speeded up.

Induced Intentional Software Views, Tom Tourwe, Johan Brichau, Andy Kellens and Kris Gybels
Andy had no difficulty persuading us that large software systems are often inadequately documented. Software Views highlight important design structures in code. Views are extensional (user defines contents) or intensional (rule-defined). In the Star Browser, extensional views are very easy to make (user drag-drops from StarBrowser) but it is hard to handle very large sets and they are apt to get out of date as the system develops.

Intensional views just define a rule. As the program changes, the view is updated. As the view gets large, they scale well. The intension rule also documents the intention (note spelling) of the view. However they require much knowledge of meta-programming to write and get right.

Induced intensional views aims to make writing correct views easy by using logic programming to deduce the rule from examples. The requirements are a set of examples and a background theory. An inductional algorithm converts the examples (‘facts’ in logic programming lingo) to an intensional view definition. (He gave an example of a derived rule. Its last check was that class inherits from itself indicating that their algorithm needed further refinement to eliminate redundancies.)

Their experiments reveal that rules can be found but the current algorithm is sensitive to the number of examples and the order in which they are presented. They might solve this by making the rule semi-automatic, getting some input from user. Performance is also less than ideal.

A prototype tool exists (Star Browser extension).

Q. I create some extensional views for my various utilities, then submit these to the tool to be given an intensional view? We did that for the visitor pattern; it produced a rule very like the one Roel wrote.

Q (Michael Prasse) if I have a UML design, can I integrate that knowledge with this tool? (I missed the detail of the answer which I think dealt with the different formats of the StarBrowser design constructs and UML's design constructs, so that the user would have to map between the two.)

Language and Reflection

Metaclass Composition using Mixin-Based Inheritance, Noury Bouraqadi, Computer Science Laboratory, Ecoles des Mines

Mixins are an alternative to multiple inheritance. Supposed you want class C (superclass A) and class D (superclass B) to share code. Extract the code to share to a mixin class X. Noury's prototype (MetaClassTalk) will generate classes X1 and X2 behind the scenes, and quietly inserts them into the hierarchy between A -> C and B -> D. The protocol is, e.g.

```
Point subClass: #ColourBoundedPoint ...
...
ColourBoundedPoint
  mixins: (Array with: Colour with: Bounded).
```

The mixin order determines the order of generation of invisible lightweight intermediate classes and therefore the order of searching for implementors in the mixins.

Problems: calling patterns such as `self class foo` and `self new bar` may not work for mixin instances. In Smalltalk, there are automatic parallel hierarchies between A -> B and A class -> B class. The basic metaclass system breaks this link for the lightweight mixin classes. There is also unwanted class property propagation; for example, B may become an abstract class because of the new inheritance. To overcome these problems, Noury gives classes properties that the system uses to avoid the unwanted class (mis)propagation of effects. In his framework, mixins avoid the undesirable class property propagation effect. Implementation in Squeak was straightforward and caused no performance loss (because it generates classes so relies on standard VW lookup mechanism).

Future plans include refactoring the Squeak libraries to use mixins (will need to move it into the kernel, which will give them a bootstrap problem).

Q. Port to VW for wider use? Noury has no plans to do so; he would welcome someone else doing so.

Q. Super sends? A mixin's super will point to the superclass of the mixee.

Q. Tool support? None as yet.

Q. Mixins are classes? Mixins are classes at present. Perhaps they could and/or should subclass ClassDescription or Behaviour but not Class.

Q. Mixins can have superclasses? Not at present. (Traits allow a sort of compositional inheritance but that is a different model.)

Q. Use mixins to do instance-based behaviour? It is possible but Noury does not want to confuse the work by expanding the feature list too much. (I note that John Brant had a utility for doing instance-specific behaviour using lightweight classes that had resemblances to what Noury described).

Q. (Andrew Black). Studies suggest that 15 - 25% of the collection hierarchy code could be eliminated by use of mixins.

Environments

Unanticipated Integration of Development Tools using the Classification Model, Roel Wuyts and Stéphane Ducasse

(See Roel's paper for much more detail on this.) So what is the problem with integration? Integrated tools (e.g. Macromedia's toolset) presuppose they have anticipated what integration is needed. Try integrating VisualWorks into this toolset. In real life, you discover the need to integrate tools you did not anticipate. They have GUI compatibility issues. Their models are different. As a result, users do the integration by hand: cut and paste, file export and import, etc.

We want to use one tool's output as another tool's input. The classification model is a lightweight item grouping model. Items are manipulated by services and the visitor pattern links these. The two axes of variability are:

- custom items
- custom services on items

This lets you impose the integration architecture from outside. The original tools do not change. The integrator, not the original developers, does the integration (example of good OO design; the right 'object' does the work).

A new service as added by adding a new visitor. Subclass Service, override methods for existing objects if you want the new service to handle those objects differently and add a method for new item type if your new service must handle them specially. Suppose you want to handle methods differently:

```
Service doMethod: aMethod
  ^doObject: aMethod

IconService>>doMethod: aMethod
  iconFor: aMethod
```

Roel showed integrating Advance (UML editor) and SmallBrother (tool for watching which methods you have browsed). They were not designed to work together but now you want to see recently-browsed UML classes. Just make a new service, AdvanceEditor. that supports the SmallBrowser MethodHistory item. It is in VisualWorks and in Squeak (where many services have been added). Class extensions are key to using the visitor pattern. Compare Eclipse: they need far more code because their packaging mechanism does not support class extensions so they must use observers.

Q. (Niall) DragDrop can be controlled by the item types dragged from and to? Yes, by doing DragDrop as a service (done in Squeak today, in VW as soon as Roel back-ports it).

Q (Daniel and others) Service conflict possibilities? You can point at different service configurations. If two services handle the same item with the same name, there could be conflict (I did not catch all that Roel said).

A Browser for Incremental Programming, Nathanael Schärli, SCG University of Bern and Andrew P. Black, CSE Oregon Health and Science University

Incremental Programming is what we do in Smalltalk. It is more productive and more fun than other kinds of programming. Smalltalk lets you program with limited knowledge of what arguments this method will want and what class it should live on when you start writing it, etc., by allowing generic protocols without type declarations. It lets you work in multiple contexts by providing multiple windows and tiling browsers. What standard Smalltalk browsers do not provide is information about the completeness of classes and their collaboration. But this information is in the code and modern machines are fast enough to compute it and keep it up to date.

Virtual categories (method categories, i.e. protocols) are computed by the browser from the code:

- -requires-: methods used by the class for which no method is defined / inherited
- -supplies-: required and provided (I wondered if it should be called ‘supplied’)
- -overrides-: overrides of inherited
- -sending super-: those that send super

The browser shows these for local visibility and for visibility to superclass. Andrew demonstrated by writing a simple app. Methods required but not yet written appeared in his browser navigator’s selector pane as a colour-coded selector in the -required- protocol. As he wrote more methods, others appeared in the -requires- list and class names also changed colour as they had or had not unsatisfied method requirements. Abstract classes should have unsatisfied requirements; at present the tool leaves them so even if you provide self subclassResponsibility implementations of them.

The browser is good for understanding and modifying existing class hierarchies. It shows which methods are core and which are support. Collection has four core methods: `add:`, `remove:ifAbsent:`, `do:` and `atRandom:`, and 110 other methods. The browser shows these four in blue. In Bag, we can see the four supplied methods and 10 others that override to disable or to improve their efficiency. A few other methods are in Bag to extend its features.

They also found accidentally abstract classes. `Fraction` should implement `printOnBase:` but noone had noticed it before. `Bitmap` should implement `primitiveFail` but does not. Why are these errors present in a codebase used by thousands of people, some much better programmers than Andrew? It is because they did not have the right tools.

Implementing -requires- was not easy; it required a lot of careful thinking to make it work in real time.

```
Behavior>>requires
  ^self reachableMethods selfMessages difference:
    (self allReallyImplementedMethods)
```

Finding `reachableMethods` is hard.

Self sends need a parse of method text. A change in Object might ripple down to change the requirements of every class in the image (Squeak base has 60,000 methods) so they cache all the self sends for each class. Nathanael had two key insights:

- For each message, cache the methods that send it, not the other way round.
- You don't need to know the required set, only whether it is empty or not.

With these optimisations, computing the required set takes 100ms.

This goes beyond existing tools. Eclipse' to-do list is only updated on global recompilation. VW decorates local properties (super sends and overrides). He's unsure if the StarBrowser can be made to work with this due to the need for a complex optimised implementation.

They may provide diagrams of the self-send info since some cyclic loops are valid while others mean 'you must override one of these methods, whichever one you wish'.

Q. Why show deliberately abstract classes as incomplete? Andrew plans to add a separate colour for abstract classes (i.e. ones with no missing methods but some subclassResponsibility methods).

Q. Sort the -all- category by dependencies? It's a partial order, not a total order so a sorted list will lose information but might still be useful.

Q. What's the use of -supplies-? It helps you see how the class interacts with its superclass.

Q. Smalllint does some of these checks. Add your code to Smalllint? Could do.

Q. VA Abt classes override DNU. How does the tool handle this? The problem is also there in Squeak. They do not recognise as self-sends cases where self is assigned to a temp or when a message is sent to self class new. Magnitude defines less than as greater than sent to its parameter, which is a self send but not recognised as such. DNU is just another such case. The tool works well in most cases, not all.

Q. Do byte code analysis instead of source code analysis? It looked hard. They did some but although the Squeak image on the disk is portable the one running on Andrew's PowerPC is not the same as the one running on another kind of machine. The Squeak tools all get it right but Andrew did not even know this was happening until he worked late every night for a week tracking down a bug. (Also, Roel pointed out that byte codes will compile away information.)

ESUG Academic Activities

Teacher Excellence Package, Gabriela Arevalo and Stephane Ducasse
ESUG offers a Teacher Excellence Package of Smalltalk books plus a CD of non commercial Smalltalks and a prepared lecture with 400 slides. It is easy to request and ESUG pays so there is no strain on the University budget. Just fill in the form http://www.esug.org.sponsoring/. The only requirement is to show that you will use what you are sent and to say how they will be used so you get sent the right books. (N.B. they send large quantities of books to poor countries, less to rich but rich universities should also register to get the slides and a book or two.)

The second package is the books plus the course. They have given it in several universities with excellent feedback.

They can also send the lecture material for local people to give the lecture. The slides are free for non-commercial use and can be used commercially (without Stephane's name if you want to customise to your company) provided you pay ESUG something. Please email Stephane if you find any errors in the slides.

Other Discussions

Vassili mentioned that he found this conference more fun than Smalltalk Solutions 2003, in part because of this atmosphere of continual discussions while looking at code. I suggested that having an explicit room for it, explicit time in the schedule for it, and the Camp Smalltalk people setting an example was what made the difference. At Toronto, the schedule was filled by three talk tracks and there was no Camp Smalltalk to set an example. Only one location had a possible discussion room next to it, the others being far enough away to be inconvenient, and that room's layout did not invite pair-programming (lots of little round tables are unhelpful when several want to sit and view a screen).

Stephane noted that the 'Smalltalk mailing list in French' was started recently with 3 members and now has 17; the message is, just doIt.

Noury is considering a one or two day annual conference for Smalltalk that is more aimed at managers (ESUG has a very technical focus). Email him if you have suggestions for the timing and content of such a conference.

Conclusions

My fifth ESUG and the best location since my first, or maybe ever.

- I like the Knights of the Square Bracket logo.
- #Smalltalk and DotNetConnect handle the challenge of .Net.
- OOVM and YNVM show you can take Smalltalk anywhere.
- Successful small commercial projects (Seaside work and smallCharts).
- Patterns matter as well as code; SUnit resource patterns help usability.
- Daniel's work on SqueakMap and mine on pluggable CM for the RB might help some convergence and Common Base Smalltalk issues.