Smalltalk Conferences in 2009

This document contains my reports of

- the ESUG conference in Brest, August 28 - September 4th, 2009 (and brief mention of the Camp Smalltalk before it)
- the virtual VASmalltalk User Group conference, April 21st - 22nd, 2009

I have put these two conference reports into a single document.

Style

'I' or 'my' refers to Niall Ross; speakers (other than myself) are referred to by name or in the third person. A question asked in or after a talk is prefixed by 'Q.' (sometimes I name the questioner; often I was too busy noting their question). A question not beginning with 'Q.' is a rhetorical question asked by the speaker (or is just my way of summarising their meaning).

Author's Disclaimer and Acknowledgements

These reports give my personal view. No view of any other person or organisation with which I am connected is expressed or implied. The talk descriptions were typed while I was trying to keep up with and understand what the speakers were saying, so may contain errors of fact or clarity. I apologise for any inaccuracies, and to any participants whose names or affiliations I failed to note down. If anyone spots errors or omissions, email me and corrections may be made. My thanks to the conference organisers and the speakers whose work gave me something to report.

CS14 and ESUG 17, Brest, Aug 28th - Sep 4th, 2008

Although you can fly direct to Brest, I reached it by flying to Paris and onward by train. After a brief tour of the adjacent buildings, I managed to locate the one where the Camp Smalltalkers were staying. :-)

Instantiations hosted a drinks party in the hall on Monday which was a pleasant companion to getting demos from the award entrants. We had a wine tasting evening on Tuesday, hosted by a majordomo who knew a vast amount about wine, albeit fewer of the English translations; perhaps some of the terms have no English translations. I learned more ways of classifying and assessing wine than I thought possible, and that you can make red wine from white grapes by leaving in the skins.

The prizes were announced during a tour of the Brest Rade (i.e. the maritime approaches) on Thursday, during which we saw the islet on which the ancient celts buried their dead. We also saw the French navy's nuclear submarine base and were warned not to take pictures of it - but only after we'd gone past it and had already taken pictures. :-)

Summary of Projects and Talks

I give the Camp Smalltalk 14 summary, then the ESUG activities reports (including the awards presentation ceremony). Next I summarise the conference talks, sorted into various categories:

- Smalltalk Development Environments
- Web Development Frameworks
- Coding and Testing Tools and Techniques
- Smalltalk and non-Smalltalk
- Managing Smalltalk Projects
- Applications and Experience Reports
- Reflection and Meta-Data

followed by the 10-minute talk track and Other Discussions. Talk slides are reachable from http://vst.ensm-douai.fr/ESUG2009Media#slides.

Camp Smalltalk 14

Camp Smalltalk 14 ran for Saturday and Sunday before the conference, and during the conference breaks, afternoons and some evenings of the five conference days. There were 25 people there by the first afternoon and several 10s of people attended first to last. All dialects had an active project (except Dolphin; Tim MacKinnon will port the SUnit latest to Dolphin).

I paired with Thorsten to port the latest version of Magritte. Thorsten also worked on porting a photo-management app to VW 7.7. This prompted us to pair on an SQLite interface for Glorp, essential now upcoming VW 7.7 uses Glorp for Store, since Thorsten had his local Store in SQLite. Thorsten made version one of the interface and I made Glorp mappings to SQLite's very limited time types. (I completed this afterwards; it's in VW 7.7.)

For the Custom Refactorings and Rewrite Editor Usability Project, I paired with Michael on his SUnitToo coverage tool which needed minor tweaks to fit into the RB in VW7.7. Michael then applied it to Seaside 2.9, seeing 70%+ coverage, which was what he expected. Based on his experience over many projects, Michael says you should never use code coverage as a management metric; you should use it to review and understand your code (his tool colours the code run/not run). To get 70% coverage via tests is easy. To get the last 30% is often easier via manual testing.

Most of the time I worked on porting the SUnit project's latest resource work - see my talk - to other dialects, helped by Michael's publish-to-Monticello VW utility, and by Yuri Mironenko and others.

A good many, like me, were working on tasks they later gave talks on, so I have integrated most remarks into those talks or into 'Other discussions'.

Cloud computing was being done, both by Ernest (see his Cloudfork talk) and by the PetroVR people doing monte carlo computations using Amazon services. Moose was being ported to Pharo while Jannick and others were using it to analyse software for code dependency cycles, etc.

Lucas, Julian et al were working on Seaside 2.9. Lucas ported 2.9 to Pharo, helped by Pharo's many bugfixes (meanwhile Stephane et al were working on Pharo adding to these). Julian was redesigning the debugger to run in a different Seaside session. Michael worked on a rich text editor (Javascript), usable in Seaside in more browsers than the current one. Philippe was generalising HTML-specific methods. Adriaan and Soemirno were polishing the Seaside-powered VastGoodies.com site.

Dale, Monty, James and Martin were working on Metacello, on importing Seaside into Maglev using Metacello, and on a Cocoa installer / UI for GemStone on MacOSX, an alternative to Gemstone's usual command-line.

ESUG Activities Reports

Conference Welcome and ESUG Activities Overview, Stephane Ducasse, Noury Bouraqadi

Stephane thanked the sponsors (see their logos on http://www.esug.org/conferences/16thinternationalsmalltalkjointconference2008); there was a time when he had to work hard to get any. Now the slide is covered with logos. He also thanked the local organisers, Loic and Alain (and gave them jars of caramel with salted butter, a delicacy). He showed a slide from Asterix - we have gone further than that famous Gaulish village to get here.

This is a second largest ESUG conference ever: 143 participants. There are tutorials on Advanced Seaside, SqueakNos, Aida - in parallel tracks to the talks. Yes, Stephane can't be in two places at once either: life is about choice. :-) The 'Show us your projects' session gives people 10 minutes each - exactly 10 minutes, so when the audience starts applauding, that means you time is up.

Stephane asked how many were attending their very first ESUG? Quite a few hands were raised. He urged us to use the student volunteers as the first point of contact for any questions about either the conference or the local area, and also urged us to help the students get in touch with the Smalltalk community by showing them our projects.

ESUG sponsors Smalltalk in various ways:

- ESUG can sponsor presentations of Smalltalk, i.e. pay travel expenses, etc. In international conferences, 8 papers in 2008 and 4 up to now (including best papers at OOPSLA and ECOOP!).
- Squeak Pharo runs on the iPhone; they paid John McIntosh to make it so.
- SummerTalk2009: SqueakOpenDBX, Safara (new text editor)
- ESUG sponsors student volunteers, so they can attend, meet the community, network.
- They sponsor other conferences: this year, they sponsor the 3rd conference in Argentina.
- Old ESUG conferences since 1993 are online.

- They have sponsored some books;
- ESUG sponsors free Seaside hosting (handled by netstyle.ch). http://www.seasidehosting.st. (will support pharo soon).

If you want to do any of this, ask: they will reply to let you know if they will sponsor.

Attending ESUG is how you sponsor all this. (Registering late is how you sponsor even more. :-) Next year, they are thinking of holding ESUG in Barcelona. Any interested would-be local organisers please contact them.

Speed dating at ESUG
The networking at ESUG is very ad-hoc; can we do anything about that? If you're shy and don't want to network, no. For the rest, perhaps we could set up a game or other interactive way of connecting. Or, could we get one minute each to say who we are, what company and our name and email - Stephane has often realised only 1/2 way through the conference that this badge name = that email. We did "who I am and why I'm here in one minute" over one lunch.

Smalltalk Awards Ceremony, Noury Bouraqadi
Awards advertise Smalltalk. Any work in Smalltalk (or related work, e.g. on a Smalltalk VM) is eligible, free, commercial, hobby. Think about your entries for 2010. Stephane wants next year's award entrants to do 5 minute videos, not (just) PDF documents.

There were 6 entries at Kothen in 2004, 9 at Brussels in 2005, 11 in Prague, 15 in Lugano, 20 in Amsterdam and 21 this year. VW had 8, Squeak 7, Pharo 3, VASmalltalk 2, VisualSmalltalk 1, GNUSmalltalk 2 (some entries were on multiple dialects). Argentina, France and Germany tied for first place as regards numbers from given countries. 10 were free, 11 licensed. 8 developers were academics, 7 were commercial and 5 were hobbyists. Entries included WebVelocity, Glamour, WikiServer&ObjectiveCBridge, Citezen, JavaConnect, Retrobjects, PetroVR, VisualGST, SqueakNOS, BoBus (interacting with city lines), NXTalk, HRworks, PhidgetLab, SqueakDBX, SqueakNxt, Iliad (web framework), PuissanceQuattre, SqueakSave, BizPro Launcher, SageTea Developer and EVA.

The winners were:

- 1st prize (500 euros): Retrobjects (commodore 64 emulation; let some of us play some well-remembered games)
- 2nd prize (300 euros): PhidgetLab: Etoys, control of sensors, robots, etc.
- 3rd prize (200 euros): Glamour (DIY browser in 5 minute; see Tudor's talk) and SqueakNOS

Books
Pharo by Example (adapts Squeak by example to Pharo plus bonus chapters), Andrés' books on Fundamentals of Smalltalk Programming Technique, Dynamic Web Development.

Wrap-up, Stephane Ducasse, ESUG
We applauded the local organisers.

ESUG is in Barcelona next year. The local organisers have translated Stephane's book into Spanish. Many teachers in Catalonia now use it and Stephane has a picture of the minister for Catalonia holding it.

Attending the conference is how you motivate the ESUG board to do the next one. Sponsors are also good not just for the money but also for the feeling that what is done here is wanted.

Anyone who attends the conference can be a member of ESUG. The board is elected every two years (required legally by ESUG articles of association or whatever). The new board is Alain Plantec, Damian Cassou, Michele Lanza, Marcus Denker, Serge Stinkwich, Noury Bouraqadi, Stephane Ducasse. They were elected by unanimous acclamation. Stephane invited people to give the new board feedback on how to improve ESUG over lunch. Meanwhile, he sought feedback on how to expand the community.

Andrew: we are a small community but very fragmented. There is a lot of duplication and porting. A while back there was an effort to revive the Smalltalk standards group. What happened to that? Can we make it happen? James: the widespread cross-dialect use of Seaside and Glorp is prompting a lot of standardisation. VW and Gemstone now both use Monticello to/from Seaside, so we can interchange code. Bernard observed that comp.lang.smalltalk is now no longer read by all in Smalltalk. Can we revive this or use planet Smalltalk or something?

Tim Mackinnon noted that it would be good if the preferred ways of publishing code for community and/or cross-dialect use were written up on a web page: a tutorial on how to commit code back to the community. Read Pharo by example. Come to the sprint.

Niall: go your local web or Ruby conference or usergroup and present WebVelocity or GLASS or similar. ESUG will sponsor you to go to such places. We are popular at Ruby conferences. Even at more general web conferences, Niall has found you can get an audience. Someone went to a a perl conference in Sweden and found that his talk on Seaside got interest.

Co-host? ECOOP suggested it but we did not want to join with a bunch of academics with type systems. By contrast, co-hosting with a Ruby or Python conference, or just being sufficiently near in time and space that some would attend, might work. That said, Stephane backed Niall's idea of individual Smalltalkers just going to a convenient usergroup or conference.

Internally, Seaside is a huge success. What should a benevolent dictator of the Seaside community now do? They should state a direction. Let's make the web page lively, do videos, etc. James is always looking for someone to do the podcast: contact Michael Lucas Smith or James Robertson if you have Seaside stuff to present (and non-VW, non-WV Smalltalks are fine).

Sherry pointed out that a good hosting story would help. If James and Ernest put the Amazon story into text then it can go on the website as a hosting story. Samuel offered them logos and that kind of thing helps.

Smalltalk Development Environments

VASmalltalk, John O'Keefe, Instantiations

(Stephane apologised for John's talk being on Friday morning three years running. John is too polite and never complained, "whereas some people complain really well! :-)"

In his talk, Georg quoted a Garter guru's assessment of programming languages which likened programming in Smalltalk to eating fillet mignon and drinking fine wine, while comparing other languages to less stylish menus. Thanks to the local organisers, we have also spent the week *literally* eating fine food and drinking fine wine; it's been a great week.

John left IBM two-and-a-half years ago, taking Smalltalk with him, and joined Instantiations. IBM wanted to drop Smalltalk and hired Instantiations to provide support, thence transferred the software to them. Instantiations has been shipping new versions of VASmalltalk since then.

VASmalltalk now has Seaside and its friends (scriptaculous, jQuery Core, jQueryUI, etc.). Version 8.0 had Seaside 2.9 alpha 3 plus some later code, Slime (an extension to RBSmalllint), etc. They have developed a porting layer which primarily serves the needs of Seaside but also helps other apps be ported from Squeak (and, to a degree, from other Smalltalk dialects). The layer will grow over time and portions of it will move into the base.

The Seaside they shipped is an alpha so API changes likely. Continuations are not yet fully supported so do `self show:` not `self wait:` (and so not `self call:` since it uses `wait:`).

The Vastgoodies.com site is in Seaside on VASmalltalk: it has had 200 uploads and many more downloads. The intent is that all VASmalltalk goodies will either be in Instantiations' distribution or will be on VastGoodies. Earlier this week, Adriaan demoed the NationaalSpaarfonds Seaside insurance site now in production.

Old VASmalltalk had the Envy browsers, Trailblazer, the Refactoring Browser, and the VAAssist browsers. John showed a VASmalltalk 7.5 browser, then the integrated 8.0 equivalent. Icons show if a tab has anything behind it. They are using or emulating native windows common controls. 8.0 did this for the main browsers, 8.1 will do the rest. They have a version graph code tool. Integer inspectors show decimal, hex, octal and binary. Inspector has a workspace pane where you can evaluate code and not lose it (or be prompted pointlessly to save it) as you navigate around. The bytecode browser is there - not too obviously so but it can be found (people who look for `bytecodeBrowser` may manage to locate it).

Niall Ross updated the Refactoring Browser and the SUnitBrowser for version 8.0 (and this week he ported SUnit 3.2 - and updated SUnitBrowser for it - for release in 8.1).

Web Services now supports doc-wrapped literals. Its documentation supplements “how to do web services in 10 minutes” with a second chapter “how to do web services in 3 hours” (useful ‘how to’ and ‘debugging tips’ information). Further chapters on serialisation and hosting a web service are in preparation.

The documentation has been revised. IBM had a lot of documentation but mislaid the means of supporting it circa the 6.0 release (2001) so they have completely reworked how it is managed. Now it is served to your web browser via a local server you install or from the instantiations website as you prefer. John showed an example of the old and the new look document. The new documentation tool has an ‘email’ button: press it to report anywhere in the documentation where you cannot understand what is said, cannot find what you need or see an actual error.

They also support windows themes on XP (John sees many Macs and few Windows laptops in this hall). They support import/export of parameters to support cross-platform development. They have added their internal set of examples to the delivered examples and made them more obvious.

Goodies are some new things and some old things moved out of the base product. Goodies that have external dependencies such as documentation (e.g. UML Designer) will be included in the product release. All others will be put on Vastgoodies.com.

What’s missing: OS/2 is officially dropped; John is now removing that code wherever he finds it. Martin Feldtmann has been given something in exchange and John suspects noone else cares. UTF-8 support is not there yet.

V8.0.1 (October 2009) will have Seaside 3.0 Beta and some of the porting extensions will move to the base. The Vastgoodies.com Tools will be included, so you can load goodies directly from the site to your repository. The release will have initial support for UTF-8. There will be a portable (i.e. emulated) progress view. Windows Server 2008 and Windows 7 will be supported (everything already works in Windows 7 except the installer - Com was hiccoughing there. `cdecl` calling convention support will be there. Install on Unix/Linux will be smoother: no symbolic links from csh to bash, X11 directory links handled, etc.

V8.0.2 (April 2010) will have Seaside flow. It will support Apache front-ending of Seaside. Candidates for inclusion include SST Servlet multipart forms. Ongoing stuff includes keeping up with Linux platforms (Ubuntu 9.04, Fedora Core 11) and performance improvements.

VM enhancements: they will look at 64 bit and incremental garbage collection. Windows services management, which currently uses a separate executable, will be moved back into the product.

Getting VASmalltalk: it used to be either a 30-day evaluation license that in fact never expired) or a development license. Development licenses are good - they bring in $$ - but now they will also offer:

- those who commit to an open-source project in, or ported to, VA, will get a perpetual non-commercial license (go to ‘company’ then ‘open-source’ on the website)
- an educational institution can have non-commercial license for teaching staff and students (go to ‘pricing’ and ‘how to purchase’ on the website)
- development builds for windows and linux will be available from their website (they will announce on their forum, and c.l.s, when a development build is available and what major things are in it)

John will send an email about the above to Stephane, who will send it round the ESUG mailing list.

Q. Any chance of a Macintosh version? There is no business case. John McIntosh will do it if someone will pay for it. Andrew Black noted that if it did not run on Mac he could not use it in his classes. John took that point and John McIntosh would be delighted to have a year of work (his very rough estimate of what was required) but in the absence of a business case or specific customer/external funding, it will not happen. Making it run on OSX is easier now that Mac uses intel but what should the GUI look like?

Q(Bernard) The APIs are based on Motif; any plans to change that? No, as noone asked for that to change; they asked for common controls support. For now, John is more focused on having a Seaside UI.

Q. Supporting Windows widgets is good but does it decouple you from the other platforms? As with the progress view, they are doing cross-platform emulation of these widgets. Their focus on Linux is as a server platform, not as a rich client platform because that is what their customers want.

Q. Seaside UI? John plans to expose more of their web environment in Seaside, but not something like web velocity.

ObjectStudio, Dirk Verleysen, Cincom
ObjectStudio was an enterprise-oriented Smalltalk from the start, unlike almost all other Smalltalks which were academic in origin. OS was file-based and initially was called Enfin. It was one of the first MS1995-certified apps.

Georg started the VW and OS integration project in 2004; now in OS8 you can work on the VisualWorks VM and be file-based or image-based. Cincom VW 7.7, OS 8.2 and Cincom WebVelocity 1.0 all run on the same VM. In these releases, we have a unicode VM, internationalisation (CLDR-based), Glorp and Store on Glorp, atomic loading from Store, better delays, new prereq engine, various tools enhancements, new icons and logo, Seaside 3.0.

OS 8.2 has a new look, a revamped modelling tool, a new mapping tool exploiting Glorp (first release only does active record but the next version will be whole Glorp) and unicode. He showed the old and new launcher (the hardhat will disappear - that change happened this week).

OS supports namespaces but they must be subnamespaces of ObjectStudio. OS handles change set notifications and will support announcements soon.

Dirk presented the modelling tool. You start with the use case explorer. There you define domain objects which you then elaborate in the CRC explorer (or not, as you wish). Then you can make interaction diagrams: they are just a documentation tool for now but may be used more formally later. The design explorer has submodels containing classes, which can be in one or more models. You work with the design explorer and/or the object diagrammer to edit the same data. From these tools, code is generated to files or to packages. You can also import existing classes into your models. Code generation can be customised by changing documented methods.

The mapping tool uses Glorp to map model classes to an SQL database. You can view database rows (e.g. show first 50 rows in the database for this class) to help you choose. Dirk showed the GUI.

OS8.2 supports LOB in Oracle and DB2, and stored procedures in those databases and in Sybase and ODBC. There is better host variable support For ODBC also support of stored procedures with input and output parameters and return values.

OS8.2 lets you drag-&-drop between tree views. The RB has an edit button for the controller (easier than looking in a long list on the launcher as the prior versions do). The source tab is coloured if the source of a method is of special OS type. A transformed source code tab shows where code is changed, e.g. where `os_add:` has replaced `add:`, and also is coloured if there has been a change.

ObjectStudio 8.3 will enhance modelling and mapping tools and will use DLLCC for GUI calls. Native tools may also be improved.

Q(Christian) GLORP migration tools? (Michael) GlorpMigration package supports migration between two different glorp schemas.

GemStone news, James Foster, Gemstone
James (QA, Seaside, Consulting, Training), Dale (Seaside lead), Martin (GBS lead), Monty (co-founder) are all here, so the cancellation of Smalltalk Solutions has had one good side-effect.

Gemstone 2.3 adds many more 64bit platforms. They find Solaris 64 very stable with very good performance (and they speak as fans of Linux).

The have improved mark/sweep performance etc., and UTF8 primitive encoding and other extended character handling. They are also moving from GemStone’s 1-based collection positioning to ANSI 0-based positioning, which will take time but they have found a way to let legacy and ANSI coexist.

In 2.4, they reduce round-trips when you have lots of logged-in sessions. It should also have a new Monticello, etc.

Version 3.0 will see a major VM rewrite. It will support DLL and C callout much better than at present. It will support Ruby, and as a prereq for that it will offer sandbox methods - methods in which the behaviour is different depending on the sender, not just the receiver. (This will be namespaces for methods, but not the same as, for example, Stephane's colleagues' selector-namespace papers.) Thus Ruby and Smalltalk methods will find the right methods in their class libraries despite existing in the same object space. James opened a window on a Maglev environment and looked at the Ruby gems that were running. (See Martin's talk on the technical challenges of this.)

You can run GemStone by downloading a VMWare in which it is installed. GLASS can stand for GemStone Linux Apache Seaside Smalltalk or GemStone Linux Aida Scribo Smalltalk. The Appliance is a VMWare download that provides GLASS all pre-installed. Alternatively, you can go straight to Mac and install native from the command line. He showed standard Seaside (the Seaside tests pages) in both these modes.

James has been learning Cocoa programming (Objective-C) to provide a wrapper so you can use GemStone without needing command-line setup. The GemStone.dmg disk image is 78 Mb; just open it, drag to your applications folder and start install. (Usual demo hiccough - 'address already in use' - probably left over from his Maglev presentation above; he killed it and proceeded). First launch opens a setup tool which tells you what you need to set up on Mac (by default sets 4Mb of shared memory but they recommend changing this to 1Gb!). James did this in his tool and in successive tool UIs created an 'ESUG' database and started it. He started a workspace, started a webserver, opened some squeak-like tools (and saw second demo hiccough - "I thought I fixed that last night").

An important component in using GemStone Seaside is Scaffolding, created by Gerhard Oberlin. He opened a Scaffolding page. Scaffolding is very like Magritte. You can create a class, give it a field, make a field required, of type text or timestamp (with 'current' button) etc., and similar commands. He stepped through setting up a simple blog post app. He showed the default UI for adding a post and showed the result in the list, then changed the Scaffolding value for the order of fields so the layout was better. He then created another class to let comments be added, changed the Scaffolding to have a 1-many relation between the post and comments classes, then added a comment to his post, etc.

Then he opened the web class browser and looked at the code that his Scaffolding operations had created: two domain classes and four views.

Metacello was created to let Dale manage Monticello packages across Squeak, Pharo and GemStone. Metacello is the package management system for Monticello. Monticello manipulates code. Metacello handles the prereqs and package groupings. Metacello will handle conditional loading of requirements that are common to all platforms, Squeak-specific, etc. Metacello will also help them work seamlessly with Monticello 2 and Monticello 1. He showed the list of all the packages in the GLASS distribution - a long hard-to-parse list. In Metacello, the list is more Smalltalk-like: its contents are reified into objects.

James browsed Dale's MetacelloTutorialProject. Specs are saved to methods, e.g.

```
version03
  ^self versionSpec packages:
    (self packageSpec
      add:'Example-Core-anon14';
      add: 'Example-AddOns-anon3';
      yourself)
```

A VersionSpec holds the usual stuff: description, doIts to evaluate before or after loading, etc., and of course the actual contents. This code is stored in another monticello package. Thus you can load the meta-data without loading the package. James loaded the Pier package meta-data without loading Pier. This lets you fix things you cannot load without having to load them in order to fix them. Any VersionSpec attribute can be conditionally modified.

They have added tools to the OmniBrowser to support this. See Dale's blog for more information about Metacello. See more at http://seaside.gemstone.com and http://programminggems.wordpress.com

Q. Metacello can handle Monticello 1 and 2; can it also handle other CM systems. Dale replied that on Saturday his answer would have been no but since he paired with Tudor in camp Smalltalk on Metacello he thinks that now it is yes.

Q. Cocoa application will be used in GemStone production releases? Although we would see it as for developers, the Cocoa part is just buttons running scripts and if clicking of the button starts the DB then that can be a production DB. But it is not supported at the moment. It exists to encourage people with Macs to try GemStone.

Q. Why would anyone not use GemStone? It does not have a GUI so in a client-server application it is the DB not the client. It is aimed at high-end industrial apps and historically the way to learn it was to be sent to a training week by an employer. It was priced only for this until recently but there is now a free for commercial use license, size-restricted. The tools are developing in Seaside but perhaps Web Velocity will make Scaffolding look weaker. There is no Refactoring Browser, etc.

Q(Tim) Some Ruby blog posts 8 months ago were refusing to believe Maglev could be so fast? What do they say now? There is no such thing as bad publicity. Some of their tests needed reexamination and of course you cannot expect to hold performance lead forever. That said, the standard ruby interpreter is buggy and slow and a stable fast VM has value. However persistence is where Maglev has real value. If you have to use an RDB for historical reasons, that is less important to you. If you are green fields, it can be very important. Monty added that they are getting a lot of good feedback. Maglev *is* that fast and the shared cache gives application performance, not just VM performance. They had expected to have it finished in a year and it will be overall two years for reasons that Martin will discuss in his talk. Where Smalltalk and Ruby match, they are there and very fast; where not, there are still some holes to fill.

Building Ruby on Smalltalk, Martin McClure, Gemstone
Maglev is a new ruby implementation. Question: why? Answer $ (except that being in Europe, the answer is e with bars). Ruby people have respect for Smalltalk and Java is a dirty word to them. Some Rubyists semi-convert to Smalltalk; anyone like that here? (Julian raised his hand.)

Martin then gave his 45 minute gemStone for dummies talk in 3 minutes (for really smart dummies). GemStone is a concurrent, persistent smalltalk implementation. Gemstone is fast, can handle 10000 commits per sec on good hardware and 2000 on ordinary hardware. Gemstone is Smalltalk all the way down. Gemstone offers transparent persistence for the web via GLASS.

Maglev aims to provide the same features to Ruby, plus a fast reliable VM.

First choice: use their current VM and extend to support Ruby or take an existing Ruby interpretation (e.g. rubinious that tried to use Smalltalk-style blue book) and write in ruby. They did the latter for Java ten years ago and the former for ruby today.

This project started 18 months ago and Martin will talk about 5 technical challenges.

Parsing Ruby. He showed the BNF for GemStone Smalltalk: two pages. Then he showed the Ruby grammar: a blank page. There is *no* published Ruby BNF. MRI (Matt's Ruby Implementation) uses a Yacc variant written in C to parse and it uses a hand-written file parse.y and it takes 130+ pages (all in the same font!). That is Ruby 1.8.6 and later people hand-edited that for later implementations. Ugh!

So what did GemStone do when Maglev started: they cheated. They had a simple parser (in C) to parse Ruby into IR which they passed to the compiler. They read Ruby from a source code file and push it to a Ruby MRI program that parses it. Then a C program, written by Brian Davis, a former-Smalltalker Rubyist, gets the abstract S expressions from the parsed Ruby. These are then got back into GemStone via a Ruby-front-end compiler (written in Smalltalk) that gets them into IR thence to compile.

Recently, they have cheated less. A Ruby parser written by Brian Davis (and optimised by GemStone) in Ruby (so they still use the old way to bootstrap) runs on Maglev and does the parse.

Arity challenges: in Ruby the number of parameters in the message send need not match the number expected by the implementation. You can pass a block with every message send (ampersand parameter). You can pass an array of keyword and parameter for variable arity. (In Ruby, Proc is the class of block and the term ‘block’ is only used for the syntactic construct.) In Smalltalk you never state the number of arguments because it is inherent and their VM never passes that info.

They looked at adding that info and decided that would penalise their Smalltalk so what did they do: cheat. The compiler synthesises bridge methods. `def foo(a, *b)` means that if I pass 5 args, the first will be `a` and the last 4 will become a 4 element array passed as `b`. The compiler compiles 16 methods foo, foo*, fo&, foo*&, foo:, foo:*, foo:&, foo::*, ... These are not legal Smalltalk methods but only the compiler cares about that. Their VM will accept anything in a method name.

Q.Lazy? They considered using DNU and creating the method but decided to generate all. You cannot analyse statically so any laziness would have to be done dynamically.

Q(Christian) these are hidden? They are real messages in the method dictionary but since Ruby has few tools noone notices. When they offer better tools, they would have to be hidden.

Invoking `some_object.foo(a,b,*c)` compiles a send to the selector `foo::*` which adapts the args and re-sends to actual message `foo`.

Q.Rubyists do this often? Yes - they do everything you don’t expect them to do a lot. Rubyists are Rubyists because they hate Java so they do everything the language lets them do a lot - meta-stuff and all - and eventually, when they’ve been Rubyists a while (but noone has been a Rubyist for long), they settle down a bit.

Coexistence of Ruby and Smalltalk: this causes issues of which one is immediate classes. In Smalltalk, the pointer to an immediate object encodes both the class and the state (Bool, Smallint, Character, ...). This is far more performant and avoids Java’s absurd ‘Int or int?’ problems. This fits Ruby which has SmallNum and BigNum like SmallInteger and LargeInteger. Now the two must be the same class but their names and methods and superclasses are different all the way up to Object.

They solved this by *not* cheating. They use environment-specific behaviour. A class has a method dictionary per class *per environment* (or you can share the method dictionary, or have the same compiled method in two dictionaries where the classes share behaviour). Senders know what environment they send in and bind to the appropriate environment implementation.

Q(Stephane) Did you look at Dave Simmons work, where he compiled the selector with the environment tag in the dictionary? GemStone did encode the environment into the symbol but they also encoded it into the send call.

They can compile the environment tag into the symbol but its identity will be the same in all environments.

Ruby allows per-instance behaviour: if you add a method to an instance you create a lightweight class (this pattern was originally shown by John Brant way back when). That was easy.

Per-instance variables: in Ruby, an instance variable is created by the act of assigning to it, whereas in Smalltalk it is created with all its instance variables, initial values nil. In Ruby an object has no instance variables and if it has an assignment to 'foo' in the next line it will still say false if you ask it in the line above whether it has instvar 'foo'. This is part of the language so Maglev has to endure this, but whenever possible they create fixed offset values initialised to 'I'm not here' value that will reply false if you ask them whether they exist.

Alas, they cannot do that always, so objects have fixed value instances and name-value pairs that must be searched. Sometimes it is just too hard to work out whether the class will have an instvar in the program. Instvars defined in the class definition are named, others are indexed.

Q. Javascript changes the pointer to the class when the class gets a new instvar; did you consider that? It is costly if you do these pointer switches a lot so we try to catch all we can upfront. Later we may as a maintenance operation do a sweep that fixes them up using this trick.

The started by passing 3,800 tests and are now passing almost 19,000 tests and there are 24,000 tests so they are making progress.

Pharo: a progressive innovative open-source Smalltalk environment for professional use, Stephane Ducasse, http://www.pharo-project.org
They want a stable system where bugs are fixed fast (professional) but that is still innovative (progressive). Researchers are paid to sell ideas so they need a language where they can try ideas fast and write papers about the results.

Stephane would tell his students "If you faint or vomit, that's normal". His students would ask, "Why does Morphic have 1000 methods?" and Stephane would reply "Good question!". Stephane told Lucas not to look at Morphic when Lucas started in his lectures - if he'd made him look at it, maybe Lucas would have left (or maybe he would have fixed morphic :-)).

Pharo removes MVC (all those `isMorphic` methods) eToys (almost all) full BlockClosures (from Eliot via John McIntosh) new UI look, TrueType character support, better Tools, many bugfixes and small improvements. The preferences have been cleaned up. The license was also sorted (MIT clean license) so Stephane will no longer get silly queries from the Linux conference, etc.

The dream is to have better tools. They want next generation refactoring (Eclipse is catching up with the RB), Smalllint++, better meta-object support (first class slots) and much infrastructure improvement: first class packages instead of all this string manipulation (maybe use Metacello).

Pharo should have an integration server: submit change, load into clean image, run standard tests, get report emailed to you next day or in 20 minutes.

Q(Christian) Namespaces? Goram had a solution years ago but people were not sure. Andreas has published a practical namespace package in SqueaKSource so perhaps it will happen.

Lucas has been doing all his Seaside development on Pharo since Pharo started. John has ported Pharo to the iPhone. Companies using it include netstyle, cmsbox and GemStone.

This is important to keep their Smalltalk flame burning inside. They know people who have been turned off by Squeak's lack of concern for good code. Stephane, Marcus et al were responsible for Squeak 3.9. They closed 700 bugs, and tried to clean the system without breaking unmaintained code. The nice thing about Mac is that it does not have all the old ports you'll see on a windows box. Trying to clean Squeak with Monticello was a pain.

They were called the random refactorer! Their reply is: what was the superclass of SyntaxError? It was StringMorph!!! If you know that makes no sense, you know they had to make these changes. Both Marcus and Stephane were burned. Roel was fixing things and they were never getting integrated so he went back to other work. Stephane and Marcus had to do this to keep their Smalltalk flame burning inside.

Pavel has created a 2Mb image (no UI, etc.) so this may be their core image. They have a tool to analyse dependencies, already applied to Morphic and being applied to much else.

Help is welcome: write tests, report bugs, submit fixes: see their slides for details.

There is a monticello inbox (much preferred to change sets). A fix that is not in the bug tracker does not exist. There will be Pharo sprints: Lille in October 2009 is the next one. There will be books: *Pharo by Example* is out (on Amazon) and there will be a Spanish translation soon (talk to Gabriela). Volume 2 (will be out soon) is not for newbies, it is for Stephane, so he can read code, write his understanding and have it for the future.

They closed by showing a slide of the huge number who signed the licence agreement (you must do this to have your fix integrated).

Q(Christian) Pharo, the name? They wanted 'Sapphire' but clashed so decided that the Pharo(s) lighthouse was a good symbol.

Q. Slots? Marcus showed that with his new compiler (that can have plugins) you can have first class instance variables without a performance penalty. The problem is not so much to do it as to sell it as research.

Q(James) Relationship between Pharo community and Squeak Board? No relation. The Squeak board was founded (by Stephane and others) as a do-ocracy: people who do stuff get a say. Pharo will be the same.

Q. What do users of Morphic do? BookMorph and Nebraska have been removed (neither were working) and EToys is mostly removed. Morphic has property dictionaries that make it hard to know what is safe to remove.

Q. You will use Monticello 2? As soon as it is stable. Lucas has students working on Monticello 2 stuff.

Q. Stay on the Squeak VM? Marcus will work with Eliot on the JIT, etc. Time will tell.

There was also discussion on Floats with Hernan and others: they wanted people to know when they were working with Floats as they are imprecise by definition. At the end Hernan said he understood their approach.

People asked about support. If you need support, you must have money.

Squeak on iPhone, john Macintosh
ESUG paid John to build the Squeak VM on the iPhone. John has 47 slides which is 50 too few to cover all the issues but may be enough. You can run any image if you can fit it in memory. John took a Pharo image of January 2009 and ripped out half before Stephane did so,. The iPhone 3G S has 256mb so you could have a 200mb image size but it takes time to move pages so that would make the app slow to start.

These figures are what we expected computers to be able to do many years ago and that is actually a lot of computing power. People are surprised when John runs a wiki server on his phone and they see instant response as the click on pages on their screen, but they should not be.

The squeak interface is hideous. The pharo people have a more business look. However he put two of David Smith's blobs into squeak and has an app that gets money (dozens every day).

John showed a Seaside Pier wiki/CMS running on his phone and, as he had remarked, responding well. John had looked at the three bridges to ObjectiveC and they were all incomplete, so he created his own, helped by SqueakProxy concepts from Avi. The plugin is in the Mac VM and you can do cocoa calls from it (and some people have been doing so). John listened to the Cincom talk on how they implemented their ObjectiveC bridge and used the `doesNotUnderstand:` pattern which is not efficient but he then made it possible to swap in a more efficient pattern.

The older bridges addressed 8 or 9 of the 18 types of data. John browsed what Python et al did and that educated him. He does not deal with *all* the complexity. You can have structures in structure such that you need a parser - John felt that whoever tried to use that could write the parser and give it to the community.

Where do objects live: Squeak/oop or ObjectiveC? John decided that objects live in one space or the other; no sharing.

John subclasses ProtoObject, so had to implement 37 methods just to make the debugger work. He discovered that noone else has done this. however Object has 400 methods in Squeak so it is a poor place to rely on `doesNotUnderstand:`.

Q(Andrew) get 37 methods into a smaller number? Yes, but one has to refactor the debugger - be my guest. :-) A protocol ‘debugger support’ would have made his life easier.

ObjectiveC calls are not the performance bottlenecks. In any iPhone app you write, look for performance problems in your Smalltalk code or in the ObjectiveC code.

The Squeak VM, being interpreted, wants to run all the time whereas the iPhone wants to go to sleep all the time, so John reworked the Pharo image to be more sleepful. You run this as a background pthread. Apple enforces thread-safety. Even properties are tied to a semaphore tied to the pthread so accessing across it raises a warning and you die. Deadlocks can also occur.

He showed the Smalltalk code that does the rotation of the screen as the tap and change page. When you split calls across multiple invocations you get multiple end-animation calls sent so to see the UI effect as you expect you need it to be a block of C code so he had to have a `rotateView:...` method.

```
infoViewString := 'WikiInfoView' asNSStringUTF8.
```

must eventually have

```
infoViewString release.
```

Objective C 2.0 provides garbage collection but it’s not available on the iPhone thanks to an evil Mac engineer who decided it was too much like work. Thus there are auto-release pools which have to be managed, which is a pain. He could have piggybacked off finalisation but that is a complex chore. His GStream plugin in eToys that he did last year *does* do the magic release, so it can be done, but today the release call is done as above.

```
ObjectiveC wrapWithAutoReleasePool: aBlock
```

We cannot have simultaneous use of two auto-release threads in two different Smalltalk processes. The monitor classes are not used in Squeak because they are broken but in Pharo someone fixed them so John uses them.

Suppose you tap a button on your iPhone and so we send `saeButtonWasTapped:`, a simple message send. However we need an

ObjectiveCSqueakProxy to do the callback. Sleep, then tap, ObjectiveC asks Squeak do you handle that? Squeak says yes so ObjectiveC locks a semaphore and waits. Squeak wakes from waiting on a Squeak semaphore and pulls the NSInvocation. It creates a messages send from its data and sends it. It send messages to the proxy. The proxy is an ObjectiveCProxy subclass of an interested ObjectiveC class. Thus you send a message the superclass understands or that a Smalltalk override of it in your proxy understands.

Eventually the Smalltalk thread gets back to waiting on its semaphore after replying to the UIKit thread. Hence you cannot do any UIKit work in your Smalltalk thread since the UI is waiting on your Smalltalk thread. In ObjectiveC, a protocol is a description of what messages the object supports. `self proxy addSigViaStrng...` you have to add the message your proxy will understand.

Asking a text view for its text object sounded safe in the documentation but that was a bug (the 23rd John suspects) that he reported because that string is freed so at the end of the block so if you reference it later you will reference a freed object so you die.

Alien is the latest greatest way to use FFI on the iPhone but there is no support. (Want support? Call John with your chequebook handy.)

Pier and Seaside have hundred (thousands?) of tests. SUnit tests for WikiServer UI: zero. he took 4 weeks (but could redo in a day) to create the 4 screens for WikiServer.

Apple uploads crash reports to Apple when you connect iPhone to iTunes, so when John says its stable and does not crash he has reason. A few non-Pier users have been blown away by its CMS and have sent John their praise. There are 6 Smalltalk apps in the Apple store (excepting the Squeak app itself which violates all the rules so you must email John to get it).

Q. Tim? Develop over VNC? If you think the Pharo browsers are slow you will not like doing that. John has a cycle that compiles and runs to the machine. You could use VNC for debugging if you get a walkback and have no idea why.

Q(Bernard) What is the ratio of Smalltalk to ObjectiveC code in your fractions calculator app? Very high ratio of Smalltalk to ObjectiveC: the latter has very little code except to put up the help view. Apple's UI auto-generates the 'on touch this icon open that app' code. WikiServer has code to do the view translations but all HTTP and rendering is done in Smalltalk.

NXTalk, Michael Haupt, University of Potsdam
The day he heard about Lego hardware he wanted to run Smalltalk on it. 256k Flash and 64k ram - that should be enough. So he looked for a student to implement it (in academia Michael is too busy writing papers and teaching).

You write your robot software in Squeak, where you have simulation of the h/w (not perfectly but the compiler can tell you when you do something wrong). The NXtalk VM is a Smalltalk VM implemented for this hardware. NXTalk is not a port of Squeak to NXT; the Squeak VM is too large. Underneath the NXTalk VM is NXOS (alternative h/w driver from a French university) and they use it only for the h/w drivers, not for its other features e.g. not for memory access.

The interpreter sits on NXOS and their own memory management and object table (optimised to store more in the object table than the objects themselves). Above this the image handles execution state, process data and scheduling (they wanted to implement as much as possible in Smalltalk).

NXTalk pointers are 16 bits wide as they are indexes to the object table, not true pointers. SmallIntegers have last bit 0 which is neutral to arithmetic and as fast to dereference as having the first bit be the tag. 1 means it's a true object. A 1 value there means you point into the object table and the object table rows are 10 bits of payload and 22 bits of pointer (only 256k, 64k, no need for all 32 bits to point). 5 of the 19 bits are GC mark, free memory indicator, moved flash to ram, indexable contents, etc.

Flash is (relatively) huge compared to RAM so most objects are store on Flash but everything that is being written to is better off in RAM as writing there is fast and writing to Flash must be done in chunks and is slow. Thus living things like the object table (as opposed to the persistent object table, which is in flash) and the heap (free space separates them - we hope and the first grows top down and the second bottom up to postpone when they clash.

When reference count of an object grows past its 5-bit store, we stop counting and instead rely on the mark-compact (this was Smalltalk-80's strategy). When objects have not been written to since the last full GC they are moved to Flash.

All bytecodes are 1 byte long, which buys them space but you cannot have more than 32 literals in a method, etc. ("The VM encourages good coding style.")

Q(Andres, Georg, etc.) discussion of whether encoding methods as small integers but they only have 6 bit integers and the smallest possible method would be one word so maybe not.

The interpreter is a simple bytecode interpreter, no JIT or suchlike, and all the dynamic stuff is objects on the heap. To avoid overeager GC, ref-zero objects are put back in a pool, not thrown away immediately.

They use green multithreaded scheduling, simple round robin except for writing processes - robots are often waiting for a sensor to reach a value and then response needs to happen fast. NXT does not provide interrupts so they had to fake them in the scheduler. Every X byte codes (a small number) the scheduler checks whether some waiting process needs to me made highest priority.

Q. NXT no interrupts? NXT by design decided to build it this way. the first thing the lego firmware does is disable interrupts. The sensors and actuators are managed by the main processor so the other processor must not interrupt (clock can interrupt). The brick has two processors, an ARM that does the main work and the other Atmel that manages external h/w such as their sensors, bluetooth, USB. A real system would probably allow interrupts, but they managed to fake it adequately.

The NXTalk system libraries are all the usual Smalltalk: classes, meta and reflection, exception handling, processes. Some things have been flattened e.g. you do not have Behaviour and Class separated, just one class NXTClass (since they do not need ClassDescription as it manages class categories and stuff that this runtime environment does not need). The library provides collections, streams, weak arrays (fun to implement :-) and hardware classes for sensors, buttons, motors, displays, event handling and USB and bluetooth. The USBManager can be started and will wait for commands. You can send it new classes to install in the image but not remote method invocation (but it could be implemented and they want remote debugging so will build it).

In Squeak, they have NXTPackageDescription which subclasses to various ways of cutting the code, all of which can be told to install themselves to the brick. The Squeak parser creates the AST. An RBNode visitor creates the NXTalk bytecodes.

He demoed the robot from farscape playing Tchaikovsky's 1812 overture and shutting itself down if it bumps into things (or the sensor fails - the ultrasound sensor on the lego brick is not robust). `setUp` enables the radar and touch sensors then initializes the tunes data. `spawnProcesses` forks off the driver, music, shutdown and other processes.

```
[true] whileTrue:
  [self drive ahead.
  radar waitTillBelow: 20.
  self backOff.
  self turn]
```

Q. How do you test your code? Upload it and run it. They want to do simulation in the Squeak image but need to complete more hardware mocks first. They will not be simulating all the hardware - that would take much time.

The standard NXTalk leaves 161 Kb (which is more than typical NXT apps). A context switch every thousand instructions = every 27 ms, taking 3 ms. The event handling latency is 4 ms with their default setting (no of instructions between scheduler checks). If an application allocates many small objects, it may reach exhaustion and GC kick in every 4 seconds.

So it works now but they want to do much more. Their bluetooth library is not yet right. The VM has a bug or two (and there are tests for them - later maybe there will be fixes). Better debugging is wanted.

The code is available from their squeaksource. They would be pleased if anyone ported it to other Smalltalks.

Don't download the VM from the webpage today; it is old. He will package the current version soon.

VisualGST, Gwenael Casaccio et al.
IDE for GNUSmalltalk. GNUSmalltalk has namespaces so the structure is namespaces, classes, protocols, methods. Below there is a the code tab and, if you wish, below that there is another pane with tabs for the transcript, the any compiler errors, any workspaces, etc. His inspector looks OK but he said he wanted to improve its design somewhat. Another button brings up an SUnit test runner that showed results and let you navigate to them. This is all written via a GTK binding so it runs on Windows, Mac and Linux.

He also showed a lego game in Cairo.

Q(Lucas) saving? You can save your files in the image or to a directory.

Web Development Frameworks

Aida Tutorial, Janko Mivšek, Eranova
Janko Mivšek, Smalltalker since 1995, founder of the company Eranova d.o.o., author of AIDA/Web web framework, maintainer of Swazoo web server and passionate contributor to Smalltalk community, is currently using Smalltalk and Aida for developing complex web-based systems to manage business processes in many industries, from gas, logistics to pharmaceutical.

For a change, the example is not a blog but a WeddingBook (i.e. a wedding register), containing WPersons and their spouses. I worked in VisualWorks development build for aug09.3. We created the model: WeddingBook and WPerson with appropriate instvars and accessors.

In Aida, presentation classes are subclasses of WebApplication, named for the model classes plus 'App' so we need WeddingBookApp, WPersonApp. This is the observer pattern: each object in the domain model has one instance of its corresponding app per session. You just create these classes (being careful to get their names right) and then register the root object of the application.

Guided by Janko, we created those classes and (corrected our typos and) saw, 'ERROR: view named #main does not exist' when we tried to reach http://localhost:8888/book. We logged into http://localhost:8888/admin, the admin page (default user admin and password password), then created `viewMain` on WeddingBookApp (not on WeddingBook) and worked on it to create a viable page. http://localhost:8888/book now showed our layout.

In Aida, you link not to the page but to the model object, which is why we must observe naming conventions in naming our classes <ModelClass>

and <ModelClass>App. We want our lists of names to be lists of the actual persons. We first created a method that displayed the names of the people in the wedding book using `addText:`, then commented that out, replacing it with `addLinkTo:text:` to display actual links to the person objects.

```
viewMain
  | e |
  e := WebElement new.
  e addTextH1: 'Wedding Book'.
  self observee persons do:
    [:each |
    "e addText: each name, ' ', each surname."
    e
      addLinkTo: each
      text: each name, ' ', each surname.
    each partner isNil ifFalse:
      [e addText: '   Married to: ', each partner name].
    e addBreak].
  self style pageFrameWith: e title: 'WeddingBook'.
```

Q. `pageFrameWith:`? Here, frame is no relation to HTML frame (ideally it would be renamed but it is an old Aida method).

We clicked the link, saw ‘ERROR: view named #main does not exist’ again, which prompted us to create `viewMain` on WPersonApp.

```
viewMain
  | e |
  e := WebElement new.
  e addTextH1: 'Person details'.
  e add: self detailsElement.
  self style pageFrameWith: e title: 'Person'.
```

Next we did MVC on the web. The *App class handles both the view and the controller, i.e. the actions. We added a menu aspect on partner.

```
  e addMenuAspect: #name
    collection: self observee book persons
    selectedToAspect: #partner
    of: self observee.
```

A menu appeared on the person page showing the list of persons. It did not do anything when the button is changed so we altered via an AJAX update which in our code is just `onChangePostAndUpdate: e.`

At this point in a real application, one would work on domain model to make it more precise and then develop the web pages further but as we were using it to learn Aida, we left it there to survey other features. We looked at Aida WebElement protocol ‘events’ and ‘events - ajax’ and similarly for WebForm, WebButton, etc.

An Aida object always has a URL and it is RESTful, the same all the time. Writing `preferedUrl` (that is the spelling Aida uses) on your model classes hints to Aida what the URL should be. A domain model is a network of objects and that maps naturally to the network of pages on the web. Such networks suggest graph-like navigation whereas in GUI apps tree-like navigation is more usual: open window, open subwindow, etc., then close subwindow to get back to super window thence to open another subwindow. Aida combines the two styles as required. Usually, the navigation is graph-like but there may be areas where it is tree-like.

```
(self ask: (WebDialog confirm: 'Delete'))
  ifTrue: [self observee delete].
```

is tree-like (and very new in Aida an still being perfected). Janko looked at `WebDialog>>buildConfirmation` to show us how this worked. The code that calls this is not a continuation; it is two processes, one of which waits on a semaphore for the completion of the other, when it receives a result. Thus treelike (or continuation-like) behaviour is achieved. Process context switch *and* web context switch occur on this return, which makes the code complex - sufficiently so that some aspects are still being tested.

Internationalisation was important for Aida from the very start. They use UTF-8 on the web and unicode in the image. Aida uses i18n for content and for text on the displayed pages. This can support (amongst other translation approaches) inline translation in pages.

He opened VisualWorks (this part is not yet ported to Pharo) and scrolled to the `viewCalendar` example and it opened in Slovenian, his computer-set language; it opened in our languages on our computers. (English is the default; there is no Russian translation at the moment so Yuri saw it in English.) Translations are stored on the class-side of the application and substitute for the English one which is written into the instance-side code.

Janko brought up an inspector on his session object, set it to development mode (`self setDevelopmentMode`) and switched on translation, then switched the language to French.

Aida runs on VisualWorks and ObjectStudio, on Squeak/Pharo and on GemStone GLASS. Swazoo runs on all these dialects as does Sport, which Aida uses for sockets, times and files. Aida/Web runs on these, which keep it portable, and the Aida/Scribo core on top of Aida/Web. The system comes with various apps such as the blog, the wiki, the website, the forum.

Future work: Aida 6.0 was released a week ago or so and is not yet in production. Aida 6.1 (out soon) will be a maintenance release. Aida 6.2 will finish translation and do more with contexts (i.e. `ask:` etc.).

```
(e addButtonText: 'Save')
  onSubmitDo: [self observee save]
```

Does the above break MVC (because you have actions inside your views)? MVC is there to avoid spaghetti code. Ajax would allow you to offer immediate feedback near the field, with submit failing if not validating.

Q.Aida is how old? It started in 1996, serving one Slovenian bookstore. An app delivered in 1998 is still being used. A logistical system has been using Aida for 9 years. Aida can scale in complexity.

Q. Aida used to create much garbage via some hard-to-release references. Does 6.0 do anything about this? When testing a benchmark, Janko created ten thousand sessions in a few seconds, creating 200Mb, so he did some work to sort that. However it does grow garbage and they have Aida GC functions which they call during the night. Also the new 6.0 contexts are not yet well optimised for garbage so that is an issue.

WebVelocity, Michael Lucas-Smith, Cincom
WebVelocity is a development environment that runs entirely in the web browser. The primary goal is to allow collaboration. Many projects use relational databases, a fact that may be strange since GemStone exists but is a fact of life. Mapping between objects and relational databases is hard and is the reason TopLink existed; its successor is Glorp.

Ruby on Rails demos building a blog server with an existing RDB so we will do the same. Michael started and chose an SQL platform. We could choose SQLServer, MySQL, Oracle, etc.: he chose PostgreSQL; the web page now shows what tables exist (he verified in pg3admin). Michael clicked and got a spotlight-window to create classes for tables, active record style, and did so. “We now have a working app.”

Scaffolding is provided in case you do not want to build the UI yourself or, more probably, want a bit of help with it. He showed the `objectClass` method showing which view a model-layer class is for. Then he opened the blog server page, showed the posts etc.

Next, he created the Comment class. Clicking the class pattern matches it to the Comments table, and there it is set up and with UI. Michael wrote a post, added comments to it. (Standard demo hiccough: he gave his post the same name as an earlier post so saw ‘conflicting values in rows’ warning).

There are some changes from standard RB behaviour. Once the code is compilable, it is compiled for you. Until it is compiled, it is in the errors. They found that non-hardcore-Smalltalkers were desperate to look at more than one method at once, so that is provided. If he modifies an inherited method, the change is saved to the subclass.

In Ruby on Rails, the moment it does not fit your needs, you are in the wilds. Here, when you go off the scaffolding you are in a mature development environment. Your app has Glorp, Seaside and Smalltalk.

A real Smalltalk development environment needs inspectors, debuggers, etc. Michael showed the inspector and “this strange button called ‘documentation’ - Smalltalkers are unfamiliar with this concept.” Mark Roberts, Cincom documentation guy, has written much documentation. It was very noticeable that their Smalltalker testers were all demanding to show source code whereas their non-smalltalker-testers were all demanding documentation. Michael showed how the environment shows both, with much text and graphics.

Q. This documentation is in the RB? Not yet. This documentation is done with DITA and XML so can be republished in PDF. Interactive documentation is on the agenda.

Searchlight has been integrated into this environment. You use the same tool to find the documentation and to find the actual method.

The undo/redo submenu lists all the edits you did on that method in this session: select to go back and forward again in your recent coding.

Michael then broke his program to show the debugger. He made a `renderContentOn:` DNU. We saw the typical Seaside walkback in the browser but when he clicked the debugger we saw a real debugger in the web browser, "and please note the speed of the response as I walk the stack, execute code, drop back and resume." Michael fixed the method in the debugger, restarted the context, resumed and we saw the correct web page again. (Applause!)

Because this is a web browser, two or more developers can connect to the same browser and work together. The CSS guy can be making the app look good while the DB guy is implementing it. If you see an error, you can send the URL to someone and they can join you and help you. (GemStone can save the error and re-present it. This is about live cooperative debugging. They may add an indication to show when two or more are interacting.)

SQLite is provided with WebVelocity so you get Store set up for you. There is a Google maps demo, a console widget, SeasideGlorp. Scaffolding provides you with an Atom feed, plus an XML and JASON output for your individual items. He went to the URLs http:// .../1.xml or 1.jason or posts.atom and saw these.

VW supports internationalisation so this does too; users can be hitting your website in multiple languages.

Michael looked at themes. A theme hotswaps the CSS to change your application's look and feel.

He opened a google map app for animals seen at various geographical locations. This was very easy. In the web development environment it is really easy to integrate video, audio, etc., so much so that it is a new paradigm for Smalltalkers.

He opened a tagged to-do list. Click on 'Smalltalk' to see all to-do tasks for 'Smalltalk'. Search for text, show completed items.

Like VW, this is supported on Mac, on Windows and on various Linux distros.

Michael invited feedback and discussion on where this should go in the future. Ernest wanted WV 1.1 to be headless on Linux. Someone else wanted Magritte; Thorsten and Niall are porting the latest version to VW so it will be there.

WV's aim is to make the tools better than the RB. This might have seemed impossible but recently the web browser has greatly improved - AJAX, JIT in the browser, etc. Michael reopened Seaside in Firefox and asked, could you make a web browser text editor that was so remarkable it was unremarkable. He opened one (on Smalltalk class documentation) that he hoped we could not tell the difference. He could word-wrap, syntax-colour, undo and redo (that will surprise the VW people present :-) and so on. Seaside has been pushing us towards an application development paradigm that could leave the GUI behind.

Q(Christian) How much do you code in WV? 99% when doing WV, when working on VW tools he uses VW GUI. WV 1.1 is being programmed in WV1.1.

Q(Annick) Collaborative production of documentation; Eclipse has concept of CheatSheets.? Michael noticed that every tester forgot that web browser windows have tabs until it was pointed it out, and then they had documentation open in one tab, code in another.

Seaside Update, Lucas Renggli and Julian Fitzell
Seaside has had 5 sprints in the last 12 months, one at Amsterdam after the ESUG and then 4 in Switzerland. Seaside 2.8.4 with bug fixes was released in June, Seaside 2.9 has had 4 alpha releases. There are now 73 packages: Flow (continuations) RSS, HTML5, etc.

Render request handling is more robust. Many methods moved from the session and the application to the request handler. jQuery support has been refactored so the stuff common between it and Scriptaculous is shared. There are 1,102 tests; good code coverage results from MLS were posted recently. The suite helps with porting; it took Michael a day to port the latest to VW. Many method comments have been added. It works in VW and Pharo (and Squeak but anyone just doing work to check and tighten that would be welcome).

The beta releases will be called 3.0. The goal for this ESUG's sprint (Friday and Saturday) will be to get 3.0b1 released.

Dynamic Web Development with Seaside, written by Stephane, David Shaffer, Lucas and Julian, is online: get it at book.seaside.st (a Pier site). You can comment on every page if you see bugs. There is a getting started section on each major platform (the book is dialect-agnostic). The book is free. (Stephane: "That does not mean that we do not accept money." :-)

Pier is extended to be a publishing engine.

Seaside Tutorial, Lucas Renggli
The code was the newest bleeding edge Seaside 2.9 and Pharo. (Go to code.google.com/p/seaside and submit a bug if you find any.) We started with a standard blog app: a simple page where you can add posts and etc.

The first exercise was on code unchanged from 2.8.

- restful URLs: make the ID of the post you are editing appear in the URL. Update the URL with the number of the post, e.g. http://localhost:8080/seaside/blog/2?_s=1awV51MQcz2t8ypv&_k=3V5YqexkzeSWw5VT where the 2 is the number of the post

```
updateUrl: aUrl
  super updateUrl: aUrl.
  aUrl addToPath:
    (blog keyForPost: post ifAbsent: [String new]).
```

- restful URLs: make the ID of the post you are editing appear in the URL instead of the usual long garbage, i.e. make a URL such as http://localhost:8080/seaside/blog/2 send you straight to editing post 2. We implement this in `initialRequest:` (so called because it is called when a request first arrives for a non-existing key).

```
initialRequest: aUrl
  super initialRequest: aUrl.
  self requestContext consumer nextIfPresentDo:
    [ :postId | self editPost: (blog postAt: postId)]
```

If the user types a number that does not exist that gets a debugger. We will instead provide an error message using the standard responder.

```
self requestContext consumer nextIfPresentDo:
    [:postId |
    self editPost:
      (blog
        postAt: postId
        ifAbsent:
          [^postId = 'create'
            ifTrue: [self createPost]
            ifFalse:
              [self requestContext responseGenerator
                                   notFound;
                                   respond]])]
```

When `initialRequest:` is called, you have your tree of initialized components. You are not in a rendering loop so you cannot render. (They also realised two days ago that you cannot call but they will fix that. Meanwhile you use `show:` and `answer:`, which is an explicit continuation - show component and when you have run do this block - that works on Smalltalk's without continuations). You will be able to do anything except render.

You can write just pure restful stuff but to use the render loop, continuations and all that you do need the _k... so it cannot be got rid of.

Session expiry: you could implement `initialRequest:` in the PostEditor instead of the BlogEditor and do things to have the session be longer when you're editing.

We changed the response handler to be a BTResponseHandler (BT = Blog Tutorial) in the blog configuration page. Then we overrode the `notFound` error to make the response page look better.

```
notFound
  self response
    notFound;
    contentType: WAMimeType textHtml;
    nextPutAll: (WARenderCanvas builder
      fullDocument: true;
"Full doc i.e. return whole structure with head tags
etc, not just body"
      rootBlock:
        [ :root |
        root title: 'Blog Post Not Found' .
        BTFiles default updateRoot: root] ;
      render: [ :html |
        self renderContentOn: html])

BTResponseHandler>>renderContentOn: html
  html div class: 'wrapper'; with:
    [html paragraph:
      'Sorry, the blog post you requested, number: ',
self request url path last seasideString, ' does not
seem to exist'.
      "self requestContext request url seasideString"
    html paragraph: 'Try another number' ]
```

This ended the first session. I was speaker in the parallel track in the next session, so could not catch the JQuery and handler parts of the tutorial.

GHPrintToWeb, Roland Wagener, Georg Heeg
How can a designer, who thinks of fonts, text and pictures in terms of desktop publishing, create a web page. Designers are artists with their own special language fonts, colours and perfect images. Asking them to express themselves in HTML is not nice for them - they think visually.

Designers use PhotoShop, Quark-Express, InDesign. All these tools are mainly targeted at print output. They do not export their work to web pages. On the web, such documents are created as PDF or Flash, not as an HTML page. Adobe page says ‘digital publishing’ but they mean Adobe Flash and PDF, not HTML. InDesign is the same: export to PDF or Flash, sure; export to HTML, no.

Ask a shop how they export to web. They export as a .png and give it to a web programmer who slices up the image and puts the parts of it the web programmer can use into the HTML page. “You’re not serious”, was Roland’s first response but he finds companies routinely do this!

Show such a designer DreamWeaver and they flee screaming. Can’t use all fonts, shouldn’t use all colours, images get squeezed and compressed, positioning - well I’m glad you asked ... , oh and learn CSS.

There are companies who have their own proprietary fonts, part of their marketing. Ask a designer at that company to use Helvetica and Arial and maybe one other if you’re good - and they are horrified.

Exporting to PDF and Flash is so easy but Flash is not available on an iPhone and not every user has a Flash player on their machine (companies sometimes limit it internally) and the catalogue is 20Mb - not everyone wants to download all that to look at the first page. PDF and Flash have proprietary formats, need a browser plug-in, are large files fetched all in one go, and Flash cannot embed Flash, but they can include fonts not available on the client's computer. HTML pages are separate giving shorter download times per page, and can embed Flash.

Once they were satisfied that designers would stay with the tools they knew, they thought about how they would help.

He showed an election poster (for the CDU in Germany). The CDU owns the font they use in the poster. He opened the 'export this document' widget in InDesign and looked at the formats. Some export formats are very good at preserving appearance, fonts, sizes, etc. They looked at the best export format IDML and the XML export and tried to combine them into an export to HTML. He opened VW (not yet the packaged version) and selected one of the exported files - the XML file. The system finds the IDML file and started converting. The progress bar gradually advanced through the 6-8 pages. This size conversion normally took 30 minutes and now takes a minute or so. The helper program they use (a Java application) to do image processing is the bottleneck, not VW, as they must process the image into the fonts and etc. (It also crashed during the demo and he had to redo the conversion).

Q. Different size pages? Yes, we must export the page several times and have a browser that switches the page to the right size. This works well and the user hardly notices that a page of the given size has been selected.

Q(Tudor) Use logical points (ems) instead of physical pixels? If you resize the browser using relative size, it is not quite right and the designers see it immediately. (They speak from experience.)

Q(Thorsten) do you end up with fonts or pixels? Pixels.

The similarities between the XML exports and the SVG files was their start point but while some browsers will show SVG directly (e.g. Firefox), most others will not or not well enough. They converted the SVG snippets to PNG files and used SeaBreeze to build the complete HTML object.

The first prototype was shown to a customer after two weeks. The customer had inserted errors into the test pages and they replicated perfectly including those errors, which impressed the customer greatly.

They found the InkSpace C++ application to convert SVG files but it took far too long to start on the command line, it was not available as a DLL and InkSpace 0.46 created wrong images in some very complex transparency situations.

Q(Annick) SVG package in Cincom OR, using Cairo? Roland thinks they looked at it and rejected it after testing. Some of the test documents have really tricky transparency situations that it did not handle perfectly.

They moved to the Java programme batik, connecting via JNIPort, and the speed was better. (InkSpace was better as a drawing tool.)

They export text as vector paths. This way, the client sees the correct fonts but this text is a .png. If the text must be editable, it would be better to use a web-friendly font, but designers are very picky and will throw you out of their office if you attempt to praise Helvetica. If the text must just be searchable, you can attach the original text to the page and arrange for it to be included in a search.

They wanted to use the converter as a web application. End-users wanted to stay inside their InDesign application. They objected to the lack of platform integration, of InDesign integration (but he has just received some Javascript code that solves that), etc.

They are preparing a MacOSX application that supports drag and drop of files (in 7.7 only) and integrates into the Mac menu (needed them to mess around with the MainMenu.mib file - they have English and German menus). Last week, he learnt how to trigger internationalisation in Mac by dragging the language labels in the list in the internationalisation preferences. VW7.7 provides an Objective-C Connect which lets them synchronise the VW and Mac internationalisation (but not dynamic during run-time).

InDesign knows about buttons and clicking on them to update other pictures on the page. He converted a very simple page where clicking on each of four buttons showed a different image beneath. The buttons on the web page worked and updated the image.

“We have more impressive examples but the web design company we built them with asked us to pay 15,000 euros for permission to show them to you.”

Q.Why many .png files, not just one per page? To allow web programming and to reduce apparent load time - the user sees that something has arrived.

Q(Annick) Resizing? The designer does not want resizing - it makes things look bad (less perfect than their intended design). They only use the browser size to determine the resolution to offer.

Q(Annick) Mac updates Java automatically; did you do any versioning / checking? Thanks for the hint. No, we did nothing and hope it works.

Q. Text v PNG? Headlines can be converted as the page loads but he thought not for whole page.

Private fonts cost money and you cannot just put them on the web if you want, so withholding fonts is in some cases very commercially useful.

seaBreeze, Karsten Kusche, Georg Heeg
Karsten learnt Smalltalk from Georg in Anhalt university 7 years ago,

where he was told that “Everything is an object”. In 2004, ESUG was in Anhalt and he mapped .ssp pages from Webtoolkit to VisualWaf. In VisualWaf you can use .ssp or the PageView API. Then he worked in the Bach project, where there were various ways to render the scanned text into HTML.

In seaBreeze, everything is an object, both HTML elements like div and span, and non-HTML elements like containers, conditions, U-tube elements, etc. SBElement has subclasses SBDivElement and SBContainerElement. SBElement knows about HTML properties, HTML structure knowledge, CSS properties and how to render HTML.

A collection container has its objects and a prototype of which of their values are of interest, and knows how to render these elements. The SBEditorElement double-dispatches to render any object. The scriptaculous file library is an object (who has every forgotten to add a file library and then seen nothing - many hands were raised including mine).

The windowSpec is a literal array describing these. Use the Web Painter to create sites. Elements know their structure and how they can be composed. Input elements know they cannot be added to Body elements unless a Form element is there to receive them. An element tree with icons to indicate e.g. whether the object has local CSS properties. The toolbar lets you embed object in others or extract objects from others, or duplicate them, delete them, etc. The Element Editor lets you assign CSS, change values, etc. Javascript is an object. You can specify which element to update at runtime which is more convenient than standard seaside coding style which has you having to know it at compile time. There’s a ‘use Ajax’ checkbox.

In seaBreeze, the scaffolding is very simple - the elements look as they did when you were editing them. You can import from ObjectStudio Designer or from InDesign (see Roland’s talk earlier today).

This runs on VW7.6 / Seaside 2.8, on VW7.7 / Seaside 3.0 and ObjectStudio 8.2 / Seaside 3.0. It sort of works in Pharo (some pairing welcome on one or two issues) and a feasibility study has confirmed it can be ported to VA.

Q.Reusable components? Yes, the application models are components.

Q. Deployed sites? The Heeg website and a picture gallery site.

Q. HTML5? What seaside supports, we support.

Karsten demoed. He created a new application ‘ESUG application’. It automatically created a component class and application and file library. he then built the counter demo. He added an anchor for the ++. It was red (no method in ApplicationModel); he clicked on it, was taken to the class and coded `number := number + 1.` He added text and gave it the aspect ‘number’ to display the current value. He showed the page, clicked and, as it was not initialised, got a walkback. He fixed and saw what we expected.

He then switched to 'use AJAX' and had the usual demo hiccough. He breakpointed and debugged into it and what he saw reminded him that to update an element you must tell its parent to update, not itself. The element had no parent so he selected both the ++ and the number and put them in a div. Then it updated. (Applause) That was an unusual demo hiccough that helped the speaker show off one of the features (embed into) he had been describing.

Iliad: a new web framework, Nicholas Petton, Sebastion Auder, University of Montpellier
Nicholas Petton is student at the university of Montpellier II. He is a co-developer of Aida/Web and author of the Iliad web framework. He talked about Iliad, a new web framework. See http://demo.iliad.bioskop.fr/browse http://iliad.bioskop.fr and also blog posts on the GNU Smalltalk site.

Why another web framework? Smalltalk has several web frameworks: Seaside, Aida, HttpView2 (little known, has a simple way to dispatch requests). These frameworks all have great ideas so they decided to reuse things from all these framework. They stole components from Seaside, elements from Aida and dispatch from HttpView2.

He showed the counter application in Iliad - very like Seaside. They wanted Ajax everywhere. Updating one counter does not need update of the whole page every time. He showed AJAX just updating the counter. In Seaside, the code for this is somewhat long. In Iliad, the code is simpler. In Iliad, `markDirty` in model accessors tell it what things to update.

```
e h2: count asString.
e anchor
  action: [self increase]
  text: '++'.
e space.
e anchor
  action: [self decrease]
  text: '--'.

increase
  counter := counter + 1
  self markDirty.

decrease
  counter := counter - 1
  self markDirty.
```

The URL is RESTful, unlike Seaside and so the back button is no longer an undo button. He showed a ToDo list in which he used URL style http://localhost:7777/examples/todo#pending, http://localhost:7777/examples/todo#completed and so the back button works as expected.

Q. Sometimes the user moves the mouse over the link without clicking, to understand the logic. Here, what they see will not be the same as what they see if they click. Discussion.

Iliad is on GNUSmalltalk, Pharo and Squeak. There is a Magritte package for Iliad. There is an Iliad-UI package.

Nicholas thanked Sebastian Bonzini who made Iliad 50 times faster.

Coding and Testing Tools and Techniques

(Invited talk) Script your browser in 15 minutes with Glamour, Tudor Girbe, Philipp Bunge

Tudor Girbe gave a superb talk on how to give talks last year. So no pressure on him. :-) Do you love objects? Hands were raised. (Tudor thinks geeks need more exercise so there were several question-hands calls in the talk. :-) ESUG loves objects and dynamic systems so it's a great place to present.

This was the master project of Philipp Bunge. Lucas Renggli and others helped.

There are many more objects than classes so the best friend of the Smalltalk developer is the inspector. Anyone know what its browser is?

You can click through many menus and windows of a generic browser to get what you want or you can build a dedicated browser that shows you what you want. The class browser is a dedicated view of the system. A generic browser is slower to use. A dedicated browser is difficult to build; usually, it intertwines the rendering logic with control flow.

```
browser = GLMTableLayoutBrowser new.
browser openOn: Metacello versions.
```

shows list of metacello project versions.

```
browser
  column: #versions;
  column: [:c | c row: #packages; row: #comment];
  shownOn: #versions using: [browser list].
```

shows a more interesting view.

```
browser
  shownOn: #packages
  from: #versions
  using:
    [browser text
      display: [:v | v spec packageSpecsInRowOrder];
      format: [:package | package name]]
```

This displayed but selecting did not show the comment. A final code change and we see the comment when we select.

```
browser
  shownOn: #comment
  from: #versions
  using: [browser text
           display: [:version | version description]].
browser openOn: MetacelloMetacelloProject versions.
```

So there we are: 5 minutes into the talk and we have created a version browser (applause).

This browser is in Squeak showing in Morphic. Maybe he wants to use it over the web. He executed

```
SGLDefaultComponent
  register: browser
  on: MetacelloMetacelloProject versions.
```

and then opened a web browser and saw the same browser in it as a Seaside component.

Q. Editable? You can edit, save changes and so on.

We still have 5 minutes left. So lets build a class browser.

```
browser
  column: #classCats;
  column: #classes;
  column: #categories;
  column: #methods.
browser
  shownOn: #classCats;
  using: [browser list].
browser
  shownOn: #classes;
  from: #classCats;
  using:
    [browser list display:
      [:classCat |
      (Smalltalk organization
        listAtCategoryNamed: classCat)
          collect: [:e | Smalltalk classNamed: e]]].
browser
  shownOn: #categories;
  from: #classes;
  using:
    [browser list display:
      [:classC | class organization categories].
```

Getting the methods from the category would be good but we need the class as well. If we just get from class we select method and do not see category and vice versa (he demoed). So we need

```
browser
  shownOn: #categories;
  from: #classes;
  from: #categories;
  using:
    [browser list
      display: [:class :category | class selectors]
      when: [ category isNil]
```

and we have two more `display:when:` calls for the cases `class isNil` and for neither class nor category being nil.

Q. Multi-selectable? Yes just specify that. (You get a collection, not a single item in return and must code for that; see below).

```
browser row: #navigation; row: #details.
browser shownOn: #navigation using: [browser custom: ..
```

`shownOn:` means please show from the selection (the 'port') so you can say #class->#entity, #classCat->#selection, etc.

```
browser
  sendToOutside: #selectedCategories from: #classes;
from:
```

Use #class->entity, #classCat->selection because you cannot access internal names directly for encapsulation).

```
browser
  shownOn: #navigation;
  using: [browser custom: self stNavigator].
browser
  shownOn: #details;
  from #navigation->#selectedClass;
  from: #navigation->#selectedMethod;
  using:
    [browser text
      title: 'Class Definition';
      display: [:class | class definition
      when: [:class | class notNil].
    browser text
      title: 'Source code';
      display:
        [:class :method | class sourceCodeAt: method]
      when: [:class | method notNil]].
```

but the order above in `using:` is wrong - select both method and class and get definition tab before source code tab - so swap it.

```
using:
  [browser text
    title: 'Source code';
    display:
      [:class :method | class sourceCodeAt: method]
    when: [:class | method notNil].
  browser text
    title: 'Class Definition';
    display: [:class | class definition]
    when: [:class | class notNil].
```

He demoed (applause).

Q. Update? Yes, that is a current draw back. If you change the source code in the other browser, this browser will not show a red code or similar. This is next to do.

Q. Can you render trees? Change `browser list` to `browser tree`.

Next, Tudor showed us the machinery that empowers this. We have named ports for the components of the upper navigation part, which we linked. Then we gradually gave them behaviour. Similarly for the lower details part and the links between them.

Objects can have many Ports. Panes can have many Ports. Ports can have many Transmission relations between them. A browser has many Panes and many Transmissions (subclasses Simple and Bundle). A Pane has many Presentations (subclass List and Tree, and also Browser, which is where `browser custom: ...` fits in). Browser subclasses are TableLayout and Finder (think the Mac finder). Presentation can have many Actions and that is how we edit and do other things. Actions can be rendered as key short cuts, menus, whatever.

Mondrian is a scripting engine for scripting visualisations (see Tudor's talks at prior ESUGs). Using

```
viewEdgesFrom: superclass
```

he showed the class hierarchy (Mondrian visual style: graphic, not indented text). For more complex visualisations, you can use Mondrian, Magritte or whatever.

He created a message tally tree as indented text, with number of calls and time in each call shown after the message printString.

```
  title: 'Messages';
  display: [:each | each sansOver: -1]
  when: ... "-1 delay before displaying calls"
```

This is like OmniBrowser but simpler to implement.

Next he looked at the Finder code, in the example of a small inspector. He can inspect Smalltalk, thence a class, thence its subclasses. But we want to see allSubclasses without spawning another pane so he lets the text editor's output from the left pane's Evaluator tab go to the right pane. Select left pane, type `sel withAllSubclasses` and see the result in the right pane, ready for further navigation.

Thus we can display the same browser in Morphic and in Seaside (thanks to Lucas - he uses JQuery).

Their VW prototype is in Widgetry because it was easier for them to script than Wrapper. Someone who knows wrapper is requested to do that bit.

(Pier doesn't run in Seaside 2.9 yet.) He showed Pier in his browser complete with the Mondrian visualisations.

This is built in 50 classes (but that includes 15 or 20 for the Morphic binding) with some 2300 methods.

Q. Browser text; the naming convention is? Discussion in which Tudor indicated the idea was to read the black-coloured code (see his slides).

Q. Dependencies? None except that if you want to use Mondrian you need Mondrian, if you want to use Magritte you must load Magritte, etc.

Q(Andres) The specification of a browser and its instance? Are the same thing; everything is done at run time and there is no spec instance, just the actual graphical representation existing at run time.

Tudor: “Morphic is a pane :-). Why do I have to create a class just to wrap a symbol. Widgetry is much better.

This is ANSI smalltalk and should port to any dialect. It is in Pharo/Squeak, and in VW (using Widgetry and not the latest version at the moment).

Specify, Simplify, Explore with ComplexValues, Christian Haider (smalltalkedVisuals GmbH) Thomas Schrader (counselling developer)
Value is an abstraction with no lifecycle; it is stateless and context-free. We use them every day.

- Immediate values are very pure and the compiler recognises this: 42 = 42 and 42 == 42, $a = $a and $a == $a. Literals are not identical but they are immutable.
- Other things are value-like but implemented as objects, e.g. ColorValue red.

A complex value may have much structure yet be a value you do not expect to change, e.g.

```
ChartText
  style: (Textstyle
    olor: (CmykColor
      cyan: 1
      magenta: 0.3 ...
    font: ...
```

The ComplexValue approach specifies such values:

```
ChartText
  style: (Textstyle ..)
  string: 'This is a text'
position: 5@10
```

is specified as

```
ChartText>>localSpecification
  <constant: #style class: #{Textstyle)>
  <constant: #string class: #{String)>
  <constant: #position class: #{Point)>
```

A ComplexValue is an immutable composite object without identity, and conversely a domain object that meets these criteria is a good candidate for a ComplexValue. Each such candidate is assigned a class-side `localSpecification` method.

Q. Existing instances when an optional attribute is added to your complex value? You care more about code than about existing values, since you usually do not hold onto them. That said, as in standard Smalltalk, you get the default value or nil for the added attribute in the existing instances.

Q(MLS) how do you compute a hash for these? Since MLS had beaten Andres in the race to ask this, Andres was asked for an answer. Discussion.

Values are created by sending a constructor to a class and then they never change. Values are always trees, never cycles, since you must put all values in at once. (Could add fixed-point operator to get round this but no cycles is the flavour they use.)

```
Trade
  time: <Time>
  value: <Number>
  volume: <Integer>
  settled: <Boolean>
```

can have a value

```
Trade
  time: 16:30
  value: 42.13
  volume: 200
  settled: true
```

Values make testing easier - you can just verify the test recreated the expected value - and it is relatively easy to translate them from Smalltalk to any external state e.g. a C struct.

He showed the ComplexValue definition for a VW `windowSpec` and for the VW Settings object, which is an XML string.

They find their systems become more reliable (values are immutable) and simpler (values are modelled). Large parts of systems can be modelled as values and at once they become simple and you can just forget about them, leaving you time to worry about the real objects - the collections you are manipulating and so on. Values are practical in Smalltalk; they are just code. A value prints itself in a literal form that can be re-evaluated.

Q(Andrew) (who paid $ to reserve the immutability bit in the Squeak VM, and so was very much in favour of this) How much work was this? Easy; it grew naturally out of their programming style. (Christian would like to talk to VM guys about enforcing the immutability.)

Q(Andres) use for changes? Yes, changes are pure values. There are two change systems in VW. There is a `className:` setter that is only there to create an instance and very dangerous to use any other way.

Q(MLS) creation methods are clean and nice, but in the `windowSpec` example, is a TextEditorSpec a ComplexValue given that we edit it with the UIPainter? This gave rise to discussion which I attempt to summarise: the immutable thing is the literal array spec (in the method created by saving in the UIPainter), not the objects edited by the painter while creating that spec. We debated whether you should see the two as different objects or as the same object in an epoch when it was editable, being created and not yet a ComplexValue, then in a later epoch when it was a ComplexValue.

Mutation Testing, Hernan Wilkinson, University of Buenos Aires
Hernan works for a Smalltalk consulting company and also for the University of Buenos Aires.

Ordinary tests verify the quality of your source code. Mutation testing verifies the quality of your tests.

- Step 1: create a mutant by applying a mutation operator to your basic code to create the mutation. A simple operator might change `and:` to `or:`, or - to +, in some method.
- Step 2: kill the mutant by running the test suite. If all the tests run, the mutant survives. If one test fails, the mutant dies.

A surviving mutant will normally reveal a case that is not tested. (It may of course reveal semantic equivalence between the normal and mutant code.)

```
self assert: (Card number: 123) = (Card number: 123)
```

will not reveal all errors in Card equality, whereas an extra line in the test

```
self deny: (Card number: 123) = (Card number: 789)
```

will catch many of those the prior line will not. In test-driven development, you may write tests that just test the positive cases. Running mutation testing after completing a test-driven development step can help improve the tests' coverage of negative cases.

MuTalk, the mutation tool, runs all the tests in the original, then applies one of various operators to the code and reruns tests. Operators includes:

- <= change to =
- `reject:` change to `select:`
- `ifFalse:` change to `false`
- remove exception handler operator

They are finding out by experiment which operators give good results.

Hernan opened the tool, generated 10 mutants, ran his test suite and saw that only two were killed. You display original code to mutated to show the mutation your tests did not kill, thus suggesting new tests you need, or new assertions in existing tests. He fixed the error that caused these failures, then ran the tool again, creating 12 mutants of which 11 were killed and 1 survived. The tool showed text suggesting the likely missing test assertion given the form of the mutant. He wrote that assertion and so killed the last mutant.

If your test suite fails on the original code, it would kill all your test cases trivially, so the tool warns you if you have any failures against the original.

Like Smalltalk, mutation testing is old. It is not widely used because reasonable complete test suites are rare, the right test-driven development processes are rare, and the tool support is rare. There is also the combinatorial explosion problem.

If you had to compile and link and run each mutant in turn for all possible cases, you would spend a thousand hours running a reasonable set of mutations, so people who do this in C++ or Java try to compile all the mutants at once with global variables activating specific mutant code. In Smalltalk, we don't have this problem. However Hernan has investigated ways to reduce the time by looking at four ways of running:

- run all tests
- run all tests that cover the changed code
- mutate only methods covered by existing tests
- mutate only covered methods and run only their covering tests

The last method is often twice as fast but the first one creates more mutants. The methods may not show the same results.

Q(Niall) Coverage itself may be changed by a mutant? Yes. The technique is an approximation.

He showed statistics comparing the basic algorithm to the coverage approach. The coverage is almost always faster but was slower for Pier, where the time to compute coverage was long enough to make the coverage approach slower than the basic approach.

Another speed-up technique is for MuTalk to look at test times. If the mutated test takes more than 3 times longer than the original, assume it will fail and treat that as a kill.

They would like to detect cases where two mutants are the same, e.g.
`a = b ifTrue:` mutated to `a = b ifFalse:` or `a ~= b ifTrue:`

A paper by K Wah (1995) claims that complex faults are linked to simple faults such that a test suite that catches all simple faults will detect most complex faults. So any fault is likely to have a set of mutants for which only a test case that detects that fault will kill them. Hernan showed statistics of the extra cases that his mutation testing added to a range of cases.

Q. Will this teach programmers to write better tests or just make them lazy? The latter kind of programmer can be treated as a mutant and killed. :-)

Q(Niall) discussion: test-driven development is about writing tests to do what you want/expect. This is a good complement? Hernan agreed; it adds tests for things not on the test-driven development path.

Q. Needs good coverage to work? If your tests do not cover your code, that is an problem in itself and should be addressed first.

MuTalk selects specific packages to mutate; they do not mutate the collection classes or the basic arithmetic since the system would crash. They test their code.

Q(Andres) Does one of your mutation operators replace a class reference with another class reference? Not yet, could do. Java and C++ people change public methods to private and etc. Which class would you mutate to - one in the same hierarchy, one with a high degree of polymorphism? Hernan does not test removing or adding return just because of the large number of mutants that would create; he will explore that.

Hernan started this less than a year ago and is still investigating. He finds it a powerful method and a great complement to test-driven development.

This is in Pharo on squeaksource under the MIT license. (Stephane: we'll be delighted to receive better tests for Pharo.)

Q(Bernard) Have you found any really obvious missing tests? Yes. Hernan showed a case in a real system where an obvious issue was not found. A colleague mentioned that the method helped him find a bug in his code.

Just-in-time resourcing: fast, flexible testing with SUnit and Friends, Niall Ross, Cincom
SUnit 3.2 will be released in VW7.7. Ports to VASmalltalk, Squeak and Smalltalk/X will be released at the same time. A port to Dolphin has been arranged and I am working to arrange GemStone and other dialect ports. In this talk, I looked at what had changed in SUnit 3.2 and why.

Once upon a time there were three classes - TestCase, TestSuite and TestResult - and then there was a fourth: TestResource, introduced nine years ago as an optimisation to avoid repeated `setUp` and `tearDown` of expensive unchanging state. However TestResource in 3.1 and earlier has several problems:

- The XP style is "Do it later" / "You won't need it". However, every resource a suite requires is set up before any test is run, and if one resource of one test fails in a suite of 15,000 tests then the entire run does nothing – not what you want to see when you return from making coffee, or come in the next morning.
- The XP style is to refactor. The rule is "first make it run, then make it right, last make it fast". When resources are used to optimise (their initial and most common purpose), that often means moving code from the `setUp` of a TestCase to the `setUp` of a TestResource. However resources do not understand the assertion protocol, so refactoring e.g.

  ```
  databaseSession := DBConnection connect.
  self assert: databaseSession isOnline
        description: 'The database is not online'.
  ```

  from the `setUp` of a TestCase that is getting slow to the `setUp` of a TestResource gets a DNU.

- Resources can compete with other resources, for example if a system can only connect to one database at a time but tests have been written that run against several databases, and each database is modelled by a resource. I coded the CompetingResource pattern: in SUnit 3.1 and earlier; this was not easy! Stephane Ducasse and Martin Kobetic also had patterns; they also found it not easy.

- It is even worse when resources rely on other resources. Tests (and resources) can need their resources set up in the order given and torn down in the reverse order, e.g. given

  ```
  MyTestCase class>>resources
    ^Array
        with: ConnectToDBResource
        with: AddTestDataToDBResource
  ```

  then the database connection resource must be set up before and torn down after the resource that adds and removes test data from that database. This was *not* handled in SUnit 3.1 and earlier.

In SUnit 3.2, a TestResource subclass has three possible states: not set up (nil value for singleton instance), failed set up (nonce value for singleton instance), succeeded set up (standard value for singleton instance).

- Resources are made available just-in-time. The first test that needs a resource prompts it to set itself up. Later tests that need it either see that it has been set up (by that first test) and so it is (assumed to be still) available, or that the first set up attempt failed and so it is (assumed to remain) unavailable.
- At the end of a test suite's run, resources used in that run are guaranteed to be torn down, as in 3.1. However in 3.2 a resource can also be reset (i.e. torn down) at anytime, e.g. in the `tearDown` code of a test case that uses it. The next test that needs the resource, seeing it is in not-set-up state, will set it up again. Thus resetting in a test's `tearDown` allows a developer to trade test performance for test isolation: the resource will be torn down and set up more often, but a test that fears it might corrupt a resource can tear it down, ensuring other tests get a clean version.
- Resources understand the same `assert:...` and `deny:...` protocol as tests do. TestResource>>setUp and >>isAvailable run inside the handler, as tests do. (The guaranteed tear down of resources at the end of a test suite's run does not run inside the handler: at that point, a call of `assert:` or similar is just more convenient protocol for the same exception-raising behaviour that would have been done by hand in SUnit 3.1.)
- Resource-processing is ordered. A test's resources are set up in the order in which the test presents them and torn down in the reverse order. A resource's resources are set up before it and torn down after it.

I walked through the code changes in 3.2 that do all this.

So once upon a time there were three classes - TestCase, TestSuite and TestResult - and then there was a fourth - TestResource - and now there is a fifth: TestAsserter, the abstract superclass of TestCase and TestResource, and of any user-created TestCase delegate class. That's enough! I am determined to keep SUnit small.

Any impact on Users?

- Anyone who overrides `isLogging` or `failureLog` needs to notice that these methods have moved to the class side. `logFailure:` is now on both class and instance sides. Niall asked if anyone did override these methods; no hands were raised.
- Profiling of tests can be affected. The time to run a specific `test...` method is unchanged. The time to run a test suite overall is also unchanged. If you profile a test's time in `runCase:`, it may or may not include resource set up time; resource set up time has been moved from the start of a suite's `run` to the start of (some) tests' `runCase:`.

Thus in SUnit 3.2, resources are in a somewhat better state than in SUnit 3.1. I thank Yuri Mironenko, Jan Vrany, Dale Heinrichs, James Foster and Tim MacKinnon for helping me port to Squeak, Smalltalk/X, Gemstone and Dolphin.

I then reviewed the aims of SUnit and various other test frameworks spawned from it.

- SUnit aims to be cross-dialect, backward-compatible and small (5 classes). It has various UIs such as RBSUnitExtensions (VW) SUnitBrowser (VASmalltalk), TestRunner (Squeak, etc.), etc., and add-ons such as SUnitResourcePatterns, SUnitXProcPatterns, etc.
- SUnitToo is a VW-specific, experimental framework spawned from SUnit where ideas, some of which may end up in SUnit, can be trialled. It is slightly larger (11 classes). You can run SUnit tests in SUnitToo by loading the SUnit-Bridge2SU2 utility.
- Assessments is a VW-specific, highly configurable framework of 40+ classes. It has transparent bridges configurable for SUnit, SUnitToo, etc.

There is also GemStone's test framework and various others.

Known issues and Future plans: SUnit will remain cross-dialect, backward-compatible, small. SUnit welcomes all ideas that fit with that goal. SUnit 3.2 is now finalised as a base point and with a view to its being in the upcoming release of VW7.7. In SUnit 3.3, some fixes are already in place:

- Tear down of test resources used by multiple other test resources has been made more robust in 3.3; in 3.2, multiple reliance by resources on a resource could still be torn down out of order.
- Circular references of resources are illegal of course, but if a developer accidentally creates such a case then in 3.2 they correctly fail to set up but still loop forever in tear down. 3.3 handles this better.

For SUnit 3.3 or 4.0 or later, here are some ideas:

- Within `runCase:`, we dispatch on the exception instead of having three nested `on:do:` calls.

- Exploiting this, an add-on utility can allow a specialised TestResult to be used to offer developers more outcome categories than just pass, fail, error (e.g. when a test that only makes sense on one OS is in a suite run on another, so is not run but should not appear as a failure).
- Look at making the `'test*' match: ... allSelectors` way of finding which methods to run as tests more easily overridable, more efficient (e.g. `begins:` is cheaper than expensive `match:`, and we can stop the search at TestCase) and able to accept pragma <test>.
- We would like to be able to tag tests as <takes a long time to run>, <expected to fail at the moment> and so on via self expectFailure, self noteLongTest, etc., or pragmas or whatever.
- In the various dialect's CMs, we would like to save a result (or a result constructor) with a test package - "Before saving I ran the tests and the result was ...". We also wish to compare results (c.f. Niall's comparison browser in VA).
- Rework `sunitChanged:` calls to make them easier to use in UI tools.

Q(Michael) The class-side `isAvailable` makes the resource available if it is not already so should not have an `is...` name? I have indeed seen cases where people have overridden the class-side `isAvailable` instead of the instance-side `isAvailable` in their TestResource subclasses precisely because of the confusion caused by this. Because the class-side should not be overridden, it *should* be possible to rename it without compromising backward compatibility since any subclasses that had problems would be revealing that they were wrongly coded and needed to be rewritten. (Whether users who met such problems would be happy is another matter.) A proposed new name is `beAvailable`.

(Michael also offered an alternative name for TestAsserter. After all, if a good name for a superclass of some Part... classes is Particle, what is a good name for the superclass of some Test... classes? I decided *not* to accept this suggestion. :-)

Smalltalk and non-Smalltalk

Why Smalltalk won the language shoot-out, Lucas Renggli and Tudor Girbe

Lucas browsed the method

```
findEmail:
  | rows |
  rows := SELECT email
           FROM users
           WHERE username = @(/\s/*(...))
  rows do: ...
```

in his Squeak image. Here we have Smalltalk, SQL and regular expressions all mixed together in a method. So if you had not seen this example, what architecture would you use to do that task?

Like Switzerland, the IT world speaks many different languages and these are not randomly distributed. People who speak French can visit the German part of Switzerland and (mostly) can communicate; we would like to do the same in the IT world. The same infrastructure is everywhere in Switzerland and ideally we would like the same in the IT world.

The system that allows that `findEmail:` code is called Helvetia. Lucas' talk was not about Helvetia; it is about what a language needs to allow that.

Minimal Syntax: transforming one language into another is a lot easier if the syntax is small. He showed java's huge list, Lisps two ASP nodes (Atom and List) and Smalltalk (Pharo has 10 ASP nodes). Parsing an AST and transforming it to the host AST is easier when this number is smaller.

Reflective Facilities: Smalltalk is very reflective but its meta-programming support is not that good. Expressions such as

```
Parser parseExpression aString, ' asRegex'.

MessageNode
  receiver: (LiteralNode value: aString)
  selector: #asRegex.
```

are not that clear to read.

Helvetia borrows from Lisp: "(',(aString) asRegex) where the red part is executed at runtime. Lucas recommends it be added to all Smalltalks to enhance their metaprogramming.

Q(Christian) Helvetia? A system that lets you build new languages and integrate them into the host. The purpose is to build a new Javascript. Some problems are solved better elsewhere than in Smalltalk so let Smalltalk be the host environment and get these others for their purposes.

Q. How did the comparison work out? Lisp was the only close competitor. Javascript is strong in metaprogramming but the AST is not available. Newspeak makes instVars more flexible but otherwise is Smalltalk for his purposes. OMeta was too slow and not flexible enough for transforming AST on the fly.

Cloudfork: cloud computing using Smalltalk, Ernest Micklei and Jan van de Sandt
Cloud computing means a dynamically-scalable computing technique that provides (often virtualised) resources. Google app is an example of a cloud that provides platform resources. Another cloud might provide software resources. A third example is infrastructure cloud computing: the cloud gives you databases or disks and you combine these.

Cloudfork is an open-source Smalltalk library that offers the cloud APIs. It was developed in Squeak and has been ported to VW and VASmalltalk. (The experience taught them how to write portable Smalltalk code.) Their first target was Amazon cloud computing services:

- S3: simple storage service
- SQS: simple queuing service (messaging)

- SimpleDB
- EC2: elastic computing cloud

All these services are built on the same architecture and business model. There is no entry charge or registration requirement. You just use the services and at the end of the month an amount is deducted from your credit card. Amazon had been a successful store for 10 years during which time it learned much which went into the design of these services. (Verner Voegels, CTO of Amazon gave a speech explaining the background: Amazon chose scalability over asset transactions since transactions do not scale. It is better to have data that is inconsistent for a few milliseconds than to have your system go down because distributed transactions don't scale. Their aim is to be basically available with soft state that is eventually consistent.) The cost is very reasonable, e.g. S3 is $0.15 per Gb per month.

They offer a SOAP interface and a simpler REST-based interface that 95% of their customers use, including Cloudfork.

S3 offers buckets in which you can store objects of up to 5Gb in size under a key - just like a Smalltalk dictionary. Each object can have meta-data with it; they use HTTP and so the page header becomes your meta-data. This is used for storage, for streaming (whether 20 people view the movie or 200,000 is fine by Amazon - the latter just means bigger bill at the end of the month), for software distribution, etc.

Their APIs are as you would expect (see slide for full details):

```
CFSimpleStorageService newWith: ...
```

SQS: Jan has a site where sometimes he has to do transcoding. Sometimes his sight is not busy and one processor is enough. At other times it is busy and he needs more. His code watches his private queue and sees when to request more resources. He would not use SQS to transfer money.

```
sqs := CFSimpleQueueService newWith: ...
```

SimpleDB is neither relational nor object-oriented; it is a key-value store. It is huge. Indexes and etc. are all handled automatically. To a Smalltalker it is just a very big dictionary. One account can have up to 100 domains. You can add up to 1,000,000,000 key-value items to a domain. You can have multi-valued attributes for your keys. You store strings only. Do not use it if you have complex joins or suchlike requirements. It is free until you go above a certain number of accesses per hour. A typical arrangement might be a testing domain and one or more production domains.

```
simpleDB := CFSimpleDB newWith: myCredentials.
simpleDB creadeDomain: 'esug-2009-domain'.
```

(Use SimpleDB protocol or Smalltalk dictionary protocol. Since the behaviour is not identical, they suggest to use SimpleDB protocol.)

They have used this for a couple of years now and find it very convenient. You create EC2 images for your Smalltalk application and just charge for use rather than making your customers pay up front. The smalltalk community could publish images and exchange ideas. GemStone could have a real opportunity here (Oracle already sells its databases as EC2 images).

Ernest then took over from Jan. Active Item Framework was build on top of SimpleDB. Ernest was doing rails applications before he returned to Smalltalk and he saw analogies between Rails' Active Record and what he wanted to do in SimpleDB.

The only thing SimpleDB can store is strings, so Active Item maps between objects and strings. Active Items have ids and let you save, find and delete your objects in SimpleDB. He wanted the same Domain Spec Language as in Rails (`belongsTo:`, `hasMany:`, etc.) and he has inheritance, etc. Domain Sharding maps between objects and domains.

(We then saw the most minimal demo hiccough ever; his Mac kept bouncing the 'update me' icon.) He opened a WebVelocity image with Cloudfork loaded. There is only one platform-specific package, called Cloudfork-VisualWorks-... or Cloudfork-VASmalltalk-... or ... . He looked at method `signStringSHA256:usingSecret:`. Every request has a timestamp so it is *very* important that your clock is right (and solving timezone in all Smalltalks was a fun task) otherwise all messages will fail.

Next, he looked at the CFActiveItemDescriber object. Ernest did a project for Flight School. They have many exams, many questions. The Question class has a `describe:` method which states that a Question has an integer 'weight' attribute, `ownsMany:` choices, etc. and returns an instance of CFActiveItemDescriber. The framework supports composition, aggregation, etc. A composition stores the subordinate objects inside the composer, flattening the structure for storage in SimpleDB.

He inspected an object and its description field, "and now you have to believe me that this object is in the cloud." He tried to get back the object and (after correcting his spelling) got it. Then he deleted it again. The object has an errors field as well as a description field: if e.g. the deletion failed then the 'errors' will tell him why.

The SimpleDB expression query is done by block syntax expansion similar to block.

He then went to a web browser and opened his AIBrow tool, a WebVelocity app that lets him explore his objects in SimpleDB. By running this app in an EC2 image close to your SimpleDB, you can minimise communication overhead.

Q(Thorsten) can the map-reduce functionality that Amazon offers could be used with Smalltalk? It sends code to Amazon who then evaluate it on each element of a big array but they do not offer Smalltalk as one of the languages in the array at the moment. Amazon would have to support it. We can interface to it of course.

Q(Stephane) Your reason for building this? Money. It's a commercial project and he is now building a flex Glare API and another interface.

Cloudfork is open-source under the MIT licence.

The future: they can complete the API, add APIs to other Amazon services and look at other providers (Google app is interesting but not Smalltalk, Microsoft is offering Azure, Salesforce is another company, Sun has a lot of white papers and have been bought by Oracle so will anything happen, and Amazon keep offering new features). What should they do?

Q. Why did you choose Amazon? We did not choose them, we found them. At that time, no others were offering this.

Amazon now offers load-balancing but do not yet offer sticky sessions so that is not good for Seaside. They now offer Flexible Payment Service (but restricted availability - some kind of legal issue).

Google bigtable is not accessible to Smalltalk images. If it were, they would look at it. Azure is in preview, not production yet.

The Eclipse 'deploy to Google App Engine' is nice for Java. A Cloudfork app engine could provide the same using Amazon features, thus evading the problem that the Google App Engine does not offer Smalltalk. Stephane offered ESUG sponsorship for this.

They could add Cloudfork support for Magritte and Pier as the persistence strategy. Continuous integration could be done in the cloud, treating it as your integration server and test framework.

A small API would let you store into Google App from Smalltalk.

Q. Encryption: SHA2 (deprecated) and SHA256 are offered. the APIs are simple to use.

Getting on the Island, James Robertson, Cincom
James had his demo hiccough at the start. The screen failed and he had to reboot. ("Conspiracy by Yann who was sorting pics back here with me - revenge for last year's ESUG photo of "Yann - man without Smalltalk").

Everyone listens to Industry misinterpretations, judging by the hands. How many people use Twitter or Facebook regularly. Few hands were raised - fewer than James has seen in some audiences. James finds "Industry Misinterpretations" useful: Ernest Micklei was the person who explained to him during one of his podcasts that Facebook and others have never heard of `asLowercase` so you must send all your stuff in lowercase. :-)

There are lots of APIs and libraries out there that we can use.

How many people have deployed web services on Smalltalk *not* on a corporate network. Not many hands went up - because when you ask your usual ISP they say they have perl, etc., etc. but not a Smalltalk image running.

James opened a browser and showed his interface to Facebook. He then executed code step by step to connect to Facebook. Facebook want you to spawn a browser and go to their page and login, not send the HTML emulating this. (After more demo hiccoughs - “Yann you’re going to pay for this” - “Ok, I’ll delete the photo”) he got logged in.

After login, authenticating through the browser, he executed the code to get a session (which Facebook says will last 4 hours but in fact will last till you close your browser). He browsed the list of his Facebook friends in an inspector. He published to Facebook from VW and showed in the browser. He executed code to upload some photos he took yesterday.

He has Twitter integration that works similarly. Facebook has FQL (Facebook Query Language) and James has an interface to it so you can do arbitrary queries and get back arbitrary objects. He did a query to get all the list of pictures uploaded by James Robertson.

The interface is small and can easily be ported to any other dialect.

Glare lets you talk from Smalltalk to Adobe Air. Ernest Micklei did this and James did the things Ernest left as “you can also do”. He showed this (Eclipse opened - not fast). White space at the end of the XML causes a problem (James googled the error he got and found someone a year ago who had met the same thing; how *he* diagnosed it James has no idea). Tudor tells James that Glare and Glamour should work fine together. He showed a Glare method finder that browsed Smalltalk methods with pattern matching just like the standard one. The specific code is very simple; Glare handles all the subtleties.

Most ISPs object to you running your server in their environment unless you buy a root. With this, you can use Amazon S3 server and run in the cloud. He executed some code to get three buckets from his Amazon S3 (Amazon bills him 2c/month because he uses so little). He opened a picture. James used the S3 browser to add things to amazon or get them back. Many other web services use S3 to store their stuff - watch for those S3.... references. So you don’t need a server farm for your Smalltalk - just use S3 as your back-end.

Language-Shifting Object in Inter-language Interoperability, Johan Brichau and Coen De Roover

This work uses JavaConnect in VisualWorks. He selected a Java string method and viewed its decompiled code in the RB.

```
zipfile := JavaWorld.java.util.zip.ZipFile new_String:
'foo.zip'.
```

We have a Smalltalk VM and a Java VM running and JavaConnect calls between the two. Often the low-level method that we call to in Java via the above can be represented in Smalltalk as something similar and vice versa.

Can we translate such a Java method into a Smalltalk method to improve performance? Static utility methods for example can be translated into Smalltalk and save many calls to the Java VM. Since one translated method will call other methods in Java, we need to translate some degree of transitive closure. The Java parser creates a Java parse tree and then it does not change in Java any more. We would like to save this structure of objects into Smalltalk so we can keep this object structure, not have to re-parse every time we start the Java VM (because the Java VM has no image i.e. no memory).

JavaConnect puts the classes in place for us so we only need to translate methods. We must translate expressions and statements. We can disregard generics, method and field visibility (Java code will respect this and we can break it anyway), lookup and overloading. We have problems with break and continue.

He had to use another machine for the talk so could not demo during it but demoed on his machine later. Meanwhile he showed slides on how it works. He showed a visitor pattern where translating the method that does the callback was a great performance gain, eliminating calls to the Java VM. (You can do `shiftToSmalltalk` and also `shiftToJava` if you want to translate it back again.)

The goal is to translate methods when needed and when possible. The translation is a visitor pattern that they are incrementally completing. managing the Java parse tree in Smalltalk is to-do.

(They tried to do it for Eclipse but their decompiler is throwing out bad code.)

http://www.info.ucl.ac.be/~jbrichau/javaconnect.html

Security on JIT VMs, Gerardo Richarte, community.corest.com/gera
He has been doing security for 15 years. Who needs security? 7 + hands. Who has ever heard of a Smalltalk security advisory. Is Smalltalk much better than the other languages as regards security - or are we years behind?

You are running an Smalltalk app on a box. The attacker provides content. The attacker makes this content execute code. Thus they escape security restrictions and access private information. EToys, Croquet, etc., can all involve mobile code.

We have a Smalltalk compiler the provides bytecodes and then a native compiler maps them to assembler so we talk abut nativizing VMs. He looked at VisualSmalltalk's VM and VisualWorks. In both, the Smalltalk stack is the native stack (whereas in GemStone the stack is a C stack but not the native stack). Instance variables are accessed directly and the Smalltalk contexts are stored on the native stack. In particular, return addresses are stored on the stack.

He demoed a tool that lets him write bytecode on the left side and see the assembler on the right side. (He also showed a VSE class browser tool that Leandro and Valeria wrote to show bytecodes in the code tab.) The attacker wants to manipulate the native stack to confuse the VM and so return from the message send not to its caller but to somewhere else. If the attacker can return wherever they choose, they can put their own program there and execute any code - any C, any assembler, whatever.

In the tool, he started writing code to manipulate the native stack. If there were a bug in the nativizer, there could be an incompatibility between what the nativizer wants to do and what the microprocessor understands.

He started by pushing something and returning. The assembler restores the stack pointer before returning so normally it would be pointing to nothing but in this VM the compiler does some optimising when it does not need to save anything so NoFrameProlog bytecode has no prologue and (very interesting) no epilogue. In Leandro's tool he coded to jump to 123456789 via

```
NoFrameProlog
PushSmallinteger 123456789
Return
```

The debugger showed a segfault as it was a poor choice of memory address to jump to. With a better choice of address, we can jump out of the VM with this. Can we do this without the NoFrameProlog bytecode.

```
DropToN 4
PopR
PushArgument1
Return
```

Before the return we have 3 pushes and the return address so we just pop 4 things out of the stack, push our return address, then let it return. We can also save the original return address in a register so we can return later. He coded this in Leandro's tool and demoed. He used the opcode for a breakpoint followed by 'jump to the content of register AA-X' code. We saved the return address in that register so this will break then continue. Run it, get to break, the windows are frozen, then do a continue and the windows resume normal operation. Replace the break opcode with some attacker code: that code will execute and you will see *nothing*.

The Cincom VM cannot be attacked this way because they have no Drop... Instead he tried

```
DropToN 3
PopR
PushR
Return
```

When the send returns, the stack will be unbalanced so further actions will overwrite stuff on the stack. He can clean things from the stack not in balance with what he put there. By declaring arguments incongruent with the arity of the selector he can bring up a debugger showing pops with the same trick - break, continue - as before. The Cincom stack was tougher but he did it.

If the attacker can transfer and execute a compiled method then they can escape the VM and access the OS and if the OS does not provide storage isolation and/or has privilege bugs, the attacker can get very far.

Securing Smalltalk could be attempted by reachability, sandboxing or a verifier. Reachability from within Smalltalk is very hard to achieve. Sandboxing is like reachability in the VM. In .net, every variable and method is decorated with the privileges needed to use it and gets checked but we also need to check primitive parameters and we also need to check nativizing as we have seen above.

A nativizing verifier must check that the method arity agrees with the pushes, drops, whatever.

How else could we escape the VM?

```
SmallInteger>>writeMemory
  lodArgument1
  StoreInstance1
  return
```

SmallIntegers have no instvars so this can be an arbitrary memory read or write.

We've heard Smalltalk is slow, Smalltalk is ugly, whatever - we will start hearing Smalltalk is insecure.

He documents bytecodes; he needs to understand how every bytecode in every variation was nativized. The best documentation is one you can debug and trace through. So all his documentation is written in Smalltalk. Each Bytecode is a class with methods showing how it works. He opened TestRunner and ran the tests.

Q(MLS) How do you jump to a more sophisticated attack? If the memory is marked as non-executable so how do you get the bytecodes to make a sophisticated attack? His company markets tools that let you evade the non-executable issue. Michael noted that an ExternalInterface method can be insecure - it can go anywhere.

Q. How long to discover this attack? Two days but it took him much longer to learn about e.g. the VSE VM.

The attacker looks for an error - usually a warning that something is not as expected. They do not need to know everything about the machine, just enough.

Bytecode Documentation, Gerardo Richarte
He had shown us (in the talk above) how unbalanced stack pushes and pops were insecure. In this 5 minute session, he showed one of his tools. He documented all the bytecodes in VisualSmalltalk so he wanted to test whether his documentation works. He hooked into the call chain to call back to Smalltalk when the nativizer is about to nativize his method. He loaded every method that has been nativized by the VM. He did arrow up

and arrow down and saw some more methods being nativized. Further moving around produced no more; all the image currently needs has been nativized.

He started the test (had the usual demo hiccough - it crashed- and restarted and reran) and saw that it worked. He flushed the memory of all nativized memory, then ran and got his break point and continued.

Managing Smalltalk Projects

Project Planning, Tim MacKinnon, Iterex

Tim started in OTI and is now a consultant offering Agile consulting. He usually joins a group to do cool stuff, then discovers they have planning problems and addresses them as well.

Tim is very grateful to Smalltalk for producing good stuff, including stuff better than Microsoft project. Adding "this task takes half a day" to this task takes half a year" makes no sense, especially when these numbers are put into MS Project. He quoted Tom DeMarco: "Software Engineering is an idea whose time has come and gone. Software development will always be experimental."

Agile is an umbrella term for Scrum (Ken Schwaber), eXtreme Programming (Kent Beck), etc. Scrum can be very flaccid - and a lot of it is. Smalltalk's eXtreme Programmers by contrast do a lot of what XP said should be done.

Planning up-front is not cheap. Agile flattens the cost curve so that decisions can be deferred. Where does planning - incremental planning - fit into an agile project.

In a commercial context, there is usually a bit of upfront planning and analysis mainly to secure budget for a period. Whole team (and customer) involvement in this is good. Whiteboarding and brainstorming end up with some ideas of what to do and what is required. Scrum calls these 'epics' and they may be represented as flip charts. A lower level output may be story cards. These items may go away, never be used, etc., so little time should be spent on them.

The incremental planning is forecasting - like weather forecasting and unlikely to be more reliable.

Velocity is a buzz word that has unfortunate connotations. XP talked about load factor (you estimate 1 week but it could be five times that) but that was not taken up and velocity was used instead but really it is range - how far will you go in the next iteration. Some consultants give you planning cards with their logo - but fingers are as easy to count on. Tim says estimate in 'ideal days'.

The number of 'ideal days' work you actually did in an iteration is your velocity. The planning game is to pick stories, then discard and so on till you iteration meets your velocity. At the end of each week you know your actual velocity for that week.

Scrum puts these numbers into burndown but that is a scientific way to think about it and most customers (and most developers) do not understand it. Burnup is better. Tim's diagram showed blue (work yet to start) yellow (work in progress) and green (work done) histograms. In the standup meeting, people put their magnetic avatars onto their tasks on the board and explain how it has moved or why it has not. Tim's software displays each day's daily standup report as a green/yellow/blue histogram. He showed us a succession of weekly histograms from a real team.

It is a lot of work to convert an 'epic' into 150 stories, so Tim has in the past asked people to offer weekly estimation for it but he has gone off that. His software helps record this breakdown.

Q(Andrew) if you cannot break it down? Then you need to do more storyboarding.

Kanban is a popular new method: let's do no estimation. Tim has not yet used it. In some Japanese gardens where only so many visitors could enter at a time, entrants were given cards, to return when they left. When all the cards had been handed out, people started queuing until someone left and their card could be reused. Kanban works by queuing. Only so many cards can be assigned to a team at a time. Whenever a space becomes free, the customer can be asked what they now want most urgently that team can handle. Tim's problem with kanban is the siloing: the method assumes you have several teams, each with a capacity. There is also the issue of when to split a card.

Q(Niall) comparison with Rob Ven's method / Georg Heeg method in which, expressing it in the above terms, the 'epic' is a smalltalk model and the cards are the parts of that model and/or the test cases created from exercising it. This led to discussion and Niall and Tim resumed later.

Q(Yann) has not found a software that replaces paper? Tim agrees; his tool is lightweight and only supplements paper and whiteboards.

When Flexibility backfires, Yann Monclair
Yann went to university in Brest where he got hooked on Smalltalk. Now he works in the Kapital project, formerly in infrastructure, latterly in financials. (Kapital was described by Georg in yesterday's talk, by Yann last year and by me in ESUG 2004, so need not be described again.)

Flexible: able to adapt when external changes occur. Knowing your flexibility: understanding it is the first requirement. If you are working with others, you also need to be able to explain it. If you can visualize it, you can explain it to a wider circle: senior managers, clients, etc.

Flexibility is technical, architectural and political. Technical: how flexibility impacts your code - which solution you implement. Architects review how code will evolve over time. Lastly, how does it fit in with the strategy of your team, how you are viewed by your customers, etc.

Where to be flexible, when to be flexible, how to be flexible? You can do it anywhere in Smalltalk but that can get you into a mess. Will you be flexible inside your application or in your interactions with others? Do you want to be flexible when debugging, when prototyping?

There are times when it is not good to be flexible. Suppose that on Monday morning your application is throwing error messages. Suppose the cause is that an upstream system is sending you extra characters because that team changed something. You call the upstream team and they say "well, we need to get senior manager sign off and he's on holiday and we can't bounce our systems so 16 hours." You can fix it in Smalltalk in 10 minutes. So you do that and send emails. Result: you look to the outside world as if *you* caused the problem.

Another example is

```
self instVarAt: indexOfInsvar put: aValue.
```

That is flexible but not such a good idea in many cases.

Yann defined `flexibility not = strict.` Strict means enforcing rules. (Yann is French so he's great at getting round rules.) Strictness can help flexibility. Yann gave an example: an old mail server was very tolerant of invalid 'from' fields whereas the new server was not, so Yann made his application enforce strict output format, adding a valid 'from' if absent. By contrast, when an input system started supplying invalid input characters that a downstream system would not accept, and the input system were slow to fix it, Yann cleaned between their input and his output. Flexible Input format, Strict Output Format let Smalltalk solve problems that were none of its making.

Be comfortable with what you are doing: if 'flexibility' makes you afraid to change your code, that is wrong: that's why `instVarAt:` is to be avoided. Sometimes it is OK to `perform: aString` because there is a pattern. Do it not because you can but because you must (Yann 's tweak of Georg's presentation): be reasonable.

Q(Andres) other examples of `perform:`? We sometimes generate methods like `..._London_secondFloor_<timestamp>`.

QAnnick) Being generic and being flexible differ how? Generic: works for every scenario. Flexible: can be adapted to the latest scenario.

Q(Andres) Cost of being flexible versus maintainable? Phone rings, bug reported, fix it immediately but these fixes accumulate, making the system slower to understand, so you must work them back into the main system.

Q(Bernard) I more often see too flexible than too strict; your view? Yes. We did an experiment trapping all unhandled errors. The log files became unreadable and noone saw errors until much later. Strict has its upside.

Q(MLS) I've seen defensive programming (nil checking, error handling) where, instead of seeing why the bad value came in, they just protect against it? Yann has seen the same: `== true` instead of `isTrue:` because they're not sure.

Applications and Experience Reports

Modelling with Smock, Testing with PicUnit, V. Verbeque, Alain Plantec, Thales

Thales is Brest-located company who have been using Smalltalk for 20 years. Smalltalk is considered exotic so their projects had to succeed to let them stay in Smalltalk. They develop code for hardware devices (military hardware and etc.). They develop their MMI in Smalltalk translating to real-time operating systems via a C translator. Since 2002, they have both modelled and tested systems in Smalltalk.

They chose Smalltalk 20 years ago. It had really powerful features (and was the only multi-platform solution back then) and TNI who are located nearby were able to give them feedback and mentoring. Their target OS is a middleware item. They do applications for police, customs, pollution monitoring, maritime traffic surveillance, search and rescue control, submarine detection, electronic warfare defence. Their defence applications are deployed on the MOSAIC architecture which contains a number of PIC components.

They model what a given PIC must or can do. An important role of these is it demonstrate to potential customers, and internally in the company. They use SMOCK (the S is smalltalk) to mock-up something, thus create a better specification of what is needed, then model (model-driven engineering) therefore design the architecture and software of the real component. Then they test it with PicUnit.

He handed over to Alain who explained SMOCK. They have evolved to an formalised approach with components, architecture, packages and contracts. All the software they produce obeys its contract which defines the services it provides. Their subcontractors can see what their components provide and require.

Inspired by the Jaguar-component modelling method, they created SCM (their variant of CCM) for modelling components.

```
MyNameSpace
  define: #MyServices
  super: nil
  with: #(serviceOne serviceTwo)
```

These models map to real components that exist in their systems so they can mock-up real configurations and connect them to real UI. Letting the customer play with a model of their intended system is very useful.

Q(Christian) how do you see that the real component fits the requirement? Smock is just to model requirements. PicUnit is to test that the real component matches the requirement. One cannot test everything of course.

SMOCK includes a cost model for pricing scenarios. A mock-up takes from a few days to a month to create.

Alain then handed back the microphone. Suppose navigation equipment talks to a PIC NAV which talks to a PIC LSE which talks to a PIC RAD which talks to RAD equipment, all PICs also talking to a DB, all using CORBA. PicUnit must validate each component and also their communication with each other.

PicUnit can simulate other components to the component under test. It can be the receiver and initiator of the CORBA messages and it can read and write to a database. PicUnit uses SUnit, DLLCC and OT/DST from VisualWorks. Above these, they implement their own packages (see his architecture slide). UIs let them define what messages will be sent and expected. A code tool in the RB (RBSunitExtensions-like with additions) shows test results and front-ends more detailed displays of them.

A Mock is a Smalltalk bundle that represents a PIC. Mock-LSE had 10 functional packages and a packages of tests. These Smalltalk Mocks are not small but of course, there is much inheritance, reuse of patterns, etc.

They would like to integrate PicUnit and Smock so that a model defined in Smock would automatically generate tests in PicUnit.

Smalltalk is powerful. Its debugger is much better than in any language they have considered. It can do anything. “Its a Swiss army knife”. When they train people, it is tricky to explain what the image is since other languages do not have this concept. The first step into Smalltalk is a psychological hurdle. Because it is not a language people know, they have to teach Smalltalk to newbies.

Q(Annick) You know DDS? They know about OMG DDS. They have evaluated it and do not see DDL as useful.

Q. When do you replace the C generation with Smalltalk? Static typing is demanded in the real products for security reasons. Some security checks use type inference. Top management will not allow that to be dropped anytime soon.

Multicore playground: how can we get the most out of our most modern CPUs, Arden Thomas
Arden is the product manager for Cincom Smalltalk. That includes keeping the product moving forward and exploiting areas where the rest of the world is moving forward.

Today it is an unusual machine that has only a single core.

Arden heard a presentation from Grace Murray-Hopper who worked in computers when mechanical relays could be blocked by moths and ‘we have a bug’ was the literal diagnosis. She showed what a nanosecond was by showing the length of wire an electron traversed in that time. She explained that when a farmer can't pull up a stump with a horse, he does not go back to the barn for a bigger horse, he goes back to get another horse. Using multi-core makes sense.

Concurrency is not easy. The area has been studied for decades, producing poor solutions and OK solutions and no perfect solutions. AMD and Intel have spent money in universities to study concurrency; they build multi-core CPUs and wish they and their customers knew how to use them.

As the power to use concurrency grows linearly, the complexity grows exponentially. So how do we use concurrency?

We can run multiple applications: that we do today. It is easy and means there is very little contention but it is not very powerful. You probably cannot use the multiple applications to solve a single problem.

We can run multiple process threads in a single application: these threads can work on a single set of objects. This allows more power on that set of objects but also the danger of unsynchronised access, object contention. If one thread goes wrong, does the entire application fail?

VW and OS8 have green threads, not native threads. Green threads were very effective (more than native) for modelling producer/consumer tasks. VW offers Process, Semaphore, Promise (evaluate forked and return value when fork completes) and SharedQueue.

We can run multiple process threads within a VM: one thread does GC, one JIT, one execution. Then in a single-threaded application we could use multiple cores. Some of the engineers ask whether the contention between these threads might be so great as to nullify the advantages? Answering that question will require building a VM that uses this to see what then happens, so this is a costly and long-term strategy.

Coordinate multiple applications: this would need only small changes to your individual applications to work. This would only solve that subset of problems where there were sizeable independent pieces.

Coordinate multiple applications sharing memory: this can solve a broader set of problems. (Niall: Kapital uses this approach). There is a distributed GC problem to solve and the usual concurrency problems.

Arden has pushed to have this put on the roadmap. Asking our engineers to solve concurrency is like asking them to cure cancer "and we deliver in nine months." The great thing about Smalltalk in that the language does not add complexity to the problem space. We can solve difficult problems. The CST engineering team have some good ideas on how to attack this problem. They have created Polycephaly ("It's not contagious: it means having many heads.") You start up multiple headless images ("should it mean having many no-heads") and hand out tasks and get results. This is *not* a general solution to the concurrency problem. It will solve some specific problems effectively. It will be in preview in a later release of VW.

Arden has experience with code to get market data and show graphs of stocks, etc. Files from Nasdaq, AMEX, NYSE, 24,000 mutual funds and etc. The sequential code to load all this information from the web took 114 seconds to execute. He did some experiments:

- He used the #promise: method and polycephaly to evaluate loading each type of file separately. promise begins work immediately. When you ask a promise for its value, the promise returns its value or waits until the fork has completed and then gives it.

  Arden started 5 promises, then requested 5 for their value. Arden's promise used polycephaly. His machine was a dual-core CPU and the result took 80 seconds, an improvement but not startling. Checking times, he found that the mutual funds (24,000 of them) took much longer than the others.

- He looked at how he got mutual funds and saw he was chunking them into 26 alphabetical calls (all starting 'A', all starting 'B', ...). He made each letter an independent task and assigned them to drones. With 3 drones, loading the mutual funds ran in 30 seconds and the sweet spot was running with 8 drones, which loaded mutual funds in 17.5 seconds.

  These times include startup of drone VMs. In fact, in the real world drone images will exist and be waiting, ready for work so the real world improvements will be greater. Arden deliberately included these times so his figures had 'no cheating' in their comparisons. These times also include object transport time which could also be optimised in a real system.

Q. New VM copies object space? No, start up of drone copies original image.

- One of the engineers pointed out to Arden that his HTTP calls were not synchronous. An HTTP call is a call, a long wait to get a reply and then a short-time processing of the results. Arden overlapped the waits by forking all the http calls in green threads for the mutual funds. This used 13 green threads and that reduced the load time to 15 seconds in a single image. (26 threads made less than a second difference.)
- Finally, he combined both, green threads for mutual funds and that being one of the polycephalic loads of the 5 types of fund. He got an overall time of 35 seconds.

Lessons learned: always measure where the bottlenecks are, do not guess. There are many ways of handling concurrency; do not get into "everything is a nail because I have this hammer" mode. Launching headless drones was faster than he had expected, i.e. noticeably faster than headfull.

Q(Georg) what operating system? Windows Vista.

Mechanisms for handling common issues make things easier. He ensured that shutting down his main image closed all of them - easier than searching for all the drones if things get into a mess. You can also ensure that if a local process starts a drone and then is terminated, the drone is shut down.

Remote errors can be retrieved locally: you can arrange for a green thread of your main image to get the exception raised in a drone image.

Q.Size taken up by drones? For Arden's experiment, this was never an issue in today's computing environment.

Q. Did you do any testing that was not I/O bound (i.e. would green threads have worked as well)? Arden has not yet tested an entirely green-thread solution.

Q. Erlang-style actors? Some disconnect between that style and Smalltalk.

Andres has done some similar experiments. When you have many green threads in VW, there is a native stack and it will not have enough space and contexts will be removed and restored, so just increasing the native stack size can show a 10-fold improvement in many-green-thread experiments. When running many images, be aware that GC is memory-intensive and if all your images do GC at once the cache may not be large enough and things will get slow.

Q. A customer used 650 images and use the same approach to testing. Their tests took an hour to run, so one of their engineers used OpenTalk to run the tests on 8 machines and that let them run their whole suite every integration.

Q(Georg) I researched dual-processors in Smalltalk-80 when ESUG was last in Brest. It is very important how you distribute the work and how many activities you do before you synchronise. Also, have you looked at the GemStone architecture?

I mentioned the Kapital architecture as an example of shared memory solution.

Smalltalk is Hot, Georg Heeg, STIC
In French and then in English, Georg said he had been in Brest twice. His first ever OO conference was in the town hall (and everyone spoke in French and very fast).

STIC are friends and sponsors of ESUG. STIC organises Smalltalk Solutions. STIC's members are Cincom, Instantiations and GemStone, plus Georg Heeg (director) and Cherniak software (treasurer).

Georg showed the oldest mention of Smalltalk. It was in August 1978 and was *not* the famous byte magazine cover of 1981 but a cover showing the 'Pascal triangle' with a small Isle of Smalltalk in the picture. Georg saw the island of Mont St Michel and thought it was very like the Isle of Smalltalk. (Is Smalltalk a tourist attraction? :-) The text described it as "a snow-white island rising like an ivory tower surrounded by shark-infested waters. Here we find the fantastic kingdom of Smalltalk where great and magical things happen."

What are these magical things? Well, Smalltalk maps the way domain experts think into classes and objects one-to-one (and so find the flaws in their thinking, when present). Smalltalk creates a virtual world. Other IT people takes the computer way of thinking - states, data structure, processes - and map that into the domain. When Georg started as a computer student, you could rent computers by the hour or programmers by the month for similar sums. Smalltalk takes the external world seriously. Examples:

- Last year, Georg solved a 130-year-old problem by modelling the existing historical knowledge in Smalltalk.
- Other web applications think of HTML as a sequence of characters. Seaside takes HTML as a sequence of message sends. SeaBreeze takes them as a sequence of objects.

Smalltalk is active. WebVelocity is a better Ruby-on-Rails, out this week. GemStone has GLASS, VA has Seaside, new Pharo, new Dolphin version.

Robert Martin's talk "What killed Smalltalk could kill Ruby too" claimed, "It's so easy to make a mess in Smalltalk" but his argument is quite wrong. The truth is, "It's so easy to see where there is a mess in Smalltalk". Robert thinks there are more messes in Smalltalk. Wrong: there are far fewer, but you can see them. Trying seeing them in Java or C++ is much harder. (James Robertson also responded to his talk - see his blog and his "Industry Misinterpretation". Georg concludes that "Totgesagte leben länger" - there is life in the old dog yet.

Bob the Builder's motto was: "Yes, we can." But Georg does not want to use that, nor the 'yes we can' of the Obama campaign because the actual convention picture showed signs of 'Change we can believe in", not "yes we can", but that too is not the slogan he wants. The ParcPlace logo was "Design for Change". *Unforeseeable* surprises happen and those are the changes that Georg is concerned about.

Currency exchange systems now effect transactions on a timescale of seconds. Get it wrong and you can lose a lot of money quickly. But JPMC made $2.1 billion in the first quarter of this year: financial crisis, what financial crisis? JPMC got the derivatives risk award in 2008 and 2009. Spiegel claimed that JPMC invented Bistro "the product of the credit bubble". He showed the JPMC definition of it - not easy to grasp quickly. :-) JPMC's Kapital system has received a ComputerWorld honours award and much other praise. Kapital went to the traders and mapped their concepts into a system and that is where its success came from.

Georg has a very old left-handed cup with Smalltalk logo from parcplace (left-handed because if you turn it round it shows an ObjectWorks for C++ logo). Gartner's Mark Driver says programming in Smalltalk is like eating filet mignon sole and drinking fine wine while programming in Java is more downmarket, culinary-wise. Smalltalk is cool again.

The ability to model everything empowers you but only you have a goal.

Q(Stephane) Smalltalk Solutions? Because of so many commercial companies restricting travel as a cost measure, they decided to support ESUG and the argentine conference. Georg hopes to have a Smalltalk Solution in 2010. ESUG would like a student volunteer programme at Smalltalk Solutions. Could we have a phone conference?

Q(Janko) So Smalltalk is responsible for the financial crisis? :-) (Niall) No, they could see the danger earlier in Kapital, whereas the others could not see and ward it off. :-) Yuri M: yes, companies using spreadsheets made a mess and, for the same reason did not see it in time. :-) (Andres) and let's blame people, not languages. :-)

Reflection and Meta-Data

Thomas Kowark, Robert Hirschfeld and Michael Haupt, Object-Relational Mapping with SqueakSave

SqueakSave generates SQL queries. SqueakSave is 20% slower than GLORP on the 007 benchmark, mainly (they thought) due to Glorp's caching mechanism. SqueakSave is faster than Glorp for complex joins due to the Glorp framework having some overhead. Essentially if everything is sucked in from the DB, Glorp is twice as fast. If everything is in memory, Squeak is faster because it is just traversing, whereas Glorp is doing some cache maintenance at the same time.

SqueakSave is easy to set up and use. They need to work on SqueakDBX usage, eager loading and performance optimisation.

Q.Why use this not Glorp? Glorp needs configuration. (Glorp ActiveRecord work makes it easier to generate that config from any data you have.)

Q.(Lucas) Change classes or multiplicities? Class new field will be added transparently, multiplicity is harder and will need config.

Experiments with pro-active declarative meta-programming, Veronique Uquillas Gomez

The source code of an application provides a great deal of meta-information about it. It would be nice to see how far the current implementation matches the design. SOUL is a logic-oriented Smalltalk-related language for reasoning about programs. The aim is co-evolution: keep design and code in synch. Currently, SOUL is developer-driven and relies on reasoning at snapshot times, not continuously.

Smalltalk Open Unification Language (SOUL) runs in Squeak and Pharo (and VW IIRC)

```
class(?class) if
  member(?class,[Smalltalk allClasses])
```

Here, the code in the block is Smalltalk. The rest of the code is prolog-like.

Programming And Reasoning About Changes Using Time: Parachut. They use the syntax of SOUL but only forward-chaining, not the backward-chaining that SOUL supports. They use forward-chaining to provide immediate feedback. Changes to the code are the events that trigger problem solving.

She showed an example: a rule that reimplementing the = method requires reimplementing hash and vice versa.

The SOUL Clause Browser (very like the class browser in the layout of its panes) is implemented in VW. The Pulsar class pattern concerns a class that keeps growing then shrinking then growing again. A simple predicate catches classes that are the same size at two points and twice as big in size at a point between.

Q(Andrew Black) Just add one method and remove it, would that be caught? The inference engine determines at what point to make the check.

Smalltalk Metaprogramming supports Probabilistic Analysis, Dave Mason, Ryerson University
He researches program reliability. He seeks paths through the program that correspond to subsets of the input domain. He therefore wants to do path discovery.

The number of paths in infinite. Depth-first may disappear into infinity. Breadth-first may spend much time on trivial paths. Heuristics that assume that e.g. loops always go round ten times and so on are used. If we know the probabilities of the inputs, we can attack the problem that way.

We need a probability density function (PDF) for the domain: a probability value for each point in the domain. This can come from data, e.g. phone records, or from a domain expert.

A pure approach might use a continuation-based model. All the parameters are in the PDF. Anywhere we have a decision point, we can snapshot at that point of the continuation and put it on a priority queue for the true case and for the false case. Then we take the first continuation off the queue and proceed. When a branch terminates, we then take the next continuation off the queue and proceed it, and so on. Many paths may be in partial execution.

The result is that paths complete in frequency order. The first path to complete is the highest probability path through the program and so on. The next path to complete is the next most probable, and so on. We know how probable because we know the input probability.

The above is very pure solution but it has many continuations in play at a time and this is not cheap. A more practical approach uses monte carlo statistics. We generate a random input point, check it is not already in the domain, and execute by just adding the correct decision point result (true or false) at each decision point. This is less certain to be most probable but you can choose the ‘random’ point to help on this.

Smalltalk has the key technologies this needs: dynamic types, first class booleans (rarer than all the other features in languages). Extending boolean to some ‘maybe’ classes lets ‘loop’ and ‘if’ decision points capture information. He uses `mustBeBoolean` to let these work with the optimised compiled rather than switching boolean optimisations off.

```
Maybe>>mustbeBoolean
  ProbabilisticExecution currentExecution
    atContext: thisContext sender
    addPredicate: self.
```

He stole continuations from Seaside. Dynamic code generation is not essential (he could interpret predicates) but the very large predicates that determine whether a point is in a domain are faster if generated.

No other language can do all these things.

Q(Niall) Probability calculation deep within the program? The probability integral at the start of program is for the input variables and computes new values for the output continuations, so the next computation uses these calculated probabilities.

Object Spaces for Safe Image Surgery, Gwenael Casaccio et al., INRIA
Do you want a self-modifying image, an image you can observe, an image with software process isolation? Then you need an ObjectSpace. An object in an ObjectSpace cannot access an object in another ObjectSpace.

You can create an ObjectSpace by cloning a running image or by bootstrapping Smalltalk in Smalltalk via deep copy of the classes and objects (changing their classes to those of the new system). This creates a system that has no references to the old system. CompiledMethods must be copied and recompiled to reference the new classes.

Reflection does not work: a mirror must be used. They still need to be able to load and save ObjectSpaces. That, capabilities and security are TBD.

He opened a window (terminal-style but responding to typed smalltalk), created a new ObjectSpace and demoed making it respond to simple methods. He showed he could arrange that

```
(smalltalk environmentAt: #Object) new foo
```

would fail whereas

```
(space environmentAt: #Object) new foo
```

would not (he having added the method in the ObjectSpace).

Q(Lucas) An ObjectSpace differs from a namespace in which all classes have cloned duplicates because...? An ObjectSpace isolates its objects.

Q(Andrew, then Niall) ObjectSpaces cannot talk to each other or to Smalltalk but Smalltalk can talk to all of them.

Q. Can I put an object from Smalltalk into the variable of an object in an ObjectSpace? Yes, but it would be the deep copied and reclassified object.

Jorge Ressia and Oscar Nierstrasz. Dynamic Synchronization - A Synchronization Model through Behavioural Reflection
He showed the dining philosophers problem: each of them needs two forks to eat and deadlock occurs if each has only one.

```
SynchronizationSpecification
  for: Philospher
  on: #eat
  interestedIn: #forks. (third param could be block)
```

controls when the `eat` message can be sent to a philosopher, This specification is registered with the main application and then uses code adaptation: modify the behaviour of the code depending on what events happen in our object model. Call

```
DynamicSynchronizationSystem current register: ...
```

to register the spec. Markus Denker built the tool Reflectivity last year. Using this, they implemented a solution. Some problems are solved 1700% slower than the optimal. Others, e.g. the dining philosophers, they can solve only 0.27% slower than the optimal solution.

He demoed. The system runs in Squeak. The Philosopher class implements `dropForks`, `eat`, `forks`, etc. where `forks` returns a set of objects: the philosophers position and the state of the forks at that position.

He ran the test, first without synchronisation (so philosopher 2 can pick up forks while philosopher 1 has them) and with (the true problem) but it had the usual demo hiccough and many questions attempting to clarify what he was saying left no time to restart.

CLIC: a Component Model Symbiotic with Smalltalk, Noury Bouraqadi and Luc Fabresse, Ecole des Mines de Douai
Clic (click) as in "the components just click together." Code may be out of synch with documentation or dependencies may be hidden inside methods. A component should be a self-documenting piece of software with explicit connections to the rest of the software.

We could reify components into OO but each component will require quite a few objects or we could start from a pure component language but then implementation is harder.

CLIC design idea is to say components are objects and (other) objects are dirty components - components lacking some features. A clic component has required and provided ports (very reminiscent of telecoms models). Attributes are private or shared, handled via accessors only and observable via ports. He showed the CLIC component code for a counter.

```
CComponent subclass #Clock
  localPrivateAttributeNames: #(count)
  privateAttributesInitDict: {#count -> 0}
  sharedAttributeNames: #()
  sharedAttributesInitDict: {}
  localRequiredPortsDict: {}
  category: #'ClicExamples-Clock'
```

and for a stopwatch

```
  ...
  localPrivateAttributeNames: #(counter ticker)
  privateAttributesInitDict: {
    #counter -> Counter @ #new.
    #ticker -> GenericTicker @ #new}
  sharedAttributeName: #(Scheduler)
  operationsExportDictFrom:
    {#counter @ #(count) -> #(seconds)
    #ticker @ #(start stop)};.
```

The stopwatch can forward messages to its subcomponents (e.g. the ticker to start ticking) and can translate messages for them (e.g. seconds is translated to count on the counter)

Q(Niall) Is this the Smalltalk way? Discussion.

Q(Thomas Shrader) But this does not provide the semantics? Their hope is that the system will infer them from code.

Detecting System Cycles with DSM, Jannick Laval
Jannick is a Ph.D student with the RModTeam in INRIA (Lille). A Dependency Structural Matrix sorts tasks based on their dependencies. A cycle is a closed path: a path that returns to its origin. Package dependencies should not be cyclic. Just looking a complex graph is a poor way to find dependencies. The dependency matrix records in each cell the dependency between the two graph elements that key that cell. Rows are using packages, columns are used packages. Cells can contain booleans or numbers expressing the strength of the dependency. Numbers can represent inheritance from a class in another package, extension of a class from another package, reference to a class in another package, etc.

The squared adjacency matrix is matrix multiple of the raw matrix by itself. If a non-zero number appears on the diagonal in such a matrix. There is a direct cycle (e.g. A -> B -> A). However it is hard to see which numbers correspond to which cycles. Path searching is rather clearer but slower to compute.

When you have a cycle, you can then regard all packages in the cycle as a single package and then recompute its relations to the remaining packages. This gives rise to a new matrix without the cycle.

Jannick showed colouring the matrix to distinguish cycles. He wants inheritance, extension, invocation and direct class reference distinguished as distinct numbers. An asymmetric cycle can perhaps be removed via extensions. A symmetric cycle can be merged.

Seaside 2.9 has a complicated graph. he showed the matrix; there were no entries above the diagonal, i.e. there are no cycles.

Next he showed Pharo. There was quite a bit above the diagonal in the lower right of the diagram. (The package right in the centre is Moose.) He then showed using Moose to display informative graphics inside the cells indicating the nature of the cycles. http://moose.unibe.ch/

Q. Invocation dependencies are static or dynamic? Static.

Q. Metrics for good package structure? The DSL is only about paths between packages.

Q. How long to do calculation. It takes 3-4 minutes to do the analysis. If you analyse Pharo as if it were not written in Smalltalk and so needed you to parse all the code into Moose first, that takes 3-4 hours.

I mentioned Travis Griggs' Browser-Prerequisites tool, released in VW7.7.

Show us your projects: ten-minutes presentations

Parallel sessions meant I only caught some of these talks.

VAStGoodies.com, Adriaan van Os, NationaalSpaarfonds
Vastgoodies.com is an open-source repository for Envy goodies. By intent, it looks very like the Envy browsers for config maps, apps, etc. You can download from the page or load an app Ernest Micklei made and download to or upload from your local Envy repository. If you upload to the site, you are asked for some meta-information (and must upload maps owned by your own user name, not owned by the library Supervisor, so this meta-information can be stored against your user name in the site's repository.

Q. Stability issues? Fine (it's a windows server so we reboot it from time to time but generally fine).

Q. Code for the site available from the site? Not yet, will be.

Seaside in VA, Adriaan van Os, NationaalSpaarfonds
Adriaan demoed the NSF auto insurance app on the web. This was written in alpha 3 of Seaside 2.9. He entered a license plate, kilometres per year, etc. He entered an invalid date to show the error reporting, then completed and the app showed a picture of the car type with the licence plate.

Interfaces on SoundWaves, Koenig
This was done in Squeak. Koenig is studying sound interference for concerts of audience of hundreds where the singers can suffer feedback into the microphones. For high frequencies, a horn in front of the microphone solves the problem but for deep cycles - 30 cycles or so - the sound is emitted as a sphere and will feedback. When sound is emitted from one speaker, the idea is to have a cancelling signal at the other speaker adjusted for distance between the speakers. The second speaker also provides feedback to the first so the calculations are quite complex.

He showed a diagram of the speakers being rotated around and the wave effects at various audience angles. Then he showed the plot of two speakers back-to-back with cancellation signals. He uses these tools for automatic testing, sending sounds to the speakers on a test rig and monitoring results.

Yuri Mironenko
Yuri is from Rostov and has finally made it to ESUG (he hoped to come to ESUG 2007 and 2008 but travel arrangements and visas got held up in slow offices both times). He has implemented a game of travelling in a rocket in the solar system. He demoed correcting trajectories for your orbit to reach your target.

Michael Haupt, Potsdam University
He and his students have implemented a game (for young children) of guiding a cow to a target in a field with various obstacles. It has 30 levels of increasing difficulty. The talk was less about the game than about the community website where somewhat older children (i.e the students and the speaker) can create levels. He created a new level, then opened the game. The cow got stuck because he had not set one of the doors to be openable; he connected that button and then the cow could open the door.

His students have devised many levels. He is stuck in level 28. He can review the levels, rate them for difficulty and fun, mark them as unsolvable, etc. If you devise a level the students cannot solve, you win a meeting with them.

This is written in Seaside, Ajax and Java script.

Nicole de Graaf
She works 40 hours per week at a Smalltalk company but also wanted to use Smalltalk in her own time. She created a DirectX interface and two others for VW. She took a month to make a multicoloured triangle rotate. Then she had a 3-d background with spheres and then a spaceship and then a game: draughts. She wanted a taken draught to explode; she can do an explosion and the draught moves but not yet put them together. She can make the spaceship make a sound. Now there is a forest we can walk through. She has code tools in the RB for working with these interfaces.

Igor Stasenko
Igor opened a Squeak image. We know what Flash is. Many people would like to draw vector graphics like Flash. A few years ago several companies created a new standard for vector graphics, OpenVG. He created an API for it in Squeak, which took him one week. There are 5 implementations of OpenVG for different platforms and luckily one of them is for his platform, so he loaded the libraries (doIt on some `loadLibraries: ...dll` code). He ran a basic tests that showed circles, lines, etc. His main demo was a tiger’s head, very detailed (just under 17,000 data points) which expanded and contracted and simultaneously rotated and so on. It was not anti-aliased and was rendered at 55 frames per second (his VGA driver could be better).

Q(Christian) the base shapes did not render very well; can you show them again? The tiger demo reset the colours so he could not at first, so restarted the image and showed and yes the lower left corner of the rounded rectangle was indeed bad. The OpenVG allows nicest or fastest and this is the latter.

Other Discussions

John McIntosh found a bug in NetNameResolver in Pharo/Squeak last week. It found the full list of interfaces, including any VMware and parallels ones, and just picked the first, which might not be the one you wanted, so was failing a loopback test. (And it was not getting the value quite right anyway but that was another bug). John has discussed with Dale making an interface from iPhone Mac core data to GemStone.

Lucas talked more about Helvetia. A LanguageBox defines `change:`, `compile:`, `debug` and other methods to say how the parse tree is changed, how it is compiled and how it should be debugged. Later Lucas will use Glamour for a more detailed debugger view but the debugger opens OK. The RB parse tree is the one used (so Helvetia will work wherever the RB is). He showed a simple case of Regex. Change recognises / followed by a literal followed by / and converts to that as a string sent `asRegex`. The SQL is more complex and bidirectional - you can put SQL statements in Smalltalk and Smalltalk statements inside an SQL 'where' clause, etc.

Maglev now handles private methods (a year ago it did not). James Foster noted that VW's Store runs over Glorp, providing a Glorp front-end to GemStone means Store could use Gemstone as the back-end.

We should code up a version of the dining philosophers problem - the power-socket-seeking Smalltalkers problem - as a solution is certainly needed at every ESUG. :-)

Richarte and Leandro can run VisualSmalltalk in Linux; just the VisualSmalltalk app plus some tweaks makes it possible.

I will look at using StoreForGlorpReplicator to do the Monticello import into VW7.7. I will send SUnit report text to Stephane, with examples and look at Keith Hodge's stuff.

VASmalltalk Virtual Conference, April 21st-22nd, 2009

Travel to a virtual conference is painless.

Summary of Projects and Talks

Tina Kyvale (Instantiations marketing manager) introduced the event and explained web conference protocol for questions, etc. Recordings are on http://www.instantiations.com/mktg/events/vastsummit09-delivery.html.

VA Smalltalk: Today, Tomorrow and Beyond, Mike Taylor, Eric Clayberg, Instantiations
Mike started early in Smalltalk (Digitalk). Eric used alpha Smalltalk at University in 1985 and began commercially working in it in 1991.

Apple's Smalltalk add: well, they were in Smalltalk once and maybe will be again. :-)

ST business is very good right now. They have 20% growth in 2008 which is very good considering the state of the world economy in general. They are seeing new licences and new customers.

Recent customer survey (in March, 921 invited, 231 responded). Smalltalk is used all over the place, in finance, insurance and other areas. USA is 47% of their business and Europe is 38%. Desktop and sever apps are common but 21% have web apps. Of course, most of the respondents use VASmalltalk (49%) or VAST (24%). 46% are stable, with 1-2 releases annually. 36% are just deployed, 7% being developed. Just 11% are maintenance-mode only.

US and Europe had 30% evolving quickly, UK 25% being built not yet deployed. Generally 2/3 legacy, 1/3 new development is the rule. Current applications are 90%-plus expected to be in use for at least 3-5 years and 20% say they expect the current system to be in serious use 10 years from now.

Most expect their Smalltalk use to stay the same with significant percentages expecting to increase it in the UK and Germany and non-trivial everywhere. Only 1/3 of customers were considering replacing any Smalltalk system, usually with a Java system.

Value of Smalltalk? 85% say it is excellent or good.

Mike handed over to Eric at this point. Eric reviewed their releases, support stats and online forum data.

VA 8.0.0 is being released today, the culmination of the 2 years since Instantiations took over VAST and re-branded it VASmalltalk. They've had ~ 1850 support cases of which 924 cases were answered without product changes; 167 were fixed with product changes. 300 are still open.

Their user forum has 1850+ posts (475+ by Instantiations) and has had 520,000+ views of the topics.

Customers want VASMalltalk to keep up with operating systems, to exploit 64 bit, to be robust; Seaside was another interest. Only 18% of respondents had not heard of it. It was critical to the current development of 4% and important to more than a quarter of them. Seaside was especially important to smaller development teams. 64 bit was a common interest across the customer base.

In the next 18 months: currency (not euro symbols but keeping VASmalltalk current on all important OSs, Databases, e.g. Oracle, and so on), web services (wizards, performance, deployment, documentation), unicode (unicode work should be finished soon, including using UTF-8 internally), look&feel (7 had VAAssist, 8 has flat-look windows toolbars, need to support 24 bit and 32 bit icons), VM Enhancements (continuation support is continuing, 64 bit is being investigated and associated GC improvements to handle massive amounts of memory), Seaside (8.0 uses 2.9 alpha and will use 2.9 in 8.1 as soon as it is released; they'll provide Seaside development tools, etc.), Performance (hotspot analysis of base classes and XML parser), Security, ANSI Smalltalk, Glorp (version available, will be included in release), Documentation (many out of date docs have been, are being and will be updated, and there are new ones for new features). There will be a VAST team, blog and public development builds.

Q. Cross-Smalltalk cooperation - monticello 2, etc? We are looking into Seaside-related tools cross-vendor.

Q. Do we have to use tabbed browsers? Existing browsers are all there so you can stick with them and turn off tabs if you want

Q. Improved graphics? Answered in John's talk.

Q(Niall) Smalltalk is surprisingly strong now (both VASmalltalk and other vendors); any views why it is strong right now, when the economy in general is weak? Partly, tough times need productive tools, partly, tough times make you keep what you have.

Q. Window builder for Seaside? To be investigated.

Q. WindowBuilder Pro or other kits in base product? WBPro yes could be considered because Instantiations own it (of course, it only works on Windows). Not the other two because they include others' libraries.

Q. Can Linux users expect UI bindings e.g. GTK or anything other than Motif? Being looked at, no promises.

Q. Upgrade directly from 6 to 8? Yes, perfectly doable.

Q. Free academic licence? Trial version is usable for academic learning and has no time bomb. Academic use encouraged.

Q. How often will the development build be available? Perhaps once a month, certainly more often than current every 4-6 months.

Q. Make it easy to exchange source code with VW, Squeak, etc. (e.g. JP's framework)? Not thought of, ask John. A talk about porting from Squeak to VA is later (VAStGoodies).

64 bit Qs were deferred to the end-conference Q&A session.

Q. Product pricing? Survey asked if lower price would increase use? Answer was that while a few said lower price would let them increase use, general answer was that price was not the issue. They *may* offer a lower-price version but currently customers report high value so cost and value is well balanced. By all means talk to them if you have "cost leading to more use" scenarios.

Q. Interaction with MS .Net? No specific plans. (Any .Net specific frameworks people are interested in, tell them.)

Mike closed by stressing that Smalltalk ideas were welcome; email the team with your advice on what you need from VASmalltalk.

RESTful Web Services in VA Smalltalk, Joachim Tuchel, ObjectFabrik

(Had to miss this talk; the one disadvantage of a virtual conference is that you can't duck out of other meetings so easily as when you're not there. :-)

VASmalltalk 8.0, John O'Keefe, Instantiations

John O'Keefe has a long history with Smalltalk. He first saw (Digitalk) Smalltalk in 1987 and was a founding member of the Smalltalk team at IBM. He was very glad to forge a relationship with Instantiations in 2007 when IBM retired from Smalltalk. He leads the development team at Raleigh, North Carolina.

Instantiations supported VisualAge Smalltalk in 2004, licensed it in 2005. For the last two years, Instantiations have maintained and sold VASmalltalk which was formerly VisualAge Smalltalk at IBM. They released VASmalltalk 7.5.0, 7.5.1 and 7.5.2 during that time, focused on tool integration and improvements to the product. They added new platforms Vista, SuSE Linux and 64-bit integration.

VASmalltalk 8.0 adds: Seaside, Browsers and Tools are enhanced. Web services are enhanced. After a long time in which the inherited documentation had fallen behind the product, they have now written new documentation and improved the documentation system. There are also various small changes.

Seaside, also jQuery and Scriptaculous libraries for AJAX-style interactions.

Scriptaculous has also been ported. A (Squeak-oriented) Seaside porting layer has been developed and provides functionality in three ways: the base is Seaside2.9alpha3 plus Core, JQuery, Scriptaculous, RSS and Slime (RBSmalllint extension that ensures your seaside code follows seaside patterns.) There are also Seaside Tools and a Seaside Porting Layer.

John showed the Seaside server control panel and the web-based tools for showing e.g. the processes being run (so you can terminate a runaway process, inspect things, etc., from the web).

Seaside 2.9 is alpha-level so it may change, and so may the porting layer.

The Seaside flow is only partially supported. There is no `wait:` method because continuations are still being completed. Instead you use `show:onAnswer:` aBlock to achieve the same effect. Seaside encoding is Latin 1 only. You must use the built-in SST HTTP Server. All these limits will be lifted in later releases.

Users wanted improved and integrated browsers. John showed a 7.5 browser and then the changes in 8.0. The 8.0 had code tabs (class definition, method source, method comment, method notes) and method pane tabs (‘public’, ‘private’ and ‘all’). 8.0 buttons are flat and are native or emulated (toggle as wished) on Windows (still always emulated on Unix). Tabs can be sticky (navigate preserves selected tab) or not (always revert to class definition). Additional tabs show class version history graphically (looked good, like the TrailBlazer utility).

These affect all EtWindows and the Class(es) and (Shadow)App browsers.

Inspectors have workspace where selecting items does not lose typed code, and show more data (all forms of integers: binary, octal, etc.)

RB and SUnitBrowser fixes: AddParameter handled better, RemoveParameter more available.

Q. Squeak-porting tools? Adriaan will discuss and also see the Seaside library package which exports packages from Squeak to VA (thanks to Adriaan and Lucas).

Q. Test suites for regression tests? They have two. One (non-SUnit) tests low-level classes. Another (as mentioned above) is used to test UI, and there are SUnit tests. John will consider whether to include the SUnit tests.

Q. Stay up-to-date with Seaside? Absolutely. John is participating in Seaside open-source activities, not just porting code. They will deliver updated Seaside in every release.

Q. Report writer update? No plans. John recalls that some asked for it to be made available on Unix platforms for example.

Q. WebService cookbook tells you how to package them, especially package headless?

Q. RB browsers? Available as separate UI or integrated (via Mastering Envy Developer map).

Q. Migration from 7.x to 8.0 issues? There is always something to do when upgrading but John thinks it is pretty minimal between 7.5 and 8.

Q. Test and Quality approaches? Some tests are automatic (e.g. VM builds have lots of automated tests, SUnit tests are run in batch process), some are manual (e.g the GUI tests are manual; change a UI, run its tests).

Q. New widgets on windows only or other platforms as well? Today, focus is on windows because most of VA client apps are there.

Q. Classic browsers still available? Yes, just disable VAAssist.

Q. Timelines for 8.0.1 and 8.0.2? Not at the moment. Plan is to release every 4-6 months.

Q(Martin) Plans for EDECL calling support under windows? Yes, there are plans. (Lost from V8 because of time constraints but OpenSSL updates mean we will need it.)

Q. Envy/Manager on multi-processor PC? It will warn you on a multi-processor PC because that code belonged to IBM till recently. Instantiations now has it, will update the code, and has never seen a problem except on Windows NT Server. Just use the -nt switch and it should be fine.

Q. REST? See Joachim’s talk.

Q. Fully automatic builds? Builds are automated now. Except when network connectivity hiccoughs, 2 hours of wallclock time is enough to do a build.

Q. Glorp port is done for VASmalltalk? A port of Glorp 0.3.178 is complete and available. An upgrade to latest Glorp is underway. The intent is to offer Glorp with the release in due course.

Q. Can we have 4 or 8 Gig addressing in a developer release (i.e. soonest; users very interested)? Nothing is held back so you will see it as soon as we do it.

Q. Improved EMSERV on linux (no more 2Gig limit)? Now we have the code, we’ll be working to fix that. (John has a large manager.dat.)

Web services: a new style of WSDL called doc literal wrapped has come in during the last two years. 8.0 supports the wrapped doc literal style in 8.0. They provide a cookbook on debugging and will add sections on serialization and hosting. They explain debugging techniques because it is easy enough to do once you know how but there was no explanation of how.

The old web-presented VA documentation was ugly and the source for it has been unavailable for several years (long story). This is partly why the documentation has not changed since 6.0. They have revamped the documentation. The documentation server is gone; they use WebWorks. Search and all other features now work both locally and through the web. CSS is used for formatting. Later, new PDF books will be written.

John showed the appearance of a page in the prior system and in the new WebWorks system. Contents tab shows structure of document and which parts can be expanded to more detail. Search can be per book or over all books. A favourites tab lets you bookmark (but does not always work locally on IE or Google Chrome - 'feature' of those browsers).

An email button lets you send feedback to developers, giving page ref in subject line and printing the page in the email.

Themes are supported on Windows XP (XD image parameter).

ENVY/Image Interactive Tests Suites are provided as additional examples of how to use widgets, graphics and printing. You can benchmark with `sampleAndBrowse` or `traceAndBrowse`. (The Performance Workbench is powerful but takes a little more setting up; these take a block and open the results in the Performance Workbench.)

In V8, they have begun providing Goodies: not-quite-ready features. Some existing goodies are also being provided, e.g. UML Designer. They will keep these goodies in synch with the V8 goodies, in the sense that they will load and are not obviously broken.

What is missing? OS/2 will *not* be supported in VASmalltalk 8. Instantiations has never formally supported OS/2 but till now has kept it running there. However 7.5.3 is the last such version. Most OS/2 code has already been removed from 8; the rest will go soon.

UTF-8 support is deferred; it will be in a follow-on release.

Get V8 from http:/www.instantiations.com/VAST

V8.0.1 will have Seaside 2.9 Beta, UTF-8, more ANSI Smalltalk (Exception extensions and Timezone support), SST Servlet multipart forms, Web service wizard enhancements, GLORP (ActiveRecord support will be further in the future), Window Server 2008 support (runs now and customer OK with it but not yet full regression test on that platform).

8.0.2: Seaside 2.9, Seaside class browser and profiler and deployment tools. OpenSSL being brought up to latest release and with wrapped OpenSSL security interfaces.

Ongoing: platforms (Windows 7, Ubuntu 8.10, Fedora Core 11), performance enhancements (John is doing hotspot analysis of the base, the XMP parser and other areas to determine where performance needs work). VM enhancements to 64-bit will need incremental GC. Look&Feel will be worked on (also .png and .tiff support). A single install/repair/uninstall would be useful, as would delta manager exports and more documentation.

ANSI exceptions are fully supported. Their old instance-based exception system is integrated with them in 8.0. You can now use the ANSI `on:do:` (class-based) or `when:do:` (instance-based) or a mix of both. John has switched the SUnit preload over to use ANSI: the rewrite was a useful test.

They will improve serialization to support the wrapped literal style which has become popular in the .Net domain. The standards are rather vague in some cases so they have studied how this works and should work and will provide working examples.

There is full support for UTF-8 locales. Most Linux platforms now use UTF-8 is their default out-of-the-box so it was a problem even to install without it. They will complete their support for UTF-8 locales.

Not Your Regular Pet Project: Using Smalltalk and Seaside in the Enterprise, Ernest Micklei and Soemirno Kartosoewito, NationaalSpaarfonds

Ernest is an old-time Smalltalker who has recently been re-infected by Seaside. Soemirno is new to Smalltalk.

NationaalSpaarfonds is a leader in car insurance in Holland. They also offer pet insurance. The new web app handles claims under this insurance. Ernest showed the screens taking you through the five-stage process: the user enters data, is advised of conditions and of process (email sent to them confirming data of their claim, another when claim processed, etc.).

Next he talked through the architecture of the application. There is a Firewall between the seaside application that talks to the web and the identical (in the model-layer) back office application where the instance created for the user is processed.

Then he introduced how these pages talk to the Seaside application. Soemirno took over and walked through the technical details. jQuery with validations drives the pages, binding form fields to model fields. Soemirno showed the page, its code and the Seaside Smalltalk code, showing how fields are bound to model aspects, ids assigned for jQuery feedback, etc.

[Rather than type lots of code here, I invite readers to watch this part of the talk from the instantiations website.]

The application uses jQuery javascript to catch when the user inputs to a field. Soemirno showed the very simply Smalltalk calls of `onChange:` and the much less simple javascript resulting from it in the web page's HTML code. By this means, the model calculates any fields dependent on the one the user changed, and so they are updated, giving the app a client-server feel.

Why use jQuery with Seaside; it is a current de-facto standard, it is cross-browser, they knew it, and its programming model is a good match for Seaside.

Problems: both the Seaside/jQuery codebase and V8 were moving targets. Some things were slow on IE6.0.

Q. Lucas' changes just for NationaalSpaarfonds or also in 2.9? Also in 2.9.

Q. How long to develop? Started December, current state is what you see. It was the team's first Seaside application and the Soemirno's first Smalltalk app.

Q. Did you miss the support for continuations? Not really; designed the app around their being absent so might have used it if they had been there.

Q. Does jQuery allow rendering of several fields that depend on each other? It can but they have used it single field only so far.

Q. Strategy decision about the use of Seaside? Yes, chose to use framework maintained by larger community rather than in-house framework.

Q. What problems did you encounter? Learning curve of Seaside (and jQuery), then the fact that the code was evolving in the v8Beta as they were developing.

Q. Marketing teams opinion of app? App is used by existing customers to make claims, not to get new customers in, so marketing guys had less say.

Q. How does it work with HTML and CSS? There is no writing of HTML; that is all created from Smalltalk by Seaside. CSS was written by a separate style designer; they sat with the designer and he told them what classes he needed.

Q. Deployment set-up? (Not in production yet) This app will have low load so may well be a few images and load monitored.

Creating VAStGoodies.com - a VA Smalltalk Seaside User Experience, Adriaan van Os, NationaalSpaarfonds
Adriaan showed the vastgoodies.com site, which looks very like an Envy configuration map browser.

VAStGoodies.com has two goals:

- Socially, to facilitate VASmalltalk open-source software, and to build a more active and visible VASmalltalk community. The VASmalltalk community is (relatively) silent.
- Technically, none of the existing open-source repositories fit Envy well as Envy is not (simply) file-based so VA goodies are all over the web and you find dead URLs and so on.

VAStGoodies is an Envy-format source code repository. It is indexable by search engines. It can be linked to by other web pages. VAStGoodies is a hang-out, not a hide-out. By all means use other sites to manage your feature requests, etc., but put them up on VAStGoodies to make them visible.

A Hosted Projects page gives data about the projects, other web pages for them, etc.

VAStGoodies sticks to the Envy model that its users all know. By using

Config maps and apps, we can be sure that things load and we're using the versions we think we are. By using the Envy user model, no additional user data is needed.

The VAStGoodies server sits on Seaside and a servlet and Envy. Seaside and the servlet sit on SST and the whole sits on VASmalltalk.

It took Adriaan 10 days / 25 hours of effort to get the first version up. Issues: Envy is a single-client-user tool with modal dialogs and a single repository file. You do not want to show all the maps on the web, just the ones that are goodies. Hence Adriaan silenced all the dialogs and serialized all repository access via a semaphore so you see the goodies repository, not any other.

He also did some caching (for performance) of names of visible config maps and their editions, and of the names of projects.

Downloads are just exports from Envy which are kept and served by the web server.

V8 beta had no continuations so no flow. There are no mime types which were wanted for uploading config maps' meta-data. There was no RSS.

Adriaan did not use continuation-passing coding style because it got ugly quickly and anyway the continuations will be there soonish, so instead he used jQuery as that was fast enough at the moment.

Adriaan ported jQuery (and jQuery UI, not used in VAStGoodies yet) from Squeak. Mime types he hacked. RSS he did a port. Lucas ported TinyWiki that renders wiki-syntax code on the web.

Aside on porting: porting used the VA Smalltalk Porting layer and the VAPackageExporter. Adriaan observed that Seaside is bringing all Smalltalk dialects a bit closer. This is good for Smalltalk and for its open-source projects. PackageExporter was originally written by Avi Bryant for Squeak-to-Squeak porting and John enhanced it to VAPackageExporter. Lukas Renggli and Adriaan created a differences exporter so you can just port the changed code after having done your first baseline port of a utility. John enhanced it further and it is now very useful for porting.

```
VAPackageExporter fileOutPackageNamed: aString.
VAPackageExporter
  fileOutPackageNamed: aString
  deltaFrom: aVersionString.
```

There are porting issues. `aNumber asInteger` is truncated in Squeak but VASmalltalk rounds as ANSI directs. Using `aNumber rounded` avoids the issue; the Pharo guys have been alerted to this. (In such cases, remember always to fix the reference implementation, not the port.) Prereqs were not always well-defined in Squeak. Look at WAPackage and VAPackageExporter>>prerequisites for examples of how they addressed that. Always run Slime, since it tracks non-portable classes and methods. Finally, when versioning, reference the source version id.

Back to VAStGoodies. The site wants meta-data on your project, such as licence information, any URLs of related sites, and any additional info for the pages. Ernest Micklei build the VAStGoodies tools, i.e. the web service and client. Load this app into your image and then invoke the Annotation Editor to talk directly from your image to the VAStGoodies server. Adriaan showed the editor UI. It demands that you provide licence info. The licence must be clear or companies won't use it. You must choose one or companies can't use it. MIT is easy.

The site will add an application page, like the config map page, and a statistics page.

Lastly, Adriaan thanked John, Lucas, Ernest and others for coding help, and Louis Andriese for providing free hosting of VAStGoodies.

Q. Is the VAStGoodies server code also open source? Adriaan wants to have a single site so it builds a community but perhaps if we grow large that would justify multiple sites. By all means talk to him if interested.

Q. Continuation-passing style disliked because?

- If you have a complicated flow, the style gets clunky.
- It is not the final state so would have been temporary code.
- Adriaan wanted to try jQuery.

Q. How to ensure not exposing internal code when using a shared Envy? The non-config maps are hidden but that's all he does currently. [Niall: the repository would naturally have just the VASmalltalk 8.0.0 base maps, the VAStGoodies server map(s) and the visible maps, so if more of these were reachable than just the intended maps, it would seem no great matter as what user would not have these maps locally anyway.]

Q. Issue tracker? Use an external one. Adriaan will not build one and using e.g. GoogleCode gives Smalltalk visibility on those sites.

Q. Download of config map with all prereqs? Yes that could be offered but it would only be a download of all visible maps.

Q. Mime-type support required what? Some dirty hacking. Adriaan looked at what Seaside-Squeak did and just grabbed what he needed, not elegantly but just to make it work by cut-and-paste from Squeak.

Q. The VAStGoodies site was built with jQuery events and Seaside canvas? The UI is wholly rendered within Seaside. Adriaan manipulates the DOM by AJAX callbacks to the server.

Q. Use this code for a distributed project? One could, but does it add much functionality to what Envy replicator can do already?

Q. Where is VAPackageExporter? Go to Adriaan's blog (http://a3aan.st/sunrise) and its location will be findable from there.

Q. Enforce comment field to make uploaders provide an overview explaining what their upload does? Now that upload is only possible via VAStGoodies Tools, that already encourages this.

Q. Harvest existing code? By all means try to find an owner of any useful utility and arrange its upload.

Q. Include 7.5.2 and 8.0 to show all prereqs? Adriaan agrees that some prereqs are missing

Q. Have a 'remove' command to deal with old code? For now, just email Adriaan. Later, he will see.

Implementing Genetic Algorithms Efficiently in Smalltalk, Bob Whitefield, Model Design Corporation
Bob started in Smalltalk in 1989, and has yet to find a language he likes better. He founded ModelDesign in 1997.

Genetic algorithms create a population of potential solutions then evaluate, select, combine and mutate them to try to produce a superior population. GAs are most useful for solving problems where no known algorithm exists or where the time to find a solution algorithmically is not affordable.

He defined genetic terms in a computer context: gene (a slot), allele (value that can be held in a slot), crossover (combining material from two solutions to create a new one), fitness (a function that evaluates how good the solution is).

Bob developed the Mendel framework so that users could apply GA to problems simply, flexibly and efficiently. He has a hierarchy of simple, composed and grammar gene classes, with operators optimised for blending and recombination. Various techniques avoid known GA issues such as premature convergence, accidental loss of the best solution, etc.

He started with a simple guess-the-hidden-value problem; all you are told is how many 0s and 1s you got right in a 100-bit number. He showed the fitness value. The correct result was found after 5000 iterations and 10,000 evaluations. A human could reason the answer faster but a mere enumeration would have been *much* slower. A 3-d contour was similar.

Next he looked at the travelling salesman problem. He listed the creation protocol that the Mendel framework offers to let you set up the problem, set population parameters (can always use default values for these but setting values that fit a problem can let you reach the result faster). The fitness function for this problem simply sums the distance travelled, seeking to minimise it. He showed the diagrams of the paths as Mendel converged on the ideal no-crossed-paths solution. this solution took 41 seconds (a rerun could take 30 - 90 seconds depending on chance factors).

Next he looked at a scheduling problem: maximise the number of widgets produced at various bays while allowing employee choices of preferred work and minimising how often an employee changes bays. An employee has 4 genes (start times, periods worked, lunch periods and bays). The fitness function minimises departures from employee preferences and bay changes via 6 criteria, each weighted to balance the goals.

Genome design needs to avoid redundant information that must be kept in synch. Simpler is better; let the solution do the work, not your gene design.

Some deviations from pure OO design were done for performance. Chromosomes (units of recombination) are not first class objects; instead, genes contain all the behaviour and they operate on the allele data. Genes are not implemented as instance variables.

His last example was a case study he did for a major agricultural supplier who must make hundreds of shipments each week. Tours that keep the truck in service for 4-5 days offer serious cost savings. He solved this with a hybrid technique: GA plus an insertion algorithm. Each solution is a permutation of all shipment legs and the insertion algorithm inserts it into a leg of a tour that saves the most money. If no saving is possible on an existing tour, a new tour is created. The resulting collection of tours is a shipping plan and the fitness function evaluates the cost of the shipping plan. He showed tours being visualised in google maps. the company believes they will save $800,000 this year by using the tool.

Mendel is very computationally expensive so he had to watch performance

- minimise instance creation: it is faster than in other OO languages but still slow and the instances must be GCed
- use SmallIntegers, not Floats, wherever possible
- use Arrays and OrderedCollections instead of Sets, Dictionaries or SortedCollections
- pre-allocate collections whenever you can estimate their eventual size
- etc.

Do *not* optimise until and unless you know it will help. The simplest thing that could possibly work is usually fast enough. Only remove accessors in very critical code. Lazy initialization is often fast enough and sometimes is actually faster.

Smalltalk's advantages were speed of development (always) and speed of execution (sometimes): C++ or Java would have run faster but been 5-10 times slower to write. Ruby, Python or Perl would have been slower both to write and to run. Bob has seen how Smalltalk applications are still working on platforms that did not exist when the apps were first written: he likes the insulation from platform changes that Smalltalk provides.

Q. Use GA to classify emails? Yes if you used GA to train a neural net. NNs are powerful but hard to train and GAs are good for that.

Q. Commercially available? No, currently it is only available as part of his consulting work.

Q. How long does it take to have working code? The truck tour optimisation took 3 months, most of which was understanding the problem, ensuring the solution did not violate any constraints, working out how to combine the tours. By contrast, if you already have a well-defined fitness function, it can take only a few days to have a working Mendel system.

Gaining Object Persistence, Martin McClure, GemStone
Martin's talk was entirely a demo (so see it on Instantiations' website). His demo was of a simple employee app that originally used a relational database. He got this app from Solveig; she and Martin stripped out the relational persistence from it. Martin will now show adding GemStone persistence.

Martin walked through the app structure: classes for employee, department and so on, with some class vars for caching AllDepartments, holding LastID (to generate unique IDs for employee) and so on. The GUI was generated by WBPro.

Martin initialized, clearing out all employees and departments, then added some sample data. He looked up employee McClure, upped his salary and so on. :-)

This data is all in-image. Now we make it persistent using GemStone/S. GemStone/S is a server smalltalk implementation: it is headless (no GUI), has shared transactional persistence (committing is like image save but multi-user, with changes being merged on commit). GemStone/S is called an object database but that is not all it is. GemStone/S is synergistic with VASmalltalk. VASmalltalk provides a GUI and other things that GemStone/S does not have, while GemStone/S has the multi-user persistence that VASmalltalk does not have. GemBuilder for Smalltalk (GBS) runs in VASmalltalk and lets the two work together.

Martin opened the GemStone session browser and looked at a GS code browser. He showed that this brand-new GS implementation knew nothing about the employee app yet.

The app UI stays on the client. Its model layer is reflected on the GS server. He published (select the classes and menuclick) and showed that the model classes now existing on the server; the classes, but as yet no methods. Connectors have been automatically created to relate the client and server classes. He showed how the Department class connector mapped the class var 'AllDepartments' between the two. Use connectors for root objects you want to connect on startup; they will lookup (by name) and perform the appropriate postLoad action (updateST, updateGS, etc.). He set updateST to show how that nilled the image's data for the class vars.

He added employee John O'Keefe, logged out, showed how the app failed (var is now nil) logged in again and could again see employee O'Keefe.

Martin showed how to add auto-login/logout to application startup and shutdown, committing whenever you save. It's that simple - unless you want to deal with some performance issue, which is the usual cause of code becoming less simple, so he closed by looking at a performance technique.

He added 10,000 employees, logged out and logged in again. All 10,000 employees were replicated, taking 5 seconds to move 2Mb from the server to the client and create all those objects. It would be better if we moved 'AllEmployees' to the server and the client only had the one, or the few, that it was using at a time. So far, we have used replicates. Now we use a forwarder. Forwarders have no state on the client and forward all messages sent to them onwards to the server. Only the result is replicated back to the client. He made AllEmployees a forwarder, logging out and giving its connector a forwarder postConnect action. He logged in again and got a DNU; the `employeeNamed:` method is forwarded to the server class which does not understand it. He therefore by menuclick copied all the needed methods to the server. Next he got another DNU; he showed how the Gemstone-extended VASmalltalk debugger walked up into the server's calls (prefaced by GS: in the stack); the server class has no data. He copied also the accessors to let data be added and now it all worked. Login was a lot faster than the 5 seconds we saw before. He showed GBS inspectors on the server-side objects.

Now we have some server-side Smalltalk execution.

Q. GemStone over a wide-area n/w - is the protocol chatty? It is not very chatty. Care in coding the application will help.

Q. Shape-changes, old and new class versions? GemStone has class histories that group different versions of a class. GemStone provides migration code and hooks to add complex migration code if needed and you can migrate all instances at once or migrate lazily.

Q. Forwarders? are never stored on the client?

Q. Classes under envy? There is a public-domain kit for Envy that lets you put GemStone classes into the client and put them under Envy control. Native source-code control is Monticello (the Squeak CM tool).

Q. How do you keep client and server code in synch? Some want the code to be identical and some do not.

Q. GBS for VASmalltalk 8.0.0? It will probably load and run right now but allow a few months for GemStone to finish testing and release supported.

Q. Methods live in both machines? You can have the same method in client and server and executed in both; in some circumstances that is a sensible thing to do.

Q. End-user reporting tools? There is no GemStone-specific reporting tools. Various customers have built very sophisticated report-writers based on GemStone. You query GemStone like other Smalltalks via select: and suchlike on collections and so on.

Q&A Session
Improve packaging performance? Not currently planned, can look at.

GTK and similar bindings for VAST? We have not looked into it. It is not on our list to look into at this time. SWT similarly has no overlap with Smalltalk that they can see. Instantiations has strengths in that area in their Java products so if there were a customer case they could do it - but they see no business case.

Backward compatible VMs? John runs version 3 images on version 8 VMs. Just be sure you run a complete set of binaries from a given release; do not mix binaries from releases.

How do you promote Smalltalk and help customers dissuade managers who say they will rewrite in Java? The cost of rewriting a Smalltalk app to Java will probably be more than the cost of creating it in the first place, and the errors-per-function point measure for Smalltalk is much lower (3-10 times lower) than in Java. Finally using automatic translation tools has not been an effective route either (block closures are hard to translate and automatically-generated code is unreadable). They have material they can share with people to show to their managers to make a compelling cost-oriented case. Actual translation of Smalltalk systems to Java is rare; more often, people decide that an Smalltalk app is 'near the end of its life' and then a new Java app is written but they have also found themselves keeping the Smalltalk app after doing so.

Multi-core such as quad-core, and forking threads to use 100% of each core? V8 is single-threaded on a core and callouts can be asynch and, unless affinity is set, these will run on other cores. They have often seen VASmalltalk taking 20% of one core while 3 other cores are maxed out for the DB and etc. keeping up with it. The 'single-core for Smalltalk, other cores for the other apps' approach is in fact very performant.

Tools to automate build processes? We do automated builds but you do not want our build process so briefly no.

Windows and Linux are most deployed-on platforms but can we count on continued support for Solaris and AIX? Yes!

What kind of code contribution do you want from your users? Any interesting things are welcome. Now that Seaside is available, VAStGoodies has GoogleCharts and there are other Seaside things

Tools to integrate with MS tools: .net, SharePoint or Office? No .net plans but they'll look at it and SharePoint. Office OLE interaction is available.

Known buglist on the website? Yes, provided we can figure out the mechanics of it. We have a nice internal system behind our firewall so must work out how to get the appropriate parts of it visible.

Support for programming against interfaces? SmallInterfaces was an interface browser. Nothing new here.

Tracking non-GCed objects? See KESStacks goodie.

Development builds? Unregression-tested so use at own risk.

Enhance Smalltalk language, e.g. namespaces, mixins, aspects, class extensions that define instance variables? The last is possible. Mixins not looked at or planned. There is in fact some namespace support (Classname::Classname) but it is not visible and not high on their list (if there were a business case ...).

GUI designer will be WBPro or the Composition Editor? They will support both and are considering bundling WBPro but be aware WBPro only runs under Windows, which is the only reason it is not already in the base. WBPro is also revenue-generating so if many customers say it is high value it may go in the base and if not they may leave things as is.

Seaside-oriented tools? We have no plans like web velocity but have thought of something like SeaBreeze; no specific plans or schedule yet.

Sandboxing? Not thought about yet. Security framework for encryption is planned.

Looking for ST programmers? Ed mentioned some American sources and providers. The Smalltalk jobs DB and the national and local Smalltalk user groups are worth using to hire Smalltalkers. The conferences are also a good place, especially ESUG if you are this side of the Atlantic.

Closing Discussions

They are thinking of offering this as an annual event.

Other Discussions

The one problem with a web conference; there aren't any other discussions.

Conclusions

My eleventh ESUG and my third VASUG:

- Seaside just keeps growing
- Being respected by adherents of now-fashionable Ruby makes a pleasant change from being rejected by adherents of no-longer-fashionable Java.
- Smalltalk's second surge was still much in evidence; given the state of the economy in general, that is doubly impressive.

Written by Niall Ross (nfr@bigwig.net).

* End of Document *