CS15 and ESUG 18, Barcelona, Sep 11th - Sep 17th, 2010

This document contains my reports of

- the ESUG conference in Barcelona, September 13th - September 17th, 2010 (and the Camp Smalltalk during the weekend before it)
- talks from the Camp Smalltalk conference in London, July 16th - 19th, 2010 and from some UK Smalltalk User group meetings in 2010.

I have put these 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.

Camp Smalltalk 15 and ESUG 18 in Barcelona, Sep 2010

I flew direct to Barcelona from Edinburgh, reaching it at 01:05 on Saturday morning. My hotel was very convenient for the site and the airport but, as several others who took taxis found, it was not very findable by the taxi drivers; mine drove past it and had to turn around, which I gather is better than the performance of some taxis that others had.

Instantiations provided food and drink on Monday evening while the award entrants demoed. On Wednesday evening, we went to the science museum, which had several fascinating exhibits, none more so than the magic tricks section. One of them was 'all done with mirrors' but very well done. The legerdemain of the three-shell game was watched ten times or more by several (including me) and while we could *deduce* how it was done I think noone could actually *see* how it was done. (As Dale and Monty both had their pockets picked in downtown Barcelona earlier in the week, we felt more than just an academic interest in improving our ability to spot such legerdemain. :-)

Summary of Projects and Talks

I give the Camp Smalltalk 15 summary, then the ESUG activities reports and the awards. Next I summarise the conference talks, sorted into various

categories:

- Applications and Experience Reports
- Agile Methods
- Coding and Testing Tools and Techniques
- VMs and Development Environments
- Frameworks
- Explaining and Enabling Smalltalk

followed by the 10-minute talk track and Other Discussions (which icludes BoFs). Talk videos are available at http://esug2010.objectfusion.fr/. Talk slides are available at http://www.slideshare.net/esug/tag/esug2010. Talks I missed that you can see from the above include:

- Georg Heeg's Excel talk on using on using Smalltalk to overcome year-by-year changes (natural as Excel spreadsheets evolve) when doing multi-year analyses
- Estaban Lorenzano's talk on Reef, an AJAX / Javascript framework for Seaside (http://squeaksource.com/Reef)
- Alan Knight's talk on recent Store changes

Camp Smalltalk 15

Camp Smalltalk 15 ran for Saturday and Sunday before the conference, and during the conference breaks, afternoons and some evenings of the five conference days. There were 50+ people there. (Some were working on tasks they later gave talks on, so I have integrated some CS observations into those talks or into 'Other discussions'.)

SUnit 4.0 Porting

I paired with Jan Vrany to port the SUnit project's latest release 4.0 to Smalltalk/X.

For Smalltalk/X, we integrated SUnit 4.0 RC1 into the base and also into a variant that remembers tests. The variant adds collection-holding class instance variables lastTestResultsErrors, lastTestResultsFailures and lastTestResultsPasses to TestCase. In each subclass, these are populated by code added to `addPass:`, `addFailure:` and `addError:`. Each TestCase subclass therefore has a list of the results of its latest tests, which ensures these are cleared when that class is unloaded or reinitialised. However they are not updated when an individual test method is changed or deleted; SUnitToo in VW uses a cache class and the change mechanism to do that. Both Linux and Windows versions of Smalltalk/X are published: navigate from https://swing.fit.cvut.cz/projects/stx-goodies. The `errorObject` method returns Exception in Smalltalk/X. Usually it is Error (and that is an ANSI thing) so Jan will check the effect of making it Error, but we assume it was set to Exception in ST/X for a reason.

James Foster and I then ported 4.0 to GemStone. This established that `sunitOn:do:`, `sunitEnsure:`, `sunitSelectors`, `sunitAsSymbol`, `sunitName` and the deprecated `sunitAllSelectors` are no longer

needed by any of VW, VASmalltalk, Pharo, GemStone, Smalltalk/X, VSE or Dolphin (thanks to ANSI), so it is time to drop them (for 4.1, and they are already dropped in Pharo). Among classes, SUnitDelay looks as if it is no longer used. Since `sunitMatch:`, `sunitAddDependent:` and `sunitRemoveDependent:` simply call their sunit-less namesakes in VW, VASmalltalk, GemStone, Pharo and Smalltalk/X, perhaps they too can be removed.

For the present, the project has no plans to remove SUnitNameResolver, `sunitAsClass` or `classNamed:` (these may make us a little more robust to introduction of namespaces in dialects, etc.).

Tim MacKinnon introduced me to David Gorisek who later helped me upload the already-completed Dolphin port to the Dolphin public repository.

I paired with John O'Keefe on SUnit. John removed some old code from the SUnitBrowser widgets. Pharo has 'expected failures / unexpected passes' behaviour (it is preserved in the SUnit40mergedWith105 branch in PharoInbox). VASmalltalk would like to use this pattern too. I looked at whether this can be made pluggable using the 4.0 pluggable pattern.

For a moment, it looked like Pharo was publishing inconsistent binary and .st files but Lukas verified from the checksum of our SUnit.40.mcz file in www.squeaksource.com/SUnit that it was just a hiccough in this one case: we published OK binary and then the connection must have dropped while the .st was being written. We published a fully-correct version, and the package that merges SUnit 4.0 into the standard Pharo SUnit in PharoInbox is fine. Lukas also added the SUnit 4.0 port to the bug/feature list; the Pharo guys will not see what you publish unless you put it in the list.

Custom Refactoring Browser
Squeak/Pharo has (long ago, I expect) renamed a number of the RB classes (ParseTreeSearcher, ParseTreeRewriter, CompositeLintRule, LintRule BlockLintRule and TransformationRule) from <ClassName> to RB<ClassName>. Now that Slime's tests uses the RB to generate temporary classes for testing, this creates a clash when Slime is imported to other dialects. John O'Keefe and I resolved this by creating trivial subclasses (named RB<ClassName>) of the few affected <ClassName>s. Then we handled the creating method call. In Envy, the RB calls `defineClass:in:`. In non-Envy, it calls #defineClass:. Making this method convert the requested category to a suitable Envy application lets the requested tests run.

Glorp
Jan van der Sandt helped me work out how to make Glorp use bindings in VASmalltalk. This work continues and will complete soon, I hope.

Seaside 3.0
Lukas et al were working on finalizing Seaside 3.0. It went gold and was released on Sunday afternoon. The 2.8 APIs are more deprecated than

removed, so porting your Seaside app should be fairly straightforward; there are suggestions on the wiki.

A check of commit numbers against Seaside, Seaside2.9-old and Seaside3.0 showed Lucas Renggli first, Julian Fitzell second, Philippe Marschall third and Michel Bany fourth.

Monticello / Metacello
Adriaan and Soemirno worked on a Monticello import facility for the VastGoodies utility. Dale worked with Jan Vrany and others on Monticello and Metacello in various dialects.

Overrides are handled in standard Monticello as in VW (but arguably the feature is far from finished - mind you, we could say overrides in VW are still being perfected). In GemStone, this feature is not supported; Dale feels it is safer simply to admit there's a problem and be forced to solve it.

GemStone
Martin paired with Alan on a GemStone backend to Store. Monty and Norm worked on various things.

Pharo
Many people were working on Pharo issues or Pharo projects. It was noted that people writing cross-dialect utilities could save themselves trouble by using hyphenated class category names for Pharo-compatibility.

ESUG Activities

Conference Welcome and ESUG Activities Overview, Stephane Ducasse, Noury Bouraqadi
Stephane thanked the sponsors: see their logos on http://www.esug.org/conferences/18thinternationalsmalltalkjointconference2010. (There was a time when ESUG had to work hard to get any sponsors. Now every year the slide is more and more crowded with logos; if the trend continues, Stephane will soon need two slides). Having these sponsors makes Stephane very happy because it means they are doing a good job.

He thanked Citilab and especially Jordi, the lead local organiser, and all the locals who assisted (long list of names on the slide; Stephane handed over to Jordi, a native Spanish speaker, to get all the names pronounced perfectly). Jordi translated all Stephane's books into Catalan. Three years ago Jordi was not doing Smalltalk; today, he's organising a Smalltalk conference. The Citilab infrastructure is really good: for example, all talks will be streamed and put on the web.

Stephane asked how many were attending their very first ESUG? Quite a few hands were raised.

The ESUG conference is the only revenue stream ESUG has. From this revenue, ESUG sponsors Smalltalk in various ways:

- ESUG can sponsor presentations of Smalltalk, i.e. pay travel expenses, etc., 2 so far in 2010, 6 in 2009, etc. You contact ESUG and they decide.

- ESUG has sponsored Morphic 3.0, Glamour on the web, iPhone development (the project is restarting now) and Dr Geo II Scripting. Stephane also looked at past SummerTalk projects, both those that worked and those that did not (the latter included a bug tracking system - but there were too many bugs in it). During the conference, we will have presentations on SummerTalk2010.
- They sponsored 15 teachers (travel, coffee breaks, etc.) to talk about Smalltalk. They have sponsored a Russian Smalltalk 2010 user group, the Smalltalks 2010 conference in Argentina, Smalltalk.cat/es, and Smalltalk involvement in OSCD 2010 and J2ML 2010.
- ESUG sponsors student volunteers, so they can attend, meet the community, network. They wear red caps at this conference (not to be confused with the red shirt worn by StarTrek extras doomed to die :-)) so if you have a question, ask them. The students are here to learn more about Smalltalk; help them get in touch with the Smalltalk community by showing them your projects.
- Old ESUG conferences since 1993 are online at http://www.slideshare.net/esug
- ESUG has sponsored some books, mostly open-source but not all (e.g. Andres' Mentoring course is not open-source).

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

This year, we have 140 participants, including 24 student volunteers and 10 free entrance tickets, 42 talks (clearly Smalltalk is the ultimate answer :-)), an international workshop, a wolfpack programming session, an agile day, two 'secret talks' and a panel on licensing (there will be a lawyer on it). There are also the technology awards (15 submissions this year) and (for the first time) an award for the best book of the last 6 years.

Lastly it was announced that Lukas Renggli is looking for work (in Switzerland, and he'd had offers by the end of the conference).

ESUG Board: election and activities
The existing board are Stephane Ducasse, Markus Denker, Serge Stinkwich, Noury Bouraqadi, Alain Plantec and Damian Cassou. Jordi and Luc are joining. All were voted in.

Serge presented upcoming events: Mozilla/Drumbeat event in Barcelona November 3-5, Smalltalks 2010 Argentina, student volunteers at Smalltalk Solutions next March.

The next 'best Smalltalk book' award will be in 2012. The marketing award will be renamed to 'best Open-Source spreading action' award and will be voted on in 2011. ESUG is willing to help those who write such books as regards copy editing etc. (and will help Andres in this way). They would also like translations (especially of the Bots book; help wanted).

Next year they will have a Summer School, sponsored by INRIA.

Local groups are good. They gave 2000 euros for new local groups with mailing list. The French mailing list started as 3 people and now has 100+. There is a German Squeak list.

If you want to be sponsored you need to set up a non-profit organisation (Serge actually said non-profitable but I suspect he did not mean that :-).

They are setting up a web archive of previous ESUG DVDs, scanned free books, etc.

A Smalltalk teacher network has been set up in the US, using the various local Smalltalk groups: Ralph Johnson, Roger Whitney, Andrew Black (Niall: and David Shaeffer). Please tell them of anyone else teaching Smalltalk in the US.

They want to provide a 'one page, one click to get to all the current solutions' info resource about Smalltalk and Robotics, and another about Smalltalk and the Cloud (vendor help would be welcome).

More projects sponsored in SummerTalk would be good. (e.g. Xtreams).

Innovation Awards, Noury Bouraqadi

The main award winner is Smalltalk; we have great energy in this community. This year we had two new awards:

- The book award was won by 'Dynamic Web Development with Seaside'.
- The marketing award, for who has best spread the word about Smalltalk, was won by Damian Cassou for Squeak-dev and Laurent Laffont for the Pharo screencasts (http://pharocasts.blogspot.com).

There were 15 competitors for this year's Innovation Awards, and 195 votes were cast.

- **Physical Etoys** (Lic. Gonzalo Zabala, Ricardo Moran, Sebastián Blanco) came first. The demo programmed a robot to avoid obstacles (to move, to respond to sensor - and programmed sensor), using a very visual display, expanding and drag-dropping into the script. It took five minutes and then the robot was roaming over the demo floor, avoiding chairs, feet, walls, etc.
- The **Cog** virtual machine (Eliot Miranda) came second. Cog is a JIT for stack optimisation. It uses the C stack when it can, the real object when necessary, and maps between them. Pharo are using it for advanced prototyping, but it needs the right version of Pharo, with some patches. (Markus accepted on Eliot Miranda's behalf and took the certificate - "but we will send him the money directly.")
- **Mars** (Estaban Lorenzano) came third. This is an MVC framework providing native UI on OSX (Mac and iPhone). There are no halos yet. He demoed Pharo browsers running with all native widgets.

(At http://www.esug.org/Conferences/2010/InnovationTechnologyAwards there is more information.)

I also made notes of the demos of some other entrants.

- For **Torch**, I combined my notes with those on Veronica Uquillas Gomez talk; see below.
- **AutoTest** is in Pharo: whenever a method is compiled it runs all tests that cover it. When a test is changed, it reruns it.
- **SmallUML** (Carlo Grigglo) creates UML diagrams from code and facts entered by the user.

Smalltalk/X jv branch and XML Suite, Jan Vrany and Jan Kurs
On Linux and Windows, Smalltalk/X code tools are being improved. The jv branch adds

- semi-modal navigation via a popup list (somewhat like RBHypertext), plus back and forward in browsers
- annotations (i.e pragmas)
- selector namespaces: VM lookup uses method slot in context

The XML Suite handles parsing, SAX2, DOM3, XPath and XQuery support. There is an XQuery debugger, workspace and inspector.

Visit https://swing.fit.cvut.cz/projects/stx-goodies to get this.

STAMPY, Martin Kobetic, Cincom
STAMPY is a Smalltalk email application developed by Martin, who has been using it as his email application for three years; he finds it to be robust. It has the basic email features of course, so I concentrated on others. Martin clicked on a PDF attachment; the tool (saved it to temporary file and) had the OS open it. Similarly HTML email opened in the browser. There is also a simple Seaside app to read email over the web.

Martin's archive is 0.5Gb (100,000 emails). Performance seems to be OK: he showed it scanning this in 15 seconds on his very slow demo machine (it is much faster on his home machine). Searching is fine on fields (subject, etc.). When searching content locally he just greps but that does not always work correctly.

The main value of this to him is programmability for cases that matter to him - e.g. find all emails with PDF attachments larger than a given size and delete just the attachment, not the email.

GitFS and TextLint, Lukas Renggli
Colin Putney has a file system library. A student of Lukas used this to create a Git repository, i.e. a file system that versioned using Git, all running on Pharo. They demoed making some changes to files, creating new files programmatically, etc., and used a Git tool to see that old and new were all readable. Thus you can see the older states of your FS.

TextLint is Lukas' own project. It finds errors in papers written by non-native English speakers. The tool has 50 rules so far. It can read text, LaTek and HTML files. Lukas advisor used to complain a lot about his papers.

After half a year of use of TextLint, she complains less. Lukas does not always change what it flags any more than he always does what Smalllint says, but the tool is usually right.

Applications and Experience Reports

NeXTPLAN*: Enterprise Software that Rocks, Johan Brichau - johan@inceptive.be

Inceptive is a new company, spun out of the Free University in Brussels (see http://www.inceptive.be). NeXTPLAN is the company's first project. After 10 years in academia, Johan is starting this company to exploit Seaside and Smalltalk.

NeXTPLAN is a collaborative resource for event organisation in Belgium, running on Seaside in Pharo. The customer did the domain knowledge and the fancy screenshots. Inceptive did the model layer. In one year, they have produced 24 versions for the customer to test. It would take him two hours to explain the whole application, so instead he will simply show how productive Seaside and Smalltalk is in making web applications that give the user the experience of a desktop application. He will focus on the challenges because the rest is all standard Smalltalk and Seaside.

He showed the calendar: rooms (rows) cross-referenced against dates (columns). An event, e.g. a film festival, may be distributed over several locations and times, so each square is a separate entity with its relationship to the overall event known. Icons on the squares show status. Users edit in-place, react to pop-ups, etc. The user can change data in different parts of the window and in different windows, and the app must update everything displayed that is related to what changed without doing a full-page refresh. Seaside is very good for that as each part of the page is a component: what you edit is a component and what requires changing in response is also a set of components.

At first, they did a global Ajax update after every operation. This was easy and OK until the user had 30 rooms and 20 weeks of data. Then one event change was followed by six seconds before the screen refreshed. In an OO app, you know which specific components need to be refreshed when other components are edited, so they switched to that.

Next challenge: this is a multi-user application. Instvar values must be reinitialised before the component is rerendered, in case someone else has changed relevant data. Thus the app captures concurrent changes to calendar. Some Javascripts need to be relaunched and others not in this case - they had to work out and manage which to relaunch.

These same issues occur in application after application so they wanted a framework, but at the moment they have a design pattern to apply to each case: `renderContentOn:` (complete render), `renderInternalOn:` (internal render), `needsRefresh` (wants to render, i.e. should the component update, has it received any announcements - and also check the DB, has another user updated this?), `scriptToUpdateOn:` (script to render). All UpdateableComponents have `htmlId` and understand these.

Local Updates: sometimes you need to update entire component, sometimes just subitem. The latter cases uses AJAX or JQuery. For example,

```
CalendarComponent>>updateScriptOn: canvas
  ^canvas jQuery ajax script:
    [:s | self cells do:
      [:c | c needsRefresh ifTrue:
        [s << (c updateScriptOn: s)]]]
```

(or, more generally, `[:s | self children do:`)

```
Cell>>updateScriptOn: canvas
  ^(canvas jQuery id: self htmlId)
      load: [:r | self renderInternalOn: r]
```

(or `replaceWith: [:r | self renderContentOn: r]`)

They separate updates into own-session changes and concurrent-session changes. For own-session changes, they use true OO change dependencies. For concurrent, they must do a degree of manual handling. Making this distinction gave a 5x speed up.

The SeasideDynamicSVG package (Holger Kleinsorgen) gives you SVG packages and brushes. He showed on the interface how they used this to get an animated tree visualisation of the event being modelled. Problem: the customer wanted the lines always to be under the dates, etc., whereas SVG tends to render in order; the last thing in the html is on top, and you can only either load new content in the SVG canvas or add content at the end. They therefore added the ability to add content to existing SVG elements (will be released back into community soon).

They have used Glorp for logging but OODBs for the rest. Magma has indexes, nested transactions and write barriers. Alas, Magma performance was not adequate for them. GOODS has none of these features (the write barrier is 'implemented' but does not work). GOODS also did not suit so they started porting to GemStone one month ago. GemStone has indexes and write barriers (but its speed was such that they hardly needed these). They created a general database access layer and this made it easy to change.

A write-barrier relates to how, when there are loaded objects in your Smalltalk image, the database tracks which ones have changed. Having a write-barrier means you must choose to put the changed objects into the write buffer to get them written. Not having one means that persistent objects that change will always be written. He showed performance for 1000 commits and for the commit of collection of 100,000 objects (in the latter, GemStone was so fast that it could not be seen on slide :-)). Then he showed the effect of making a single-object change and committing: magma took 6 seconds, GOODS with the automatic write-barrier was the same, GOODS with their manual write-barrier was much faster. GemStone was far faster. GemStone only took 0.68 seconds.

The move to GemStone was in fact motivated by the resource-adding tool. Their demo test database has 30 resources. Their customer has 5000 resources. GOODS takes 20 seconds to load these, so takes 20 seconds when they first visit the interface. In GemStone it was almost instantaneous. (They could have pre-loaded in GOODS, of course, to dodge the problem.)

Their GemStone experience was all positive until they had two users in concurrent transactions. If user B changes the event's name, and User A tries to change the same name, they want to catch this so as to notify the user. In GOODS, they captured it as a transaction conflict because in GOODS they had a database session per seaside session. However in GLASS, they have only one session for the database. Another issue is that Seaside applications integrate with OODBs at request level but in fact this interaction is more like a conversation. Thus GOODS has its plus side for refreshing though it is very slow for loading.

In GemStone, loading is fast but refresh cannot be trapped in the database. However having implemented a manual write-barrier for GOODS, they were able to reuse it for GemStone; they can warn the user when they change an object someone else has changed by looking in the write-barrier. The API for it is

```
self session db
  executeAsTransaction: [...]
  forObjects: #(...)
  onSuccess: [...]
  onFailure: [...]
  retryIf: [...]
```

This call is the only database access call and is the same whatever the underlying database.

Using hybrid components: when a user is searching 10,000 components, they want to see first results fast, not after all 10,000 have been searched. Seaside has good Javascript that can help with this. They want to show the list of found items with an in-place edit wrapper on each; if 300+ lines were found, it took 2 seconds for the interface to become responsive again as those in-place edits were being registered.

You can register the URL for a callback with immediate accessors:

```
assignCallbackScriptOn: html
  ^(html jQuery ajax callback: [...])
     asFunction assignTo: 'someJsVars'
```

Thus you need client-side programming and callbacks to get the responsiveness.

Testing: unit testing and SeasideTesting. He ported the latest version of David Shaeffer's SeasideTesting framework from VW to Pharo.

```
testDeleteGroup
  ...
  self executeCallBack: [...]
```

lets him test, but you also want client-side tested so he recommends a web tester that uses Selenium (but David's latest version offers a connection to the browser that offers an alternative for client-side testing).

Experience: use Grease and Slime when porting. GemStone's compiler is stricter (no empty statements allowed, no shadowing, etc.) They had to map Exception classes (made ExceptionA classes). BlockClosure is called ExecutableBlock. The Date/Duration/Timestamp protocol is not perfectly compatible. Assignments to temporaries in older GS (before 3.0) was also an issue.

They expect to release their first beta version by October and be fully live by the end of the year: if you organise events, or know anyone who does, please consider it.

Q. Use Koch? Not yet stable and their Magma interface is now outdated but they have heard of it and will probably try it.

Q (Georg) There are similarities to issues confronted by SeaBreeze. Did you look at how SeaBreeze handles these issues? He had not known that and would be interested to look offline.

Q (Janko) Use image-based consistency? We wanted concurrency from the start so always knew we wanted a database, and the transaction conflict in the database was an obvious accelerator to handle the refresh problem. Their customer has 20 users already and 2 years of usage; a database-less image could have grown to 1-2Mb in that time.

The application is not open source but many of the things they have done have been moved back into the Pharo/Seaside community.

Design Principles behind Patagonia, Hernan Wilkinson, University of Buenos Aires

Patagonia is a conference management system used by ESUG and others. When you registered for this conference, Patagonia is what you were using. It was build on Seaside and has the MIT license; download it from www.squeaksource.com/Patagonia. The talk is not about the system but about design issues and solutions.

Abstractions are created from the technical domain not the problem domain. In e.g. the Reef Seaside component talk, *when* a texting is required is a business rule. Business rules are not always clearly modelled in software systems.

We want to write robust software that helps humans understand it. His approach is always to make humans do less and computers do more. He has learned these tips while teaching and working on business systems.

A first axiom is to have a computable model of the problem domain. Software development is a learning process and the hard part is to go from a natural language description to a formal description.

A rule of the Patagonia domain is that no two conferences can have the same name. No two talks in a conference can be of the same name. So how do you model this. A junior programmer might say that name is just a String. Hernan made a class Name in Patagonia that knows to compare itself non-case-sensitive.

Are symbols the same as selectors? No, symbols have uniqueness and only some of them are selectors. `#isBinary` is always a symbol but it may or may not be implemented. A programmer should never have to think "Is this a symbol or a selector?" Hernan wants programmers to think less. In fact, Smalltalk has many cases of one class representing more than one thing.

"Perfection is finally attained not when there is no longer anything to add, but when there is no longer anything to take away." (Saint Exupery)

So the overall rule is: don't be afraid of creating classes. Map the problem domain to classes one-to-one and never have one class representing two concepts. N.B. this does not mean create a class for each *role* that a concept plays in a context. Some people say this will cause confusion: he quotes Alan Key, "Remember, it's all about the verbs."

Another problem: how to create objects? Usually we create an instance, then send messages to set is values. During this time, the object is incomplete. A year cannot answer `isLeap` when it has its day set but not its year. This makes the user of the object responsible for telling what messages can be sent at a given time.

His answer is always to create valid objects from the beginning. Have instance creation protocol that creates complete objects only. When is an object complete, you will not see `nil doesNotUnderstand: ... .`

Q (Noury) Concurrency: your creation is not atomic so if you do allInstances in a concurrent thread you could still get invalid instance? True; Hernan thinks that case is rare enough that the culture of having complete instance creation protocol helps.

He showed the instance creation method for an Attendee (the class should be called Registree because an instance is created when a person registers but you do not know whether they will attend [Niall: also it should have been called Attender as it is the subject, not the object of the verb attend]).

What about objects that must be filled out later? Split them into two objects.

Should we check that our complete objects are valid? If an object represents an entity in the problem domain, then an invalid object represents nothing. So we should validate objects. It is not good to have the object do it or the UI. Hernan believes the creator, i.e. the object's class should have the responsibility to validate the object.

The domain model has business rules. The fee needs to be positive (we

want the conference to make money) and the overall fee must be greater than the one-day fee.

Thus we now have only complete and only valid objects. Thus the model tells us when we make mistakes; new people joining the development team will be taught what is correct by the system.

If we only have complete and valid objects, do we need nil? Hernan argues that no we do not need it. C.f. Tony Hoare's talk: 'Null references: the billion dollar mistake'.)

Never use the name Abstract in a class name (Leandro taught him that).

Can we use `assertion:description:` to pass text to the UI? Yes, since Error descriptions are in a single place, with the assertions. Thus notifiers can open with the description. Hernan wants to let them open with multiple descriptions, showing several errors in a single place.

Do we need setters? An object could become invalid after a set so have immutable objects. But some things change. An invoice cannot be changed after you send it to the customer. Have an object to represent the invoice form, on which you build an invoice, and then another for the immutable invoice thus created.

You can create a new valid object and then synchronise with the new object in an atomic operation. So his tip is, when changing objects, synchronise atomically. As a result, the UI is simpler: if the user cancels, nothing needs to be changed.

Objects are valid in context. We must model objects that handle sets of cohesive objects. Hernan calls such objects 'subsystems'. He recommends modelling the system architecture explicitly. For example, the Attendee must be from a valid company and country. He advises always checking everything - e.g. if someone gives a talk, check they are attending; this is important for concurrency. Structure these subsystems; have "one system to rule them all".

[Niall, post-talk: issue of circular verification. You need to be able to move to a new consistent state without creating a spurious difficulty of finding a path of steps each of which will individually result in a consistent state, i.e. it can be dangerous to make verification too aggressive and/or automatic.]

His final slide had a tombstone with the inscription, "Ran out of Time". These rules help newbie programmers, and to create culture.

Q.Do not use setters except for tests is better way to put it? Hernan has a system with 23,000 user tests and hasn't needed it yet. [Niall: I have needed setters for tests.]

Q (Noury) Do you teach the null object pattern to your new programmers? Good idea. His fundamental tip is "It is worth the time to think about what

it means for something not to be there". When migrating, they check for nils; if you see any, this means the migration was not correct.

Q. You advise a one-one map between classes and domain but actually you can have more classes because of implementation details, so do you advise public and private classes? No: private adds unneeded complexity.

Hernan uses test-driven development.

Q. Sometimes, initialisation order varies (e.g. you must evaluate the first parameter expression then second in one case, second and first in another)? Hernan initialises them in local variables. If you do not yet have enough data to create the object, you should split the object and model the event that will create the new data and you should model that event; then you create the new one and forget about the old one.

Q.Actually a comment, to recommend Martin Roberts' book, and also the book 'Object-Oriented Heuristics'.

Q (Boris) Only valid objects exist in the system then the rules change so they are no longer valid, so how do you change them to be valid? The validation system will make it easy to find invalid objects and what to do with them is another business rule. The update could be null object pattern (e.g. now everyone's birthday must be entered but some customers don't have it so they get the null date value) or by providing essential data or initialisation or whatever.

Smalltalk debug lives in the matrix, Loic Lagadec, Damian Picard, University of Brest

A hardware matrix of programmable connections is connected in specific ways to realise functionality. Such hardware is flexible, so lets you get to market quickly, but it is hard to program and debug. Simulation is not enough; you must directly debug the hardware.

If a circuit runs at 200Mz, how much can you pay per cycle to support debugging - obviously, not much. Debugging requires extra circuitry. Debugging facilities are circuits. Step, next, resume and reset are inputs (he showed the circuit diagram).

Could you use variables instead of signals? Full observability of the hardware is not scalable: you must focus on a point of interest. A key testing decision is to select your point of interest. They aim to perform several runs and stop at various breakpoints. Once 'bullet time' has been stopped (film 'The Matrix' used as illustration here) you can examine and then choose to restart. Thus you take classical code with conditional breakpoints and then extract the underlying state while time is stopped.

You implement the application in Smalltalk and generate a CSP-like description of it with inserted probes. At a higher level, you encapsulate circuits as Smalltalk blocks. So they make a model of the matrix. You can model a virtual matrix layer implemented over the real hardware. He

showed an example of a circuit display where he swapped in a block.

When investigating state, they freeze not the clock but the rest of the activity in the hardware, so this is real hardware debugging, not model simulation. He zoomed in (massively) on a hardware diagram and showed the input lines that enable this (and it was also a Smalltalk tool showing this). Finally he demoed.

How long till my feature will be ready?, Leandro Caniglia
This is a bad question because we do not know. Better questions are, "What is the probability the feature will be ready in one month" or "In what time is the feature 80% likely to be ready?"

Leandro reviewed some metrics. Lines of code (loc) is a poor metric; different programmers make more or fewer lines for reasons unrelated to functionality. Bytecodes may be a slightly less arbitrary metric. You can count the number of messages in source or in byte code format equally. Message chains (incursion) can be counted; are messages in the same chain. You can count arguments (arity) or sub-words. You can count lots of things. Stepping back a little, when you think metrics you think of getting a number as 'the answer'. Leandro thinks that is the wrong approach. These measures make sense for methods and only a few (e.g. lines of code) for classes. On classes, you can also count instvars or whatever.

Try thinking of these not as numbers but as probabilities. What is the probability that your system contains a method that has less than 100 loc, or a class that has10 instvars. Without this, you are taking about averages. He showed examples of probability plots and explained how to generate random samples: select randomly several points on the probability axis and project them onto the values. In summary, metrics are not numbers, they are probability curves.

He showed 6 graphs plotting the curves for various metrics. He looked at his method loc curve and showed it was very close to log-normal curve. The same is true of other code methods. Thus you can approximate your metrics well enough if you know the mean value and standard deviation and use that to create the log-normal curve that would fit them.

Development metrics apply to the code - how many methods per class - and the change set - how many methods in it. For the development team, Leandro would like to measure speed and efficiency.

- By speed, he means the number of change sets that get integrated each day. Leandro has looked at the timestamps of the change sets of several years of development (and this is real elapsed time data so it includes holidays, etc.)
- Efficiency: not every change in a change set remains unaltered. Suppose a change set of three has one method deleted, another edited and only the third left unchanged. (Niall: this is what I call the 'churn' metric; how much time is spent churning around as against moving linearly forward.)

Q. So renaming a method is inefficient? Yes, you created it with one name (apparently, the wrong name) and then spent time thinking of a new name as against getting the name right first time. [Niall, post-talk: I note that one of the values of Smalltalk is the very low tax it imposes on refactoring, compared with other languages. This is a good thing so our metrics must return 'positive' numbers when measuring code where this has been done.]

Q. Time-frame? This is not something you are measuring all the time, just when you get asked 'how long my feature'. (The correct time frame is in fact the duration of the task, but that is what you are calculating: you guess the right order of magnitude.)

How can we estimate the size of the feature before we can do it. He guesses the number of classes and uses his probability graph to get a possible size. To get the number of classes, sketch a design of the feature. You will have some existing and some new classes. Weight existing classes as 0.5 the size of new classes in a system you know (perhaps reverse this in a system you do not know).

In real conditions, the development team has only so much focus (% of their available time) on the feature. Thus you can put all these metrics together in a stochastic model and make a monte carlo simulation. You can rerun this simulation with different random samples 10,000 times (takes very little time: his model is fast). Order the results by time and see the curve which is your answer.

He demoed by assigning values and running the tool, showing the result: a probability-against-time curve.

Box and Cocks (yes, those are their names :-)) wrote a paper on this.

Q. So this feature will cost between 8000 and 50,000? You can translate time into money directly but you cannot convert money directly into time: He can't build the feature in a second however much money you offer.

Q. Where is this linear? Nowhere, because though the individual runs are linear in all the variables, the graph is of many runs with log-normal probability.

Q (Niall) you will underestimate the number of classes, but at least you will easily see when your class list first exceeds your total? Underestimation always happens. Leandro is showing us a new method for underestimating.

Q. How to apply this to a new development, new team, new system, as you have no data? You could use the log normal distribution and estimate as best you can its two values.

Domain-Specific Modelling, Cristophe Allegrini, Bird Technology
There are two types of language: general purpose languages, such as Smalltalk and Java, and those focused on a specific domain.

A domain-specific language represents the rules of the domain in the language. Salary calculation rules or insurance rules are complex. Having a language in which the words are the actions and subjects of the domain simplifies the user's task. Domain experts can contribute where their ignorance of a general-purpose language would block them. Thus we specialize the general purpose language to create the domain-specific language.

Having a homogenous set of words for the concepts of the domain is important. In a DSL, reusability is an obligation, not just a goal as in generic OO.

Bird Technology use VisualWorks to create their semantic middleware from which DSLs can be generated. Smalltalk is a real OO language. Java is not. When we speak in English or French, we can see the meaning in a dictionary and the explanation is also in French or English. You can do that in Smalltalk, not in Java.

He then demoed.

Q. People sometimes ask about tooling for DSLs; debugger, editor? They use VW because they can create these tools.

Agile Methods

Lessons Learned in Agile Development, Johan Brichau

Johan was forced to be agile when working in a University. Then he went into the real world and this means adding a customer into the loop.

Aside: there were legal issues on their project so they were working with a legal firm in Belgium and had side discussions on the legal issues they had to handle to do with failed IT projects (and learned they did not want to be sued by these guys). These lawyers were having to learned about agile but the courts and the laws are not agile, so for them you may have to have a paper trail to prove you did formal acceptance, code quality monitoring and have traceability to show you did things to make the project work.

Pair programming, *if* it is done on a per-need basis is a great help - it saves time and increases the learning curve - but they did not try to enforce it at all. Tests were also much used. (Tim: in London, pair-programming is a non-issue; the banks are used to seeing it.)

Requirements are the bottleneck. Waiting for feedback can slow you. Doing things and then reverting when the prototype you sent them for evaluation two months ago is finally looked at can slow you. So monitor that. Have clearly identified first point of contact but also make sure who else is looking at it. Don't have the system be looked at only by your contact. Help them to avoid getting stuck on some early point that prevents them seeing others.

Q. When the customer does not give feedback, perhaps also be alarmed? Maybe always put one bug in; no report of it means they never looked at it.

When you demo, you get different feedback from when you just give it to them. It is hard for customers to assess the time-cost of features (in both directions - at times they were reluctant to ask for something that in fact was easy to do).

Q (Niall) give them 100 points (or jelly beans or chocolates) and get them to allocate them to features to say how important each is to them, not how much it will cost? Yes, informally they did but be aware your customer may be being influenced by their subjective (and wrong) ideas of how hard it will be for you to do.

Over the course of one year, they build up a test database of what they must do when they start, so be aware of the value of that. Maintain that database from iteration to iteration and describe the functional delta to the customer. This is work and you get 'bugs' due to bad migration not bad development. (Lucky the customer is good-tempered.)

Agile process + Fixed budget = uncertainty. Split the project into modules and see how much of each module is done so you don't use all the budget.

Q (Diego) We made customer responsible for budget; offered features with points and they chose from the overall budget? (Joachim) Would the customer feel you are gambling with them? Every change needs to prioritise and also to monetise but customers are unused to those processes. (Tim) Perhaps build trust by doing fixed price features first, thus get trust to do agile for the harder parts. (Cor Venter) We graph requested features and put them in a matrix with estimated cost size, then another whose axes are how important and how well it is being done now (and accumulate the dependency cost into what they choose). (Niall) Developers cannot well estimate how long implementing a feature will take. If the customer has rated how important a feature would be to them then at the end of an iteration you can map what you have done to how much you have earned. At the start, your estimate of how much you will do is unreliable. (Hernan) In fact, it is all about trust; your strategy must be what earns trust from your customer. perhaps it will see you working some extra hours.

Q (Annick) automate tests for your GUI interaction? Routine testing using Selenium and Seaside testing.

Agile Web Development and Seaside, Lukas Renggli
Who have not used Seaside? One hand. Who has written a component in Seaside - many hands (one of them mine) but far from all. So some people are afraid to admit they have not used Seaside. He will summarise Seaside, then talk about agile. (Lukas has heard about agile but not read the books or attended the courses.)

Cmsbox is written in Seaside, as is dabbledb (recently bought by Twitter). In Seaside everything is built from reusable components. He showed a management application: displayed components connected to their models. Seaside control flow is simpler and more natural than that of many other web frameworks. You can show the development tools in the browser,

bring up halos on components, etc. Seaside integrates with jQuery and prototype, and recently Dojo and DojoX have been integrated (with scipt.aculo.us a bit receding for current use).

Lukas structured his presentation around the agile manifesto to focus on individuals and interactions, working software, customer collaboration and responding to change.

1) Prefer individuals and interactions over processes and tools: the manifesto targets businesses and Seaside is very dispersed. He showed a world map with locations and commits - no two of Seaside's developers live in the same city. Thus they must come to conferences like ESUG and have sprints (he showed the Amsterdam sprint pictures: it started in a cafe from which they were evicted for thinking too much and drinking too little, so they found an ideal place in the Amsterdam public library). They want to do more of these - if you can host a sprint, let them know.

An open-source project has motivated individuals behind it: he showed a picture of Lukas, Philippe and Julian, and another picture of people presenting at conferences (Lukas, Jim Robertson, John O'Keefe, Dale Heinrichs, Adriaan van Os et al). Seaside's web presence is visible outside the Smalltalk community (he showed pictures of webpages) as are books and articles.

All this fits well with agile but as they are distributed, they must have processes and tools. The issues and commit list currently has some 13 frequent contributors and they are the core team. The bug tracker has many bugs, some marked easy, some are for specific platforms - maybe yours, so give it a try.

Squeaksource is where they host their code. The repository is no longer public - having it so worked well for six years but recently there were accidental commits - private projects accidentally published by people who had the repository in their list. Just ask in the mailing list for commit rights and you just have to add your username to the monticello configuration.

2) Working Software over Comprehensive Documentation: in an open-source project people really need documentation. He showed the numbers of commented classes in 2.8 and 3.0; all the public API classes are commented in 3.0 although, since 3.0 has more classes, the percentage commented has gone done. So we must not take this agile doctrine to extremes. (Stephane recently browsed some code of Joseph and liked the fact that every non-trivial method had a comment.)

3) Iterative Development: the Seaside 3.0 release was not incremental - it took two years to be released (it was released last Sunday). We will avoid doing that again (applause). However they also have lots of iterative cycles in development (as Joseph said, the Smalltalk debugger lets you do this and Seaside shows the stack trace on the web, etc.). They are always using the latest Seaside to develop Seaside. Thus Lukas has been using Seaside 3.0 alpha and beta all the time.

4) Working software is the principal measure of progress. The Seaside project agrees.

5) Continuous integration: they use Hudson (which is written in Java but it works really well and needed only a little work to make it work with Smalltalk) Yanni Chiu, Lukas Renggli and Philippe Marschall created this process for Seaside. Lukas gave his students projects at the University of Bern. Philippe Marschall told him to replace the bash scripts that managed these with Hudson, which Lukas never thought of using in Smalltalk; it just made his assistant work easier in Bern. Yanni noticed Lukas was using it, so wrote scripts to build Pier, etc., and Lukas noticed that. The source code is a collection of Smalltalk and bash scripts. The bash starts Pharo images, and the smalltalk scripts load code, run tests and collect coverage. Another bash script builds one-click images, another resizes images, etc.

He pointed the browser at the URL of his home server's project list. When something is published in any of the long list of projects, he can create a new image to load it and exercise it. Another page shows what image has been built (http://hudson.lukas-renggli) and what the current test state is (all run and passed) and the checkstyle results (some issues, a few red).

To set up your own server you need to know how to glue these scripts into Hudson. He showed the config page. First, specify the projects that this project depends on - that tells Hudson to trigger this build when one of the other projects build succeeds. Second point it to the squeaksource repository and whenever something changes there it starts build. It checks every two minutes; usually, if Lukas commits and then goes to check, he sees that Hudson has already started the build.

The scripts are shown in the page and need the input image and the output image, e.g a line to load Seaside, another line to load Comanche, a command to load Swazoo (will test against both). The last line builds the one-click image (the scripts are documented to help you). Then there is some meta-info for Hudson and lastly how to show test results and etc. on the Hudson page. The Smalltalk code generates JUnit reports so Hudson can understand them. He showed a coverage analysis on PetitParser, and a Smallint analysis (displayed as CheckStyle-Java format).

The PDF for the Seaside book is on the web: buy it and every week you can get new versions (online version is free, PDF version is 14 euros, print version is 28 euros but they get less profit).

Customers Matter: in an open-source project, primarily Lukas is his own customer. Other customers are the consulting clients, the mailing list participants and the platform vendors (Grease was written for them). So if you have a problem, ask in the mailing list (it may already be fixed). If not, create an issue in the bug tracker. Then wait for someone to fix it, or fix it yourself and submit and it will probably be in the build next day, and/or earn the commit rights.

People matter: who committed the most code since Seaside started? Avi,

Julian, Andrew, Michel, Lukas, Philippe. It was down to the last two in 2008 then Julian came back and now Nick has joined too. Seaside started on Squeak. Michel ported it to VisualWorks. By 2006 it was on Dolphin, 2007 on GemStone, 2008 moved the base platform to Pharo and ported Seaside to VASmalltalk. The Javascript libraries appeared in Seaside in 2004 with Seaside Asynch (Seaside-specific) then Prototype and script.aculo.us in 2005, then Comet in 2006, then JQuery, JQueryUI and WidgetBox in 2008 and so it carries on.

The logo: he showed the very first 2002 logo, then the 2005 logo, then when web 2.0 was available they got their current logo.

He showed the first website page - very clean and minimal. In 2003 it was less minimal and even less in 2005 and 2008. Thus we Respond to Change.

Where to go: "We've improved performance and memory footprint - no, we've reduced the performance and memory footprint - no - well, you know what I mean." Many other things are better. Some things are to fix (e.g. lightbox only works with script.aculo.us).

Q. Were the SUnit output to JUnitReports easy to do? The report list all the tests and results. All the magic is done by Hudson. They just converted SUnit test results to XML tags. You can download this code from SqueakSource. They had to do some tricks; JavaReport wants to know what namespaces a class is in so they used the category name.

Q. Magritte 2 and Pier 2? Not too many changes. In Pier, markup has been extended. Mainly it is just a new version to keep in synch with Seaside. Both are fully backward-compatible.

On the Extremes of Extreme, Julian Fitzell and Jason Ayers, Cincom
Julian explained they do not know where the work they are presenting is going but it is clear that when we work out how several people can work in the same codespace, it will change how coding is done - they have seen interesting results so far.

Wolves are packs of 6 -25 and the alpha male and female are usually the father and mother of the pack. They patrol their territory looking for herds to target and, once they have found a herd, which animal to hunt. The latter task begins by each wolf testing individual animals to see which one to hunt. Then all concentrate on the selected animal. If the prey runs, they chase it. If the prey stands, the wolves circle it to contain it and two wolves at a time worry it while the others rest, changing pairs from time to time.

The wolfpack approach is to have 6-8 programmers, two of whom are the alphas, addressing a problem in the same codespace. First each person explores a way to implement the feature. Then the prey is selected by the alphas, i.e. which implementation strategy will be pursued. When it looks like they have a handle on the problem - the prey is running - everyone codes on the solution - chases. When the prey stands - if there is an obstacle - two work on it while the others 'rest' - watch, write unit tests, think.

Kent Beck: “XP is best practice taken to the extremes”. They think he was too timid and they are being extreme. When you put a whole lot of people in a space, that is like living in a virtual reality

```
theCode: is
  ^[:your | reality]
```

Pairing is a restricted view (like having one avatar shared in a virtual world) as you have one keyboard and mouse and so on. In wolfpack programming you have 8 people in the same space each of whom can be looking at one or another method, coding, whatever: n-way programming instead of pair-programming. (One thing is missing in their current environment: you cannot follow where someone else has gone. You can see their changes to the method you’re looking at but if they go elsewhere that is not visible.) Within this process, pairing and (especially) tripling is common. Groups of four or more are less common, probably because of the limitation of physical space.

Communal living forces you to be a good citizen. Massive changes to the class hierarchy have to be deferred and communicated - you must communicate what you are about to do. You must be cleaner - (un)like living in a student kitchen; if you leave debris - code that does not work, you need to clean it up.

Continuous integration isn’t: it is like the edit-compile-run loop. If you live in a common code space then integration happens on a method by method basis. (WebVelocity compiles any method being edited whenever its state allows compilation). As soon as you change a method, you affect everyone else. They believe that as teams get use to this they will find mechanisms. The wolfpacks they’ve observed struggle with it. “Please tell us how to do this.” “That’s what we’re studying - what you choose to use”. People subclass or put name prefixes/suffixes and these are temporary solutions while they think of what the support should be.

Collective craftsmanship: people pursue individual spikes, then alphas choose a spike and the team pursue it. There was much discussion; “found this, how do you do this” happens a lot (so in a virtual environment the communication channel would have to work well). In a typical example (in a group not knowing Smalltalk) in the sudoku problem, someone said “I can do this for a cell, but how for the whole board”, someone else said “I know that” and so it progressed.

If you program on your own, you hit roadblocks. In a larger group, it’s more likely someone else in the team moves you over it.

Shifting sands: the code is continually changing underneath you. “Someone just changed my method” It’s not your method, it’s the team’s method. Your method may change or your test suite may be changed. No one person in the pack can follow what everyone else is doing. Strategies to solve this:

- continuous testing: each test needs awareness of its scope so when something in scope changes, test is run, but that scope changes, so better just run all tests all the time.
- test goes red, what changed and who changed it. 'Heat' represents where people have been (hot means changed recently, cooler means changed longer ago). 'Scent' is being able to see who did the change.

Wolves do not own territory. Wolfpacks control territory. A wolfpack may control the code it fixes.

Issues: what should be the role of the alphas? They have observed three kinds: delegators ("You do this, you do that"), first-among-equals (who let people get on with it, facilitate, reign in people who go to far), and equals (who are not really alphas). They are asking 8 strangers to use a process they don't know on a tool and language they don't know with 2 minutes to pick alphas; this is not much like the process in a real team where people know who are the alphas.

"You will have problems selling this, since it's hard enough to make managers allow pair-programming." Everyone works on top of each other. Everyone works on different spikes and then 3-5 get thrown away and one is pursued. During 'standing', two wolves work on the problem while the other 4-6 are 'doing nothing' - but in fact much of a programmer's life is thinking and writing tests and standing back from the problem. Just as with pair-programming, they think the end-of-day productivity will be higher.

There are also business situations, e.g. a crucial bug in a live system, where 8 guys would be given the problem.

Most trials were of people who had no Smalltalk: they gave them 9 slides and got them started. They've seen people talk for 3 hours on how to start Haskell and you come out thinking "interesting, but the speaker lost me on how actually to program". By contrast, people actually programmed in their scenarios and were saying, "I've never programmed {in Smalltalk} and here I am contribution to solving a problem" (Warning: maybe Smalltalk is cool, not wolfpack.)

Dispersal: as the junior wolves get more senior, they get more argumentative, leave the pack, find a spouse and try to find their own territory and form their own pack. So perhaps people could leave the pack (*provided* they have one other person) and find a bit of the codebase to maintain.

Most of the participants seem to have enjoyed it

Q (Christian) what problems fit this method best, and how sure are you that selecting just one spike will find the best solution? What kind of problems - no idea (their trials use sudoku puzzles) The alphas can switch between modes so you can junk the chosen spike and go back. In one trial, after the spike argument, the alphas picked spike A but a spike C pair thought it was still was the best so the team got A working and the C pair contributed but

still in background worked on C (and did get it working).

Q. Benefits are long term; solution is better and that will appear; how to test it more? Reply was also a question: do you know a company that will try it? Julian would like to find a team of 8 that will commit to doing it a few times and/or in an existing team already set up and knowing each other.

Q. Process is intense; do this every day? Real wolves do, but fair question.

Q. Wolf can catch mouse by itself; pack is needed to tackle what is too large for one wolf. The sudoku problem could have been solved by any one participant. Use harder task? They have two challenges. When people have never seen Smalltalk, you must expect less. Their first ever trial asked participants to build a test rig to test the ANSI-standard class hierarchy. In the trial's 3.5 hours, that was just too much for those 'wolves' and their level of experience of the process.

Q (Niall) Is the difference between standing and running precisely the difference between "If we run now we would conflict" versus "The intended solution is clear enough that everyone can follow their nose on it without conflict?" Jason thought no, the solution is better tools. Julian thought tools would reduce conflict but yes he agrees, for a given level of tool support the alphas decision to switch from running to standing should be exactly that criteria.

Q. Feedback from participant? Yes, solve sudoku is a better trial puzzle than e.g. do a Monticello increment because participants start with similar levels of domain knowledge. His trial group were crazy geeks and got excited about the solution rather than exercising the trial; maybe try to do the trial in 20 minute iterations to keep the focus on the process? They bounded the process by time in the way they explained it and should avoid that; 5 minutes for the spikes is fine if that gets the info. (And not solving the problem is not a failure - the experiment has yielded results.)

Q. Finding a friendly company will be hard. Try a friendly university? Good idea.

Q. Long ago, the questioner offered a very visual approach teaching a development method to Siemens. "What do we need to use this method?", they asked and he replied as joke, "A room with wall-to-wall whiteboards." They took them seriously and built it and it turned out to be a great project room because you could put everything up and see all the info. Make all the info visible in the virtual environment? Physical face-to-face is good.

Wolfpack Programming, Julian Fitzell and Jason Ayres, Cincom
Wolfpack programming is a development process for when 8 or more people are working on the same Smalltalk image. Smalltalk's history is to influence how people build software: agile, CRC cards, etc. Wolfpack is an experiment in progress; if it is successful, we may do so again. They have run this workshop several times already.

The two areas it pushes are:

- pair-programming: now you have a 'wolfpack' of six or eight, not a 'pair' of two or three
- continuous integration: we are all in the same image (the environment is WebVelocity, which gives you a single image accessed by many people through their web browsers)

The workshop is about the process, not about Smalltalk.

We are going to find a solution and implement it as wolves find a prey animal, hunt it down and kill it. ("If it sounds rather ruthless, hey, it's just an analogy." :-)) Wolves must hunt most days to stay alive. Wolves are family-oriented; there is an alpha male and alpha female, usually the parents of the others. Similarly our packs will have two alphas who will act like slightly dominant parents. Firstly, the wolves search for a herd of animals. Our analogy is to select a problem on which to work. Then individual wolves test various prey animals seeking the weakest. We will individually (or in pairs or triplets) test a possible solution for 30 minutes, ending with up to 8 partially-implemented solutions. Then the team discusses which approach look like working and which do not, and the alphas decide which solution the team will pursue. The next phase is the chase, which has two modes: running (the mode you start in), in which everyone works to solve that problem, and standing, in which two work on the problem while the others discuss, write tests for it, whatever. The alphas move the team from one mode to the other and in standing they swap from time to time which two wolves are directly attacking the solution.

Most people who have gone through this have enjoyed it. Julian handed out sheets on the process and on how real wolves hunt, and on the problem, a sudoku puzzle. As we were all on web browsers, he warned us against looking for solutions on the web (that would defeat the exercise) and also to avoid brute force solutions (so apparently, we're not trying to replicate *every* aspect of wolf behaviour :-)).

Julian then took over and explained WebVelocity (also gave us a 5 page handout on WV for later reference). Web Velocity 1.1 is a light Smalltalk environment with the debugger but not all the refactoring tools. The link at the top takes you home. The link below takes you to the sudoku app, with its test classes and domain classes. Click on a class to see its methods and/or run its tests. If you restart and proceed in the debugger it will effectively start a new debugger so yours will say the process is terminated; you must go back to top. (WebVelocity will alter that soon but the fix was not available to us at that moment.)

The console is your workspace (no automatic variable bindings) in which you can execute code and inspect variables.

Whenever you edit a method and reach a compilable state, it is compiled; there is no save button. ("If you don't want to change it, don't edit it.) A red border shows that it is not compiling; as soon as that disappears, it is compiled. The history shows what everyone has done, not what separate

individuals have done.

The wolfpack I was in introduced each other: 3 Pharo-using teachers, one VASmalltalk user, Jan (Smalltalk/X), two student volunteers, and me.

Our first discussion was to choose alphas. The rest of group bullied the two who admitted they were teachers after Jan vigorously refused. (Question: are the ones the rest of us can bully into doing it really alphas? See Julian's remarks in the 'extreme of extreme' talk.) The alphas told us all to take 10 minutes to explore the problem model then share, after which we ran the process more or less as defined.

A key issue that emerged from this run was the importance of the sonics of the environment when a team of 8 people are communicating. The curved brick ceilings and solid floors of the Camp Smalltalk room contributed to remarkably strong echo and amplification of sound, and the fact that we'd set up the four teams right next to each other (in retrospect we realised setting up in the four teams in the four furthest corners of the space we were in would have been better, although it would have obliged Jason and Julian to run from team to team) meant that intra-team communication was having to compete with the communication of the three adjacent teams.

Then we discussed and looked at the system for ten minutes, and sorted out some usage issues. Breakpoint / debug titles were long, so we suggested having the 'x' that deletes at the start, not the end as it was tedious to scroll to the end screen and tedious to keep moving mouse to get onto the next x. After the conference, Michael made this change in the next version.

Time, the echoing sonics problem and some other hiccoughs meant we did not complete the solution. This was the first time the exercise had been run with experienced Smalltalkers (IIUC); not being able to swap between the WV environment - which they liked for various features - and the classical one that they knew, was unnatural to them and thus a distraction. Like other teams that have done this, they demanded advice on avoiding/resolving code clashes. The participants' being experienced Smalltalkers may have meant they each coded faster than in other runs, thus making this a more serious issue.

Overall, the conclusion of those participants I spoke to was that while this might well be a good way to get non-Smalltalkers to encounter Smalltalk (because the wolfpack programming analogy was easy to grasp and might be interesting to explore in a case where the simple invitation "Let's do an exercise in Smalltalk" would not get the same sign up), it was a bad way to introduce WebVelocity to experienced Smalltalkers. The unresolved issues of Wolfpack programming - especially the "Who changed my method?" issue - made grasping the WV IDE more complicated (as noted, Smalltalk experience may make it more severe).

Pomodoro, Stephane Ducasse
Conferences are best when you get something you did not expect as well as learning what you expect. Hence these secret unadvertised talks.

Pomodoro is a technique to use to deal with getting the important tasks done; when you find you would rather do anything than the task you *should* do, that you are cleaning the kitchen instead of doing the next task on your todo list. The pomodoro techniques work: Stephane knows someone who was doing Pharo work instead of finishing his thesis. Stephane taught him pomodoro and he finished it.

Pick a task. Work at it for 25 minutes with no interruptions. Stop. For 3 minutes do something else - anything else. If a pomodoro is interrupted, it is not valid. Every four pomodoros, take a longer break. Have a sign on your screen or whatever to say, "On a pomodoro, do not interrupt." See http://www.pomodorotechnique.com/ (can be read in 25 minutes).

Stephane then showed two videos. The first, by Tom Wujec, talked about the marshmallow challenge: build the tallest tower you can out of tape, string, spaghetti, and a marshmallow. Tom has run this 70 times. Usually, a team builds the structure and at the very end they put the marshmallow on it and that causes the whole structure to overbalance. Tom says kindergarten kids do it better than business school graduates. The latter are trained to find the right plan and then do it. The kids start with the marshmallow and build successively bigger prototypes. Architects and engineers do best (they average 30 inches), kindergarten children next, business school attenders worst, lawyers below average, CEOs better, but a mix of CEOs with executive administrators better. Offering a money prize made the first run poorer but a repeat run better (because they understood prototyping). See marshmallowchallenge.com. "Design is a contact sport." This task exposes the hidden assumptions that underlie projects and teams.

The next video was about motivation. The lecturer gave MIT students challenges (memorise digits, throw ball though hoop). When the task was mechanical, bonuses worked as expected. When task required more than rudimentary cognitive skill, the experiment reported that large rewards gave poorer performance. They repeated the experiment in India. Those who were offered the very higher bonus on cognitive tasks did poorly while the lower and higher median bonuses were about equal.

The lecturer argued from this that if you want engagement instead of compliance, self-direction is better. Paying enough to make money not an issue is motivating, but more is counterproductive.

[Niall: I, by contrast, felt that he was neglecting the temporary stress effect caused by his experiments. A golf club pro once remarked that while there were amateurs at his club as good (when on form) as he, he could always win by suggesting half-way through the game, "Let's make it ten thousand pounds a hole, shall we?", since he was accustomed to playing for such stakes and they were not. The "Who wants to be a millionaire" show works on the same principle. His subjects would indeed be stressed by bonuses well outside what they were used to *at first* but several of us, talking afterwards, felt that they - or we - could rapidly become accustomed. :-)]

An Australian company offered its developers one-day of complete autonomy: every Friday, people worked on whatever they thought best. Result: lots of fixes to their systems.

Mastery is getting better at things. People will do a lot of work to get better at what they do. Linux and many other open source projects are done because people want to master skills. This is the purpose motive as opposite to the profit motive.

When the profit motive becomes unhitched from the purpose motive, poor work results. People are purpose-motivated as well as profit-motivated. Using this when designing your organisation can help it.

Retrospective Coherence in Scrum, Joseph Pelrine, MetaProg
Joseph was in at the beginning of Agile in Smalltalk and has been continuing to develop Agile. He held up his membership badge: the Smalltalk-80 book.

The Icelandic volcano with the unpronounceable name saw several people stuck in places they did not want to be, including one man who missed his own wedding. There is a lack of predictability in the world but either you deal with it or you get swept from the market. Joseph's work with Nokia after the iPhone came on the market is a good example. Heracletes: "Change is the only constant".

For many things, we will know more tomorrow than today, so delaying the decision till tomorrow is better, and the key question is: how long can we wait. Errors in software are often the result of making decisions too soon. Joseph recalled my remark that static typing is an example of premature optimisation.

SCRUM is a great way to diagnose issues for your company. SCRUM does not tell you what to build or how to build it. Bad requirements or process will see you build bad product with SCRUM as without it; with SCRUM you may recognise the situation before the market does. Joseph has used SCRUM to organise museum exhibitions and rock concerts (and his wedding; he delivered on time and on budget and is still happily married).

The first vector is what to build, the second is how to build it, and the third is the people. Many companies have an old fashioned requirements method even when they use SCRUM to build the software: the customers get one chance to say what they want and if they get it wrong there's no process for correction.

How do you get a group of people to become a team. Just get the people together and they'll be a great team? Well, look at the Italian team in the world cup. (Joseph used to use the Swiss team for this example, but they went and beat Spain, spoiling his talk.)

Joseph clicked on and off a light switch and showed us that we could predict what would happen: a predictable system. People also have

causality but there are differences. Michael (professor at the local University) is good at organising great parties. So Joseph asked Michael how Joseph's upcoming party could be made a success. They talked about food and drink. But then Michael explained, "If you want a successful party, you have to invite my friends." But Joseph wanted to invite *his* friends. Michael offered him the playlist from the last party; just play that same list. And give everyone a piece of paper telling them when to eat and drink and when to talk, based on what they did at the last great party. Where people are concerned, doing the exact same thing does not get you to the same place; on the contrary, you are virtually guaranteed *not* to get to the same place.

So why *do* Michael's parties work. Michael's party has boundaries (his single-malt collection is locked away) and some things people like: good food and drink, TV with world cup on. Why do people congregate in the kitchen? Because the kitchen is an attractor. Set up attractors, watch the patterns, if they work well, amplify them, if they work badly, break them up. Michael monitors this, takes a person on their own over to a group, etc.

The factors we notice, e.g. who was at the party, can be mistaken for the causes of the success. The DJ is watching who are on the dance floor and deciding what to play: the list of what was played is not repeatably successful. The host is watching what people are eating and drinking and replenishing what runs low. It's the process that makes the party a success, not the artefacts that are left at the end.

Joseph sometimes tests people walking out the door, normally, then with some chairs in the way, then blindfold with (hypothetical) broken glass on the floor. People walk differently in these three scenarios. Repeating what you did in one case is not good for one of the others.

Many of us are experts, meaning we know what to do in certain specific coding and debugging situations. But in a complex system, you will know if it was a success only afterwards.

Joseph's rules: we don't make mistakes, we learn. If you punished children when they fell off bikes, they'd never learn. Babies learn to walk because they are praised for their first wavering steps; they get positive feedback and noone doubts it is possible.

Imagine your boss gives you a specification and asks for a time to implement: how long? You think maybe a week, so you tell the boss two weeks or three weeks. If you estimate too low, you'll be working after hours and weekends and will have to say, "We're not done yet." Joseph persuaded a recent client to drop the word 'estimate': instead they used the word 'forecast' because everyone knows weather forecasts are unreliable; they get better the closer to the time and you know afterwards what the weather was.

SCRUM is about getting feedback as quick as possible and responding to it without being 'punished'. Smalltalk's strength is in giving feedback

quicker than static compile-link languages. Unit testing gives feedback quicker still.

SCRUM says whoever takes the risk takes the decision. Your risk of having to work late means you need to own the decision.

When do you want to know you have a problem? After two weeks or after two years? (For large companies, sometimes you can omit the word 'when' in this question.) The sooner the better, obviously. XP is about having a rapid feedback loop.

Joseph learned from Kent Beck to ask: how much are you prepared to throw away? You don't know the future which means you may have to throw away something you're doing.

Java has made some major steps forward - towards becoming a legacy environment. Java has adopted good things but Smalltalk still has competitive advantage, and one of the great ones is that Smalltalk still has the most rapid feedback loop of any language.

Going to meetings made you agile like going to a garage makes you a car.

Q (Stephane) Retrospective Coherence? Tim is one of the people who pushes the 'retrospective movement'. It's only a mistake if you don't learn from it and 'learn' means you change your behaviour because of it. Do retrospectives at the end of every iteration. Half an hour for a week's iteration is adequate. Tim's wife Arlena worked on the eurostar. Every day they reviewed whether any problems happened that day. Joseph uses quattrostagione - breaking his day into four quarters and you can talk whether there's a problem every 90 minutes.

Joseph dislikes KanBan because they say continuous learning but they have no process for taking time out to do it.

Q (Niall) Pair-programming can be the hardest of the three XP practices to introduce? Test-driven design, refactoring, pair-programming: these are defined in order to give small feedback loops. Pair-programming is often hard to implement because people think working on you own gives twice as much time. Think about two buddies working out together; they work much better than when on their own, not least because they motivate each other. People study the psychology of pair programming (for example, he has an Open University friend and has seen the results of her experiments). It's a people issue.

Q (Annick) Wolfpack programming: 8 people share the image? Joseph thinks that having several people sharing the same *codebase* is pure XP. Having 8 people sharing the same *image* - hm!

Q (Noury) There is also the work of the team on multiple problems and their dynamics; retrospectives on that? At an agile conference attended by Luc, participants played roles working for Santa Claus and afterwards

analysed what worked and what failed. They started by stating their comfort level of discussing issues. If it was low, post-its were produced and put on the timeline.

That sounds like standard retrospective techniques (discussion with Tim). What you do differs whether you are in the middle or at the end. Is the group going to disband (and you want closure) or will it carry on and do a fresh project. Retrospectives are like looking under the water at the 9/10ths of the iceberg. How did we *do*, and how did *we* do?

Q. Is agile for software only or does it work for engineering? Joseph has used agile to optimise the emergency room of a hospital. The Eisenhower method splits things into the cross-product of important/unimportant and urgent/non-urgent. Important and urgent, do now, Important and not urgent, plan (with SCRUM). Urgent not important, delegate it. Neither urgent nor important, ignore till it vanishes or moves to another category.

Coding and Testing Tools and Techniques

A Parser in my Pocket: Modern Parsing in Smalltalk, Martin McClure, GemStone

When doing DIY, there are always two or three tools you use a lot that always end up in your pocket, not in their place in the toolbox. It's the same in software.

Martin wrote a recursive descent parser long ago just to learn them, then left them in the toolbox. Recently, parsers have become easier to use in Smalltalk, so he has brought them up to his pocket.

The paper "Parsing Expression Grammars: a Recognition-Base Syntactic Foundation" by Brian Ford combined some 70s ideas, some 90s ideas and two ideas Brian himself had into something neat. Following on from this, Gilad Bracha put an executable grammar into NewSpeak. In NewSpeak he could make the code look a lot like the BNF; we can't quite do that in Smalltalk.

Parsing Expression Grammar Parsers (PEG Parsers) are now being written a lot by Martin McClure, by Lukas, by Michael Lucas-Smith and others.

For many years, we've had context-free grammars, usually BNF or similar.

```
conditional ::= IF cond THEN statement
              | IF cond THEN statement ELSE statement.
```

These rules were made for statement generation, not for statement recognition, whereas now we want humans to generate and the computer to recognise, e.g. in the above it can be hard to see whether an ELSE should be expected after an IF or not. The generator just chooses the branch it wants but a recogniser must have some rule about which branch it will try to match first. In a PEG, the rule is: match the first branch, then the second and so on. Hence in the above we must do

```
conditional ::= IF cond THEN statement ELSE statement
              | IF cond THEN statement.
```

Doing incremental backup for a file system is not where you normally use a parser. Martin has a file server at home and he backs it up every night. One of the not-so-visible features of rsynch is

```
time ( rsynch -aHxi --numeric-ids --delete --link-dest=
/mnt/backup/caboodle1/2010-08-03 /caboodle1/
/mnt/backup/caboodle1/2010-09-08 )
```

where he is backing up e.g. /mnt/backup/caboodle1/2010-08-03 to /mnt/backup/caboodle1/20101-09-08. It matches files and just puts a hardlink if the file is unchanged.

So this is a fixed command except for these dates, and occasional changes in the overall directory reference or date format. So he wants to parse the files and age out the old ones and so on.

Martin opened a Pharo image with Ometa2 loaded, plus his BackupListingParser class, which is a grammar class. Each production in the grammar is a method (Ometa calls these methods 'rules'). Ometa is written in itself, like any good language.

```
listing =
  directoryLine*anything* -> [backups]
```

This is scannerless parsing: there is no lexical scanner tokenising everything upfront. (You can have a simple grammar that is in effect your lexer if you wish, of course, which then passes its output to the main grammar.) The `-> [backup]` is a semantic action attached to this rule; within the block is Smalltalk code that is executed when this rule matches.

```
directoryLine =
  (-backupDate anything)* backupDate
```

He only cares about dates so matches anything not a backup date as much as possible, ending with a backup date. PEG parsers get power from this 'not' predicate -. As long as you can't match a backupDate, it will advance one character. Eventually, it can match backupDate, the 'not' expression will fail and the backup date will be matched, whereas (anything)* would just have greedily matched the whole line.

```
backupDate =
  yearNum:yr $-monthName
```

We have an instance of the grammar class and it has instvar 'backups', initialised to an OrderedCollection. We can have Smalltalk methods and Ometa methods in the same class because the grammars are sufficiently distinct.

```
yearNum =
  digits(4)

digits
  ^digit:d -> [d digitValue]
```

His superclass has an inductive rule for digit(0) and digit(number) and the rule above calls that and gets the value from the string. The method

```
listing
  ^BackupListingParser matchAll: stream with: #listing.
```

just returns 'backups', which has been added to on each matched line, so he ran it and saw his list of backups.

Next, he looked at the listing method in decompiled form

```
^true ifTrue:
  [self many: [self apply: #directoryLine].
  self many: [self apply: #anything].
  backups]
```

Allessandro has all Ometa methods starting `true ifTrue:` so that he only needs to generate expressions.

Q (Annick) How does it differ from Prolog and from implementations with cut and ordered rules? Martin has not seen that but probably Brian Ford took those ideas when he wrote his paper.

All PEG parsers can detect left recursion (i.e. that the left of a rule is the same rule so matching would go on forever) and move on to the next rule in that case; this makes writing parsers easier.

His next example was a BadZipReader with a bad zip grammar. A zip file keeps its directory at the end, so any truncation means standard tools won't look at them. Martin had some very large but truncated zip files, so he wrote a grammar to get as much as he could out of them. Zip files have a file entry header and file data over and over again, then finally the directory. The header is fixed length fields plus the extraFieldLength followed by the file data.

His first attempt worked, but blows memory up because it is caching the entire zip file in memory in case it needs to backtrack: by the time it had extracted 7-8 files it was using 250Mb. So he changed it to do one file at a time. But he now think it should parse just the file headers, then act on the files (2 hours more work he thinks). Meanwhile, he showed a jpeg of the ESUG in Brest last year that was extracted from the zip.

Allessandro has documented on http://tinlizzie.org/ometa and http://tinlizzie.org/~awarth/ometa/ometa2.html. The documentation is mostly for Ometa not Ometa2 and for the javascript version. The smalltalk is www.squeaksource.com/OMeta and Martin has written a doc (someone tell him where to store it).

Lukas' project is PetitParser. Lukas gave him the following example:

In BNF,

```
  ID ::= letter { letter | digit }
```

In Ometa2,

```
  id = letter ( letter | digit )
```

In PetitParser,

```
  id := #letter asParser,
  ( #letter asParser | digit #asParser ) star.
```

PetitParser grammar expressions are wholly Smalltalk code, thus you can do Smalltalk processing on the grammar itself and do neat things (see Lukas talk on Helvetia).

So maybe people will now find other uses for these.

Q (Martin Kobetic) Have you tried PEG-parsing Ruby? As mentioned last year, there was *no* published grammar and there was 186k of code in the Yacc parser. Now, someone else has produced a grammar.

Q. Ometa works with socket streams? It should, because it does not require streams be positionable (that's why his zip reader was caching - it caches whatever it may need to backtrack). If the socket closed? Martin McClure suspected it would just fail but it would depend on the particular error.

Q. How does the Ometa2 bootstrap work? There is a preload and postload. The preload is Ometa2 written in Smalltalk. Then the load overwrites it with the same methods in Ometa. The preload methods are just the decompiled Smalltalk ones, so it can be regenerated before each save.

Q (Tim) How fast are these things? He has not measured it, but he understands Lukas measurements indicate PetitParser is faster than Ometa and faster than an LR parser, but slower than a hand-written top-down parser. PetitParser does partial memoisation (in complete memoisation you consume a lot of memory but get guaranteed linear time).

Q (Martin Kobetic) In Michael Lucas-Smith's PEG-parser for Xtreams, he wanted not to have a method per rule because he wanted to see several rules at once. What is your take on that? It is inconvenient to see each rule in isolation. PetitParser therefore has a GUI tool that addresses that.

Q. The questioner has used Ometa and found that debugging was not much fun ("I was forced to write unit tests"). How is Ometa2? Not much better; when your parse fails you lack info. He hopes to help improve that.

Q (Stephane, Q for Bert Freudenberg and Martin McClure) what is the value of bootstrapping Ometa in itself? Conceptually, is there a value in having your parser able to parse itself as well as other things? When you are writing a grammar, then Ometa is much more natural to write grammar than Smalltalk. We get lots of advantages of having the Smalltalk compiler written in Smalltalk, not in C. It's the same for Ometa. It also helps cross-dialect portability.

Monticello 2, Colin Putney
(Colin once relocated from Toronto to Vancouver by bicycle.)

Colin started by reviewing Monticello I. It was a light overlay of Squeak, providing a simple package structure: a convention used in naming packages. A Monticello I repository is a collection of versions, held in .mcz files, which are zip files containing two representations of the code, one as a binary file (this is what loading and reading in Pharo use) and one as a .st

file (used by squeaksource and other older tools).

Merging is key for a version control system; without it, why not just save your image regularly. Colin showed a typical merge scenario 1.0 -> 1.1 and 1.0.1 -> merge to 1.2 -> 1.3 but also 1.0.1 -> 1.0.2 and now one must merge these to 1.4. He demoed doing this in VisualWorks for a one class one method package being published to an empty repository from two images, the trunk and the branch. (This had the usual demo hiccough because Colin had not already published the base to the repository so at first Store was not offering him the versions he expected. Colin could not see the display on his local computer, making it difficult! He changed the colour of the branch image to clarify - to himself as well as us - and then could proceed.) He published a 1.0.1 branch with a change to the one method, returned to the trunk image and published a 1.1 with a different change, then merged the branch and published. He then made a new version on the merged trunk and published, and a new version on the branch and published that, this time adding a method to the branch. Then he tried to merge the 1.0.2 branch and showed that although we had already resolved the changed method, Store was now asking us to resolve it again.

This is typical of version control systems, not a criticism of Store. This is a case that Monticello I gets right.

Another scenario was cherry picking; you want to merge some changes, not all. Store does this well. Monticello for years did not so it so well. However it lets you do ‘backtracking’: you create a version ‘as if’ you had backported changes to the trunk from which you started. This was always a bit confusing and so they have come up with Monticello 2.

In Monticello 2, ancestry information is stored for each method, not for each package. A method Version has a Variant which has an Element and Properties. The ancestry is a set of hash stamps referring to other versions of a method in the past of its current state. Monticello 1 finds the nearest common package ancestor, so 1.0.1 and 1.1 have common ancestor 1.0 and both are ancestors of 1.2 and it remembers both ancestors (unlike Store which only remembers 1.1 as the ancestor).

Monticello 2 can look at the hash stamps of the methods and see if the repository is in the image method’s hash stamps or vice versa. If either is true, there is no conflict (or a conflict that was resolved in a prior merge). If not, there is a conflict.

Q (Martin) hash stamp includes timestamp? Yes. It was originally 4 bytes of timestamp and 20 bytes of hashed content but the latter made it hard to change the model since the model determined the hash, so instead it is now a random number, not a hash of the source. (And yes, source might naturally be returned to during development so old system had to have timestamp and new still has timestamp.)

Thus when you merge two packages, as far as Monticello 2 is concerned, the contents of each package is a collection of method versions. Each

method version has a unique id and a history of the ids of previous versions of that same method, i.e. the versions from which this method is descended. If one method is in another's list, that method is the direct ancestor of the other, so there is no conflict.

As a result, if you are merging two snapshots, all of the merging is done down at the method level: package structure is less important. You can merge things that do not come from the same package. Many versioning systems are sensitive to package boundaries. This was a major issue in Squeak where 5 years of gradually splitting apart the spaghetti image caused a lot of shifting package boundaries. In Monticello 2, you can rework the package structure all you like and the merging will still work.

A slice is an arbitrary selection from the image: it defines what is in the slice and what is not. A package is a slice, a change set is a slice and the user can also define an ExplicitSlice: a collection of methods they put in the slice.

He demoed in Pharo. Suppose standard development is going on in a package and at the same time someone is doing a cross-cutting refactoring that touches many packages. The cross-cutting developer uses a change set slice and the demo is to merge their change set slice with the latest package slice. He changed a method and added a method in a change set, saved a snapshot of it and 'push'ed it to a repository on disk. In another image, he changed the method differently in the package and snapshotted the package slice. (It should be possible to use your local version as one of the merge bases but that is a known limitation, to be fixed.) He opened the merge tool on these slices and saw the extra method and the conflicted method. (After the usual demo hiccough) he resolved the conflict (by choosing one version; he could have edited to create a third implementation and have it supersede both merged versions).

Q. When you rename a method, is the ancestry kept? Not in this version.

Q. Encoding; if you save from Pharo and load in GLASS, might you see bad characters because the encoding changed? Repositories use utf8 (or can use utf16). When loaded, it gets serialised back to the local encoding. It is the platform maintainer's job to make that serialisation work correctly.

Q. Classes? Every time Colin said 'method' in the above, we should think 'method, instvar, class comment, etc.'. This means that in Monticello 2 you can extend a class with an instance variable although the tools do not support that at the moment, but he intends that they shall in future.

Q. Do you store text style? No just the text; no text style information is written to the repository.

The beta is not quite ready yet; anticipate a new release in a month or so.

Metacello, Dale Heinrichs of GemStone and Mariano Martinez Peck of INRIA

Metacello is a package management system for monticello. Dale built it to manage problems he had in GLASS porting Seaside. Load this .mcz - it depends on that one so load that - it depends on these - and each of these depend on ... . Metacello is for the users of a project, not so especially for its developers (they already know what's in it).

A configuration file defines project versions. A version has a list of .mcz files.

Mariano then began the tutorial. he used the Grease example. Grease is a portability layer. The convention is that you name your project's Metacello class ConfigurationOf<ProjectName>. You can do it from scratch or by starting from a Metacello template MetacelloConfigTemplateClass. Then define the `project` method. Then you create a version method

```
version01: spec
  <version: '0.1' imports: #('0.1-baseline')>

  spec for #'common' do:
    [spec blessing: #'release'.
    spec author: 'MarianoMartinezPeck'.
    ...].
spec
  package: 'Grease-Core'
    with: 'Grease-Core.al.231';
  package: 'Grease-Tests-Core'
    with: 'Grease-Tests-Core.')];
```

Dependencies do not changes as often as versions, so the pragma imports the baseline. So you have a baseline method (coded as e.g.

```
spec repository:
'http:///www.squeaksource.com/Seaside30'.
spec
  package: 'Grease-Core';
  package: 'Grease-Tests-Core'
    with: [spec requires: 'Grease-Core.'];
...
```

that can defines the dependencies and the repository, etc.

He then created a new baseline and added platform-specific dependencies for Gemstone and Pharo.

```
spec for: #gemstone do: [...].

spec for: #pharo do: [...].
```

The Grease core has to be loaded before the platform-specific stuff, but the specific stuff for the platform has to load before anything that uses Grease loads. If you had e.g. a requirement that gemstone core be loaded before gemstone core for 2.1, then you would need a requirement for that, otherwise you would just load in the order of encountering the dependencies.

All this could be in XML but noone else could read it. Using a stylised Smalltalk method to define the information is easier. The pragmas are grabbed directly by their system; they do not need implementors.

Q (Veronica) How do you avoid circular dependencies? Dale has a circular dependency checker and finds and fixes bugs in it from time to time to handle weird cases.

Julian notes that Seaside-Core depends on Grease-Core and Seaside-Pharo-Core depends on Grease-Pharo-Core and that is done manually at the moment. Dale and Mariano have a solution of that issue. Pharo-Core requires Grease-Core but Dale also binds them together by setting `Grease-Core includes: Pharo-Core` which lets the system figure it out.

Q (Janko) Why split version and baseline? Splitting into a version and baseline is a convention. It is the recommended approach since version numbers change much more quickly than project structure.

Lastly you want to load things

```
(ConfigurationForGrease project version: '0.1) load.
```

A baseline can define groups:

```
spec
  group: 'default' with: #('Core');
  group: 'Core' with: #('Grease-Core');
  group: 'Tests' with: #('Grease-Tests-Core');
  yourself.

spec for: #pharo do:
  [spec group: 'Core' with: #('Grease-Slime-Core'); ...
```

A `postLoadGreaseCore` method can do post-load actions. You can also reference other projects in the baseline:

```
spec project: 'Refactoring-Browser' with:
  [spec
    className: 'ConfigurationOfRefactoringBrowser';
  loads: ...;
  repository: 'http://www.squeaksource.com/rb.
  ...
```

and in the version (you must indicate which version you want)

```
spec project: 'Refactoring-Browser' with: '1.2'.
```

The 1.0 release will be out by end year; it is still in beta at the moment Feedback is welcome; go to the google code project or the mailing list.

Gaucho, Fernando Olivero, University of Lugano
Gaucho is a package/code browser in Pharo. Click on a package, see its classes. Type in superclass and class name, get new class in package. Click on a class and see an icon widget with subicons for the number of methods, vars, etc. These widgets expand and contract so he can break apart his browsers and drag his various views all over the screen. Active areas /

fields let him add methods, etc. (there was a slightly WebVelocity feel, with the addition that the widgets can be dragged here, there and everywhere). Accessors are automatically created.

Q (Stephane) How would you create a lazy accessor? That would need a setting, or of course you can generate a normal accessor and then edit it.

He generates test scopes based on code references. For example, RBAssignmentNode is referenced in 53 tests; he opened a view of them.

This is a code browser whose component UI widgets can be dragged over the screen, contracted and expanded. Behind it there is a change model so every action the user takes is recorder. A menu item opens a standard Pharo browser on any item viewed in Gaucho, a desirable backup since Gaucho is not finished yet.

Q. Showing objects? At the moment, he is focusing on classes, methods, packages, etc.

Q (Noury) one usually selects a package, then a class, then a method, then some code in the method. Any optimisations for that sequence? Not yet.

He can search for packages to import. His session object remembers what he was working on so can recreate it in a new image.

To do: replay mode to help you remember what you were working on, where you were looking in the system.

Q (Stephane) Widgets go grey when you drag them; why? It is an optimisation, which he will soon drop.

(The usual demo hiccough occurred when he tried to show automatic accessor generation.)

Stephane advised him to set up a real experiment, get Pharo programmers to use it and focus on how it helps or hinders development.

Object-Oriented Software Configuration Management, Hernan Wilkinson, Jorge Silva.
This conference has seen several CM talks already, showing the need for better software configuration management tools. They have a new idea and want to discover whether it will work (and will build it in Smalltalk to promote Smalltalk).

Jorge wants to know where a change was integrated, by whom and when, and to give the programmer feedback about whether, why and how it was integrated. He wants to know what tests were run before the change was committed. He wants to address code formatting.

He wants to avoid or resolve conflicts. He wants to recognise when changes are not conflicts (e.g. two changes each of which add a different

instvar to a class are not conflicting). When refactoring occurs, e.g. a class is renamed and a method is added to the old class, this is another kind of spurious conflict that in fact is not a conflict. Thus he needs to reify the concept of refactoring.

Q (Stephane) You know that in the RB there is tracking of this? Yes.

Seaside releases have had many method renames. Automatic application of these renames to his Seaside-using code would be good.

Lastly, he wants to track the development process. Did the programmer write the test first or after. How does the system architecture evolve. He wants to be able to query the repository for reports.

Hernan then took over and looked at what can already be done with the current tools. Traditional tools such as SVN or GIT give little of this. They are archive-oriented, not OO-oriented. Monticello/Metacello has more but they do not manage the change lifecycle. Hernan does not believe it will be possible to deduce refactorings from the changes file (see Veronica's talk); it must be captured when done.

Store (Hernan does not know Store well) does not capture refactorings and does not manage the change lifecycle. Envy is always on and captures every change, not just the 'commits'. Envy also has problems: its security system has can be a hindrance instead of a help, and it is proprietary. However it was what Hernan liked best. (Audience: and it also stores Java code - but you can't really buy it any more.) He showed their tools, which add code lifecycle display to Envy (colour-coding, symbols extending names, etc.) and record results of tests as a serialised TestResult in the config map which then can be unserialised and put in browser so you see the result they got when the map was saved. (Stephane: this capture of test results is very important to have.)

They also have a smart integrator with a hierarchic structure of changes.

Q. Do you have a separate process where you can configure how you will work (without that it will never be accepted)? Yes, they appreciate the importance of configurability.

They will build a first version for Smalltalk but trusting they can apply it to other languages (Java, Eclipse first). The architecture is client-server, multi-repository built on Pharo and GLASS. A subsidy from the Argentine ministry of technology is available to help the work.

Q. Very ambitious plan. Existing modern tools have issue trackers and quality integration, build server, etc.? Valid point. They want to do in Smalltalk and learn, and hope to do better (because in Smalltalk, etc.)

Q. Are you planning to implement all the parts of a CMS; Monticello and Torch and etc. already exist? They will not reinvent Torch but they will reimplement Monticello.

Q (Niall) Store 771 handles method renames - by deduction, not by capture.

Object Graph Swapping, Mariano Martinez Peck, INRIA, Mines de Douai

His PhD is on memory management for devices with limited primary memory such as robots, but the techniques can also be used on iPhone, desktop apps or servers as and if convenient.

OO apps usually use much more memory than they really need. Objects that are referenced but unused make up the bulk of this. In OO, the primary memory is an object graph from roots to all referenced objects. The Garbage Collector collects only objects that noone references.

This graph may contain subgraphs of referenced but unused objects. He wants to swap them out. LOOM back in the 1980s first looked at this but at that time disk was far far slower than primary memory and LOOM did many tricks to make the system bearable which complicated it. Melt is in Java; it was intended to support investigating memory leaks by preventing the application crashing as it leaked memory via swapping the unused objects to disk. These earlier systems do not solve all the problems: they are not performant and they do not group the objects cleverly.

One key challenge is not to use more memory doing the swapping than you gain by having swapped (so no backpointers of swapped objects to those that referenced them). The swapping must be performant and above all the objects must be grouped cleverly.

Their approach marks and traces unused objects at runtime. Proxies let them load back objects automatically. They group unused objects by finding subgraphs. Because they replace each object with a proxy, swapping one object releases very little memory. Freeing a whole group that has few external references is where the gain occurs. When you have only a few proxy roots for a large graph you gain a lot of memory.

They modified the Pharo VM to mark and trace object usage. They use Moose to visualise which objects were used. With these, they collected statistics from several scenarios. In one Seaside application, they found only 4% of the objects were being used by the application, occupying 15% of the memory (many used objects were bit maps). So we can save 85%?

You can lose a lot of time finding the graphs and avoiding cycles, encoding and writing to the file. Loading the objects should rarely happen but can also take much time.

A subgraph has roots and shared objects (that are pointed at from outside the graph or by non-root objects that are themselves shared). Inner objects are pointed at by roots and inner objects. Their plan is to swap roots and inner objects only, leaving shared objects. This forces them to keep an array in memory for the offset between the memory address of the shared object when the graph was swapped and what it now is (since GC can move it around). The graph's root objects are copied into a WordArray of

serialised objects after which the process walks the graph from them. When a shared object is reached, it is put in the shared objects array (and the objects in the serialised object array that point at it are updated to point at its relative address in the shared objects array). An inner object goes to the serialised WordArray, and references to them are similarly updated to point to their offset in the array.

This means that when the serialised WordArray is populated again on loading, all the objects in it with references to other objects in the graph will be pointing to the right place (see his animated slides for an illustration).

The speed is good, the graph traversal is done in the VM, and the GC facilities are exploited. However you need to remain aware of shared objects and the granularity is poor.

Q (Michael Prasse) did you look at other techniques e.g. compressing strings in memory: transformation time may be shorter than write/load time. Read-only strings can be replaced with a single string, likewise dates, etc.? The research is on this specific technique.

Q.How do you define subgraphs? That is their biggest problem. Their current approach, called 'shooting' is to pick objects randomly and see what the results are and so find subgraphs.

Q.Looked at GemStone? They have talked to Martin McClure and others.

Spy: a framework for profiling, Alexandre Bergel, Felipe Banados, Romain Robbes, David Rothlisberger
He opened a SpyBrowser in Pharo and entered a profile name. This generates a category (interacting with user) in which it will place some classes. A typical way to profile is to run the tests of an application: Alexandre ran Mondrian's tests. When the tests completed, he opened a profile; it showed large squares as classes, small squares within them as methods and lines showed connections between them.

He then started to instrument his generated profile methods.

```
beforeRun: methodName with: listOfArguments receiver: anObject
    numberOfExecutions := numberOfExecutions + 1.
```

He initialized to zero, create accessors and

```
isCovered
  ^numberOfExecutions > 0
```

He ran again and saw all the squares are still empty and white as he has now collected data but not yet changed the display to show it.

```
visualizeOn: view
  view shape rectangle
    width: [m | (m number of executions + 1) log * 10]
    ...
    nonCovered ifTrue: [... fill: Colour red]
    ...
```

He ran, opened and saw the size larger for much run and red for never-executed methods. He can see some classes that are not tested at all - all their methods are red. Now his visualisation shows that, for example, any very tall white classes contain much code and are very seldom executed and are executed on few receivers.

He created another visualisation. Recently, he noticed that Mondrian scrolling was slow, so he created a specific profiler to track down where the time was going. His KaiProfiler runs twice, once tagging messages to know how much time is spent in code and again to see whether methods have side effects, etc. (At this point, the standard demo bug occurred and he had to revert to a slide of what the profiler would have done.)

```
KaiProfiler viewProfiling:
  [:view | view := MOViewRenderer new.
  view
    nodes: (1 to: 100)
    forEach: [:each | view nodes: (1 to: 100)].
  view root applyLayout]
```

The width was the number of executions, the length was the execution time. The colours show what the methods return: grey returns self, yellow returns a constant value, white does neither. He used this to decide on a caching strategy that obtained significant performance improvements by reducing the amount of yellow.

Next he compared Mondrian versions 4.5.1 4.5.7 and 4.6.6 on whether methods were slower or faster, with red for changed methods that are slower and pink for unchanged slower methods. As it showed the calling structure, he could easily locate red changes that were within pink chains of slowed callers.

Q (Niall) Port to VW? This does not use grease or similar but is of course derived from an older version of Mondrian that came from VW. (And of course profile data in XML form from anywhere, e.g. from Java, can be visualised in this tool).

Q (Alan) In Glorp, the big performance cost is DB queries. They can be slow because you do many or because they are slow or for other reasons. There are some interesting visualisations but they are not that much related to call graphs? Alexandre worked with Lukas on profiling the debugger. That raised slightly related issues but yes it's a hard problem.

Alexandre gave a further demo on profiling in the 'ten minute talks' section with an additional visualisations: the method rectangle dimensions were execution time and number of calls, and the rectangle was filled with a darker colour if a method was executed on many different receivers. He also showed a memory monitor tool.

Torch, Veronica Uquillas Gomez, VUB and Univ of Lille
(Unfortunately I missed most of this; these are my notes from Veronica's demo combined with what I caught of her talk.)

Veronica got started in Smalltalk in VisualAge. Now she is an inter-university-programme Ph.D. candidate with the VUB (supervised by Theo d'Hondt) and the university of Lille (supervised by Stephane)

Torch is a code analysis tool, written in Pharo, analysing code changes (working from a Monticello I back-end). It helps integrators understand changes by characterising them and presenting them in context. A dashboard gives an overview of changes, from which Mondrian-style visualisations are opened. A popup on each class shows colour-coded changes. The initial version is at soft.vub.ac.be/torch.

Q. Any work showing conflicts? No, just showing how changes happened. The final goal of her PhD is to work on merging support but this first step is to visualise and contextualise the changes.

She is working on capturing refactorings but the RB refactoring objects are not in the change set, so she must try to find them. She also plans to work on distinguishing semantic and non-semantic changes.

Q (Christian) can we recognise patterns of changes and then see when a step is omitted. For example, when you add double-dispatch, you have several methods to add. By comparing with the pattern, you might see if one was missed? She agreed it was worth researching.

Q (Travis) method renames?

Torch can inspect any Monticello repository.

Smalltalk VMs and Development Environments

Cincom Smalltalk Roadmap, Arden Thomas, Cincom

Since ESUG 2009, we have had a major product release and a maintenance release for VisualWorks and WebVelocity and ObjectStudio. All these products share the same core. Cincom wants to give its customers what they need to justify continued use of Smalltalk. (They also want new customers of course.) They also want to benefit Smalltalk.

Some changes are necessarily disruptive. We have much improved our unicode approach. We want to avoid disrupting our customers as much as possible. Smalltalk customers can be pressured: if a development team say, "We'll need a year to port to the latest release of <dialect> Smalltalk", they may be told, "Why don't you port to <fashionable language> instead?" Put out a new framework that solves some problems but not all and we would have 50% of customers on old, 50% on new, neither doing it all and two frameworks to support. Arden's job, as Cincom Smalltalk product line manager, is to avoid our getting into such situations. Arden talks a lot to customers to valid our idea of what they need and what they will do.

Arden worked for a VW customer for 6 years who ran 100s of applications 24 x 7. This customer could only upgrade every 2 years. Cincom therefore tried a longer product line, to match this and also to ensure a new release had features that could be sold to management - if we upgrade we will get

X. So 7.7 had a long cycle and that meant we had more but it also took longer to get fixes out, etc. The development and build process has become more agile, and agile says more frequent releases, not longer. So future cycles may be shorter. Our testing is more automated, our builds are better.

Unicode is now standard. (We sent out a flier confirming this. After someone in marketing had given it a final spell check, it said that Cincom Smalltalk was now 'fully unicorn compliant' - we write magical code. :-))

Store now runs on Glorp (O-R mapping style) which will allow some more improvements, for example atomic loading. 64bit VM and image has been improved. Delays and the time system have been much improved. The icons on all products have been redone.

ObjectStudio 8.2.1 is a Windows-centric business modelling tool. There has been much work on the mapping tool. The windows message loop has been moved into Smalltalk from C which solves some rare errors. OS is Vista certified and Windows 7 certified, the only Smalltalk dialect that is.

WebVelocity is a web IDE that combines the active record pattern with the ability to read and generate scaffolding code from an existing DB, so it is very fast to create new apps using existing legacy. It is build on Seaside, Glorp and VisualWorks. WebVelocity 1.1 addressed cloud solutions. The editing tools now colour syntax. Collaborative editing is supported.

VisualWorks 7.7 customers wanted internationalisation, so it has CLDR and Unicode. Georg's Excel talk illustrates how COM has been revamped. Look at James Robertson's screencasts on interfacing to iTunes. A Grid component is in preview. Web services were upgraded to WSDL 2.0 / SOAP 1.2 (many customers wanted it). The project launcher makes it easier for newbies who don't come from an image-based background.

The future: Windows does 64 bit differently from other systems but we know we must do it. Store will now exploit the Glorp substratum and some WebVelocity work. IPv6 will be done. UI work (see Travis talk): Cairo and Pango are available to our customers and we have tested letting the OS render the widgets. More web services, work on polycephaly (see Arden's talk at ESUG 2009; the work on his slides has already been progressed and will continue.)

Arden wants input. You need something, make sure he knows.

(There was no time for Arden to show the WebVelocity video; see it online. WebVelocity is easy to use.)

Q. Monticello format role? VisualWorks has tools to read Monticello now. (Post-meeting: Michael Lucas-Smith has done a video on this tool and posted about it in vwnc.)

Q (Christian) the public store repository gets the newest tools but they only work in newest image so can we have store for each version? Arden will

look into it. Offline we discussed using version naming to make it clear which release a version was compatible with, and having a Store DB whose versions were always for the latest release, never for development.

Q. Will you add project templates to the launchpad? Yes, we are working on that now.

Q (Janko) Electronic signature support? (Martin answered) It is not in the schedule now and is a moderately complex problem. It could be done. Telling Arden is the way to make things happen.

GLASS and VMWare, GLASS goodies, Monty Williams, Gemstone
Monty talked to five VMWare vice-presidents last week about the GLASS model. They said "If this benefits the community of Smalltalk, it is good, so why don't you just give it away!" After recovering from his shock, Monty felt very pleased - and so did the conference when he showed the new GLASS licence web page!

(Later, Monty explained to me that the low-end VMWare sales are all handled by a distributor company. For VMWare itself, collecting a $7000 license fee for a GLASS that had become too full would be so far below their usual transaction size as to be not worth doing.)

Visit seaside.gemstone.com. GLASS runs web apps, not huge client-server apps. It does not run on Solaris, support GBS or the rest of the GemStone enterprise suite. To run a web server, you do not need 4000 virtual machines, a 256 GB shared page cache, etc.

Monty showed the key file. They do not in fact limit to 16Gb in such a way that you would crash when it expands beyond, but the other limits prevent you from going much above that without noticeable slowing.

If you want to play with GemStone 3.0, remember it is pre-alpha, with partial continuations implemented by Dale last week so it is very pre-alpha. ("If you have problems, I'll be on holiday next week.")

Q. MagLev in Ruby community? There is a lot of respect for Seaside in the Ruby community. They think Ruby is the greatest but they have a lot of respect for us.

Q. CPU limits? You can run it on a 4CPU box. It will just not use more CPUs. Dale handles 500 RESTful URL hits per second on his 2CPU machine no problem. A CPU is whatever the platform considers a CPU. They have some software stuff in GLASS that enforces that limit.

Dale: the 3.0 GLASS config has a GemStone 3.0 component but has not been heavily tested.

GemStone/64, Norm Green, Gemstone
As session chair, I introduced Norm Green, the Research and Development Director for GemStone. The previous afternoon, I overheard the reply he

made when asked what he was coding in at that particular moment: “I write C code so you don’t have to.” This is the man who writes C code so we don’t have to; I invited the audience to show our appreciation, which they did, emphatically. :-)

GemStone was acquired by VMWare on May 17th 2010. VMWare is a $2bn revenue with 8000 employees and offices right across from the old parc place. Almost everyone at VMWare is or recently was technical.

What has changed for the GemStone guys? Not much; it is the same team, a new reporting structure, better snacks and lunch room, more paperwork, more funding. VMWare are aware of Smalltalk - several people have said, “Yes I did that in the 90s and liked it”. They are sometimes asked is MagLev replacing GemStone? MagLev is built on top of Smalltalk, is a complementary product with shared code base and no way is any kind of replacement for GemStone.

20% of the world’s shipping is run in GemStone. Kapital uses GemStone. For 24 years they’ve heard rumours about ‘Gemstone - or Smalltalk - will not be around in X years.’ They are used to it.

GLASS is now free for 2 cpus instead of one, for 16Gb instead of 4Gb, for 2GCVB shared page cache instead of 1 (applause).

Norm outlined the improvements in the latest release. Smalltalk byte code is now converted to native code on first method invocation (JIT-like) which can get a 2x performance gain. Gemstone has improved dynamic networking support (this was mainly an issue for laptops). GemStone used to think its IP address should never change; now it can tolerate a change.

Many of the cross-repository processes are now multi-threaded; the user specifies how many threads. The global GC is now an order of magnitude faster in 3.0. `allInstances` takes longer when you have 2 billion instances - it is now multi-threaded and faster. The GsObjectInventory reports classes and instances; it now has primitives to make it run faster.

3.0 has FFI, letting you load 3rd party shared libraries without having to write any C code.

3.0 supports 3rd party authentication, so end users do not have to do a separate GemStone login. For now, use unix password or LDAP; more will be added as and if customers need it.

The various vendor OSs had asynch IO bugs that always bugged them so they’ve written their own.

James Foster has improved ANSI exception handling. It used to be merely a layer on native GemStone exceptions. It is now supported down to the bytecode level.

ScaledDecimal has been renamed FixedPoint and there are new faster

implementations of ScaledDecimal, LargePos/NegInt.

The Gem-to-Stone system communication no longer needs to serialise responses, giving a 30% improvement for some really busy customers. Session priority has been introduced so you no longer have to accept 'first come, first serve' for two sessions in the Stone's run queue at the same time. (Q. Commit? The commit is a special case as it's always at the head of the line and beyond that there was never control over it.)

64bit intel Mac will run GemStone.

That's what's coming in 3.0. Beyond 3.0, they've been asked for nested transactions and are thinking about it. Hot standby would allow every commit to be immediately replayed to a hot standby database (Oracle calls this database clustering). ODBC support is no longer supported (for various lawyer reasons) but could be supported again. IPv6 is coming. Secure sockets (SSL/TLS) are closer because the LDAP code needed it. Automatic load balancing is often requested by customers. At the moment, customers have to roll their own assignment of tasks to servers. Single-sign on (get into the corporate and then signed on) is being thought of but it is tricky because it touches many things. The S in SNMP should not stand for simple (JMX looks better though still not simple and recently customers have said, "Do that anyway") so one of those may be done.

3.0 is due for release in June 2011, 3.1 in December 2011.

Q (Christian) Why is windows not there? It is very high cost to support a Windows version because Windows does everything differently, so much of the code must change. Take 1 million lines of code and double it. The client of course runs on Windows fine, but the commercial argument for the server to run there is just never adequate.

Q 32 bit? Still there, little used by GemStone now ("I guess it still works").

VASmalltalk, John O'Keefe, Instantiations
Just in the last month, there has been a fairly significant change in the company Instantiations. It already has an interesting history: it has been bought and spun off again. It has been a Smalltalk company that redid its Smalltalk tools for Java, then for Eclipse. Google has been using these Java tools for 4 years so they decided to buy the tools. So now Instantiations is a purely Smalltalk company again with a lot of money from selling off these Java tools. Their website is now all about Smalltalk and they are hiring Smalltalkers and/or someone good who wants to work in Smalltalk.

In May 2010, VASmalltalk 8.0.2 was released with Seaside 3.0 alpha, an initial Glorp release, various platform concurrency updates and SUnit 4.0. The VASmalltalk port of the Seaside 3.0 release is on VAStGoodies.com (site developed by Adriaan van Os and others, and supported by Louis Andriese of OHRA). VAStGoodies.com was the very first Seaside application built on VASmalltalk. OHRA also uses Seaside commercially; John showed a screenshot of their site.

VASmalltalk seaside does not do `call:/answer:`. This may be supported in 8.0.3 - or it may not; John will not commit himself yet. Most Seaside apps can be built using the workarounds in the Seaside book.

Glorp is an open-source database mapping layer. An initial version in 8.0.2 is being improved.

They have digitally-signed Windows executables, so you can build Windows-logo'ed applications. They support Ubuntu 9.04 (version 8.0.3 will support Ubuntu 10). Windows power-management events (putting your laptop to sleep) are recognised so now the image no longer has trouble reconnecting to the manager automatically after sleep events. They support Oracle 11: direct use of function calls, support for anonymous SQL blocks.

Julian's pragma implementation stores pragmas in the first literal slot instead of the last, which caused major performance problems when John ported it. He moved it to the last literal slot and all worked. (Their VM uses the first four slots for high-speed stuff.) Exceptions are better handled. SUnit 4.0 is in. There is much more documentation, in a new doc system.

Future plans: they will do two releases a year, major or maintenance as the schedule requires. They work in three-month timeboxes. The next release is in November. It will have Seaside 3.0 (and continuations?). It will have MessageExtraction, an internal tool for mapping code with text in one language to translations. Their VM will run on Ubuntu 10.4 and Fedora Core 13.

Unicode/UTF-8 is always in the future. (It's convenient that John is speaking after Arden detailed some of the issues.) They are working on it. They will work on web services debugging tools (their codebook has tips, debug support is always wanted).

They have started a collaborative project with HPI (university) to improve the look on Linux: they will work on GDK 2.x bindings on Linux. (This is their first university collaboration.)

They will rationalise their settings which are currently to be found in the settings workspace or VAAssist or ... . The change browser is old and needs improvements. A Glorp tutorial is in the documentation. A programmers reference is being written.

Security: they currently use OpenSSL and will move to OpenSSL 1.0. Their Windows services management was in C and will be in Smalltalk. They will look for performance hotspots in their class library, and at better incremental garbage collection. They will also address 64 bit.

If people uninstall just by wiping the directory then there can be issues. They want to improve their install tools to deal with the issues people call them about.

They have many collection classes that differ only in their hashing and

sorting. They'd like fewer classes with policies that either the user or the underlying system can change to use the best policy for the content.

Get VA in one of the following ways:

- register on site and download evaluation copy
- buy it (their preferred way, of course)
- get a development build. They offered (on Windows and Linux only) one for 8.0.1, two for 8.0.2 and there will be two for 8.0.3. They are announced on the VAForum and comp.lang.smalltalk
- contributors to open-source projects can have a license; releasing something to VAStGoodies is the usual way you indicate you are such a person (but other open-source contributors can ask)
- be an education establishment

Q (Janko) Following all Smalltalk forums is hard so is it good to be on a distinct forum? Instantiation is considering converting the forum to a nabble mailing list.

A JIT Smalltalk VM written in itself, Richarte Gera, Javier Burroni
The aim is to write a Smalltalk VM in Smalltalk. They wanted it to be compatible with Digitalk's VSE VM and leverage that environment. So they started by using the primitives, GC and JIT of the VSE VM while getting the basics of their system. They want to run test cases at every step so they always have to have these frameworks at every step of the work.

All is implemented in Smalltalk. He showed some pieces.

```
GenerationalGCBuilder>>populateInternals
  addAllMethodsFor: GCSpace
  addAllMethodsFor: ObjectgenerationalGC
  at: #isSmallInteger default: Object>>#isSmallInteger;
  at: #isSmallInteger add: SmallInteger;
  at: #bitXor add: SmallInteger;
  ...
```

This is frozen code lookup. The first time, a dynamic lookup is used. The second time, the cache is in place. If you already have a cache, look there. (Lookups are chained from class to superclass.) For example, Object and UndefinedObject implement `isNil`.

```
nativizer
  at: #isNil
  addAll: (Array with: Object with: UndefinedObject)
```

You also need `_isSmallInteger` as that is an immediate object.

What does a Smalltalk VM need? Probably more even that they realise but at least .exe generation, method lookup, object formats, memory management (object creation and copy, GC and `become:`), primitives, FFI, processes and callbacks.

He demoed, opening a browser and executing code: "you won't see anything; that shows the demo worked". After they got the JIT working, they took 3-4 months to generate .exe; it's not as easy as you think. In the

end, they were glad they took the long road to it.

For the GC, you need to be able to reclaim memory from the OS. All the code on their slides is real but lacking initialisation of access to the OS and other such complex things. (Such things as

```
callTo: 'VirtualAlloc' from: 'Kernel32.DLL'
```

took them a long time but were worth it to avoid creating primitives for such outside calls.)

He then handed over to Javier to talk about GCSpace. It contains an old space and two flippable spaces, A and B. `shallowCopy` moves objects between spaces. From the perspective of the GC, we call the subspaces 'old', 'from' and 'to'. GenerationalGC:

```
collect
  self followRoots; walkStack; rescueEphemerons;
  traverseWeakContainers; flipSpaces.

flipper purgeRoots.
flipper follow: object.
flipper moveToOldOrTo: object.
flipper findReferenceOrSetToTombstone: weakContainer.
flipper addInterrupt.
```

At end of flip, copy data from B to A (from 'to' to 'from'). The VM assumed that new space was lower in memory than flip space but now they flip pointers instead of copying, this assumption is no longer true. They have not yet implemented mark and sweep, just copy and flip.

They have lots of tests. Richarte opened the environment (in Cincom VSE), and ran a test which switched on their GC, exercised it, then returned to the VSE GC. He looked at the

```
copy: object to: space
```

then saved a count method ("I've never tried it in this order, but since demos are meant to fail ..."). Then he looked at senders of `flip` and brought up the singleton GC object. Various `profileFlips`, `profileBytesFlipped` and similar methods wrote data to the transcript as he coded and executed.

This will be released (in whatever state it is!) at the Smalltalks conference later this year. This is a good environment for testing your VM ideas.

Q. Benchmarks? It's slow - we don't need benchmarks to see that. There is one unresolved bug that is causing more flips than are needed (just finished this current version two weeks ago).

They still have to do `become:`, `new` (at the moment, `shallowCopy` is their cheap `new`), and mark and sweep, also better method lookup strategies, and to add closure to the JIT (find closure over method sends).

Q. Catching circularities, e.g. GC firing GC again? Mostly, if the GC fails,

it crashes - not hard to spot. They have counters to catch some cases. At the moment, there are no exit points from their code that would allow the particular case raised by the question.

Q. Debugging GC in the new environment? While you wait in the debugger, the space is in a dirty state so changing flags in the debugger is to be avoided, etc. Their simulator is the background Smalltalk environment.

Q (Georg) why not let symbol tables access GC from the native debugger? They do exactly that so they can use the native debugger.

Q (Martin) do tests run in background or in your environment? That is why they have two test methods on the slides. The upper method is the native one, the lower is proxying 'from' space from flipper into the block in the test (see e.g. the `testFollowObject` code on the slide)

Pharo, Marcus Denker, http://pharo-project.org
The people working on Pharo use it every day for research, for teaching and for commercial purposes. Some of them have used this system for 10 years and all of them expect to be using it in 10 years.So it occurred to them that they should continuously improve what they use. Small simple fixes, the next step even if imperfect, are useful, even just a student saying "here's a typo in a comment".

Every improvement has an effect. The Refactoring Browser lets you make significant improvements e.g. via code critic or using the refactorings. It is easier to improve the refactoring browser once you have the refactoring browser to use.

Pharo wants to be a flexible environment to support innovation in Smalltalk. So Pharo is not the system with innovation in it, it is the system that supports innovation. It is a second step to put good innovations back into Pharo. Pharo also wants to be a system where you can get the job done in the real world, so that needs robustness, and this is not a conflict but a common ground between the two.

Think of research as like climbing a mountain. He showed a picture of a cloud-free mountain and stressed that research is like climbing when the mountain is enshrouded by clouds and you do not know how many peaks there are or where they are. Mountaineers climb up some distance and build a base camp. When the base is supplied and stable, they can go further. The base is essential to the further climbs that find the peaks. Pharo aims to be the base camp for daring research explorations, much higher than the boring plain of IT but nevertheless stable. If you have a research student for 4 years, you do not want to give them a rubbish system.

Have we succeeded? Pharo's first release was in May 2010. He showed slides of many research groups who are using it, and of the 30 companies listed on the web site who have indicated use of Pharo (amount and kind of use not known in some cases, and there may be users not on the website).

He showed what Pharo 1.1 looks like, then looked at Seaside and Pier in 1.1 (pre-built images for these can be downloaded). GemStone / GLASS uses Pharo as the IDE. Moose uses it. Pinesoft (London-base company) contributed pluggable look and feel to make Pharo look like windows or like old Squeak, and MBagger which can photo and enhance barcodes hidden under the plastic wrappers of shippers, so solving a real commercial problem. Estaban Lorenzano develops on iPhone (project on hold until last Wednesday's announcement!). Two years ago, John McIntosh ported to iPhone, now Estaban has made it possible to use native iPhone widgets in our Smalltalk apps in the Mars and Deimos projects. Now you can put apps in the iStore and not be rejected because "you used the wrong button".

Pharo started from Squeak 3.9 in 2008. The first task was to make major cleanups. MVC has been broken for years so they dumped it; why have two rival mechanisms. They really like eToys but its source code is beyond its lifetime. They integrated the new look of Pinesoft, Eliot's BlockClosures, Lukas' tools, the Preferences class (was referenced in 800 places!) and many minor bugfixes. They closed > 1200 bug reports and did 470 updates and released in April 2010. That was (too) long so they released 1.1 much quicker. The 1.1. focus was performance and a smaller core image (6.2 Mb). They now have a settings framework so you can add preferences without a global preference class; menus are similarly changed. In 1.1, 883 bugs were closed and 410 updates were made; it was released in July.

The current development has already closed 300 bugs and 141 updates. What the next release will have will depend on what people do, of course, but some things are fairly certain.

Lukas has worked on a build server: if tests take too long to run, you need a server-based infrastructure to load fresh and run tests in background. Every commit will verify that it loads - or tells you it will not - and will tell you what tests failed. He showed the page for a project of his that Lukas configured for him. The plan is to use it for every project in Pharo, and for the VM too if they can manage it. The system they use is Hudson which is in fact a Java system but very flexible, and many teams such as INRIA have Hudson installed.

Reporting is also being worked on, based on Moose.

Opal is a project aiming to have a new more understandable compiler. It will use the RB AST to do intermediate representation of bytecode and bytecode transformations. The old Smalltalk compiler has some old ('70s) and strange code that makes it hard to do some things, e.g. if you want proxy classes the fact that new for class is a bytecode makes it hard. (Just try to change the compiler; you will soon find how obscure and hard to locate some things are in the old compiler.)

Everyone can help: report bugs, confirm bugs, write tests, write examples, write comments, contribute simple fixes, and all this leads into deep discussions from which ideas emerge. Look at the bug tracker. Pharo sprints have been run in Bern, Lille, Buenos Aires, Brussels, London and

Lille (and Barcelona), sometimes piggy-backing on other Smalltalk meetings. A very small-font slide showed all the contributors.

'Pharo by Example' volume 1 is out, and volume 2 is in preparation.

Q (Laurent on Twitter, forwarded) anyone working on eToys on Pharo? Marcus thought that eToys was not packagable.

Q. Morphic 3? It is sponsored by ESUG. Marcus did not know when it would be released; because there are significant advances, it needs wider visibility and exercising.

MARS, Estaban Lorenzano
(Talks usually start with 'who the speaker is': Estaban says he doesn't really know, he's a work in progress. :-)) MARS has a new logo, a red planet with logo, and now it has the 3rd prize award logo too!

Once upon a time there was Squeak and Squeak had a lot of colours and it hurt his eyes. Estaban knows someone in the room who refused for two months to use such a garish system. There were also no native widgets, no good key-bindings and no native UI (no Polymorph in those days), no feeling for the final user.

Then there was Pharo and it used Polymorph and cut down on the garish colours and had some key bindings (cmd + w, etc.) and many of these changes are now in Squeak too.

So why do we need MARS? Because native applications rock. The Mac experience matters; let's have it in Pharo. MARS is an MVC UI framework for building native widgets for OSX iPhone and Mac. He uses the old MVC framework but rendering native widgets (Cocoa/CocoaTouch).

The Objective-C bridge was made by John MacIntosh and is now present by default in all the latest Pharo VMs. It is used on real apps in the iPhone; John has 5-6 apps in the mac store. However this bridge is just a way to interact with Cocoa objects and C; you still deal with memory issues, etc. Estaban wants to program in the Smalltalk way where his VM handles this.

A MARS app has two objects, one in Smalltalk and one in Objective-C. The object in Pharo forwards a message to Objective-C and a proxy in Objective-C talks back. Semaphores control the callbacks; coordination of that is tricky but the VM handles it for us. A MARS app has hundreds of threads running, each waiting on its semaphore for much of the time.

Any view has a corresponding Objective-C object. Everything is a window or a child of a window or a dialog and is inside an Application and this is how we manage memory as an application keeps track of its windows. When you close the window, MARS flushes the memory for you. Estaban has created a lot of controls already and more are coming. (Many similarities with his talk on Reef.) A controller relies on the callback proxy.

The model is in Smalltalk and is just standard Smalltalk; you subclass MRModelAdaptor.

He demoed. ("Most of you saw it in my awards demo - and know it can crash at any moment due to a known bug that will be fixed next week.") He showed some UIs with MARS widgets, then opened a workspace to show

```
MRWindow new;
  title: 'Demo Win'
  bounds: 0@0 extent: 200@200;
  add: (MRButton new title: 'OK'; onAction: [Object
inform: 'Pressed']; yourself);
  center;
  show
```

Q. You can just debug or need to integrate something for the debugger? The Objective-C object is just a reference inside Smalltalk so you can inspect it by sending it messages from within the debugger.

Q (Kwai) About the iPhone side of MARS, iPhone objects to apps that use too much memory; does MARS handle this? MARS flushes memory on window close. This is triggered in the callback, which tells iPhone.

He showed a MARS random name generator for role-playing games on a largish iPhone-style screen.

Q (Johan) Access to orientation of the device? Cocoa fires an event; you can catch it, and so get a callback to Smalltalk.

Problems: the deadlock problem has been solved (by changing the VM structure) and John and Eliot are working on another bug. It is between 'make it run' and 'make it right'. Make it fast is to come - several optimisations are possible.

Towards Small Portable Virtual Machines, Noury Bouraqadi, Luc Fabresse, Mines de Douai
Ocean is the project name. Noury explained that it is an experiment with a design for a VM in mind.

They work with multiple robots or multiple hand-helds, and these need to communicate. The OS network API provides the VM-layer socket plugin which provides the image layer network library. The socket plugin is written in C and mixes-in TCP, UDP, IPv4, IPv6, etc. It is hard to port.

The image level is not so different. SocketAddress is a subclass of ByteArray, for example. The test coverage of the network Kernel was zero in Pharo 1.1 and is only 4% now. Diego Gomez found problems with it.

The above will be hard to fix. The Ocean project therefore plans to redo it instead, this time 100% test-covered, fully object-oriented with a properly wrapped OS library.

The image has the Ocean library and Alien library, which sits on the Alien

plugin in the VM that talks to the wrapped OS. Socket (subclasses TcpSocket and UdpSocket) bridges to NetworkLibrary, whose subclasses are the various platform-specific libraries. Luc started work in Spring and now they have sockets and are almost 100% tested (and these are real vigorous tests: socket connected? data sent? socket closed? data received?, and there are also tests of concurrent connection).

Testing needed a reliable client so they used NetCat. They set up an OSProcess library sitting on the OSProcess plugin and System API so they could send command line socket calls for testing.

Yesterday at midnight they sent and received 10Mb of data 10% faster than the existing sockets. Thus it looks good so far. Next they need to put Ocean into Pharo and clean it up there. They will have to keep compatibility to the old for one version at least.

They want to make the VM smaller by pulling stuff to the image side, keeping performance. They suspect some other plugins are there because of performance issues. An FFI plugin that supports multiple system processed.

Q (Leandro) The NetSqueak project replaced the OS with a wholly Squeak implementation, which would help you break free of the OS; there is also the SqueakNOS project? Noury knows of these but wonders about the performance: they want a smaller VM not a slower VM.

Q. Is there a security concern when relying on the Foreign Function Interface? If you raise everything to the image level perhaps you can solve the issue there.

The OSProcess test use NetCat. The command line for NetCat starts a server on a port: you type the string and send it to the OS process just as if you were on a terminal but wrapped so you can see what stdout is.

Q. Firewalls? Should not be affected by any of this; sockets are sockets as far as firewalls are concerned.

Q. Do you plan to support blocking reads? It is (only) blocking at the moment. Concurrency is important because Pharo FFI does not support multiple-threading yet. They plan to fix this and then they will have non-blocking.

Frameworks

Xtreams, Martin Kobetic and Michael Lucas-Smith, Cincom

(Martin presented at ESUG, Michael at a UK STUG meeting two months later. This write up incorporates both talks. Both executed all the code snippets below as they described them so we could see how they worked.)

Martin's slides opened in a strange yellow colour. With help from Jordi, they were returned to something more normal in appearance. Martin works primarily on networking and security in the Cincom engineering team.

Michael Lucas-Smith and Martin have been working on Xtreams for two years. It is a different take on streaming protocol. Martin and Michael started exchanging war stories about horrible stream things "and what were they thinking when they did that" so inevitably it got to the point that they tried to do their own. Martin went into the project with many opinions, most of which he has now discarded. What they have today is very unlike their first version.

You can use an Smalltalk-80 stream with an Xtream but Martin and Michael have no plans to replace Smalltalk-80 streams as there is so much code out there they could never replace. Reimplementing Smalltalk-80 streams on Xtreams is on their 'consider whether to do or not' list.

Their issues with classical streams are:

- They are not consistent enough. If you do something interesting with e.g. InternalStream, and then try to use it elsewhere in the hierarchy, you are likely to run into problems. (For example, they are not consistent in the way that incompleteness is handled; you never know whether the stream will read ahead for you. When you encounter an error, if you resume you don't know if you are on the RHS or the LHS of the problem.)
- Many streams are not positionable so they wanted to de-emphasise that.
- Read and write streams need to be better separated.

Xtreams basics are terminals, read streams, write streams and transforms.

Q. How prevent write to e.g. socket? The out is not the same as the in for a socket so that's OK. Generally, there are no hard and fast rules. One can use buffers to avoid issues.

Q (Stephane) Damian Cassou reimplemented Traits for read and write streams and had that problem. They were told "often we want both read and write so they said use both traits." Did you find this? The Xtreams library has not been that much used. Martin thinks a read-write stream is a fake: it is two streams in one. Is 'the position' the read position or the write position. The only exception is file streams where, since you can read or write at a given point within a file, a single position is OK for both read and write. However in most cases that is not sensible. Martin therefore feels it is more appropriate just to put two streams inside a wrapper class.

- Scalability matters for network protocols. They wanted to avoid expensive assumptions: standard `collect:`, `select:` and suchlike do not scale; things that evaluate lazily are possible.
- Lastly, they wanted to handle transactions and stacking of streams better.

There was also an API design decision to make the methods consistent, easy to remember or guess, easy to type. Xtreams makes nomenclature changes in the basic protocol for ReadStream (`source`, `get`, `read:`, ...)

and WriteStream (`destination`, `put:`, `write:`, ...), i.e. for ReadStream, they tweaked protocol to use `get` for `next`, `read:` for `next:` and `rest` for `upToEnd` and `source` for `collection`, and similarly WriteStream uses `put:` and `write:` instead of `nextPut:` and `nextPutAll:`. They also changed the convenience creation methods:

```
stream := 'hello world' reading.
```

Michael showed that reading past the end, i.e reading to the end then sending `get`, raises an error.

Use of the word `reading` indicates the progressive lazy nature of read. Another difference is that doing `get` on an empty stream raises an exception. In VW, `next` returns nil (unless wrapped to catch the EndOfStream Notification that it raises, whose `defaultAction` when not caught is to return nil) whereas `next: 1` will raise an exception. This is an annoying inconsistency that pollutes using code with nil checks.

A new terminal must have `read` and `contentsSpecies`; that's all it needs. The `terminal` returns the non-WriteStream object that is decorated by one or more WriteStream (or subclass of) streams.

Terminals are where the content comes from or goes to. Terminals can be collections, blocks, files, sockets and pipes, buffers and shared queues. Martin showed using a block as a terminal.

```
current := ObjectMemory someObject.
[current := ObjectMemory nextObjectAfter: current.
current == 0 ifTrue: [Incomplete zero raise].
current]
  reading inject: 0 into: [:c :e | c + 1]
```

(He raises the Incomplete exception to tell the stream it's at the end; that exception is how a stream knows.) Closing a write stream trims the terminal to the actual content, so if you ask for it twice you get the same thing both times.

```
String new writing
  write: 5 from: 'hello World' reading;
  conclusion
```

This means you can implement the operation more efficiently as you do not lose information.

Insert is new functionality not in Smalltalk-80 streams. It lets you position yourself back into the stream and insert. `conclusion` is also new.

```
buffer := RingBuffer new: 4.
```

(Have you tried implementing a ring buffer in Smalltalk-80 streams?)

Xtream terminals can be Collections, Blocks (0-arg for reading 1-arg for writing), Files, Sockets and Pipes (`Stdin reading`, `Stdout writing`), Buffers and SharedQueues, e.g. the heap.

Martin showed using a stream on a block to compute a fibonnacci number:

```
a:= 0 b:= 1.
fib := [|x | x := a. a := b. b := b + x. x] reading.
fib ++ 99; get"returns the 99th fibonnacci number"
```

and demoed further, making a another block print to the graphics context, recomputing its position at each position of the stream. Then he looked at pipes.

```
[:in :out |
out writing write: 'Hello'; close.
in reading read: 5]
  ensure: [in close. out close]
    valueWithArguments: UnixPipeAccessor openPair.
```

He showed a block on a CPointer (writes to the heap):

```
buffer : CIntegerType char malloc: 50.
[buffer
  writing length: 50;
  write: 'Hello World'.
buffer reading
  contentsSpecies;
  read: 12]
    ensure: [buffer free].
```

Michael has used this for creating 3-D models of floats.

Lastly there are TransformStreams. Transforms can be created in the 'collecting' style: `doing:`, `selecting:`, `injecting:into:`, `collecting:`, etc., and they have specialised ones for `encoding:`, `encodingBase64:`, `compressing:`, `interpreting:`, `hashing:`, `marshalling:`, etc. (Their base64 encoding is a single method, a far cry from the giant class that is standard VW base64 encoding.) Substreams have protocol `ending:{inclusive:}`, `limiting`. General transforms can discard elements or turn one element into several. Limiting lets you say things like, "this string stops after 10 elements", "after we see $C", etc.

```
random := Random new reading.
random := random collecting: [:f | (f * 256) floor].
random contentsSpecies: ByteArray.
```

The first stream becomes the source of the second stream; all we need a pointer to is the top of the stack.

We can compute primes by having an infinite stream of 1s that we convert to an infinite stream of sequential numbers by adding them and then make a stream of primes by rejecting any that are not.

```
sieve:= OrderdCollection new.
ones := [1] reading.
twoAndUp := ones injecting: 1 into:
  [:previous :next | previous + next].
primes := twoAndUp rejecting:
  [:i |
  (sieve anySatisfy: [:p | i \\ p = 0])
    ifTrue: [true] ifFalse: [sieve add: i. false]].
primes read: 10.
```

Character encoding is done just as VW does it. It would have been a lot of work not to reuse what VW already had. Writing

```
stream contentsSpecies: String.
```

can avoid the overhead of the general encoding structure if you happen to know the file you are streaming over is OK with latin1. That is the fastest way to read from a file i.e. without doing any transforms.

```
(#[10 13 13 10] reading encoding: #ascii) rest.
(ByteArray new writing encoding: #ascii) cr; conclusion
```

Xtreams can be used in cryptography (below, the `-= 0` is just saying 'read to the end'):

```
(ObjectMemory imageFilename
  reading hashing: 'md5') -= 0;
  close;
  digest.

key := random
((String new writing)
  encodingBase64;
  encrypting: 'aes-128-ecb' key: key iv: nil)
    compressing encoding: #utf8)
      write: Object comment;
        conclusion.
```

You can layer transforms upon transforms but they must be compatible: if one expects to read Bytes from its source, the prior must write bytes to its destination.

Then he looked at using this to translate morse code. Morse code can be read via a decoding tree, where long means take the left branch and short means take the right branch.

```
('. ... ..- -- .....' reading transforming:
  [:in :out: |"in is a real stream, out is virtual"
  node := MorseTree.
  [beep := in get.
  beap := $]
    whilefalse:
      [node := beep := $.
        ifTrue: [node at: 3]
        ifFalse: [node at: 2]].
    out put: node first])
      rest
```

Writing morse code is the same.

```
String new writing transforming:
  [:in :out |
  out write: (Morse at: in get); put: $ ])
    write: 'ESUG BARCELONA MMX'; close; terminal
```

Martin demoed limiting: you can say limiting: 10, read only 10 elements, or bounding conditions

```
input := (1 to: 50) reading.
messages := Array new writing.
```

Martin showed not being able to write more into a stream than its size:

```
output := String new writing.
(output limiting: 40) write: Object comment.
output conclusion.
```

Time did not permit Martin to talk about Substreams or the whole issue of position and positioning streams and how they deal with it; Michael said something about it. Substreams also have `ending:`

```
... ending: [:e | '.!?' includes: e]) rest
```

Slicing is using streams to produce streams. Martin showed:

```
slices := (input limiting: 10) slicing.
slices := slices get.
slice rest.
```

Michael's example was of limiting output to things of 3 items and slicing.

Stitching is combining streams into a single stream. Starting with

```
directories := ElasticBuffer new: 10 class: Array.
```

Michael stepped through an example of reading filenames and combining them into a total directory listing. He then went on to positioning. `++` jumps forward and will also work on non-positionable streams. Only positionable streams support `position`, `position:`, `--`, `+=`, `-=`, `available`, `length` and `explore:`. Doing `... input -=6; rest` returns ' World' from 'Hello, World'; it means 'go back 6 then read from there to end'. Doing `+=6` would do the same because 'Hello, World' is the same length either side.

You can wrap to position non-positionable streams. Michael demoed. You use a default buffer - so you can go back forever - or a RingBuffer - so you can only go back as far as its size. (Someone suggestion renaming ring buffer to sausage machine. :-))

Exploring is like peek but as far as you like for the condition you choose.

What have they achieved? They have strictly separated read and write. They can compose streams better. Reading or skipping past the end of a stream always raises an exception (with a great deal of information: how much it read, etc.). You can never ask `atEnd` - Smalltalk-80 was simply wrong in imagining you could do this. Only reading past the end tells you that you're at the end. They have dropped `upToAll:` which is replaced by

```
(... ending: ...) rest
```

i.e. a substream that finishes at the right point is used; this gives you more with less API.

Michael has done many experiments using Xtreams for parsing. The API is stable and safe to use. The documentation (generated from the code, so it should be up to date) is at http://code.google.com/p/xtreams. The code is in the XtreamsDevelopment bundle in the Cincom Open Repository.

Q (Georg) The Smalltalk standard has 23 pages on what streams are about; how do you relate to that? The standard makes some of these mistakes. For example, class structure is not part of the standard but the read stream has to support the positional interface and every write stream must support contents. This is Xtreams and perhaps it can never be called the standard or morphed into it. It will always be Xtreams.

Q (Annick) Do you know the "Implementation of backtracking for Streams" paper? Yes, it is of interest.

Q. Licence? MIT.

There was some discussion of the syntax. Their choices were to indicate the lazy/progressive form of Xtreams. They rewrote it from scratch four times so only now can they really invite outside scrutiny.

Q (Niall) What did you think wrong with Streams when you started that you now no longer believe. They thought `atEnd` should never be used but now accept that that is not practical. Instead, in Xtreams, you have to handle the exception. In networking, connections break so you should never rely on end of stream being end of message: you should always have elements being correctly self-delimited.

Q (Richarte) Someone rewrote streams (and called the result Flow); any use? They looked at it but it was more about process synchronisation. It had different goals. Xtreams is about being able to process the characters coming out of anywhere; it is about the abstraction of 'this is just a simple character stream' no matter what is underneath.

Q (Tim) This is great work. Is Cincom is offering this on the MIT licence so all Smalltalkers can use it? Yes, indeed, that is the intent. Alan confirmed that it has been duly released on that license after discussion in Cincom. They believe that if a better streaming library were proprietary to Cincom, that would not be less beneficial. Martin believes we cannot make dramatic progress on the existing library so we have to have both and just accept it; the old is not going away anytime soon and probably not ever.

Michael and Martin are looking at XML parsing because that needs streams of streams.

Michael then started an image and showed that the whole framework is documented. The classes have comments with examples and explanation. He scrolled down all the terminals they have (e.g. SharedQueue's are peculiar - if there is nothing on the queue, reading will block). They will move Cycles and compression into the core if Squeak is OK with that (i.e. with the FFI for the cryptography).

He showed a PEG parser that let him receive file-ins from Martin over the web and, as the Smalltalk parser has to rewind, he could have read infinite files of Smalltalk code. The PEG grammar is shorter than the Smalltalk grammar. His PEG grammar is the bootstrap that creates the Smalltalk

objects that do the parse. (I cannot resist quoting Michael's sentence, "ParserParser is a parser parser for parsers, i.e. it creates the objects that let you parse." :-))

(In PEG, !. means not something, i.e. ensure you've parsed to the end of the expression.)

Michael has an IRC client using this stuff (IRC sending is weird!). Martin has implemented SSH2 in this stuff.

Since Smalltalk-80 streams implementations are incompatible between platforms, reimplementing old streams on top of Xtreams (if they do it) will itself have to have platform-specific aspects.

Q. Anything you want but can't do or have not yet done in Xtreams? They want to reimplement NetClients using Xtreams. Michael would like to do a JSON parser.

Helvetia, Lukas Renggli
This is *not* a presentation on Seaside. Smalltalk is a highly dynamic but the syntax is of course not dynamic. You can change the compiler but then the debugger is broken, the tools may be broken, the source code management may be broken. Lukas wanted to change the syntax of Smalltalk dynamically.

Switzerland uses different official languages and has different language areas, and people live all over the place, but the infrastructure is everywhere the same; the same money, supermarkets, laws.

He opened the normal Pharo debugger. and showed it parsing

```
rows := SELECT id
  FROM users
  WHERE username = @(aString |= /\s*(\w+)\s*/)
```

So how is this done. He wanted to adopt (change the way the language behaves), to extend (add new syntax as a first class entity) or to overload (make the syntax behave in additional ways). The result is having a different set of languages active depending where you are in your code. However all code should use the same data and code abstractions. And of course the tools must work on all of this.

The lowest level of Helvetia is a macro system (some LISP similarities). Above that you have a language-box system that lets you have several languages in the same method. WikiParser is the technology powering this.

Lukas demoed on a fresh system he downloaded last night. Introduction of closures in Pharo has forced him to rewrite over the last half-year and this newest image still has a couple of bugs; it runs perfectly in a pre-closure image. He noted some terms: a pidgin is a simplified form of the host language that adds new constructs and semantic meaning. A creole is a new language formed from several languages. He showed a language for placing elements against coordinates in a simple diagram:

```
row=grow
row=fill

column=grow
column=fill

(1,1) = label
  text: [:each | each name];
  borderColor: #black;
  borderWidth: 1.
...
```

The above code makes no sense in Smalltalk. Nor does a pidgin related to it, e.g.

```
shape [cols: #grow; #fill; rows: #grow; #fill]
...
```

He progressed to a creole. The traditional Smalltalk compiler inputs source to its parser. Its output, the parse tree, is analysed and the result used to generate bytecodes. Helvetia makes any part of this model changeable by adding hooks so the output from any step can be modified before input to the next. Thus a pidgin uses the normal Smalltalk parser, then transforms then analyses and generates. A creole takes the source and parses it, calling to the Smalltalk parser for parts of the process (see his slide for a diagram of this architecture; Georg noted that ObjectStudio 8 was integrated into VisualWorks using the same approach). Rules govern what interventions are made at what points.

Lukas then demoed. His first demo did Roman numerals.

```
self assert: XVI +VI = XXII.
```

via the method

```
transformRoman
  CHTreePattern new ...
```

(He got a bug - it may have been a bug in the latest Pharo, not in Helvetia - but worked through it.)

Lukas never understood the Turing language ‘Brainfuck’ (Niall : why would you want to? :-)) until he could run it stepping through the Smalltalk debugger. In BF, + increments the number on head, > moves the head to the right and so on (unreadable isn’t in it!).

Q. What happens if you press ‘into’ in the debugger? Lucas did so and showed we were in the implementation of the BF machine, in method

```
incrementValue
  ^data at: pointer put: pointer + 1
```

Go to the home page of Helvetia to find info on these and other examples, research papers, etc.

This system works well if you have distinct language extensions but not if you have two language extensions interacting in the same method. In many real scenarios you want to use several languages at once. LanguageBoxes are a delta to the original host language, not just rules on the compile process.

```
change: aGrammar
  LBChange new
    before: ...

compile: aToken
  ...
highlight: aToken
  ...
```

Lukas showed how he integrated regular expressions into Smalltalk.

```
self assert: 'abbbbbbc' = /ab+c/.
```

His browser lets him see which language boxes are active in a method. The order of adding language boxes determines their precedence; the construct is matched to the first language box if possible, then the second, and so on.

A language box must define how the language is to change, its concerns (how it behaves) and its scope. A Regex has primaries of Strings or Numbers or other Regexes, and has / as terminators. The compiler uses `aToken asRegex lift` (lift was 'lifted' from Lisp). Highlighting the matched expressions in the extensions is useful to see what is a Roman numeral, what is a regex, etc.

Code completion can also be extended; that works in his system. You can scope the language extension to apply only to packages, classes, methods, or any subset of the system you can describe.

When in scope, each active language box in turn has its chance to change the grammar. Each method to compile gets this process and so is compiled in the resulting changed grammar. (See Lukas paper in SLE in 2009). Thus the system can handle a mixed extended expression such as

```
findEmail: aString
  rows := SELECT email FROM users WHERE
      username = @/\s.../ matches: aString at: 2 ...
```

Fernando did OpenGL and found keyword args a pain. He asked Lucas to provide support for an arg-based calling, `self assertSeen: #()`

```
LBChange new before: (productionAt... )
```

Lastly, Lucas looked at PetitParser. A grammar is a set of rules. A dynamic grammar is one that executes at runtime those tasks that other grammars do at compile time (the standard Smalltalk approach: 'as late as possible'). This is what PetitParser does, to apply deltas to the host language. Traditional grammars are written to have their process hard-coded and cannot have this easily done to them.

Martin McClure's talk showed an example. Like Ometa, Helvetia lets you specify BNFs but Lukas prefers the longer Smalltalk way of expressing (see slides). PetitParser is a scannerless Parser (you can have a scanner if you want but it makes little sense since composition becomes harder; when there is only one thing there is only one point where you need to connect changes, not two). PetitParser is a Parsing Expression Grammar. The final parser is built from smaller parsers which are in turn built from even smaller parsers. PetitParser is a packrat parser which gives you linear time in most cases. A grammar is defined in a Smalltalky style

```
identifier
  ^#letter asParser, #word asParser
```

A tool presents a visual diagram of specific expressions and a graph of the whole grammar. His example tab generates a random string that can be parsed by the selected string. (His random Smalltalk method looked very strange :-). A random example of a keyword was more comprehensible.)

He has a debugger which produces a trace of the parse of a method, showing which productions were consulted, bold if they accepted the input, not bold where they backtracked. He also had a visual representation (blue for chars, white for blank spaces) showing the time the parser used to progress through an input (the example could be seen to have been parsed efficiently with little backtracking).

Q. How to sort conflicts? Lukas has slides he can show offline and include in the uploaded presentation.

Q (Georg) ObjectStudio did much of this; how does it compare? Helvetia does not subclass the parser, because that would make it more complex to support multiple simultaneous extensions.

Q (Tim) This is great stuff; does it ever end up confusing you? Syntax highlighting is key to avoiding that.

Q (Martin) This works down to the method level? It can go further down because a language box uses method-level granularity *but* it can just change something in some block within a method.

Q (Martin) Save code and reload elsewhere? Lukas annotates class-side with which LanguageBoxes are active. Then you load the code and the LB model ensures the required LBs are installed or else generates fake methods that throw an exception when called.

This works in the Pharo 1.1 core image only. If you want to play with this, install it there.

Bonding with Pango, Travis Griggs, Cincom
Travis heads the Tools/GUI team of VisualWorks. This is an experience report on Pango. Experience and expert have the same root but in Travis' case they do not have the same meaning. There are twenty Pango experts on the planet. He dropped the Asian Text Rendering book to let us hear

what a thud its many pages made.

Pango was named from ‘Pan’ (meaning ‘all’) and ‘go’ (from the Japanese, ‘solution’) so the all-solution. Owen Taylor started it. Behdad Esfahdod then made it represent old Persian and etc.

Smalltalk’s baseline is classic Roman text rendering, left to right, with one-one mapping of code page values to graphemes. In VisualWorks, ComposedText does our tabs and word wrap and so on.

Now we have glyphs from Eastern Europe, from Asia, and from many specialised sources. He showed code for a letter fibonnacci sequence starting from $a - in base VisualWorks we very soon started seeing unprintable characters everywhere. By contrast, with Pango loaded, all the characters printed OK.

He then showed the code page scrolling down and down. It talks an hour to load all the fonts from debian but then you have 12,000 and you can show anything.

He then showed a right-to-left rendered slide of English - this is the experience readers of Hebrew, Arabic and so on have even if we get their characters right. He then displayed some Arabic and some Hebrew with correct characters and displayed right-to-left.

BIDI is short for bidirectional text support. Some web pages will mix text of both kinds. He displayed a list of <language name> ‘hello’ (in that language) where some were left-to-right and some were right-to-left. Arabic text reads right-to-left except their numbers (arabic numerals) go left-to-right. He then showed an Arabic date list where the numbers are left-to-right but the month and day names run right-to-left. (Noury: this left-to-right is modern Arabic - it was all right-to-left in ancient arabic.)

The diacritical twins e.g. A and Â-diacritic. Try sorting BÂA - you get ABÂ. The new VW7.7.1 sort gets it right: AÂB. In fact Â is the a plus the codepoint for the circle. Sorting with unicode is hard because you have to break apart such characters. Shaping is recognising that two characters (or three or four) occupy the same cell. (Unicode consortium would like us to store them broken apart but that is not convenient for anyone). Almost every character in e.g. Korean is done this way.

It is a lot of work to write an index shaper so getting it for free from Pango is very useful. The same script in different languages may shape differently e.g. commas in Japanese shape differently and some Tamil characters modify others when applied to them.

There is also vertical text: Chinese characters, for example. He opened a slide with English, Arabic and Chinese. He rotated it and it did the right thing but only for one font size (he needs to talk more to Behdad). Characters have gravity and Pango knows how to handle Chinese (gravity points down) and so on.

Pango can emit vector versions of all its font lists, on which one can then do a lot. He opened a small tool he has written, inputted the string ESUG and made the font large. (He broke his fonts down into submenus and even then they go off the screen because there are so many fonts.) He then made the curved path of the very large S he was displaying be the wavy line along which he displayed a message (he has not quite solved the handling of characters on sharp corners - "the interested student will solve that").

Pango people then told him to write an editor. He wrote a simple one which he demoed. Pango gives you the APIs to go from points to objects, and where to put the strong cursor and the weak cursor. He entered 1112 and 1161, and it displayed, and 1100 and 1173, and so go to get an 'almost correct' rendering of Hango (the Korean language). He typed letters and numbers in Hebrew and showed the right-to-left / left-to-right switch.

Travis' Pango binding has been implemented similarly to his earlier CairoGraphics. He has stuck to Pango names as much as possible, so that people working on it from Pango documents can easily understand it. `toPangoScale`, `fromPangoScale` convert to/from the 1024 scale that Pango always uses. Pango can use several back-ends but he has only focused on the Cairo one which it prefers.

You can do a lot of stuff with the characters and the fonts. Pango has an iterator (think stream) so you can enumerate over structures in many ways.

Pango is a C library so it has lots of constants. He uses class side methods e.g. `EllipsisMode right`. Like RunArray in VW, you have attributes in Pango. He showed a slide where italic means VW already does it and non-italic where Pango supplied us with something wholly new. Attributes can be got from markup (e.g. HTML).

Callbacks are handled by adding blocks to a Smalltalk registry.

You cannot annotate Glyphs (Travis has suggested adding that to Behdad).

Memory management in Pango was not as good as in Cairo but he thinks he has it sorted now. Since strings and characters are 1 or 2 or whatever long, he had to think of a C-like pointer advance rather than attempting to reference `at: 1` or `at: 4` which might put you in the middle of a complex character.

He opened a VW in which Pango did all the font rendering. He loaded it and pixellated fonts became smooth but now the non-pixellated fonts are not aligned as VisualWorks architecture is trying to align them to integer coordinates.

Q (Christian) unicode does not address tabular digits (unicode with fixed space to align in a table), or rather unicode does but font designers don't care about it? Pango lets you choose between different glyphs.

Q (Christian) how to deal with cross-platform moving of graphics; the

roadmap? Pango is the native solution for Linux, Uniscribe is the native solution for Windows, CoreText is the one for OSX (there are builds for Pango on Windows and OSX). Do it in Smalltalk? Much would be easy but the shaping engines are very complex - it scares Travis to think of doing that. Pango is open-source so Travis can see it all and also the Pango people know Travis is doing Pango in Smalltalk and that validates Smalltalk to that community. (Alan) part of the decision to make is what works for customers. Cairo is now available for customers to use. We want to see people using it for real; if that works out, we may take the *big* step of deprecating some current stuff - many old customers would need the old stuff supported for years.

(Bert Freudenberg: Scratch uses a very simple Pango plugin - give it text, get render, and eToys also hands the Smalltalk text object to the plugin and gets the render, but it does support the look and right-to-left when needed.)

A Method of Reflective Web Application, Reza Rezavi, Ambiant Activity Systems

Ambiant is a startup company in Luxembourg. Reza is working on a framework for implementing web apps. He wants to manage functionality as content, in the same way as you manage your content. The domain is care providers providing services to senior citizens, re-engineering legacy applications for this onto the web, etc.

He uses Pharo, Seaside, Pier, Magritte and Dart. The application is pure Smalltalk (and Seaside's generated JavaScript). He does not think he could have implemented this in any language except Smalltalk with Seaside and Pier. His architecture has four pillars of which two, online adaptation and Execution / Lifecycle Management are the subject of this talk.

From Loic Lagadec and Damien Picard's presentation of 2009 on hardware control, Reza took the idea of a matrix of components with a flow of control between them. If you change the flow of control online, you get different output. Users want predictable results and want context for information.

He embeds a domain-specific modelling language into the application via a meta-language framework. The meta-language is called Pontoon; in it, you write a Pontoon DSL. This is then connected to the web and the idea is that specific care providers program in it (he envisions training for this) to specialise their products for their clients. He presented the ontology. This work grew out of work he did at the University of Luxembourg in 2005-6.

This framework is for apps with requirements that change unpredictably post-requirement and whose end-users are willing to do a little programming (the care providers say, yes, they are). There is an online demo at http://www.afacms.com (ask him for the password if interested).

Pier comes with a root and structure of nodes: Pages, Components, Files and Templates. He adds Concepts, Contracts and Activities to Pier's set of nodes. He implemented many Seaside JQuery examples in two weeks with

no prior knowledge of SeasideJQuery or of JavaScript; just starting from the examples of the SeasideJQuery library, he found it easy.

Each component can render itself so his Activities can render themselves. Using the Camp Smalltalk London traffic info component and the Twitter component (created by Andreas Raab) and created control flow components such as 'WhileTrue'. In the web application, he constructed: LoginToTwitter WhileTrue GetUKTrafficInfo SendToTwitter.

He has set up the website Pontoonity.com to create a community.

Q. Recursion? We could implement it but he has worked in many end-user communities and they need sequencers and iterations, not recursion. An action can be the argument to an operation.

He demoed, logging in on a page that is for the administrators rather than the final end-users, for whom it will already have been customised.) He went to the shopping activity (which has only one action, the standard Sushi shopping which he has reused). He expanded to show its structure, starting with: WhileFalse FillOutCart.

He added a new activity - TwitterExample - and clicked on the Design Activity to create: WhileTrue UndefinedAction. He then started to define UndefinedAction action in terms of other operations, selecting his existing traffic and twitter work and so building up the action. The application tweeted to his account that there was congestion on the M27. Twitter won't accept the same string twice, which terminates his WhileTrue.

In 'show my projects', he demoed at more length, selecting the Contract category (which had concepts such as Address, Credit Card and so on) and creating actions. There was discussion about whether he was expecting a lot of his end-users in requiring them to program, and about prior failures of the visual programming paradigm which some argued this resembled.

BLOC: A Trait-Based Collections Library, Tristan Bourgois (and many collaborators in INRIA, University of Lille)
Traits deliver behaviour to structure. A Trait is a block of methods that provide some behaviour, e.g. Magnitude (<, >, <=, ...). They want to study how to handle traits, when to use them instead of inheritance, and whether they can be used like blocks.

He has studied Pharo Collections. TOrdered has 9 traits: adding, accessing, collection, copying, creation, enumerating, etc., of which some, e.g. adding, are not required by sequenceable classes like Arrays. To create a class, you select the required traits, e.g. OrderedSet requires TOrdered and TSet, then implement the primary methods that the trait methods require.

There remains much to discuss about trait granularity.

Q. Would life be easier if Traits had state? State creates constraints.

Q (Niall) Use the meta-programming collection problem (ESUG 2005

design discussions) to study whether state in Traits is useful? Alexandre Bergel will read the text and then discuss with me.

Q. Will this replace collections? The speaker has been a Smalltalker for 5 months; ask someone more senior. Someone else confirmed that Pharo has started to replace the Stream library with Nile, which uses Traits. There was debate whether Streams were used in the Traits library - perhaps not - whereas collections are used by Traits, so any replacement attempt would have to handle that circularity.

Explaining and Enabling Smalltalk

Scaling Application, James Foster, Gemstone

James handles QA, Seaside, Consulting and Training in GemStone.

Working with Ruby exposed them to what people expect of program environments. What makes the Smalltalk learning curve steep? File-based versus image-based is one of them. Presenting GemStone or Smalltalk in general, you must deal with this. Files are external, compiled, tools are separate, the executable is external, persistent data is external.

To describe image-based, compare it not to programming but to a DBMS. In a DBMS (Oracle, SqlServer), the schema is internal to the system you are working on. You create your database schema and it is in the database. Stored procedures are in the database. Backup and restore and you'll see the code, the data, the structures are all still there in the database. Make a change and you get it immediately applied to the database; you go incrementally from one stable state to another stable state. You may have external tools but mostly you work within the database. Lastly, the initial state of the database has some initial capability: a user table, a meta-schema of RDB structure.

Another example is a spreadsheet. The expressions are in the cells of the spreadsheets. You are coding inside the spreadsheet and when you save and open again your code is still there in the same place (it could be VB scripts). Likewise it has inbuilt capability: the library of functions you call, the default workbook with three sheets.

A computer is another analogy. Computers have built-in capabilities. When you edit and save a file, shut down the computer and start up again, the file is still there. Put your laptop to sleep, then open it again; the windows are in the same location. A virtual machine is especially similar: you can suspend you machine, save it and later resume it. You do not create a brand new computer whenever you do a new task; you start your existing computer personalised to you with various things in various places.

Smalltalk is a programming environment, not merely a language. Objects combine code and data. The image is the disc, the VM is the computer, the object space is the RAM. You work inside the environment, save and restart. The image contains all your tools: debugger, inspector, etc. Everything is an object.

Next, he introduced Gemstone. Traditional Smalltalk has limitations. Your object space must fit into RAM. Only one VM can see it. Sharing between two VMs is doable by various approaches but a challenge as these approaches are non-standard, have overhead of effort and performance, and above all, object identity is not preserved. Object state is lost if the VM exits without saving (in crashes or in transactional cases).

In GemStone, these limitations vanish. The image size is limited only by disk. You can have thousands of VMs, hundreds of hardware servers and terabytes of data, with database transaction semantics. Thus it is a Smalltalk environment *and* a database.

Georg Heeg, STIC
STIC (Cincom, GemStone and Instantiations) is holding the next Smalltalk Solutions in mid-March in Las Vegas. STIC is looking at helping people declare things common platforms. Xtreams (Martins talk) was an example of this.

Teaching Adventures with Smalltalk, Nick Paez, University of Buenos Aires
The University of Buenos Aires teaches computer science and software engineering. Software engineering means you know how to work in industry. Nick works in software engineering and his overall department also teaches civil engineering, electronics, etc. Thus Nick's students will start by studying maths, physics and chemistry (unlike the computer science students). In first year, they learn nothing about programming. In their second year, they learn (for 25% of their time) algorithms and data types (in C and Pascal, and occasionally a bit of Python). In their third year, they learn object-oriented programming.

Thus Nick starts each year with 150 students who know C and Pascal (and maybe Python). The Argentinian software industry mostly uses Java and Microsoft technologies, and the students mostly expect to work there.

Q. It is the same at other Argentine universities. Industry expectations aren't all; many of the students are already working so the students come to the class thinking they already know OO? Nick has encountered designs of objects with only data and objects with only methods from students who think they know OO.

Q. My colleagues keep asking me, "Why do you teach Smalltalk?" (Rest of talk was answer.)

Nick teaches UML, eclipse and Smalltalk tools (in Pharo). He has one 2-3 hour lecture per week, then one 3 hour practical session: the first hour explains how to use the tools, in the second hour the students do exercises, during the last hour teachers answer questions from the students, usually related to the follow-up homework.

Nick is one of a large team. They teach Smalltalk and Java/C# and they mention but do not use C++/Object Pascal (e.g. to show how C++

implements multiple inheritance). The students have two programming tasks in Smalltalk, both with a fixed delivery deadline (failing to deliver means failing the course): one week to do task, returning source plus report (UML diagrams and text) and unit tests.

The tasks are usually games because they see that as fun and so motivating for the students: chess, pac-man, guitar hero, tower defence, battleships, galago, lemmings, bomber man, etc. The language is specified as Smalltalk for two tasks, Java/C# (student chooses which) for a third. The students have to read Evans (Domain Driven Design), Ingalls (Smalltalk Design Principles), etc.

They have a mailing list in which students ask questions and the teachers and other students answer questions; they encourage students to answer questions. There is a public website. They give the students the checklist they will use to mark the programming task.

Every semester they have a review (involves students), followed by a retrospective (done by teacher; they use Diane Larson's book about how to do retrospectives), after which they plan the next semester.

Q. Relearning is sometimes harder than learning; have you tried Java/C# then Smalltalk? Yes, when he was a student in the course, they did Object Pascal/C++ and then Java/C# then Smalltalk. When he started as a teacher, they tried Java/C# then Smalltalk but now they have switched. When the students arrive they know C and are very memory management oriented, so they tried Java first, but they find they get better results by teaching Smalltalk first.

Q (Noury) similar experience in his University. When he started, it was C++ and Java. He threw away the C++ for Smalltalk, and now he does Smalltalk first because Java has so much noise it's hard to teach the exercise. Nick found the same: in Smalltalk it is quicker to finish set up and start teaching the exercise. Noury also tells his students upfront that the Smalltalk market is small, but he helps them find Smalltalk jobs and internships where he can. The speaker confirmed there are companies in Argentina who use Smalltalk. At the last review, many students said "Please do the whole course in Smalltalk", but teachers think best to learn the whole OO paradigm by keeping the mixed-language approach.

Smalltalk in Enterprise-level Applications, Andreas Tonne, NOVATEC

Andreas joined Novatec two months ago and wants to share what he has learned about the scale and kind of problems that enterprise architectures face. Business architecture sits on information systems architecture which sits on technology solutions. JEE is all over this space so where can Smalltalk get into this space. Answer: Smalltalk needs to not be 'not-JEE-compliant' for all 10,000 apps.

15 years ago Smalltalk was mature and Java was a joke to Smalltalkers. In 2010, Java is the biggest player after standard software applications such

as Oracle. Novatec was founded as an Enfin Smalltalk company and now noone in the company knows Smalltalk. So he sees his talk as a wake-up call for the whole Smalltalk community about a situation that will not be solved by any one group (he notes GemStone have done much).

Enterprise JavaBeans are going to be here for a long time: EJB is becoming the next Cobol. This is where the really big money is. We need to be able to make Smalltalk a first-class citizen, not the alien for which you must always argue. So his talk is about what needs to be changed and what needs to be added to integrate Smalltalk to JEE, because we cannot ignore it.

Smalltalk is way ahead of Java as a language. However Andreas argued that at the architectural level, this is not so true; JEE has less boiler-plate, etc. Andreas used to believe it was much worse before he studied JEE 6.

For Hello World, you need a bean that returns Hello World. This needs two annotations, `@Stateless` and `@Remote` at the head of the code and now it is accessible from the outside as an enterprise bean. That is EJB-style today. On the client side, it is much the same, just a head-of-code annotation `@EJB HelloBean remoteBean`.

Now we have a scalable system, robust for thousands of users, and the lifecycle of how beans are created and destroyed is handled for you, and is now completely portable between thousands of systems.

The EJB 1.0 specification was a disgrace to the computing world and took kilometres of paper to show the same example as was on his two slides. It took them 15 years to get here. Java had 4.5 million developers world-wide over 10 years. What did Java learn from this?

- Plain Old Java Objects: separate business logic from framework-specific code as not doing so makes it not portable.
- Beans: components are a good thing. Write it in the right style and let the system discover what it means.
- Dependency injection: Don't do plumbing: your business object should not know how to create your database object. Let a framework/tool inject that.
- Annotations are good.
- Containers are good. The container system is the OS for beans and makes it possible for you to find them, and for them to find each other. The declarative annotations are recognised by the dependency injection.

Smalltalk is do-it-yourself: it would be a lot of work to do the above again in Smalltalk. Smalltalk has no standard model that everyone uses for this, so every project Andreas has seen over the last 15 years does it differently. For example, Ramon Leon posted about Seaside: he needed thread-pooling. James Robertson replied that he has the same in his BottomFeeder. It's too easy in Smalltalk so people write their own instead of being compelled by the difficulty to use a single common approach.

Smalltalk is great for the complex business modelling and Java is (*very much*!) not. Smalltalk is great for rapid domain changes; that's why many banks use Smalltalk. Smalltalk is great for translating between domains; often, when Smalltalk is part of a larger architecture, Smalltalk ends up doing all the integration as that is easier for everyone.

Andreas diagnosis is therefore: don't try to replace JEE; instead, do what Smalltalk is good at: integrate! Then you are part of the architecture. Model the JEE world in Smalltalk. Then we would enable third party tool vendors. Specifically, we need to introduce a component model, add pragma-implemented meta-programming.

One possible architecture is a Java bean proxy for Smalltalk but this does not inherit the Java scalability as the Smalltalk server must still provide that for us. Session beans are, he thinks, the best place to start. Smalltalk should export session beans. Class-side pragmas can handle the annotations.

We need containers that can find our Smalltalk beans. He showed a number of slides of possible API for bean pragmas and deployment descriptors.

```
greetESUG
  <object: #HelloWorld into: #hello>
  | hello |
  hello name: 'Barcelona'.
  Transcript show: hello sayHello; cr
```

The Smalltalk component is never accessed directly; from the Java side, everything is a Java bean but our beans do not compute themselves. Java has multi-threading and this has to be synchronised with Smalltalk green threads. In JavaConnect and etc., doing this synchronisation is painful and so there is work here. Can we start a Smalltalk image from a Java server bean? (The alternative of beginning in Smalltalk and then having it start dependents is not available as the JEE server must be the first started and then start all else.) How should we communicate: by RMI is best, via web services is possible.

In summary, Smalltalk lacks a general server enterprise architecture and so never gets past the: "What standards do you support? ... Goodbye!" cycle. Andreas role is to give existing Smalltalk applications a longer life and to find new opportunities for Smalltalk use. A first task is to use Smalltalk in testing.

Novatec is a fast-growing company (its emphasis in Germany but also looking elsewhere. e.g. China) and it sponsors students theses in relevant areas (in hope to hire them later).

Q. A Dutch enterprise server uses an enterprise server bus, communicating with the Smalltalk server via message queuing. This makes it easy to integrate without caring about green threads etc. All the Smalltalk side needs is a good XML parser and then integration is easy. Does the speaker agree? If you accept this slower-integration-approach, yes that works, but Novatec have customers running JEE environments for whom any slower

connections are not acceptable. Also, with the possible exception of GemStone, you still need to manage scalability. Of course, when Java itself has to integrate with legacy systems then they must accept messaging or similar.

Q. If Smalltalk is to have standard models for these things, JEE is not the only one, e.g. in US market, integration to Microsoft technologies is also big, and Ruby also plays in that space. So should we thing of JSON and/or web services rather than specifically Java RMI? It is really helpful to mirror the architecture of the other side on the Smalltalk side so concepts can be moved from one side to the other without disturbing anything. To achieve the same in Microsoft, you therefore need a different model. Yes, Novatec is making an explicit decision for Java.

Q (Janko) A first step towards this is web services, i.e. on all Smalltalks not just on VW where it is already (Niall: it is also on VA)? Yes, web services is a good first step but they are the hidden transport layer, not the top-level application communication layer for this scenario.

Q (Lukas) Some University of Bern work by Fabricio Perin has already done something on this using Moose etc., so you want to look at his work.

Q (Alan) the pieces of this are not too hard: RMI over IIOP is already there. Making the server robustly scalable is more work but two people for six months could create a very good start on this? Yes to create it but the important thing is that it be accepted and used in the community. The follow-up tools effort inspired by acceptance is a key part of the vision.

Q (Tim) As regards inversion of control and dependency injection, many of Tim's Java friends are backlashing against that because you cannot track who injects what where and it is very hard to maintain, so they are reverting to constructor objects? Swing started injection early and did it in XML and it is an enormous pain. (Andreas) Today, the advice is to write annotations and treat them as default documentation. (Tim) They are a pain to find and the constructor is no less indirect. (Hernan) Annotation is coupling: you use stateless and then you want to use stateful but now the annotation is in your way, so these JEE solutions are not good, just because of the mistakes they have made. (Someone) JEE has captured the market, so we can model JEE in Smalltalk without thinking all its answers are best.

Power to Communities, Stephane Ducasse
Stephane has noticed some patterns and anti-patterns to do with people and collaboration in open source projects.

He also warned us about the video he showed yesterday; he's not too keen on its message not to pay people. :-) (much discussion)

Sharing is positive energy. Participating is rewarding and fun. You freely learn something for free.

Communication is important. Be transparent, no private emails. He

discussed examples when Stephane et al did Squeak 3.9 and others did 3.10 because some had not such good English (and perhaps Stephane also had been the same). Janko sent a fix. Stephane said put it in. Bert warned wait a moment, this fix breaks X, and having the commit-diff was useful to get that communication. A "Feature X does not work" guy sent emails to Stephane for 3 weeks whereas if he had emailed to the list, he'd have discovered it was OK in half-an-hour.

Public emails also shows that the outside world sees what is happening and that is how people get in (by listening), and simply how they learn.

If a mailing list gets too many emails, people slow up reading and sending emails so these lists auto-regulate. However don't write long emails. It is better to write three short emails than one long one. Write *short* emails. (Niall: this is a message that I need to take on board!) Related to this, do not break the thread: one thread, one topic. Do not use your reply to change the topic of the thread. That is the way to cause long unproductive threads.

Fun is allowed, it is not essential that *every* email be useful.

Answer the question, do not turn aside. Always take another few seconds to read the question in the email. If you are an English native speaker, be aware that idiomatic expressions may not translate (even French and French Swiss can lose each other with different idioms).

There is no "I want that", "the maintainer of the package should do that". In the community, "we should do that" as noone is working for you.

When reporting a bug, always remember to say which system, which version, etc.: give the context.

To companies: if you rely on a software system, it will not magically get better; invest in it.

85% of bugs are really trivial: focus on them. And don't keep your fixes on your hard-drive; don't expect people to look on your hard-drive for fixes.

Ignorance is not bliss: licensing in software, Panel Discussion
Julian Fitzell and Jason Ayers presented a few slides to provide context and some suggestions, after which the panel started. Panellists were Bert Freudenberg (Squeak), Markus Denker (Pharo), Henrietta (IP lawyer), Julian (Seaside) and Jason (corporate).

Slides (25 in 20 minutes): open source projects are better if you get license issues right upfront. Julian and Jason are not OS experts - their slides may or may not be right.

"We'll deal with this when it comes up." The trouble is, you and your users have been infringing and the problem does not start when it is discovered. If the software has no value, it may be easy to sort out; if not, not.

Oracle / Google example: Sun had patents it was uninterested in, but when it was purchased, its value included these patents. Penalties can be commercial, professional and even criminal.

Copyright is what it is about. The GPL only works *because* of copyright: the GPL can only restrict forms of future usage because the GPL licenser owns it. Copyright is about a tangible, not abstract, creative work with some originality in it. The moral rights (right to be identified as author, etc.) are not transferrable. There is also the right to exploit.

Today, you do *not* usually have to put a copyright statement on something in order to have copyright on it. Copyright laws are very country-specific and international treaties have standardised it only to a limited degree.

Patents: Oracle is suing Google over android. People have built up arsenals of patents and this cold war may go hot. GPL3 offers some protection from patent rights but no protection from 3rd party claims; if your GPLed code infringes another patent right, its GPL licence won't save you.

Copyright ownership means you can copy and distribute and allow others to do this. You can assign rights permanently or license others without assigning away your rights. In an employee-employer relationship, your employer may own what you do at work, and it may be your spare time is also owned by your employer in some places or cases and by you in others.

If many people were involved in an open-source project, each can only transfer rights to what they own. It may be worth setting it up so all rights are transferred to one entity upfront. Then you must be clear who it is who owns. Did your ten contributors have employers who were the real owners?

IP licensing means the owner of the IP authorising third parties to use it, usually restricted by time or what they can do. You can agree anything but without some agreement you have no legal right to use something for which someone else has copyright ownership.

A license is between an owner and a user. It does not involve a distributor (who however may accept warranty duties).

There are original works and derivative works. Forking a code base or translating a book (that has no word the same as the original because it is a different language) creates a derivative work. Doing something based on a translated book, you need to consider the rights of the translator and of the author. What is and is not a derived work can be the subject of much argument. Everything you do will in fact be derivative of something, but Julian urged us to go the extra mile(s). Be aware that doing a true clean room implementation is *very* difficult in practice. Google hired programmers from the Sun team - and now Google has problems to show their work is not derivative from work that Sun owns.

FOSS licensing is evolving. There is GPL, GPL2, GPL3, Apache, MIT and in the future licenses will evolve further. So you need to know who your

contributors are in case you need them to re-license. You need to know what their employer contracts are. Freelancers can also in fact be on contracts that assign to their client everything that they create during the life of the contract.

Licence state of future releases can revert. Oracle have pulled most of the SUN Java projects from OS back into commercial state, mostly because there was not enough of a community to keep these projects going. So you may need to consider ongoing availability: a future licence claim could see you saddled with fees you can't pay or being stranded on old hardware or, if an OSS is pulled back, maintaining it yourself.

The best protection is a commercial vendor warranty. Mitigation is next best: have an escape route. Due diligence is third best: just try and know nothing too worrying is there. Some software - e.g. BlackDuck - will scan your software to see if anything that should not be there is there.

You need to weigh for yourself whether these risks apply.

Risks to a developer include viral infection from some software pulled into your software that has a licence problem. Oracle say none of their software offerings include open source. Pollution is very hard to clean up. A 'derivative work' may have many implications and reimplementing may still be claimed to be 'derivative'.

Most commercial software provides an IPR indemnity, including an unlimited liability. The user is then sure *they* will not be sued. Most FOSS licences do *not* include such an indemnity. Thus in Android, all its developers, not just Google, are, in principle at least, in line for a visit from Oracle's legal team. Thus the commercial customers of Redhat are requiring them to provide indemnities.

Most European jurisdictions require software suppliers to provide a warranty (for 2 years in Germany but it can be limited to 1, less elsewhere). Usually FOSS warranty cannot be enforced or not fully enforced within these jurisdictions so *the actual developer* has a warranty to any user. That is not what was expected or desired but that is the case. (You only really know where you stand as, if and when you are in front of a judge and they make their decision.)

Q (Monty) If you use MIT, the license says you must maintain the copyright notice so please do that. Monty had to pull software from GemStone. Also no licence means not usable. Someone in Denmark therefore sent them 'you agree this software has no licence' as the licence and that was usable.

Henrietta confirmed it is important to read the licences. MIT does indeed say what parts you must reproduce. Many very liberal licences let you do a lot provided you comply with what you read in them.

Q. As vendor, what license do you prefer? Monty finds MIT fine, BSD fine,

later Apache is fine. If your company is ever bought, you will go through the wringer.

Q (Stephane) French and US copyright law differences? Henrietta is a German lawyer, so her knowledge of other law is limited. The general concept of how software can be owned, licences needed and usage restricted, all that is the same. The definition of a derivative work in US law differs from that in German law, and probably differs from that in French law, and that is why GPL does not define the acts that are covered but instead refers to local jurisdictions.

Q (Stephane) Employee work owned by employer? Julian negotiated with Cincom to have a waiver for his Seaside and other ongoing OS work.

Q (Julian) if owner is in one country and user is in another what applies? The license will usually say if it exists. If not, international agreements will say. The party providing the relevant characteristics will decide for them.

Q (Niall) what are the limits of 'mental pollution', e.g. if you see some pattern presented in a public domain paper or conference, perhaps with code to explain it, and you think it is a good pattern so write similar code into your system? New code is not copying, so rewriting corrects copyright but not patent. There is also trade secrets law, which may create the implication that in given cases you were obliged to be confidential. (Niall) So if patterns presented with code at conference, that is free of copyright, not of patent? Answer was essentially that it would not be a problem, but of course there are many defensive software patent portfolios with much in them. Also patents have a long period before they become visible. Henrietta's clients have been very surprised at patent research. (Jason) look at Google patents. (Bert) Be aware that patents apply to things you invent yourself.

Bert: by now I know much more about licences that I ever wanted to know. No amount of due diligence will protect you from the possibility of being sued. It's all about risk. The larger companies and communities get, the more risk.

Marcus built Debian squeak packages long ago. The Debian license was intended to be OS but in fact was not (the lawyer who wrote it did not understand OS) and so there were many arguments ending in a major re-licensing effort. It is worth sorting your licence early.

Jason: Cincom has things put into OS and has thought carefully about the OS licenses it uses.

Q (Stephane) Can Cincom people sign MIT license? Henrietta: there are strict rules but it depends from state to state, but the degree to which it reaches into their spare time varies. Generally every contributor needs to know what conditions they are under and what they can sign. Spare time software may relate to the job so usually the differentiation is whether it belongs to what you do at work or not.

Q (Joachim) Changes in licences? What must Joachim do if, after selecting an OS item, its developer changes the licence. (Bert) Licence applies to thing you downloaded and a change can only apply to future versions. You may be unable to upgrade but you can keep what you have. Also, the GPL is fundamentally incompatible with Smalltalk, as it talks of linking and so on, which makes no sense in Smalltalk. If the license makes no sense in some part, does the rest apply? (Henrietta) whether the GPL applies does not depend on whether it fits or not. (Bert) MIT is easy to understand "do what you like and don't sue me" whereas Apache is similarly liberal but longer and harder to understand. (MMc) GPL2 does not mention linking. The free software foundation says it means linking but the license does not says so, however avoiding it is wise since there is doubt.

Q. We have a licence from Cincom. If we use Seaside, is our licence agreement with Cincom or with whom? Jason believes Cincom provides warranties and support so the Seaside part is with Cincom. Henrietta was unsure. Do users have to read all the licences of all the components in our public folder.

Q. Andreas Tonne: if you load the component, you know have Cincom + component license.

Q. The questioner has the impression that the GPL is the most popular license in OS community and the most legally advanced, so should we have dual licenses and ask the software foundation to use the GLP? (Markus) GPL is more complex than MIT and simplicity wins. The community is tired of licence discussions. MIT is also practical in letting you use it in products without worrying about viral. (Julian) GPL is aiming to use the threat of licence fees as a form of prevention. (Bert) To re-license Squeak would mean starting with Apple and then going through 500 contributors; it will not happen.

Q (Martin Kobetic) If we upload our software, are we obliged to comply with any warranty requirement of any country where it was downloaded? (Jason) whoever said the law was sane! There are ways to mitigate. (Martin Kobetic) But under some jurisdictions some issues are un-mitigable? (Henrietta) In a commercial distribution scenario, there is a sales transaction with remuneration. This could be a combination of software and service. Thus you have local law demanding a warranty and the MIT excluding all warranty. There is a conflict of laws that is unlikely to turn against you when you contribute on a private basis, but is more likely where OS is included in a product.

Q (Christian) I am now terribly confused. I use Cincom Smalltalk and components they distribute and some extensions I have made and intend to open-source; how does this play out? (Henrietta) You must make sure what you open-source is free of proprietary rights, i.e. your own work. (Alan) This case is no different from VisualWorks and ObjectStudio projects that depends on Microsoft libraries or whatever. We can open-source our work but not the Microsoft libraries.

Q. Glorp license is LGPL(S) and also in squeaksource under MIT? It is under LGPL(S) and Alan is moving it to (probably) MIT. Squeaksource is blank by default and people must choose licence.

Q (Nick) MIT preamble include: where? Class-side method. Start of sources file.

Show us your project(s) in 10 minutes maximum

First Experience of Smalltalk

Alex has 10 years of Delphi and Java and C# and has now started a Seaside Pharo application, deployed on GLASS. He really likes Smalltalk and expects to keep working in it for a long time.

The image concept took some getting used to, and the menus took time to learn. After a while he realised you need a workspace to interact with your image. The debugger is great.

He started using Monticello to deploy packages. It works well and is portable between Pharo and GLASS. You only have your package; if you break it into several packages, Monticello does not follow. It has cross-dialect issues e.g. GLASS objects to the euro sign "somewhere in your code".

He was often pleasantly surprised to find a feature in the user interface - that he should have found days or weeks earlier, so this is not in fact praise for how the Pharo UI is laid out. ChangeSets are very useful and very easy to lose. Someone who has used Smalltalk a lot will find the change sorter tool. So generally, discoverability of tools is poor.

He wanted to use Pier. It looks great out of the box and the demos look good but after a few hours he had made no progress towards doing the one simple feature of Pier that he wanted to use.

Collection protocol naming - `contains:`, `includes:`, `includesAll:`, `includesAny:` ... - was not as immediately clear as he would have liked.

The one-click tutorial is really good to create your own application but you really need a mentor to go further. We need to improve libraries and tools.

Bibliocello, Dale Heinrichs and James Foster, GemStone

Bibliochello, guided by a configuration, looks into a repository of .mcz files to do directed searches. Philippe Marschall and others were working on squeaksource 3.0 which has a very nice RESTful interface, and so he decided to add issue tracking and then he thought "doesn't Pier do that". So he put it all together to do context-specific searches. He took .mcz files, unzipped them, cracked them open and parsed the Monticello definitions so he knew enough to have senders, implementors and references.

Bibliocello is more like an incubator for a replacement for squeaksource than a completed replacement. The website is certainly crashable.

He demoed ("There are passwords but I think you can guess what they are"). He went to the Seaside project page and into 'Repository Versions'. He showed all the packages in the configuration for Seaside 3.0. He showed lists of classes not defined, messages sent but not implemented and so on (some of these are defined in prereq packages, of course).

He looked for `foo` in source and found 22 mentions of it, some of which were senders of it. He found that TestErrorInitialization shows up in two packages.

Get it from http://a-bibliocello.gemstone.com/.

Seafox, Nick Ager, GetItMade
Seafox is a Seaside plugin for Firefox. He demoed, opening the 'Welcome to Seaside' screen. At the bottom of the Firefox UI, there is now a starfish: if you click on it, the page transforms into the Seaside render methods that created it. He swiped them, pasted into a new method and (after a quick `WAAdmin register: DemoComponent asApplicationAt: ...`) ran it to prove this worked.

When Nick was learning Seaside he would have wanted this, so he created it as a learning tool. When a web designer, a graphic designer and a Seaside designer work together, they do not see things the same way so, after thinking about a DreamWeaver plugin or a Smalltalk parser, he decided on a Firefox plugin. Mapping from e.g. ... to WAUnorderedList is complicated by the fact that many tags don't create their own brushes; they just create a generic brush. He used ObjectAsMethodWrapper, iterating over all the methods on the canvas, to wrap the brush method, so he can see whether it was a custom or generic brush and from that he can get the html tag. Thus the tool obtains a list of tags and their associated render methods, so generates information that feeds into the tools that create the plugin.

His mapping is not yet perfect. It can fail if there are more than 256 literals. It can find attributes it cannot understand ('unmapped attribute'). So now he wants to integrate with Firebug. He wants to add syntax highlighting and to serialise the DOM into JSON in Smalltalk.

David Gorisek
He has is working on Ajax integration into Seaside, a Talkback system for Dolphin 6.1, and other things. He talked about his Dolphin project as there are lots of Seaside projects here. He showed his 'Doing business in Slovenia' site. He started developing it in 2002 and it was a little ahead of its time (people were unused to online applications, unsure they could trust it to store their data) but now they are the biggest provider in Slovenia (and also have translated the software to Spanish and Portuguese).

You login, popup a list of products, select one, identify the buyer, issue an invoice, print it, etc. The application will also do eBanking, payroll and other such accounting tasks. This software is very lightweight and all Smalltalk: no 3rd party components whatever. His demo ran on a Linux server with VisualAge VM although they developed it on Dolphin.

Christian Haider, smalltalkedVisuals
Christian started up a VW image. Christian is a graphics guy. He has a product which does graphics for newspapers and businesses in that area is moving from postscript to PDF. He therefore implemented a PDF system and will open-source it under MIT (it is in VisualWorks and can easily be ported to other dialects). PDF had a good spec so it was fun to implement.

He showed a PDF - a graph - generated from his program. He looked at its Smalltalk. He read 10 objects of 14 - his reader reads lazily. There is the doc info and it starts with a trailer. PDF is an unordered list of objects and then data on the cross-references, which you do not actually need - you can recreate them by finding the cross-references in the earlier information.

There are basic types. The spec is huge, 1000 pages, but don't worry; it is easy to find what you need. Just read the two pages you need and implement that object. He has implemented what he needs and when you have implemented the basic structure you can read it all.

Q. Tests? 160 tests.

His multi-pane display shows the file's structure in detail. Attributes are either required (shown as ! in his UI) or optional (shown as white if present, grey if absent in the file we are reading). He drag-dropped the PDF of the PDF specification onto his tool. It read 125,000 objects. Outlines he shows as a discretionary because he has not implemented it from the spec but you can still read it (usual demo bug, which he managed to proceed).

Serge Stinkwich
PlayerST is a client for the Player/Stage robotic device server. Stage simulates a population of mobile robots or can control real robots. Player is an interface to a robot. He opened the simulation, dragged the robot here and there and then let it wander under the programs control.

In fact, this is a difficult program to use so there is a new program called ROS, http://www.ros.org.

His other project is PharoSugar, porting Pharo to the Sugar environment, Sugar is a free desktop environment designed for children used in OLPC laptops and can also be used on Linux of MacOSX10. Etoys is already there and some weeks ago they started porting Pharo. They reused Bert Freudenberg's debus work. There is a SugarTheme for Pharo. There is a Metacello config for it.

Bert Freudenberg, Etoys introduction
Bert is the Etoys development team leader. Etoys is for children (and adults). Go to the squeakland showcase at squeakland.org and download it. There are lots of examples of use, also downloadable from the web. He showed a game where the computer plays chords and you must sort them into their keys.

The start screen shows you tutorials and demos, and a gallery of projects.

The first demo is just an object that moves around. Kids like to spend a lot of time drawing their object and then give it some behaviour. He showed a one page 'land on the moon' rocket game plus complete source code for it in eToys.

He went to the gallery of projects and looked at an electrical inverter diagram. Another simulates 2000 particles in a gas simulation, showing how pressure is affected: increase the particle count too high and the box will explode!

There are 2 million OLPC laptops today, which means 2 million potential Smalltalkers; he switched to textual mode and showed the Smalltalk. If something is missing in eToys, you can switch to Smalltalk and add it.

He added a pen to the car and drove it to etch a sketch on the screen. He drew a road for the car and then gave it a sensor. He dragged a colour test onto the car behaviour to make it move forward if it saw the road and turn a little if it did not; three lines of code and now the car is following the road.

Translation Support in Smalltalk, Janko Mivsec, Eranova
You have natural language text in your code, e.g. for display on web. How to make this code multi-lingual. He showed an AIDA method and the web page it displayed. He converted a textual string in the method to an association: `#eng -> 'the text'`. He then went to the web page, put it in French mode (no translation yet), and clicked a button that made the text editable; he then typed in the translation. He went back to the Smalltalk image and showed that the translation had been stored on the class side of the method and now the French page displayed the French text.

The inline editing makes it easy for professional translators to work on the web page.

Q. Reuse default translations? If it's a reused component, it's already translated; if not, we want the translation done in context.

Scratch, Marino
The video of Marino in Spanish was explained by Jordi of Citilab. New programs are added to a Scratch image to run robots. The display shows reading from sensors. Software in the image drives two boards which drive robots, turning light on and off, reading infrared sensors, moving robot. We saw the robot's camera displayed back on the screen. This work is in Squeak 2.8.

Altitude, Colin Putney
Altitude is a framework that Colin wrote with code from Seaside. He showed the seaside counter demo as usual except the URL was the same as he steps from 1 to 2 to 3 and back to 2. Call and answer worked - he went to a multiply and divide component and came back. Therefore we have the usual ugly Seaside URL but it is RESTful. Several people viewing the same component can share the resources of the server, so the size of your app is not affected by the number of concurrent users on a setting. An initial

request has state of counter, comes in, registers callbacks, is serialised to bytes, hashed and given a URL based on those hashes. When you get to the URL, it loads that callback and calls that callback, serialises it, redirects the browser, renders component at URL by reinstantiating it and rendering it. So we have content-based addressing. The URL of a component is dependent on its state. The trick is serialising callbacks (which is tricky).

The architecture can specify various ways of associating components with URLs. Serialisation is one way. He showed another: to put the whole state of the app in the URL - very long URL with encoded data. (*Very* RESTful URLs. :-))

You can encode as an integer; he showed the counter integer being in synch with the tail of the URL.

It works with forms. It does a sub-render of everything in the form and creates a subURL for just that form.

He wrote a toy blog application to test these ideas. He demoed. A custom locator examines the state of the components and produces the URLs. The code looks much like Seaside.

He has not done load testing or written any complex apps yet. This is proof of concept and the concept works. Client has state, can change state, could go back to server with altered state.

GSOC2010, Platform Namespaces, German Leiva
Hernan's summer of code project was on platform namespaces. His mentors were James Foster and Goran Kampe. The project was to design and implement an approach to namespaces in Pharo. Pharo prefixes WA*, MC* , ... to avoid name clashes. The result is a reference implementation, not production-ready. It was a requirement not to change the syntax.

He opened an empty Environment Browser. He defined an Environment 'Africa' and a class 'Elephant'. Elephant was unknown in the workspace but `Africa Elephant` can be found, as can `Africa at: #Elephant`. He defined another in namespace Asia. Then in a third namespace, he defined class Zoo. He tried to reference Elephant in code in class Zoo, but again he needed `Africa Elephant`.

He created a workspace in the environment of Zoo so he could reference it without qualifier. `ZooEnv shareEnvironment: Asia` imports the namespace and so makes the Zoo see the Asian Elephant.

GSOC2010, eToys, Ricardo Moran
The projector did not work for 5 of the 10 minutes. He was improving the eToys set. He worked with Bert Freudenberg and others. It was a lot of fun and he learned a lot. The release will be in two weeks. He showed a calendar he made, then a graphing tool, and a variant that plotted an electric field, and another that showed solar flare data. He showed a caterpillar moving and thinking of being a butterfly, controlled by eToys, which he

changed to be thinking of going to ESUG, and some cartoonish animations. He also worked on making the drawing tools of eToys better with alphas and etc.

SqueakNOS, Richie (Richarte) Gera
Two students are now being directed by Hernan on theses on SqueakNOS. The SqueakNOS had no way to save the image from within it (so it had to run on VMWare). They made mouse scrolling work. He booted on VMWare (beautiful multi-Smalltalk balloon picture). They have access to the hardrive and to the 32-bit file system. So now they can see the sources of their own Smalltalk methods. He saved an invitation to come to the Smalltalks conference. (Saw an intentional halt and proceeded.)

Saved image could be inconsistent if the real image is changing in save. Easy solution is to have a primitive that gets copy of whole image and saves that while real image is changing. The audience clapped when it was clear he had managed to save the image from within the image.

They were using FFI but imported Alien and Alien had callback support and so they made callback into the VM work, so now they can do low-level memory management. Calling `Smalltalk at: ... at:` let him get to the page cache and change its values, e.g. to be not writeable, thus forcing a page halt, which duly came up in the debugger (the system keeps running!) and he fixed and resumed.

Seaside on EC2, Jan van der Sandt
Jan showed us how to host Seaside apps. We start a browser on Google. He has pre-registered (he only has ten minutes for this talk) as a customer for Amazon webservices. He started the console, logged in to Amazon Web Services, and saw tabs for the clouds. He clicked on Amazon EC2 tab. You see AMI (amazon machine instances). There are thousands you can choose from. Every AMI has an ID. He's prepared one based on Ubuntu (took him a week but now his Amazon machine instance with Seaside is published in the catalogue). It takes a few seconds to search for a given instance.

There are instances based on Windows 12c/hour or Linux (10c/hour).

His image contains Smalltalk and Seaside (GLASS). The page asked him for parameters. How many instances to start: 1, 32, 64? He chose 32, and he took the small size. You secure your image by logging in with a public and private key, behind a firewall (Amazon calls that security groups - he chose one that allows traffic between ports 22 and 80). Amazon created an ugly long hostname for him - to host anything commercial you must choose a hostname yourself by reserving an IP address then going to your domain name registrar and pointing your domain name to that address.

When the image starts, it is not immediately available. Be patient. Then he managed to connect to it from the browser.

You can manage the system through ssh (you have a private key so you don't need a password) and he has set up a tunnel for VNC. Now you can

load your own Seaside app and have it running within another ten minutes.

Get it Made, Nick Ager
He showed an animation of his app illustrating ordering and prefabrication of the individualised ideas of inventors. The site aims to bring together inventors, manufacturers and 'discerning shoppers'. The inventor keeps the IP. A sufficient number of items must be pre-bought to justify its being manufactured.

The site is built on Seaside and Pier 2. Pier has great content management, does restful URLs, has a blog built-in plus twitter and search, etc. JQuery WidgetBox is a fantastic resource.

Camp Smalltalk London, Tim MacKinnon
It was great. 40 people came. James Robertson filmed interviews - see his blog. 8 people were new Smalltalkers. Food was delivered (thanks, sponsors) and that kept people together. EventBrite was a useful aid in organising it. Name badges are good but do them double sided (same problem at this ESUG). The event should have had a 'show our projects'.

They did tutorials using Prof Stef which was good (but take out a few of the detailed slides), then a short Seaside exercise.

SUnit 4.0 (and 3.3, 3.2): what it is and where it is, Niall Ross, Cincom
SUnit 4.0 has been ported and released in VisualWorks 7.7.1 (and ObjectStudio and WebVelocity), VASmalltalk 8.0.2, Pharo 1.1, GemStone, Dolphin 6.1, Smalltalk/X and VSE, thanks to James Foster, Tim MacKinnon, Jan Vrany, Leandro and Valeria, Yuri Mironenko and a number of others. Is there anywhere else it should be (if so, will you port)?

Some dialects have their own widely-used extensions to SUnit. When porting, we aim to provide a version of SUnit 4.0 integrated into the dialect's standard stream, and another that is pure SUnit 4.0. The latter may see little overall use but is useful when someone is porting a utility between dialects and wishes to run tests in identical SUnits, to eliminate a possible cause of variance. Our tested port of pure SUnit 4.0 into Pharo is available in www.squeaksource.com/SUnit. Our tested port of SUnit 4.0 to the standard Pharo test framework is in www.squeaksource.com/PharoInbox. (N.B. this port does not call `cleanUpInstanceVariables` by default in the basic merge; add it if desired. All other features of the standard Pharo SUnit are preserved) Similarly, in Smalltalk/X there is a pure SUnit 4.0 version and a 4.0 version that also captures the last result of a test run (for use in documentation, etc.).

A major task of the SUnit 3.2 through 3.3 to 4.0 evolution was to make TestResource more robust. In contrast to 3.1 and earlier, a test's resources are set up in order and torn down in *reverse* order. Likewise, a resource is set up *before* everything that uses it and torn down *after*.

A call of `tearDown` on a TestResource is ensured if `setUp` is *entered* not (as in 3.1 and earlier) only if `setUp` completes successfully. This means that TestResource now has the same behaviour as TestCase (the principle of least surprise). This may require some rewriting of resources' tear down calls from

```
tearDown
  instVar release.
```

to

```
tearDown
  instVar isNil ifFalse: [instVar release].
```

(as is already commonplace in test case tear down). We believe this is preferable to the risks in the earlier approach. (The only effect of not doing so is that an early-failing resource may then fail again in its tear down, just as a test can fail twice if such guard clauses are omitted from its tear down.)

Resources understand the `assert:…` protocol, just as tests do, making it easy to refactor between MyTest>>setUp and MyResource>>setUp if code developed in a test becomes slow enough to need moving to a resource.

Resources are made available just-in-time.

- In the run of a test suite, the first test that wants a resource sets it up; all later tests that use it merely check whether, thanks to this first call in the run, set up of the resource either succeeded or failed.
- MyResource>>setUp is now called at the appropriate point in the run by the framework, not by MyResource>>new on first creation of a resource. This unchaining of `setUp` from `initialize` solves a number of problems that the earlier pattern inflicted on developers trying to broaden the use of resources.

  N.B. this means that if you have scripts that manipulate resources programmatically, beyond the normal framework pattern of assigning resource classes to test classes, you may need to replace `MyResource new` with `MyResource new; setUp` in some cases.

Thanks to all the above changes, we believe that TestResource now offers a robust flyweight pattern to testers. Using a resource trades test isolation for performance.

- TestCase>>setUp, testSomething, tearDown
- TestResource>>setUp, isAvailable, tearDown

do similar functions in different epochs. Moving code between the two makes no *functional* difference to any single test. It repartitions behaviour between what happens at the start and end of the test run (strictly, the start of the first test using that resource and the end of the test run), and what happens at the start and end of every test that uses the resource.

Usually the decision on repartitioning is binary: either the code is in a test case and its `setUp` and `tearDown` are rerun for every test, or it is in a test resource and its `setUp` and `tearDown` are only run once per run of a whole

test suite. However if the developer sees that a widely-used resource, otherwise invariant for all the tests in a suite, is corrupted by the specific test they are coding, they can explicitly tear down that resource in that test. The next test in the run, seeing the resource's current instvar is now in the first of its three states (not-set-up, successfully-set-up, failed-set-up) will act like the first requesting test in a run and set it up again.

A second task was to make exceptions pluggable. In 4.0, `runCase:` dispatches on the exception by sending it `sunitAnnounce:toResult:`. Error adds an error to the TestResult. TestFailure overrides this to add a failure to the TestResult. Developers can create subclasses to plugin specialised behaviour (see, for example, the TestSkip subclass in SUnitResourcePatterns in the Cincom Open repository).

There are also some trivial improvements. The code controlling the inheritance of tests via `shouldInheritSelectors`, `isAbstract` is more consistent, and there is a new method `lookUpHierarchyRoot` for rare inheritance cases when you do not want to look all the way back to TestCase. There are some tweaks to aid pluggability, e.g.TestResult calls `self addPass: …` instead of `passed add: …` , so users can subclass it more easily. (More needs to be done to improve pluggability.) A failing resource now logs the calling test. (More logging improvements needed!)

As well as helping code be refactored between TestCase and TestResource, their common superclass TestAsserter is also available as a superclass for any delegate classes that a test needs.

Future plans: SUnit 4.1 will exploit the spread of ANSI to retire the compatibility methods `sunitSelectors`, `sunitOn:do:`, `sunitName`, `sunitEnsure:`, `sunitAsSymbol` and the class SUnitDelay (does anyone use it?) and maybe also `sunitMatch:`, `sunitAddDependent:` and `sunitRemoveDependent:`. TestResult may switch to initialize pattern from lazy initialization, as it already has in Pharo's main branch.

Our aim is to have an SUnit core (the Camp Smalltalk project) without preventing the various dialects from experimenting. The SUnit core will allow robust cross-dialect test running and will be the compatibility base for SUnit variants in each dialect. Therefore our aim will always be to merge each release into dialect's main variant as well as just porting the core.

Our hope (not least because it will reduce our workload) is that pluggability improvements in the core will let tweaks be moved out of dialect variants into dialect add-ons (extensions and subclasses) from time to time. However new ideas may put new tweaks back in from time to time, to be move out again later … or occasionally to be moved into the project core if they are that good and that backward-compatible. Such cases will be an application of the "first make it run, then make it right" principle, where we replace "last make it fast" with "last make it cross-dialect" (i.e. in the core).

So SUnit welcomes ideas, but will be cautious about moving these ideas

into the cross-dialect core.

Q (Hernan) when restarting in debugger, can you restart resources? Yes, explicitly: sending `reset; isAvailable` to any resource class will cycle it. Previously, if you restarted a method in a test that was below the outermost level at which resources were setUp and their tearDown ensured in the framework, then their state was as you left it (or as you had changed it in the debugger). In 4.0, every test sends `isAvailable` to all the resources it needs, so explicit resource tearDown in the debugger is safer. It is also less costly to leave a resource in a failed-to-setUp state - if you then resume, only tests that need it will fail.

Arthur van Schijndel
He works for a Dutch care company (starting size 10, now 150) after a non-Smalltalk history in business architecture software and similar. He found Smalltalk, solved a few problems, created a few frameworks. He uses Pharo and Seaside with Apache and a proxy in front of it. He uses Glorp for persistence, and found it worked well. He has meta-descriptions to create his business objects and his UI.

His app manages company data and people. He showed the page for quotation for a new client. His workflow tells the people what work they have to do today. The workflow has got more and more complex but he just changed the workflow data and his system still works fine.

Hwa Jong O
He is from Korea. This is his first time at ESUG (and in Europe) - it will not be the first time next time. He makes money out of iPhone development. He showed a Squeak bouncing ball of himself connected to other balls of Smalltalk, ESUG and so on.

Another set of bouncing balls showed his projects: LazyRabbit, SmalltalkQuiz, MorphoPhysics (which does these bouncing balls). He showed his ToolDAD project. The DAD is for drag-and-drop. Smalltalk tools have drag-and-drop, often just within each tool. He wanted drag-and-drop from debugger to workspace, from one inspector to another. He showed coloured balls and windows (coloured to distinguish them) and drag-dropped a colour setting from one to the other. Then two workspaces - he dragged a value from one to the other. Then he tried to do the debugger one, had usual demo hiccough and reverted to the Jing recorded demo he'd prepared earlier. He inspected a rectangle and a slider. He dragged a value from one to the other, then opened the browser and dragged a value from a method to the inspector.

Other Discussions

Mike Taylor (Instantiations) enjoyed the energy of ESUG. (this was his first visit). Instantiations have sold the Java part of their business to Google. The Smalltalk part is now even more their focus and better funded than ever. John remarked that putting Smalltalk into Google would have been like putting it back into IBM (I so agree). This way is much better for Smalltalk, and for Instantiations.

Stephen Egremont is working on improving the row-of-buttons widget in Pharo. He has a commercial Seaside-on-Glass application.

The Exegis Seaside system is in GemStone, and wants to use PostgreSQL to backup its transactions.

Seaside BoF, Lukas Renggli, Julian Fitzell
What do Seaside users want, how do we encourage contributions.

There is lots of good stuff in squeaksource; can we indicate what is good and what is not? Can we have a place where users of the UI plugins could vote on their quality and or a location where brief videos on plugins are made? A central location is important (maybe link from Hudson). The first cut could be a Pier site with a page per plug and a link for where to get it. Summary: create an easy way to contribute. However not everyone uses Monticello; e.g. Boris publishes in VW to the Cincom OR. Dale thought GemStone could host that Pier site; he will put something together.

Another thing wanted is one-click loadable plugins from the website. Julian would welcome help on that; what is the full list of what is a plugin? However discoverability is the priority; the instructions can then be followed by whoever wants it.

Martin did the integration with the Opentalk HTTP server. It looked like 3.0 addressed issues but he recalls Seaside was trying to write headers and that is not Seaside's job; the server should have control of headers and when they are written. Seaside's job is to inject the headers the server writes: (1) call Seaside to say what kind of headers it wants; (2) callback to get payload. These should be distinct as they may not happen at the same time. Lukas asked Martin to create a bug report (server adaptor).

Joachim Geidel: everyone wants widget objects on the server so we can write to them, so use ids to get Ajax callbacks. Perhaps have control of the whole DOM tree. The more we go from simple input fields to partial replacement of more complex things, the more we are interacting with the DOM tree: first render does div, second will put div in div, and so on.

Lukas is aware of the problem but requirements differ for different projects. Some frameworks try to handle this by generating a lot of boiler-plate around your widgets, and that is not the Seaside way. Grease, and the work Joachim has done and others, have all come up with similar solutions for this. (Halos sort of generate boilerplate - and breaks most CSS.)

The mailing list is fine for discussion but the decision must be documented somewhere else.

Shall we have a sprint on it? From the point of view of sponsors, what are the requirements to sponsor a sprint? (Arranged an off-line discussion.)

Generally, the increasing use of Javascript has implications for Seaside. Johan suggests every DOM element be a Smalltalk object (intelligently

converted to Javascript by an automatic process).

Reza Rezavi argued for a new web trend of user-generated services, complementing user-generated content. Can we have a repository of reusable Seaside components that end-users select and compile on line? Several people though that was asking a lot of users. Niall said the past history of attempts to use it suggested that the visual programming paradigm tends not to scale. Cmsbox and Pier show the way to go.

RESTful components? Dale finds the stuff he uses on the bibliochello site and it works well. It recognises which URLs are RESTful and before it gets to the Seaside handling it extracts the RESTful stuff. Dale would recommend pulling it into the Seaside base. Julian agreed: better support for RESTful URLs is worth having. Julian hopes to make it possible to use the session even when not in the render loop so the session would persist through RESTful URLs.

Andreas found (in VW) that `isolate` was handled at a low level so you were kicked out of the component, and that can puzzle developers. Sometimes you want to isolate a session and sometimes just you want to isolate a component. If you can't reconstruct the component, you may want to configure what happens and plugin a specific component to show instead of just seeing the root component. (Julian) A planned refactoring of session will fix it (and it's out of Seaside 3.0 to come back when fixed). Also, `lightbox` needs work and configurability could be improved.

Can we have the back button work in the context of AJAX update?

(Tim) While developing in Seaside RC3, when suspending and opening again, Comanche threw a socket error that raised a walkback in his image; is there a bug report? Comanche is old, and rarely maintained; we need to find a new solution.

Can we have larger examples to start with - no more blog servers and suchi stores? Can we choose which render loop to enter based on the header? If we use RESTful more we may need this.

Boris needs a beginners guide to Monticello (see CS talks at end of this document for some info). There is Monticello with VW and he needs a beginners guide to that. (Alan and I accepted an action to ask Michael Lucas-Smith about that. Post-conference: a post, with link to video is at www.cincomsmalltalk.com/userblogs/mls/blogView?entry=3418634144.) Also, he wants a better Seaside page to explain what the submission process is, linked to the pages on the specifics of how to submit in VW and other dialects. If you do not have commit rights, you send a .mcz (preferred) or a change set (acceptable) and they review it.

There is the Seaside mailing list and the Seaside dev mailing list, the latter for those who contribute, or may contribute; subscribe as appropriate to your activities. (Julian thanked all those who have answered questions on the mailing list.)

Hosting is the first question newbies ask. See Jim Robertson's webcasts and Jan der Sandt's 10 minute talk for answers.

Pharo BOF, Stephane Ducasse
There are various infrastructure tasks. What will we do with Morphic? We should integrate Veronica's Torch tools. We have Announcements and the RB has models of changes and undo. We have OPAC.

What is the state of Alien (FFI)? Alien is in the Mac VM at the moment but the VM team need to integrate the changes need for the other VMs and there is standstill at the moment. There were some questions about licenses which are now resolved but nothing has happened since. Partly the issue is just knowing how we should write it. Igor built the plugin 7 months ago.

Martin McClure: Alien is good but the design is not suitable for anything except x86. It needs a better type system. If it is to be improved, it must be done by someone new. Generally, Pharo needs a good way to communicate with C. They have arrangement for an engineer to work for two years on Pharo and this is a good item for that engineer to start on.

Pharo has gone through a counter-productive discussion about WebClient and licenses and so on; we learn fast so we know now what to do and not do. WebClient will not be in the system but it will work (it works now with Philippe's changes). We don't want to have comma converting everything to a string because we do not want date and time to be equal to their string representation; soon we'd have DNU everywhere.

Underscore for assignment: if old code has this, just fix it: switch settings to load it, then fix it, then switch settings again and save it. A clean-import utility for loading old packages from Monticello would be useful (it could also clean WideStrings).

What prevents people participating? Time was what most people said as the reason. There are the things we would love to have and there is the time we have to do it, and there is always a mismatch.

PharoTaskForces is a special repository for work that changes large parts of the system. First versions of major cross-cutting tasks should be put there (e.g. Stephane's rework of the package system was put there, reviewed by Lukas, etc.)

The review process is to minimise code rot by reviewing and integrating without too much delay. Their concern is not to control what goes in so much as to ensure the system is working. Thus the integrators are the bottleneck; if anyone wants to help integrate, talk to Stephane.

Markus Denker explained that change sets can have preambles that change things and deltas are created on the server so that could actually corrupt what you were getting, so the process where you get everything uploaded, not just a delta, is slower but safer. The process checks which packages have changed, pushes them to the server and then it verifies that all these

changed packages load.

The split between PharoCore and PharoDev is just a first step to a structuring that will let them load tool changes easily.

They removed MVC to remove all the `isMorphic` sends, not because they disliked MVC. It made it very hard for people to add their own windowing system.

Do we keep tool builders? Do we use Glamour? Stephane is very supportive of Glamour business but he's not sure for Pharo. He has the same issue with Mondrian. This message will affect the stuff on the stack - Stephane finds that kind of programming hard.

Michael Prasse' build process
They organise their bundles to separate their code from VW base code. They have a deep hierarchy to keep a finite number of top-level bundles. There is one bundle that contains everything of theirs. Other top-level bundles contain only code relevant to a specific customer, one such bundle per customer. Bundles are published once a week, packages at any time. A tool finds packages that have been changed.

When changing the base they create

- preBundle - stuff from VW, unchanged
- postBundle - overrides, changes to VW

After they publish a new top-level bundle, a tool creates a new image and developers then work on the new image. You start with a clean VW image, load the Loader package, then choose which project to load. A dialog asks you to choose versions manually or just pick the latest version of each bundle (Michael always chooses manually). They avoid prerequing to simplify the choice process; everything is in the bundle hierarchy. He selected and then the image loaded.

The class-side method `projects` defines a set of projects. The defined structure contains the names (only, no trunk-match strings or other criteria in their approach) of the subbundles. He showed the overall bundle project.

Another tool can collect methods that have changed and add them to a parcel of the patches.

The RMB application
Cor Venter works for RMB in South Africa. A Gemstone-oriented team built the 'real' system over 2 years while a 'demo' VisualWorks and Sybase system was shown to traders who said "give us this immediately." This demo system was used daily by traders, so was forced to let traders in to provide reports. The Gemstone-oriented system did not focus on reports, so the demo system became the real one and still is.

Later Calypso was brought in. "It'll just take a month to configure it. and replace the old Smalltalk system." It was really hard for him to persuade

management to let the Smalltalk system continue while this happened, but eventually he found the words: "You are traders, you hedge. Let 4 guys carry on in Smalltalk while 60 work on Calypso and that is a hedge just in case Calypso goes wrong." Now, years later, 80%+ of the system in still in Smalltalk, and the 60 Calypso guys are going nowhere.

GLASS and GemStone
The GemBuilder in the VisualWorks distribution is 32 bit (with expired key license). GLASS is 64 bit and has the traversal buffers to VW/VA switched off. Thus you can't connect them. If you need to evaluate a GemStone system with GemBuilder, contact GemStone for an evaluation license.

GemStone Seaside has added support for background transactions; you can fire off potentially-blocking transactions (e.g. call to a payment gateway) to a separate Gem that handles them all and responds asynchronously.

JPMorgan drove them to implement mid-level caches; JPM have a network of 500 servers so could not do all to one cache.

Intercontinental now do 13,000 write transactions per second.

Other 2010: Camp Smalltalk in London, and UK STUGs

We had a great Camp Smalltalk in London in July with 40+ attenders. This Camp Smalltalk included 'Prof Stef' Smalltalk training for the newbies and some talks. One idea that emerged (use at ESUGs too) is that name tags should always be double-sided, since they so routinely get turned around.

Monticello explained, Julian Fitzell, Cincom
A .mcz file is a zip file with structure (prior version, contents, ...). There is a file per version of a package. Each file is a full copy (unless its a .mcd file - a diff file - but these are little used). Package dependencies are also hardly used. Support for branch version naming was broken and is now fixed but is hardly used.

The id's of all previous versions are soft-linked. Operations include:

- Load
- Save
- Merge gives a dirty working copy with two ancestors.
- Adopt adds the code to the ancestry. "I took everything valuable from this dead-end branch which was in fact nothing so I just showed it as the ancestor of the current loaded."
- Backport: take part of one branch into another so load previous version, do diff, load diff into second branch.

A repository is a loose collection of package .mcz files whose relationships are dynamically (re)calculated when you view. The differences between arbitrary versions are not UI-intuitive as you must go into the history and walk it to get these.

Seaside has 600 versions of some of 60 packages and that stressed Monticello 1. In Monticello 1, packages were not first class, and it had no way to associate other files (e.g. database scheme descriptions) with the source.

Monticello 2 was started by Colin Putney 4 years ago. Julian joined in the work 1.5 years ago. A self-hosting version proved a dead-end (had to load it to load it). Julian aims to pull out some of the abstract and recursive aspects of the design (like omnibrowser).

The Model layer of Monticello 2 has Elements, Variants and Versions. An Element is a uniquely-identified thing you want to version: class element, method element, pool var element, inst var element, etc. These contain no source as such, just enough to identify them.

One element can have many variants. A Variant specifies state. Variant has subclasses DefinitionVariant (it exists, these are its properties - held in a property dictionary) and AbsentVariant (it does not exist).

Q. Could use properties for cross-platform transformations, e.g. to Envy? There is no application property so must calculate it from package name but user can choose (choice dirties package which must then be saved, etc).

A Version has one Variant (one Variant *may* appear in many Versions and has no back-reference to any version) and one Ancestry linking it to many prior Versions.

This model can live in Gemstone as precisely the above model classes, or can be serialized out to files as in Monticello 1.

Facade has subclasses SqueakFacade, VWFacade, etc. A Facade can be given a filter that does true or false on any element passed to it defining what to version and what not to version in a package.

Path is Environment -> VersionSpace -> VariantSpace -> PropertySpace -> (platform specific) Facade -> actual dialect, actual bytecodes. If a PropertySpace tries to load stuff that does not make sense on the dialect, the PropertySpace will remember them. A filter (block) is called with every element (facade) with every element and property dictionary (property space), with every variant (VariantSpace). A VersionSpace gets the file, builds block with it, passes it to next space along, and so on - then the whole thing unwinds back up again to apply to things.

Monticello has an http repository and a local repository; you synch to your local repository. As Monticello 2 develops, there will be a concept of a line-up. Versions have an ID that lets them recognise that an item in GemStone database is the same as another item in a file or in a relational database (maybe glorp-mapped).

Slices are filters plus identity. (Today, a package slice talks to a package in the image and makes elements for it. Monticello 2 will use a facade to

indirect this.) A Slice has one Filter at any point in time: over time, the filter may be changed. A slice is the set of packages we want to version. There are Slice Elements, Package Elements, etc., so you can version these things (c.f. the config map versions of Envy).

A Snapshot has many Versions. If a Slice says to an Environment, 'save me now', the result is a Snapshot for the Slice. The Snapshot has many versions and it also knows what its Slice is. In Monticello 1, a .mcz file is the serialization of a Snapshot. (In a typical operation, you would tell the Environment the Slice and the repository where you wanted the Snapshot saved.) A Snapshot has a Historian which knows the last loaded Snapshot(s).

On loading, the system looks at all versions currently loaded, and looks at all being loaded, then does the minimal change to move from one to tother.

You could have multiple Environments, one for your image, one for your file system, etc.

Q. Rename Element to Name? Element subclasses to ClassElement, MethodElement, etc., and these are mostly just reified wrappers on Symbols.

PostgreSQL, Bruce Badger
This was Bruce Badger's 2003 Smalltalk Solutions talk, updated for 2010.

PsotgreSQL blows away MySQL; it was the most advanced open-source SQL DBMS in 2003 and still is in 2010. It is descended from Michael Stonebreaker's Ingres. It has advanced administration: in-flight dumping, live cross-database synchronisation, and lots of options for Security (Kerberos is still good).

It uses local sockets or TCP/IP so there is no need to have binaries to access the database. (There are binaries for C programmers and others who want them.) For debugging this, Bruce noted that TCPDump underlies Wireshark. Bruce used the former when debugging but only because the latter was not then available (and/or does not work with Parallels or VMWare).

The super-user for postgres is supostgres. The first (and perhaps only) thing to do with this user is to create a user for yourself who is authorised to create databases and then logout.

To get started, do

```
apt-get install postgresql
createuser -d -a guest
```

then as guest, or whatever user you create, do

```
createdb myTestDatabase
```

When you run these command line tools then you connect over the UNIX

socket, not the TCP/IP socket, so letting you do the above without needing to supply a password.

FrontEnd/BackEnd protocol: the current protocol is 2.0 but Bruce still uses the prior protocol (would be easy to update but the old protocol is still supported and works fine).

- Message flows: startup, query, function call, termination
- Notification responses: asynch

The terms in the spec (AsciiRow, etc.) are also in the Smalltalk interface code to make it easy to cross-reference.

Back in 2000, Pete Hatch persuaded Bruce to implement a Smalltalk interface; it is now shipped with VW (there are also others out there). He started by looking at the spec, which he found better than looking at the C and DLLCC which is how others do it. He had to use change sets for his CM because he was working in 5i, had no Envy and couldn't use Store till he'd got the interface to Postgres working. :-)

Next he wrote an EXDI layer on the driver. Finally, he wrote a Store application.

Using StoreForPostgreSQL to manage driver development is doing brain surgery on yourself, so can go horribly wrong. He therefore makes major and minor version numbers. He also tried to make it work for Lens but noone would pay him and it was too hard to do without that enabler of time.

(The ...PostgreSQLMods parcel in the VisualWorks distribution was done by someone else; Bruce does not know what it does).

The driver is now ported to Squeak and GemStone. (GemStone is good at handling large complex structures, Relational Databases are good at quickly searching simple data structures, so OpenSkills aims to combine both.)

When you first run the unit tests, it asks you for your database, username and password, thereafter all just runs.

To configure the socket, just change the pg_hba.conf. Use ALTER to set password, i.e.

```
alter guest password 'password'
```

if you want to use a password to login.

VisualLauncher > Tools > end menu submenu ‘PostgreSQL Connection’ shows the SQL sent and what comes back.

Host your Website on Gemstone and EC2, Nick Ager
(Go to Nick's blog and read his post for full details.)

Gemstone offer GLASS free with 2Gg of RAM, etc. GBS, which moves

objects between VW or VA and Gemstone is not free. GLASS has no GBS, just Seaside in Gemstone. One stone serves pages to three Gems serving your app. They are separate virtual machines using shared memory. Gems are fired up by Topaz and connects to shared memory (shared page cache) and interact with the stone and other Gems.

In November, Amazon announced a free startup offer. Amazon's offer is 10Gb of disk but only 640 Mb or RAM which is limiting. They give you 2 CPUs on a bust but average is less than that. (Bandwidth is also limited.) And it is limited to 1 year - after that you must pay them some money (might be £50/month but Nick can't quite remember).

Nick uses Amazon for his trial server and backup server - he uses a virtual service provider at £70/month for his production server. It provides Redhat-like linux and it is good.

Go to aws.amazon.com. It will ask for credit card info - scary but its OK. Nick logged in and looked at his existing running instance.

Amazon has very discrete regions that do not share anything. If you create an instance in one region then it is not easy to move it to another region; Amazon has 4 data centres and they are fairly autonomous at the moment. So pick a region that is close to you and launch an instance.

You select an AMI (Amazon Machine Instance). Nick chose a linux 64bit. He selected an empty 64 bit and installed Gemstone and then created an AMI from that. He has one AMI in EU/West (actually based in Ireland) and one in US/East. He tried to start EU/West but had a hiccough so he started US/East.

If you want to be free, you must pick the microinstance option. (So later you could open the same thing with 20 Amazon ecu - enterprise computing units?) You can scale it up, but not automatically.

(DoS attacks - are you firewalled so you just pay for genuine use?)

You get an RSA key to let you ssh into your instance. The default way is amazon give you a file and you specify it on the command line but Nick fins it easier to specify his public key in user data (bottom widget in form). Nick skipped a step (never bother with that - don't know what it does).

Next he created a new security group for the firewall (not essential - just done to show how). He created a group and added HTTP and SSH. This relates to an amazon firewall that exists outside your instance.

Amazon then provisions your new instance (takes 30 secs or so) and then boots it up. The web widget showed green so we copied the public DNS and pasted it into his browser. The front-end webserver comes up before Gemstone, so you see an error page first, then the Seaside welcome screen.

We then duly saw “sorry the site is down for maintenance” which is Nick's

public-visibility phrase for “it's not working”, after which we got the seaside welcome screen, as promised.

Q (Bruce) Backups? That's another blogpost.

Nick's seaside is hardened a bit - no toolbar, can only get to config through localhost (so must PortForward in .ssh/config file), etc.

Next, Nick ssh'ed into his image - and discovered that something (local to the venue AFAHCS) was blocking port 22. (Bruce recommended having a machine that listens on port 80 for ssh requests; he usually has one of those hanging around.) He therefore connected to the image on his local machine (in VMware) to continue the talk.

Nick develops in local Pharo + Gemstone, then tests on VMware running locally (Ubuntu), then moves to his Amazon (which alas he cannot download and run locally at the moment because Amazon is using Redhat in a modified Zen that is not available). Nick could build the same redhat / sentos and same tools and be pretty close. His production server runs on Ubuntu for legacy reasons so he is mirroring that rather than redhat.

To load code into the image, ssh into it using X forwarding and run gemtools. Nick has installed the minimal set of libraries. His default linux instance has no GUI and just enough X libraries to do X forwarding.

Gemtools is a modified Pharo browser on what is in Gemstone (it is *not* replicating anything). Dale has ported Monticello so it is easy to develop in Pharo, save your package and load it into Gemstone. You can also open a local Monticello browser (so you need to remember which windows belong to which). Dale has done a good job of abstracting. Dale has implemented Grease on Gemstone.

He runs a webdav server that is the one both his local and his amazon VM uses.

Issues: Pharo allows you to use UTF8 but Gemstone will not (so no euro symbol). Gemstone is fixing this. Nick is running on Gemstone 2.4 (64 bit - must have a 64bit AMI). Gemstone 3.0 is coming. They hope to run Ruby on it and Dale was talking about importing the whole of the Pharo class library. There are also streams (1-based versus 0-based), and String hierarchies not being the same (multibyte strings are the issue - ByteString, DoubleByteString, ...).

The Gemtools is based on Pharo 1.0. Norbert moved it to 1.1 and Nick does most of his development work in 1.1. (Tim was impressed with the way Pharo 1.2 is now looking.)

Conclusions

My twelfth ESUG:

- In these difficult economic times, it is heartening Smalltalk is doing so well. Some of this is lucky admin (which maybe we deserve when you think of the unlucky admin Smalltalk had in the 90s): Instantiations has divested itself of Java and is using the cash for Smalltalk; GemStone seems to have found a happy home in VMWare. Some of it may be that all the Smalltalk projects that could be killed were, years ago; we are now a mix of unkillable large projects and agile smaller projects.
- While this year's ESUG continued the higher numbers of the last three years, I want to bring more people in from well outside the core groups. We'll see what next year's ESUG can achieve.

Written by Niall Ross (nfr@bigwig.net).

* End of Document *