CS10 and ESUG 13, Brussels, August 13th - 20th, 2005

I arrived at Brussels and confidently set out to walk from the train station to my hotel. Fifteen minutes later, I turned around and started to walk back to the train station. I then set out again, this time walking along a street pointing 15 degrees further north than the one I picked before. :-)

We were treated to a beer-tasting evening (of 13 distinctive local beers), a chocolate-tasting evening (generously supplied; we all ate lots and still had chocolates with every coffee and tea break for the rest of the conference), a walk round Brussels ("Please arrive on time or it will be a run round Brussels and it is too hot for that."), a brewery visit and a banquet.

Style

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

Author's Disclaimer and Acknowledgements

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

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

Likewise, the talk descriptions were typed while I was trying to keep up with and understand what the speakers were saying, so may contain errors of fact or clarity. I apologise for any inaccuracies, and to any participants whose names or affiliations I failed to note down. If anyone spots errors or omissions, email me and corrections may be made. A single programme track (save on Friday) meant I caught most talks; apologies to those missed.

My thanks to:

- Katerina Barone-Adesi and Felix Madrid for working with me in the Custom Refactoring Browser project, Reinout Heeck for working with me in the SUnit project, the organisers and/or university for providing straightforward network connection, and the student volunteers for ensuring that everything was present and working when required.
- the speakers whose work gave me something to report, the conference sponsors (Cincom, Georg Heeg, Instantiations, MetaProg), and the ESUG organisers (Stephane Ducasse, Noury Bourakadi, Roel Wuyts)
- above all, Roel (again), Maja D'Hondt, the team of local organisers and students, and BSUG and PROG, for a conference that ran smoothly.

Summary of Projects and Talks

I begin with ESUG Administration and some Camp Smalltalk 10 project summaries, followed by the talks, sorted into various categories:

- Web Frameworks
- Applications
- Analysing Smalltalk
- Developing Smalltalk
- Environments and Tools
- Design Discussions

and then the Academic Track:

- Awards Candidates
- Research Talks

I close with Other Discussions and my Conclusions. Talk slides are reachable from http://www.esug.org.

ESUG Administration

ESUG Programme, Roel Wuyts

Roel explained we would use the same format as in Bled; if you wanted to here more from a speaker, buttonhole them in the CS room during the breaks and arrange a demo, etc. As at Kothen, there was:

- a business day
- a parallel-track education day
- an academic research half-day
- technical innovation awards prizes

Roel welcomes feedback.

ESUG Activities, Stephane Ducasse

Stephane thanked the audience for coming: attending this conference is an easy and, he trusts, rewarding way of sponsoring both ESUG in particular and Smalltalk in general. The student volunteers help run ESUG but also get young people aware of Smalltalk. (ESUG pays all except their travel expenses.) For example, Philipe was a student volunteer two years ago; now he is a commercial Smalltalk programmer.

We need active local groups. Just do it. Stephane listed the Italian, Swedish and Netherlands mailing lists. Sweden started with 3 and now have 30. The French SUG started with 3 and now they have 135 people presenting Squeak in French, writing books (and translating them), etc. (One such book is ‘Squeak: Learn Programming with Robots’; it supports teaching programming to children and is by a certain Stephane Ducasse. :-)

ESUG offers a free week of teaching and free books to any suitable University; just apply. Stephane gave the list of recipients in 2002-4. After a break of a year, this programme is now restarting. Tell any university you

have contacts with who might be interested.

ESUG will consider helping you present at Open-Source conferences (e.g. RMLL2005, LinuxDays, etc.); contact them and they will evaluate what you offer and may pay, help or whatever. (They sponsored Marcus when he presented.) They will also pay if you get Smalltalk articles into magazines (100 euros).

They are going to set up a Seaside server so people can put up a Seaside system fast. They plan to advertise the conference in magazines. They will advertise Smalltalk books. They plan to make the technology award more exciting - a gadget of some kind maybe. They may sponsor a summer of code: a student project to build something useful.

10 reasons I hate Smalltalk, Theo D'Hondt, dean of the university
This witty speech (very briefly summarised here) enlivened the banquet of local delicacies. Theo expressed amazement at the idea of using a language whose syntax has not changed in decades and that uses a period (a full stop) to end its sentences. He also dislikes real-time debugging; computer programmers should not imitate those lazy chemists, engineers and others who use computers to help them do their jobs instead of demonstrating the power of the human brain (and the value of job protection).

Smalltalk can't be a real OO language if it has no nested classes. Instead Smalltalkers talk about things called blocks - clearly some shoddy means of covering up the fact that they failed to provide sufficient control-flow structures. Also everyone knows that modern programming languages strive to outdo each other in the number of visibility quantifiers they propose; Smalltalk's everything-global-or-local approach is just not trying. Above all, where's the static type-checker? This is especially important today - how can you have internet security without a static type-system?

He made several other points but closed on his clincher, "Smalltalkers visibly have too much fun to be using a real programming language."

ESUG Business Meeting
The new board was elected; see the website for the list.

ESUG Conference Administration
Roel and Stephane talked through ESUG organising tasks with groups of possible future local organisers of ESUG conferences in Prague, Spain and Lugano. (The Spanish group is mostly expatriate or from the new world and would be glad of contacts from Smalltalkers located in Spain. Info of local Smalltalkers in the Czech republic or Lugano is also welcome. Interested groups in other locations should contact the ESUG board.)

This conference was unusually early because Brussels hotel prices triple when the eurocrats return from their holidays. Generally, ESUG aims to be end-August/start-September, some months after Smalltalk Solutions (and Smalltalk Solutions is usually earlier than it was this year).

You need several local organisers (four is a good idea) because there are things to be done at various times during the year and any one person may be on holiday or busy at that point.

The mutual ergonomics of the lecture room and the Camp Smalltalk room are very important. The CS room must have sufficient tables (suited for pairing, so e.g. not small round ones) and plugs. Some network connections and cables (capable of serving almost any position in the room) are also essential. The CS room for the weekend before the conference need not be the same as the one during the conference but of course it must be equally accessible from the accommodation. The CS room during the conference must be very near the lecture room and where coffee breaks are held. The ideal is that a large room has both the Camp Smalltalk area and the coffee area, and the lecture room is next door.

The weekend Camp Smalltalk is self-organising. You must provide the room and must sign it well by Saturday morning. Having something to drink in the room is essential: a coffee urn, plenty of water and plenty of paper cups. Some nibbles are also a good idea. Advice on nearby places for lunch is also useful (and check which are open on Sunday).

There may be a single day of parallel business and education track talks, needing two lecture rooms. Otherwise, there is only a single track so only a single room is needed, capable of addressing in excess of a hundred people (ESUG has never exceeded 150 in recent years).

Cheap accommodation is essential for the student volunteers and desirable for the rest. It needs to be close to the venue or travel well worked out. People usually arrange their own, except for the student volunteers, but need to be advised on where to stay and probable costs well in advance, Having a single main location for accommodation promotes interaction.

The student volunteers must be managed by the local organisers. They must man the conference and CS rooms during breaks (2 x half of break time). They must provide coffee and tea during the breaks, clean away, etc. They must get the slides from each presenter promptly after each talk and put them on the ESUG server.

The social event(s) must support interaction, and preferably also fairly free choice of whether to leave early or stay late. A walk, visit to a site or boat trip is good, allowing people to move around, and to talk or be silent. A coach trip is bad. One or two small evening social events can be added. Again, these should promote moving around and talking, so buffet meals, tastings (wine, chocolate, etc.) and suchlike are good.

Sponsors are of two kinds, local and ESUG. The ESUG board is responsible for ESUG sponsors. Local sponsors need to be found by the local organisers.

Things to be done very early include choosing the logo and making the budget projection. A safe budget is one that will break even with an

acceptable registration fee of 1/40 of the budget projection.

The web site travel information should be done a good few months in advance so people can start thinking about attending.

Camp Smalltalk 10: Project Summaries

Camp Smalltalk 10 ran for Saturday, Sunday and Monday (a holiday in Belgium) before the conference, and during the conference breaks, afternoons and some evenings of the five conference days. There was much activity in the room, only some of which I caught.

The Custom Refactorings and Rewrite Editor Usability Project

I spent more than half of my CS time on this.

Katerina Barone-Adesi and I paired on creating new refactorings. Our first was a refactoring to split a cascade at the statement indicated by the selection point. If the cascade's receiver was not `isLiteral` or `isVariable` then an ExtractToTemporaryRefactoring had to be added. We knew to pass the overall refactoring's model to it but it took a failing test and a debug to realise that because (unusually) the two refactorings affected the same method, we also had to pass the parseTree between them.

- Every refactoring has a variant of its standard creation method that starts `model:...` and must be used whenever the refactoring is being used as part of a composite, to initialise every refactoring with the composite model. (This is so that all changes are assimilated in the model which is then compiled atomically.)
- When multiple refactorings of the same method are applied in this way, the successively-rewritten parseTree of the method must also be passed between the refactorings.

Passing the parse trees was easy in code, because the parseTree is lazily initialized later than refactoring creation, but was less immediately obvious to us because there is no protocol for it and because the instVar has not been pulled up to MethodRefactoring. The various subclasses had their own and some called it sourceTree. With this mastered, we got it working with the usual hiccoughs and test rewritings.

I then made an RBCommand and menu item for it. Meanwhile, Katerina refactored the hierarchy to pull up parseTree and its lazy initializer to MethodRefactoring. The way John Brant envisioned this kind of operation being done was first to rename sourceTree in those subclasses that used it, then pull up the now common instVar, then pull up the relevant method(s). Katerina, naturally enough, started by pulling up parseTree and then sought to assimilate sourceTree - whose renaming the framework forbade since parseTree was now in scope in the superclass.

John's is the theoretically cleaner approach but what Katerina and I wanted to do made sense and was a situation that often arises when people reparent hierarchies or fail to anticipate this situation and do work they do not want to reverse. We therefore started work on our second refactoring of the camp: PullUpToExistingVariableRewrite.

(There are a range of things the framework can do that can be known by the user to be refactorings in a given context but are not provably so; we call these 'rewrites' as opposed to refactorings and name their classes *Rewrite instead of *Refactoring. Our intent is not to expose them at top-level in the RB but to make them available when a RefactoringError means the refactoring is not provably so but the user may wish to proceed with a rewrite. It is a choice, and a philosophical debate, whether and when to allow these. The project is interested in its users' views; post in c.l.s and/or the enhancements mailing list on customrefactor.sourceforge.net)

We created it, then 'Create-Sibling-ed'an abstract superclass for it and RenameInstanceVariableRefactoring. We rewrote the objecting scope condition in the latter refactoring to raise a refactoring error with a parameter block (i.e. something the user could choose to do instead) that warned them that continuing was not known to be a refactoring. A PullUpToExistingVariableRefactoring was performed by running the block. Thus to the user it would seem that they had been allowed to proceed after a strong warning though in fact the refactoring had been aborted and a rewrite substituted.

We gave our new refactoring its own precondition warning if any methods in the assimilated hierarchy already referenced the superclass' instVar. If that were so then the risk of its not being a refactoring would be greatly increased. When it is not so, the operation is a condensed form of the sequence PushDown target instVar, remove it from non-using classes, rename rivals in those classes, pull up. These make a provable refactoring when the superclass' methods do not use the instVar (or when some do but no method of any class in the rival sub-hierarchies could call them.)

We coded it all, passed model-layer tests and then tried it in the browser; nothing happened! We stared at it, walked through a debug and satisfied ourselves it was running our parameter block. Then we swiped the refactoring error's block code, passed it the test values as literals and evaluated the block in a workspace; it worked. We reverted and ran it from the browser again; no effect. We breakpointed the actual compilation in the workspace-case, worked up to a method we'd seen called in the browser case, stepped forward again, and saw that our refactoring had nil model in the browser case so the RefactoringManager instance did nothing with it. We suddenly realised that we were passing the first refactoring's RBClass to its successor, not the original Smalltalk class. The second one, already having an RBClass, never lazily-initialised its model to get one. Changing the `class` parameter to `class realClass` fixed things. When you compose refactorings this never matters since you pass the model. But we were not composing refactorings, we were substituting one for another.

(After the Camp, I realized that we actually needed to pull up the parseTree instvar to a new subclass, ChangeMethodSourceRefactoring, sibling to ChangeMethodNameRefactoring, and not to MethodRefactoring. This is in our latest releases.)

Meanwhile, Felix Madrid and I paired to produce a widget that would let

you select multiple methods and see them all in a single text pane, cutting and pasting between them but also able to use all the standard refactorings and features of the single-method code tool. We first reused the rewrite tool pattern (RewriteMultiMetaCodeTool and RewriteSingleMetaCodeTool) creating BrowserMultiCodeTool and BrowserSingleCodeTool subclasses of the BrowserCodeTool. This pattern overrides the basic parsing calls as

```
parse...
  self alternativesDo: [super parse...]
```

so that the offsets (that map selections to their relative position in a method) are captured before parsing that method and errors are similarly unoffsetted again before being inserted at the right point in the overall text.

With this working, I then looked at the alternative solution of embedding multiple BrowserCodeTool instances within an overall MultiCodeTool that simply displayed a variable number of other tools. The latter is probably better in most cases but requires more dialect-specific code. The former is easier to write in dialect-neutral code and may be easier to use when the aim is to make one method into two or vice versa, to do a find-and-replace in several methods at once, etc.

We also looked at 'extract with holes': the ability to select text for 'extract to self' or 'extract to component' and then select within it those parts that were not to be extracted. In the model layer, this is ExtractToMethod composed with a number of ExtractToParameters on the extracted method. The cascade work on multiple refactorings of the same method had shown the issues to watch. In the UI, adding a pane to the extract dialog window showing the text being extracted, and a button to prompt 'unextraction' of selected text to the parameter list, would let the user choose the 'holes'. Work continued after the CS; we hope to release it before the next Camp.

SUnit

In between Smalltalk Solutions and ESUG, Michael Lucas-Smith found problems in my recently-released cross-process test case utility (in SUnitUtilities in the open repository). He was using it in his WithStyle test suite (much WithStyle code has multiple processes) and he sent me tests that opened a window and sent it uiEvent or uiEventNow in a subthread, all of which hung if run under my utility.

I paired with Reinout and briefly with Chris to resolve this. There were two issues.

- Firstly, uiEvent and uiEventNow create DeferrableAction events and these are then added to the event queue and run in a UI thread. If this already exists, it will not be recognised as a spawned subthread of the test process. The solution was to wrap DeferrableAction>>block: as well as Process class>>forBlock:priority: so that any deferred blocks created in test processes got wrapped with the test handler.

- This was discovered while investigating but was not really the problem. Window managers started in formed blocks leave a thread that is not then cleaned out after the window closes. Only when kicked by the next window event does it die. If however the test was launched in the main window thread (the usual case), then the UIProcess window manager waits for it to complete before kicking anyone. The result is deadlock: my main test thread waits for this thread to die and the UI process waits for my main test thread to complete before killing it.

Reinout and I refactored the code to have the key test thread processing done separately from the (now multiple) wrappers and to hold references to test processes instead of renaming them. We also refactored the shared queue of semaphores to be a single ProcrastinatingSemaphore (written in Smalltalk, using `valueUnpreemptively` but it should be easy to clone parts of Semaphore's `signal` and `wait` primitives to implement its `signal` and `unsignal` methods as thread-safe primitives). Finally we looked at giving the procrastination loop some kind of loop to let the window manager (and anything like it) kill threads. (Reinout persuaded me to rename `procrastinate` to `unsignal` in the semaphore; he pointed out that the former may be a word fairly recognisable to non-native English speakers but the latter conveys more quickly the method's intent.)

Other Projects

Avi worked on an OmniBase-on-non-Windows bug with Mike Roberts and others. If you write the optimised b-tree structure to OmniBase 500 times concurrently, it works on windows but Mac has problems. It emerged that it was a file locking issue. OmniBase was developed on Dolphin (windows-specific) and ported to the other dialects, adding some file-access methods to replicate Dolphin helper methods. On Unix, the file lock is requested but no check is made to verify that it was obtained; the code just carries on blithely assuming that it was. This seems to be the cause of the problem. There was also an issue of the computed file size under Unix, whether it might be changed by something else and need recomputing.

Michael Kinberg has built an improved file chooser in Squeak. He will use this with his application which is a children's game to define characters, have them interact and show the results in comic-strip style, e.g. moving an ice cream to a sad face turns it into a happy face.

John McIntosh and David Shaeffer paired to improve Squeak's poor handling of connection requests. David wants to serve 100 connection requests per second and had observed problems. They found two infelicities in the code:

- The code was putting the last bit in the connection and closing it, trusting the OS to flush before closing. In fact, the OS might wait only a short time before closing and preempt the flush. Thus problems as the number of connections grew were to be expected. They fixed this and most of David's tests to expose the problem ran green.
- The access loop ran at too high a priority, so the code was accepting new requests more readily than handling the ones it had. One test showed it taking requests till it hit the system file-handle limit, then

processing 200 and accepting more again, and so on (c.f. the classic 'priorite a droit' problem of traffic coming onto a roundabout having priority over traffic on it, so of course the cycle blocks under load).

Much more was going on, both during the weekend and in breaks in the conference. This year we had a large break room that had tables for Camp Smalltalk at one end and the coffee, tea and cold drinks tables at the other. Thus it was very easy to demo or pair-programme during breaks and in the evening. (The drinks table also provided a kettle, so I drank passable tea for the first time ever at a conference outside the UK! :-)

Web Frameworks

WithStyle: opening a new frontier on the Smalltalk user interface, Rowan Bunning, Software WithStyle

Software WithStyle was founded in 2003 and is located in Canberra. It has been involved in a range of things known to the community: NetResources and NetResources WebDAV, LibTidy and LibXSLT, BottomFeeder widgets. Rowan Bunning fills in the gaps that Michael and Steve, the core developers don't do.

(Rowan also works for Wizard Information Services. They have a web framework product called WebCanvas, very much Seaside-inspired but in VisualAge, so it has not got continuations but it does have callbacks and all the rest, plus a good test framework.)

Rowan has been developing web sites for the last 8 years. Everyone was interested in AJAX in the last talk. It is easy to see why it was popular; it makes your web apps look better and the web browsers haven't changed much. But is this really the place we want to be; getting exited about ways of working round the dullness of the browsers.

WithStyle is an XML UI with CSS rendering 100% in Smalltalk. It aims to have the abilities of the Smalltalk GUI and of a web browser. WithStyle lets end-users customise the user interface. XML has a lot of mainstream mindshare (and with wider use of XForms will have yet more; W3C know that HTML forms are poor so offer a standard for next-generation forms).

Many owners of client-server applications want them to become web apps which means a huge step backwards in usability and flexibility today. It is also hard on the developer. Rich internet application is jargon for making web applications as good for the developer and user as client-server.

WithStyle targets business applications rather than public sites. It can be deployed embedded in your desktop app (c.f. BottomFeeder), as a thin client, or as an intranet or extranet application. That it has the usual Smalltalk openness (all code visible) makes it very flexible to integrate.

At Smalltalk Solutions 2004 they had 600 tests and 35,000 lines of code versus Mozilla's 16 million LoC. Today they have more tests and less code, ~2150 tests and 33,500 LoC (good!). They also have CSS3, scripting support, and several commercial applications. WithStyle uses LibTidy and

their own code to tidy the HTML and XML they are given.

He opened an image and inspected a WithStyle.SpecializationDocument. He then browsed it, showing a Zen Garden document. Yes, WithStyle does have a web browser but no they are not competing with Firefox; their aim is not to build a web browser.

Rowan walked round the slideshow app (which he'd been using since the start of the talk) that Michael coded during StS2004. Then he showed the tool set architecture, network management (download manager), etc.

XML handling: LibTidy and wrappers around it do their best to tidy the (X)HTML it is given. They also use XML events to hook in Smalltalk scripts (v. similar to Java script but XML events are more generic).

Layout handling: their CSS parser is much better, much more robust to small CSS errors on the sites they browse. The Pollock GUI framework is fully integrated underneath this, giving potential for very interesting hybrid layouts. They are adding rendering services for GIFs, JPEGs (mostly there on Windows, not yet all on the Mac he's demoing on here).

Editing tools: these let you backmap so when what you see is massively XSLT-transformed from what was originally served you can edit and get your change backmapped to the original. One of the backmappings supported of course is to Smalltalk, so you can edit any Smalltalk object as XML. They allow structure editing and are enhancing to support data types in schemas, which can be important to enforce in XForms. Text editing is trickier than it at first appears and must work nicely with the underlying structures. Spell checking has been added on Windows.

Development tools: you get an example browser and diagnostics for links and for layouts. The RB now has a WithStyle tab.

He listed application areas (quotes are from customers):

- Embedded Browser: an example is BottomFeeder. "A working beta was ready two days after the migration started" (and had to deal with all the buggy XML/HTML that there is out on the web, especially in hacky buggy free blog apps). Rowan demoed.
- Embedded editor: an example is BottomLine. He demoed editing the post he'd just shown in BottomFeeder, popping up hierarchic context sensitive menus (operations you can do on word selected, on paragraph selected, etc.). On Windows, the hierarchic context being considered is outlined by a dotted line (not on Mac yet). "The new editor not only simplified the posting tool; it simplified the back-end server."

- Web app client: an example is Seaside with rich UI. They can script on the client side in Smalltalk, not in Javascript, do validation on the client-side etc. WithStyle client with Seaside server is a very natural and powerful match. You could also extend the interface on the client dynamically under the server's control. You can use the Pollock form widgets, etc. He demoed using CSS styles to turn tags (name, text input field, text editing field) into Pollock widgets. These widgets are rendered by WithStyle like everything else they handle so flow around as you resize the page just as you expect.
- WYSIWYG XML editing: an example is EzyXML, the editor they have developed. They have delivered this (via Wizard) to 6 Australian government sites and they have heard no bug reports, just praise, since they started delivering in 2004 (in contrast to the expensive large-vendor solution those site used previously). Currently they support XHTML (all 3 variants), Simplified DocBook, Resum XML and Scrum Backlogs. They will add DocBook and HR-XML.
  - — Cross-platform dialogs for business users
  - — Beta version of it is downloadable.

They are talking to several possible customers for WYSIWYG XML-editing. One is an HR department, needing to handle forms, resumes, etc.

He demoed. They've re-labelled XML names to give end-users something they can understand (Division, not , paragraph, not <p>, etc.) which many other XML editors still don't do. Backmapping lets you edit the original, regardless of the wrappered look-and-feel and the XSLT changes. He edited, changing look and feel between operations.

They are working on hardening it so end-users can configure it without seeing interesting exceptions. They integrated WebDAV so you don't edit and have to go somewhere completely different to edit the look-and-feel to match. Various files hold the vocabulary definitions, the schema, the menu filters (can have their organisation differ from the schema default).

Their roadmap:

- new CSS engine
- new styled XML objects to minimise recalculation
- drag and drop
- `before:` and `after:` is CSS' way of saying 'put this here'. This will be easier than using XSLT to do that kind of layout.
- zooming, paging, scrollbars,
- they want to use Pollock embedded widgets for HTML forms. XML translation of Pollock widgets is the starting point but they also want to map the names of elements so they can apply CSS to them in detail.

Rowan opened a WithStyle browser on a Smalltalk string, edited and showed that the string had changed.

Q. (Reinout) Licences? Announcement soon on licence details. There will be a difference depending on kind of app (very simple like just a help window as against very complex) and whether internal or external usage.

Q. (Christian) Speed? WithStyle 3 on the Windows VisualWorks VM has great performance. WithStyle 4 will have yet better performance which will also look fast on the Mac's slower VM. [It looked OK to me.]

SmallWiki and Magritte, Lucas Renglii, netstyle.ch (www.netstyle.ch)
Magritte is the meta-model of SmallWiki2, which is the re-implementation in Seaside of SmallWiki. Why SmallWiki? Well, there are some 200 wiki implementations and they all use s///g, they all hold huge meaningless strings and they all duplicate the parser everywhere. He has SmaCC parse the wiki into meaningful objects.

He compared SmallWiki2 to WikiPedia (written in php). The WikiPedia parser is 117,000 Lines of Code; SmallWiki's is 555 LoC. WikiPedia's unsophisticated query engine is 20,970 LoC; SmallWiki's is 195 and gives a better search engine. (To be fair, he noted that the WikiPedia LoC includes comment lines and 100 lines of the 3-page licence, etc., but this does not change the order-of-magnitude comparison.)

He showed some SmallWiki sites, including the ESUG site. He showed editing, creating a link, getting a form to say where to link to, etc., embed or link (+ or *), edit page, upload file (inline or link), with nested lists embedded as subpages in the main page.

SmallWiki is implemented in Seaside so he can put any Seaside component or application in his wiki. He added the sushi store to the RHS pane of his page. The Wiki editing app is itself a Wiki so can be used and changed in the same way; he added 'hello world' to all the editing pages.

SmallWiki is neither small in what it can do nor just a wiki. It is a content management application and can be used to build applications. There are a range of widgets provided, some at Stephane's request: a detailed search, a calendar, a photo-gallery, a refactoring engine for wiki pages, a plugin to import other wikis, etc. View-model separation has been done rigorously so you can plug in any view to your components. This also allowed better testing. SmallWiki has 170 tests, Magritte has 1000 tests.

SmallWiki 1 won first prize at last year's innovation awards. It was implemented in VW and has been ported to .Net by John Brant, to Squeak (latest version released yesterday) and other dialects. There are several users, some major, some just using the parser. The production version of SmallWiki 2 should be released near the end of this year.

Seaside is now the default view of SmallWiki2. The new storage framework is a changelog-based framework where you do snapshots from time to time. He captures changes so can go back to an old image and replay.

The basic architecture starts with root WikiObject with subclasses

DecoratedObject and Decoration. DecoratedObject's subclass Structure has only Children (subclasses Page and File) as decorations (the other Decoration subclass is Security). This part of the framework is stable. The Kernel and Views are ongoing (those things never end) and he is still working on Security.

Q. (comment) We've just finished the first version of an external storage for SmallWiki and will release the code in 2 weeks (demo version here) because we've been nervous of the image snapshot storage.

Magritte is SmallWiki2's meta-modelling framework. Lucas soon realised that it can be used in other Seaside projects and indeed beyond. Magritte lets you build views, reports and validated editors. An edit command in SmallWiki2 (Command subclass: EditPageCommand) has description a TextDescription, part of the Description hierarchy. Send `asComponent` to an EditPageCommand to build a Seaside component, a simple form, or `asMorph` to get a Squeak view of it. He showed this in Squeak (new work, updating still needs you to click it - and the usual demo hiccough - he had edited his pages from two different locations during the demo so got DNU).

```
EditpageCommand class>>descriptionTitle
  ^(StringDescription selector: #title label: 'Title'
priority: 100)
    beRequired;
    yourself
```

This is Magritte and lets you add a new field to all the above views, the morphic, the Seaside, etc. He can qualify them as #accessor, #label, #comment, #default, #readonly, #required, etc. He has a deep Description hierarchy. Magritte lets you build a range of views and editors from a single description (say once, get everywhere) and then attach your own preferred view to it.

He then demoed. The UI for Magritte descriptions is defined in Magritte. He edited a description using Magritte, adding a comment text field and a colour field to the name description. He opened the new form and could not instantly save because the name field is required. He added a name, set a colour and saved.

He then showed how to extend SmallWiki2's commands. He wants to let people tag edits as 'minor change' i.e. not put in the generated RSS feeds. He added a new field 'Minor change' that is a boolean description with a large priority number, so it appears at the end, and #auto setting so he gets the accessors, etc., automatically. It generates the methods and adds the UI element to the page-editing form. In the Squeak OmniBrowser morphic form he has the same new field. He can now tie it up to the RSS behaviour.

Don't let php, python, .Net and Java win the race. Join, test, help.

Q. (Niall) Relationship to MEWA? It's a new implementation. MEWA did not describe itself and was not storable.

Q. (Niall) Am I right in thinking that your Container class duplicates the Collection class for the usual meta-programming reasons? Yes. (Rest of discussion is in my Design Discussions issue below.)

Q. When in VW? By all means help port it. It's just a question of who works on it; at the moment more Squeakers are in the project so it advances in Squeak. As SmallWiki1 is in VW, the port should be easy. Lucas loaded Magritte in VW and out-of-box 80-90% of the tests ran (not the Morphic stuff of course). From his SmallWiki1 experience he can help whoever ports SmallWiki2. If you want it in VW, contact him and offer to port.

Advanced Seaside I, Lucas Renglii, netstyle.ch (www.netstyle.ch)
Lucas is a student at Berne but also works at netStyle building Seaside apps for customers. He showed a picture of him and Adriaan at ESUG 2002 in Douai, when it all started. Avi having recently done the first Seaside, he wrote an app on version 0.9. By ESUG 2003 in Bled, they'd written a huge insurance app for a client in Seaside and presented on it. At ESUG 2004 in Kothen, Avi was there (and he was at Camp Smalltalk this year but had to go back to Canada yesterday).

Lucas has a tutorial on the web and there are several others. The slides are all on www.lucas-regglii.ch/.seaside/advanced. This tutorial assumes you know the basics. Lucas invited people to ask questions during the talk. His slides give references at the end of each section.

Topics:

- backtracking: The client goes from page to page and then back to a page and then from that to a new page. What should the server do, given that the browser takes the back-tracked page from the cache so the server receives no new request.
- Ajax
- the Presenter hierarchy
- REST URLs
- The Call/Answer continuation framework: he will show what lies behind it, how does it work.
- Customising Session, Error handlers, Renderers
- Professional Apache Serving with Seaside
- He also recommends David Shaeffer's testing framework (subject of the next talk) and his talk on SmallWiki2 which uses Seaside.

He then had the audience vote on which to present now, which on Saturday.

Seaside supports Ajax live updates. Google uses it in gmail, etc., doing all kinds of Javascript tricks to avoid updating the whole page, just the part that needs it. Seaside supports this.

WAPresenter subclasses WADecorator, which decorates WAComponent which has children other WAComponents, some of which are of subclass

WATask.

Seaside URLs are not REST compatible. The session key and continuation key are embedded in the URL e.g. ...seaside/counter/_s=TVSOVzhb... so it cannot be bookmarked. However Seaside provides means to make the URLs REST-compatible., You must write additional code and Lucas will show how it works.

AJAX (Active Java something :-): HTTP (1989) and HTML (1992) are very old but customers want UIs that feel like client apps. One solution is to use Javascript to just update a part of the page in response to a user request. This feels much faster to the user since they do not see a whole page reload. AJAX is used in Google gmail, google maps, a wiki, etc.

If you use AJAX with Seaside, you don't have to know any Javascript (unlike with other frameworks). Seaside is server-based so uses no Javascript. It includes a small Javascript library into the resulting HTML. The Smalltalk renders your controls with all the rest.

```
<a onClick="javascript: ...">My Link...
<div id="unique">foo</div>
```

Javascript sends request to Seaside, via the unique div. Javascript parses the response and puts it in. You put live update in your Seaside app with the obvious helper methods:

```
html
  anchorWithAction: blockEvaluatedIfJavaScriptDisabled
  liveAction: blockIfJavaScriptEnabled
  text: ...
```

For example:

```
self renderTimeOn: html "whole html page"
html
  anchorWithAction: [...]
  liveAction: [:renderer |
    self renderTimeOn: renderer]
  text: ...
```

One common very-hard-to-find mistake is to pass the old rendered to the live update block.

```
self renderTimeOn: html "whole html page"
html
  anchorWithAction: [...]
  liveAction: [:renderer |
    self renderTimeOn: html "WRONG"]
  text: ...
```

He demoed this with and without reload of full page.

```
html
  textWithCallback: [:value |...]
  liveAction: [:value :renderer |
    ...]
...
```

The current framework just supports updating one simple div. It would be

easy to make it support updating multiple divs each with its own unique id.

```
  scrollBarWithHeight: [:value |...]
  size: self numberOfItems
  liveAction: [:value :renderer |
    self renderValue: value on: renderer]
...
```

updates based on where you are/where you select in scroll bar. You can also update automatically:

```
html
  refreshDivNamed: ...
  with: [:value :renderer | ...]
  every: self duration
```

Update time is a constant in Javascript (200 millisecs) but can be changed.

Lucas is no fan of javascript but finds this an excellent way of improving his commercial apps UIs. More reading: ajax.pattern.org, plus Avi showed some patterns on his blog, plus look at exercise 24 in the web tutorial.

Q. example of using the javascript-disabled block? Lucas added a

```
self inform: 'Sorry, javascript not supported'
```

to a test of Avi, switched off javascript in his browser and showed the effect. In real commercial apps you have to store the value you want to update in a variable and pass it to the code that will do a whole-page update. (Or you can choose not to use Seaside and find yourself, like Google, building effectively two web applications.)

REST: the idea of Representational State Transfer is that every URL is a fixed resource that can be used with GET, PUT, POST, etc. independent of the client and the server and at any later time. Seaside embeds the session key and the continuation key in the URL so it is not REST-compatible. Suppose you want it to be. Every Seaside component can update the URL by adding new stuff to it. `updateURL:` lets you add path elements (i.e. subdirectories), parameters and/or fragments:

```
aURL
  addParameter: 'language' value: 'en';
  addToPath: 'subdir/...';
  addFragment: ...
```

N.B. the default behaviour is to reuse the session and continuation keys. Only if they have expired does the REST-behaviour kick in trying to set up the model from the parameters.

You create a subclass of WARenderLoopMain and override `start:` to add whatever you need. (New framework posted yesterday so it may be easier now than in his example.)

Lucas subclassed the counter application. He set up count 3, bookmarked it and revisited the bookmark while the session was still valid (no problem) and after flushing the session (usual demo hiccough, "how do I do that?"; Ian Prince and David helped out) whereupon he got 'Page Expired' instead

of ‘Page Not Found’. He had implemented an override of `updateURL:` on his subclass to handle the bookmarked URL.

Q. If the model is very complex to store? Yes, you have to think about that. Maybe you could have an id. But usually you just want the expired bookmark to take the user to some simple start point in your app.

See exercise 41-42 in his tutorial and Avi’s blog posts (URLs on slides).

Advanced Seaside II, Lucas Renglii, netstyle.ch (www.netstyle.ch)
Presenters: how to nest components and decorate components. WAPresenter subclass WADecoration decorates WAComponent (with children other components), subclass WATask. Whenever you want to build a part of a page you subclass Component. The Task defines flow of pages by overwriting `go`. `next` returns another component.

Components are nested by the composite pattern and displayed using WAHtmlRendered>>render: You also have to initialize your children, e.g.

```
SomeComponent>>initialize
  super initialize.
  myCounter: WACounter new.
```

However you may often do it lazily as you may not know your model at the right time. Seaside must know for every component, what its children are, so `children` must return a collection of all the components that it *might* display, so at a given moment `children` may return more components than are actually being displayed at a time. It needs to know this *before* rendering happens.

If you fail to specify children correctly you will eventually see a walkback ‘components not found’ when processing callbacks. Children render themselves via

```
SomeComponent>>renderContentOn: html
```

and are rendered by others via e.g.

```
html heading; render: myCounter
```

noting that you must *always* use `render:` not `renderContentOn:`, when rendering children in their owners.

Always register components for backtracking (see below) i.e. if you do

```
SomeComponent>>reset
  myCounter := WACounter new.
```

instead of just resetting the counter to zero, you must re-register your new child component.

He showed a children method that lazily initialized four store components rendered one under another. This may not impress you but it wows Java and Struts programmers. He can buy things in the various stores and it all works, with no need to change anything within each store component.

Decorations can change the behaviour or the look of the decorated component; some decorations change both.

- Look: WAMessageDecoration just renders a title above a component. WAFormDecoration renders an HTML form tag around a component with buttons (OK, Cancel) below that send messages to the component. WAWindowDecoration gives you a border and a close button for a component.
- Behaviour: WABasicAuthentication protects your component with a basic password check. Lucas wrote WASessionProtector because sessions can be hijacked in Seaside if you know how. When protected this way, it verifies the request came from the same machine.

(Q. We use net-oop pool, so if I leave my session for 15 minutes I would probably get a different net-oop to get out, preventing hijacking.)

WAValidationDecoration checks the answer arguments and renders error messages. WADelegation does `call:`, WAAnswerHandler does `answer:`, WATransaction does `isolate:` (these used to be hacked into basic Seaside; now they are done more elegantly as decorations).

You can do your own decoration by overriding `renderContentOn:` to emit XHTML around the decorated component. Call `renderOwnerOn:` where you want owners to provide their content. Example:

```
AlignDecoration>>initialize
  super
  self be Centered.
...
PerformanceDecoration>>processCallBackStream: aStream
  “rare example of overriding processCallBackStream:”
  | timing continuation |
  continuation := self session escapeContinuation.
  self session escapeContinuation:
    [:response | self logRequest ... ].
```

Q. Adding decorations - need to handle as children? No, they are not children of the component. Decorate child - need to do anything? No, that is chain of responsibility pattern, not composite pattern.

To Customise sessions, rendering, error-handling, Seaside into Apache, etc., you create a subclass of WASession and register it in the configuration interface. He demoed calling up the config interface on the web; you just go to the session class field and put in your class.

Q. Several components with different sessions? No; you will not need this. PHP programmers love session classes but in Seaside your components are more powerful and do not need them. It is e.g. the current user or the database connection that you want to hold because all components, not any single one, want to use the same connection or be for the same user.

You clean up (e.g. to close your database connection) by overriding `unregistered` (be aware, it will be called when the GC collects it, so not at some defined point).

WARequestHandler has subclasses WAEntryPoint and WAErrorHandler. When starting a new session, requests comes to the dispatcher which chooses the right application which starts the whole render loop of the Seaside session. You can easily customise the error handler e.g. to let your customers see not the stack trace but some more useful dialog to continue the session or to backtrack the state or to send an email to the developer with stack-trace and debug link (useful if your developer is fast enough to remote-connect before the session times out; maybe in future Seaside will have various time outs for various conditions, so will wait longer for the developer to connect :-).

Use class extensions to add new tags or repeating layout constructs to WAHtmlRendered. Alternatively, you can override `renderedClass` in your components (subclasses of WAHtmlRendered). Or you can avoid further growth of WAHtmlRenderer (187 method and growing) by using WARender-Canvas. This returns a brush which you configure by sending messages to it. This new framework looks similar to the old. Old examples:

```
html
  anchorWithAction: [self click]
  do: ‘link’

html
  anchorWithAction: [self click]
  liveAction: ...
  do: ‘link’

html
  anchorWithAction: [self click]
  text: ‘link’
  submitFormNamed:...
```

versus examples of the new approach:

```
html anchor
  callback: [self click];
  do: ‘link’

html anchor
  callback: [self click];
  liveCallback: [:r | self live: r];
  do: ‘link’

html anchor
  callback: [self click];
  submitFormNamed: ‘formid’;
  do: ‘link’
```

Q. Why call it brush; it looks like an anchor? It is an instance of the class WAAnchorBrush. The name brush came from someone else (from another web framework’s term, Lucas thought).

The result is you don’t have more and more methods, you just combine options. You get your brush and send it whatever combinations of messages you want.

Apache: you may want virtual hosting, security and SSL (which you can

do in VW but Squeak has no library for it yet), to serve huge static files (gifs, etc.). You my also want to hide huge long access paths and you may want to map Seaside applications into some existing directory tree. These are all reasons for using Apache.

Apache handles incoming (including SSL) and the file system, and it then talks to the Smalltalk, proxying its server to the outside world. He showed an example configuration file:

```
<VirtualHost myapp.server.com>
  ServerAdmin myEmail@myLocation.somewhere.ch
  DocumentBook /var...

  RewriteEngine On
  RewriteCond %(REQUEST_URL
  ...
```

Q. You use the rewrite rule rather than the proxy path rule? Yes; he finds it works better. Some things are not possible in proxy-path.

Call and Answer: noone would admit to not knowing what a continuation was :-). The issue is that one page means one script to run (e.g. in PHP request = start PHP, do processing, generate page, return, quit program; this happens over and over again). In Seaside `call:` transfers control to a component and `answer:` returns control to original so A can call B which can then answer back to A. In your code this looks just like a program.

Why do other frameworks not give this simple paradigm? A continuation is an Escaper and a Context. In Smalltalk, continuations use no primitives, it's just normal Smalltalk.

```
Continuation class>>currentDo: aBlock
  | result continuation |
  result := Continuation currentDo:
    [:cc |
    continuation := cc.
    false].
  result ifFalse:
    [continuation value: true]
  self assert: result.
```

So who can explain this code? (Hey, you all said you all knew what a continuation was. :-)

Continuations are used for several things in Scheme and elsewhere, and are good for web control flow. He showed the scheme 8-line form:

```
send-suspend: page-builder
  return continuation: [:cc |
  co-url := create unique url.
  register
  ...
```

Continuation takes a block, captures the state of the application, evaluates the block immediately and returns the continuation. Later, if you evaluate the continuation, you get back to where you were, even if the process has

terminated in the meantime.

```
WAComponent>>call: aComponent
  ^Continuation currentDo: [...
    ... show: ... onAnswer: ...

show: aComponent onAnswer: aContinuation
  “just adds as a decoration and on answer removes”
delegation :=
```

Code from top to bottom is not answered from top to bottom. At first it just adds the delegation showing the new component which does what it likes till it receives an answer whereupon it evaluates the block and removes the decoration inside the continuation which returns the answer.

The bottom line: you don’t have to care about this, just use it.

Q. Need to know about it when debugging? Yes, and also if you step into a call the debugger locks; you must go to the browser and do something.

Seaside Testing Framework, David Shaeffer, Department of Mathematics and Computer Science, Westminster College
[See my Smalltalk Solutions 2005 report for an earlier write-up of this talk. The text below is brief on things already covered in that version.]

They use Smalltalk a lot in Westminster and there is a course on Seaside. David needed a testing framework for the students to use. David is also a consultant and needed it for his work.

The tests run on the same server as the component being rendered so you get detailed access to its state. In standard web pages it’s easier to ask about the contents of the web page so people often do but it’s not always the most useful thing to test.

You also get the Smalltalk debugger. He has a web test runner. Thanks to Michel Bany, this is all in VW as well as in Squeak.

He started with a simple single component example.

```
self newApplicationWithRootClass: SCTestComponent1.
```

Every seaside testcase must start by creating an entry point

```
self establishSession.
```

Start the session and keep track of the most recent response.

```
self assert: self lastResponse ...
```

Now we can test what Seaside rendered.

He tags everything in components he wants to test.

```
self cssId: ‘first’
self anchorWithAction: ...
```

Now we can test it.

```
sef followAnchor: (self lastResponse ...
self assert: self lastResponse ...
```

The response is parsed to an XML.DOM tree and this is then turned into an SCSeasideResponse. You often search for things like anchors. He has naming conventions anchorWithId: (uses CSS id), anchorWithLabel:, ...

He showed his webtest runner. He adds a new halo icon to components which runs the unit test on a fresh instance of that component's class (*not* on the instance on which you clicked the halo). The test runner keeps the history so you can see the last thing you got back from the server. You can then work back in the history using the left arrow and the history is locked but you can interact with it, e.g. click the OK button again, while the session is still valid (rerun test, lose this and the old history).

He showed a more complex component and its test, which invoked a component with a form, got the form, submitted it and checked it responded correctly.

```
form := self lastResponse forms first.
...
self submitForm: ...
self lastResponse ...
```

David finds that testing the state of the component is less brittle than testing exactly what has been displayed. He finds you often want to test that what was answered was what you expected. And he wants hooks to callback to components.

State: `component` answers the component that satisfied the last request (because Seaside has backtracking). He showed that he added an accessor for counter to get its state in his tests.

```
self followAnchor:
  (self lastResponse anchorWithLabel: '++').
self followAnchor:
  (self lastResponse anchorWithLabel: '++').
self assert: self component count = 2.
```

Detecting Answers: he showed a component that was a YesOrNoDialog that answers a boolean. He set up a test to confirm this:

```
  ...
  self
    submitForm: form
    pressingButton: (form buttonWithValue: 'Yes').
  self assert: self answer.
```

It would error if it didn't answer, and in this case happens to answer true.

Callbacks: he demoed the WAMiniCalendar. To actually use this you would create an instance and give it `select:` and `canSelect:` blocks. But his test framework asks Seaside to create this component: he's not

doing it. So he provides an initialisation block:

```
self
  newApplicationWithRootClass: WAMiniCalendar
  initializeWith:
    [:calendar |
    calendar
      canSelectBlock: [:date | true];
      selectBlock: [:date | selected = true]].
...
```

You can also get the session. There are configuration hooks. You have the live history. How do you test the appearance? Well, two different browsers will have two different appearances. Using `markForInspection`, he can mark tests to keep a record of what the rendered page looks like (i.e. the actual HTML) in a given browser and he (actually, his wife; she has a better eye for this) single steps through to see there are no differences.

He has a Seaside tutorial at http://www.cdshaeffer.com\david\Seaside.

What he would like to fix: visual appearance, as noted, and live updates. When he spoke at Smalltalk Solutions, he had no answer but now he has at least a demo of a Selenium driver he has written for Squeak. Selenium runs in the browser and can invoke Javascript so he now needs to merge these; at the moment, his Selenium tests look very different and are usually distinct test classes because Selenium provides very few accesses to things to test - “Is this on this page”, is about its level.

SmallHttpUnit is a VW thing that runs outside the server and has an excellent API (that he will steal sometime).

HttpUnit is a outside-server framework written in Java. (There is also Struts, etc., for such as want them.)

He then demoed some simple tests in Selenium (from ThoughtWorks, selenium.thoughtworks.com). It uses an awkward interaction mechanism and he sometimes has to have tests wait on semaphores, etc., see his example code. He hopes to use this to test live update.

Q. Can you use this to create scripts and watch the test run. He has the XML doc for the page in Squeak but not a web browser at present.

Q. Maybe do string compare and use WithStyle to display in browser? Usually it is browser-specific (e.g. OK in Mozilla but not in IE) plus he’s unsure he could tell when to save (maybe do string compare until value).

He has overwritten the Seaside examples he tests to be much more tagged; his overwrites of Seaside code make no other changes.

Applications

Rule-driven Workflow Enactment, Adriaan van Os of Soops and Tim Verwaart of LEI

See my report of the talk in Kothen last year for the background to this

system. Last year, they presented the Artis system. This year they want to present how they compare the actual database data to the business workflow rules to see what they must do.

They have 60 employees in regional offices collecting data on farming in the Netherlands and putting it into a central database. Some 10 to 20 researchers access the database and extract data profiles for themselves and for others. These customers generate questions which defines the data need.

The data need is first represented as data model (standard UML-ish model diagramming). Adriaan took over here (at every demo stage, Adriaan and Tim swapped roles, customer to implementor) and created a model ('Farming at ESUG') which modelled assets, who held them and who paid for them. (This is a typical, albeit rather simple, model; others might look at usage of chemicals in farming, crop implants and yields, etc. and most would be considerably larger.)

The model alone is not sufficient to capture the data need. They may not care about assets worth less than £10. They may only care about hardware assets. They may only care about purchases at certain times and they certainly care about the timeliness of the data. Thus they have relevance, integrity and actuality rules for each attribute.

Adriaan created the rules showing the UI and then inspecting the resulting structure. The rules held blocks and he extracted the source. The aspect that a rule applies to becomes an argument to the block. They use a special operator that can handle nils on both sides, since the data may not be there.

By attaching rules to the data structure model in this way, they define the data need very precisely and so can discover any violations, thence the person responsible, and so discover what they should do to remedy the data. A rule violation gets added to the workload tasks for the responsible employee (can be a shock to that employee but then they get on with fixing the issue). If the value is not current, or it is current but does not satisfy the integrity constraint, they must pursue the correct value.

Adriaan added constraints to the model. He added prices of tractors and noted they would change from year to year. You can export the resulting table, which asks for a value for each year, to excel, html, or whatever and work on it there.

Tim explained that this is then entered into the workload of the employee. The database also has a set of procedures and notes their applicability to the various rules. Thus a violation can be mapped to an applicable procedure and the system then simply enters the first task of the procedure into the employee's to-do list. Adriaan demoed building a prodedure called visit farm, within which some account analysis is done. The rule says that if the farm has assets satisfying the rule then the book values should be recorded. He added an example asset to the example farm and saw some rules violations; the tractor has an undepreciated value, etc. Adriaan fixed

some violations.

They find it straightforward to create workflow by running business rules over the database. They have used this for 5 years and it has let them reduce their staff count; fewer people (60, and 4 defining the models and rules in the central office) can do more work. The temporal data model is essential. You also have to model responsibilities of people and applicability of procedures.

Q. Can you track outstanding actions for given people and offices? Yes, there are statistics of the programme of work of employees.

Q. Your query language; is it a standard? No, it is completely invented by us. We train new users in it. It's just like learning a simple programming language.

Q. How data-driven is the model? Wholly; there is no need to change the Smalltalk to change the model. It is implemented in our own meta-language (implemented in Smalltalk) so only deep semantic changes need Smalltalk changes. The model is wholly temporal but omitting the time period would it a non-temporal model.

Q. Rule to model mapping? You can use different rules for the same thing and vice versa. Rule assignments are temporal too; this rule for this year, next year use that rule.

Q. Reasoning and forecasting workflow? No, it is purely reactive to rule violations. (You could of course use past data for forecasting. They did not want to reimplement the various tools that do that; you can extract the data to those tools if you want.)

Robots, Serge Stinckwitch

There is an annual robot contest in France. Contestants want to run on Linux, not just Windows and to have subtle control. MEC (Module Electronique de Command) is a very easy-to-use module to communicate between the computer and the module. Each controls one group of sensor so several parallel modules control the overall robot. SuperMEC is an extension of it. I2C is the protocol it uses.

Squeak utilities have been written to support this module: drivers for multiple parallel ports, and I2C protocol support.

Serge then showed pictures and videos of using this with children. The children put together various robots: explorers, caterpillars, walkers, etc. Sometimes the kids come up with ideas that are useful to the researchers. Etoys controls the robots remotely. It is always connected to the parallel port (these are not autonomous robots). A video showed a boy building and operating a walking robot. The we saw a girl build a caterpillar robot and use Etoys to experiment with how to make it move correctly.

SqueakLive (based on Linux Gentoo LiveCD) boots directly into Squeak.

It loads the tools and all the teaching material from the CD. This will be available soon.

They are also letting the children interact with real robots. Koala is made by a Swiss company. Unfortunately, Koala's s/w support is very bad (any language you like as long as it is C compiled by GNU and can fit in small memory). They tried to use Spoon, a small Squeak-derived VM, but have difficulties and are now looking for another platform. Meanwhile they are simulating Koala, scripting with Etoys and using the ODECO Squeak plugin for the physical engine. He demoed programming the robot with various behaviours and running the simulation.

Info is at www.iutc3.unicaen.fr/serge/SqueakBot (page is currently poor, will be rewritten). The code is in SqueakSource.

Q. This behaviour is in Smalltalk, in Etoys, where? Serge opened a robot's script and showed the script items which just presented Smalltalk in a Squeak UI. (See the Kothen talk: drag-drop script items, etc.)

Smalltalk and Hardware, Bernard Pottier, LESTER Architectures and Systems - University of Bretagne Occidental, Brest
Unusually for ESUG, Bernard studies hardware, not software. Stephane introduced him as the Asterix-like inhabitant of a small(talk) village resisting the surrounding power of C#, Java et al. in his domain. The LESTER lab researches system integration, power consumption, device reconfiguration, etc. Bernard's subunit (AS) has 5 full-time and five sabbatical people.

Bernard started to work on Smalltalk a few hours after the release of the spec (he built a VM for it). He found the microprocessor performance poor at running it. In 90-95, he worked on reconfigurable hardware and parallel hardware; it is very hard to program these. Since 96, he has concentrated on reconfigurable circuits and systems (the Madeo workbench). You can spend a lot of time learning the industry tools, build something on them, and find the industry has moved on and your tools are obsolete. So they switched to using VW to build their own system and since then have found industry coming to them instead of leaving them behind.

Rapid development is demanded by industry but conflicts with current hardware development methods. The industry tools are inaccurate. The frontier between software and hardware is becoming less clear. Multi-processing is essential.

They run their workbench on PCs. They find Smalltalk powerful. They can build compilers, models and interactive tools, quickly. The ease of extending the language is key to this and the fact that it is not bound to a fixed type system is key to that ease of extension.

They would like to evolve Smalltalk to use emerging technologies:

- (they already have in their lab) numeric processing with new hardware primitives (arbitrary floating points, fixed points, Galois fields, etc.)

- automatic synthesis of circuits from code
- implicit parallelism

The electronic industry expects continuing exponential increase in transistors and memory (1982: 64k RAM, 100k logic gates; 2005: 1Gig RAM, 1.7Gig logic gates). It also expects nanotechnology to provide more flexible integration components. From this, they deduce a future of massive parallelism handled by synthesis in software rather than hardware developments. Tools will map your software computation to the parallel arrays.

This strategy raises various problems.

- need more reliable characterisation of hardware behaviour
- must handle exploding state space
- you must integrate 3rd party components, so your design tools must interoperate with their design specs
- time-to-market may be short, causing high levels of risk for projects

LSI (low scale integration) is long dead and VLSI is on the way out (does not scale for microprocessors). Processor styles:

- A time-shared multi-tasking processor runs straightforward time-slicing on a single processor.
- Simultaneous multi-threading processors lose part of the processor power but do real multi-processing.
- Multi-cored processors can share memory while running two threads on two different cores.
- System-on-Chips (SoCs) are made for mass markets (otherwise they are not economic) such as PDAs, phones, set-top-boxes, cameras, etc.

FPGAs are more generalisable systems sold to do a range of tasks.There are three kinds, oriented to logic, to single processes (dsp) and to systems. He showed the characteristics of each type. The idea is to have a set of operators run with (random) logic integration of multi-processors using high-speed communications; you can buy these today.

Moore’s law will carry on as before but we are at a turning point in how we think about parallelism. He presented two points of view being debated in the academic community today:

- One (David Patterson’s ‘Call to Arms’) holds that obsession with performance and ridiculous reliability figures should decline in favour of processor design and architecture effort, and that current methods will not scale from where we have now reached. Patterson also wants more attention to human-oriented factors of interest: security, privacy and specific applications.

- Alan Kay compares today's software industry to pyramid building: the brute force construction of layers without vertical coherence. (This is very true of hardware development.) Personal computing has caused a retrogression of systems (c.f. the B5000 and pseudo-codes of Wirth, rejected by industry developments.) Kay stresses the importance of late binding.

Bernard then presented his own point of view. If you can speed up your system by using cheap resources from a range of sources then it makes sense to model the whole translation and execution chain transparently, i.e. using the same tools, because a high layer may need to get feedback from a lower layer (e.g. one in or near the hardware).

The Madeo project was started in 1995 to let the lab survive the rate of change of industry tools. They have explored modelling technology, logic and system synthesis, portable physical tools, structural design, etc.

A reconfigurable architecture has a lot in common with the way nodes are organised in a program tree. Basic units such as switches, buses, logic cells and memories are organised in replicable hierarchical patterns. He showed a UI of logic cells connected to form a directed acyclic logic graph. Equivalent graphs can then be placed in the architecture. From such an architecture, you can do physical synthesis by mapping the graph to the resources of an actual circuit. Algorithms find circuit components able to perform primitive nodes in the graph and allocate them and their connections accordingly. Floor-planning comes next: real systems are hierarchies of graphs and you must minimise the use of resources across the graphs. Their physical tools for FPGAs are mature and have been applied in several experiments and commercial developments.

Logic synthesis: logic blocks are produced from Smalltalk methods. A compilation context holds the data that will be operated by the resulting circuit. The aim is to reduce the graph of complex nodes to simpler graphs of synthesisable nodes. Smalltalk has been powerful in enabling these tools. Currently they only do logic synthesis, not numeric synthesis; they are working on this.

You can cut and paste designs; they reused framework code from the rather similar area of interface building.

Their Morpheus project is extending the current workbench to model systems of different grain addressing the same resources.

VMs: Dorado used microcode, which matches current hardware styles well. Six stages of computation were cascaded: the instruction-fetch unit gets the microcode address to the micromachine up to 6 bytes in advance so the CPU runs at nearly 100%, much more efficiently than an interpreter.

Lastly, he showed a typical modern architecture: four 256 x 8bit RAMs connected to two 16 bit multipliers, etc. Using Microcode on the data path can make this a reconfigurable unit. He presented a suggested outline Dorado architecture configuration on this architecture.

Their task is to build tools now to model these upcoming architectures that have concurrency at circuit level. They cannot wait for the C compiler that will produce parallelism and portability (it will not be easy to map C onto these new architectures).

They would like Smalltalk to have the ability to describe spatial parallelism and processes, time constraints and type systems.

Q. Before building Madeo in Smalltalk, you could not keep up with the rate of technology advance; does that mean academia was (is?) not influencing industry? Most of their projects are done with industry people working on basic technologies.

Analysing Smalltalk

Typing the Image, Roel Wuyts

The previous talk ran over, so Roel had to squeeze his presentation. It is therefore lucky that it only takes him one-and-a-half minutes to type the whole Smalltalk image otherwise we would all have missed lunch. Conveniently, Roel's name is pronounced 'Rule' so RoelTyper makes a good name for his tool.

Have you ever wondered what ProgramNode's instVar mapEntry is about? Probably not but if you are doing something with the compiler you will. Roel just clicks on the typer and 3 milliseconds later he knows. He only finds types for 80% of cases, so you know more than you did before and are no worse off the remaining 20% of the time. (VW has 54,000 instVars of which he finds types for about 80%.) His aim was to provide correct useful information fast to the coder, not to be theoretically complete. (Compare the RB's class comment tool which finds types if a class has no comment. Because this can lock up for several seconds if called on a class whose instVars need difficult type computations, coders rarely check it.) He demoed by implementing

```
Point>>double
  x double + y double
```

The tool instantly told him that Point's vars x and y are SmallIntegers but SmallInteger does not understand `double`.

Roel talked through the heuristics that find types. He starts with those that the RB's typer uses but he has ported them directly to Smalltalk bytecode so they run faster. A subclass of InstructionClient is used by his bytecode interpreter to do symbolic evaluation of bytecode to see which messages are being sent to instvars, in assignments, and so on. For each selector, his algorithm then asks Object whether it implements it. If not, it finds all roots that do. Repeating with the next selector works down to a set of roots that can understand all the selectors in a given message set. (For example, in Point, RoelTyper quickly finds that ArithmeticValue is the only possible root for instVar x.) Root-finding typically takes 27 milliseconds, so the whole takes an average of 30 milliseconds per instVar.

In assignments, he looks at literals and literal arrays, direct assignments of classes and 'instance creation' protocol (he assumes the receiver is the

possible class). This assumption about methods in the ‘instance creation’ protocol is one of his heuristics; he has experimented with subtler rules.

RoelTyper runs in VW and Squeak, with the same SUnit tests for both. The InstructionClient differs between the two dialects since it must know how Squeak and VW handle blocks, etc. Likewise the TypeCollector must understand Squeak versus VW differences in SmallInteger handling, etc. Given the correct instances of these, all else is dialect-neutral. VW and Squeak have very similar bytecode sets. His slides list the few differences.

Squeak only has the core part at the moment. In VW, he has begun adding tools. He has a SourcesAndTypes tab in the RB, an editor to let coders edit the captured types list (and the tool will warn later if your choices become statically no longer possibly correct), an explaining tool (show why the tool thinks this type is possible in terms of messages sent) and a logging tool.

He would be very glad of help, noting that he insists all collaborators keep the VW and Squeak versions compatible, so if you are working in Squeak, for example, you must either touch only the Squeak-specific classes or else always save to both Monticello and Store.

Q(Niall) I have seen methods in protocol ‘instance creation’ of the form

```
newWith: parameter ...
  guardClause ifTrue: [^super newWith: parameter ...].
  ^self newMoreDetailed: parameter ...
```

and

```
newWith: parameter ...
  ^(guardClause
    ifTrue: [OneSubClass]
    ifFalse: [OtherSubClass])
      newWith: parameter ....
```

(there are examples in the RB). Your heuristics would probably give a sensible answer for this as the alternatives are super or subclasses. However there are also examples where the new class does not share hierarchy with the old (though likely to share protocol, obviously). How well do they capture this? Roel will check. He noted the aim is to get useful type info fast rather than complete type info too slowly to help coding.

Q(Niall) Should this be integrate this with Michel Tilman’s Dynamic type work (the Zork-Analysis browser). He has not done that yet but it could be done and seems worth doing.

Q.Temp handling? He has done some experiments.

Q(Michel Tilman) If the typer finds that the message set is satisfied by two distinct classes? He just returns the set of classes found.

Q.Could type conflict checks be run as a test? Yes, but usually you want to know while coding, not later. RoelTyper warns when you accept a method.

Understanding your code assets with Moose, Stephane Ducasse, Tudor Girba and Adrian Kuhn of the Software Composition Group University of Bern, and Michele Lanza of the University of Lugano
Moose started in 1997. The aim is to understand legacy systems that have been dragged far from their original clean design by developer turnover, hasty implementation and misunderstood requirements. Reverse engineering means creating a high-level view of your code by stripping away details. Which details to strip away depends on the problem and the purpose for which you are doing reverse engineering. Moose analyses the code and provides a range of tools for presenting various high-level views.

The easiest thing to do is just count: they count classes, methods, ratios, etc. Visualisations convert counts into pictures. Browsing code gives you no sense of spatial orientation but humans are good at reasoning spatially. Moose uses a technique they call polymetric views. You have nodes and edges. On the nodes you can set five properties (width, height, colour, and x/y positions).

He opened a Moose repository, containing some ten system models, in this case ten versions of the same case study which is Jun, a 3D framework written in VW. (These models are exportable and importable between Moose databases.) They have kept the UI very simple. In the right-hand pane, you see something, you select it, you right-click to see what you are allowed to do with it. In the left-hand pane you have the StarBrowser tree which displays selected in the RHS pane and lets you manipulate the tree and drag-drop to/from it. He selected some classes and opened a visualisation of them.

(Long ago, an analysis revealed lots of duplicated code which they pointed out to the customer who replied, “Isn’t that good? It’s code reuse, isn’t it?”)

He then showed a class blueprint view, showing the class initialization, public interface, accessors and instVars. Edges (i.e. lines) showed usage and were colour coded to show usage types. From this you can see e.g. an instVar mostly directly accessed but occasionally accessed by accessor from within the class. (You can browse the source directly from any of this if the code is loaded but the browsing the Moose analysis does not require that the code be loaded.)

He noted some interesting classes, put them in a view and saved it. (The editor has a window at the foot that does search and select.) In metrics, a DotClass means a class that encapsulates too much, does too much and is therefore much used and dangerous to change. They have query that searches for DotClasses in terms of metrics. He found 23 classes in the total of 334 that the query offered as suspected DotClasses. He called up a view of how these classes were distributed in the system.

At this point in the talk, we have looked at Jun metrics but we don’t really know much about what it does. Systems have much knowledge in the names of classes and methods. They have a fuzzy string match, synonym match, etc., to find concepts. He showed two uses of this, one to find

concept in the system and another to see how similar the vocabularies of two classes are. Running the match returns a correlation matrix, whose individual dots are correlations between pairs of classes and whose axes are ordered into class similarity clusters. The view had two class list panes on the right showing the two classes of any dot you have selected and, in a third pane, the correlated terms. He selected a correlated pair and noted that the terms were mostly VRML-related. Another pair had some correlated names of methods; these were uncommented but opening a view revealed a well-structured hierarchy that was clearly handling a single concept.

Time contains useful information for reverse engineering. History can tell you which parts are change-prone and which changes are time-correlated. They have a history object (once talking to Java programmers, he found it hard to explain why one would want an object for a thing when wanting to reason about that thing). The UI for reasoning about histories is much the same as that used to reason about classes. JunBody is the most changed class. We can inspect it in a view where node colour indicates age and node size indicates frequency of change. Colour and thickness coded edges show what relationships have changed.

Moose is not just a tool. It is an environment for integrating code analysis tools. It handles VW directly and Java and C++ via external parsers. They have applied it to large systems (11,000 classes). Moose is on the VW distribution CD and at www.iam.unibe.ch/~scg/Research/Moose/.

Q. Does the tool correlate semantic info (the concept-finding text/synonym match) with hierarchic info (the counts)? Not now, because it is fairly easy to navigate between them by hand on a system you're studying, but they should and Tudor will (should take him two days or so).

Q. Could this be used as a code checker when coding like Smalllint. For example, it could show instVars that are mostly directly addressed but sometimes self-addressed by accessors, to prompt the user to decide whether refactoring to all one pattern or all the other is appropriate.

Intensional Tools, Frederic Pluquet, Universite Libre de Bruxelles
The Intensional Model is a collection of set definitions, called Views, and relations between them. Relations also have definitions, e.g. all methods in first view must call at least one method in second view. Using this model, his group analysed each of three versions of the SmallWiki tool.

He talked through SmallWiki briefly (structure of Pages, actions on Pages, etc.) then showed it in the StarBrowser with the view tools shown, starting just with the default views of all classes and all methods. He opened the view editor tool (RHS of StarBrowser). In the centre pane of the tool you write SOUL logic expressions, with embedded Smalltalk as required, to define the intension. A button lets you run it to see its current extension. He defined views and a relation, one view being a subset of another.

SmallWiki has protocol 'action' and many 'execute*' methods; these two views have the same extensions and this relation can be set up. He

calculated the extension; a green bar indicated that the relation held. Other views defined naming conventions e.g. action classes are the class acted on suffixed by 'Action'.

He then showed the graphical tool (from Moose) that displays views in rectangles whose heights indicate the number of methods and tints indicate the number of classes; relations are lines between the classes, coloured to show if the relations hold. He selected a violated relation and brought up a list browser on view elements to show the missing element. Sometimes it is appropriate simply to note an element as a known exception to the rule. The rule now runs green (and will run red in future if the element changes to be an exception no longer). Example: all classes in package satisfy rule, class that doesn't is later moved to other package. Rule will now run red again and you remove the known exception to fix it.

He then looked at the next version (one month later) and recomputed the view and relation extensions. Only one change to the captured relations is now visible in the graphical tool and can be inspected.

Q. Can I use this in production tomorrow How stable is it? It is a research prototype. It has been much used and now is fairly stable for our use. We will gladly give it to you to try.

Developing Smalltalk

SCRUM: Mastering your Process, Joseph Pelrine, MetaProg

(Christian and I started taking to Joseph at breakfast about Ginsu, iViews and other interesting subjects, and I should have said 'Time to go' five minutes earlier than I did. I had timed the subway journey from the hotel with an adequate margin but I forgot that the walk from the metro to the venue was a good ten minutes. We got there within a minute of the start time, and Joseph gave an excellent talk about the importance of time-keeping in meetings, the strictness with which it should be enforced - and how Niall's journey-time-estimation should be a bit less agile. :-)

Joseph started by saying how nice it was not to recognise so many people; new people are coming into Smalltalk. [Niall: Several other people made similar remarks to me during the conference.]

Joseph opened with a story about inviting us all to a party. He has limited time to prepare as he must work for a living so in his hour-long lunch break what is the most important thing he must do in that time? Maybe he'll give us lasagna with asparagus, white wine and shrimps and ... (it sounded like it would be a great dinner). So he must buy some of these ingredients. Next time he goes shopping, what's the most important thing? Maybe he'll give us a salad as well; peaches (which he'll rip, not cut; a rough surface holds the olive oil better, yummy) but they have no peaches. OK, he can switch to using plums. But if he can't get the asparagus (maybe it's not the right time of year) then the basic idea of the dinner is in trouble.

Suppose he takes a few friends shopping with him. They could optimise going through the supermarket by splitting up to get things and meeting

again in fifteen minutes to see if they are all on track or else have to switch some ingredients. That's SCRUM: people doing things and meeting from time to time to see if they are on track, or must switch something small, or must switch something big.

The party metaphor points out that there are various tasks. Some tasks need to be done in order. Some tasks have an innate time that cannot be hurried (the pregnant woman principle: if one woman takes nine months to create a child, nine women will do not do it faster).

The odds are against us. The Standish group regularly interview people who do big projects to see what percentage succeed and what fail, and why. They do this to educate us about the odds (or maybe they're sadists :-). He showed the 2000 statistics. The 2004 statistics have just been published and are even worse. The larger the project the greater the chance of failure. But this is not all bad for certain kinds of IT managers. If you owe the bank £10,000, they have you. If you owe the bank £10,000,000, you have them. The bottom row of the most complex, never succeeding projects are run by Accenture and suchlike; their 0% success rate does not hurt them because such huge projects become uncancellable till the overspend compels it.

A Harvard business school study (in 2001) found that managers thought that they could plan projects, could get the plan right enough that it would not need to change, and that it was a good idea to run projects like that.

Joseph works in the field of organisational complexity, basing his work on the research of David Snowden. Your ontology often influences your epistemology which often influences your behaviour. Joseph showed some 2x2 matrices (lower-left = bad, upper-right = good):

- systems can be ordered or un-ordered (c.f. undead: not dead or alive) un-ordered means order may spontaneously arise but is not imposed
- systems can be causal or un-causal

Lean development is the latest attempt to move ideas optimised in the manufacturing domain to the more people-oriented domain of software. To a first analysis, these attempts fail because people mostly do not act according to rules. Managers who know this nevertheless think they can express a vision and construct rules that influence people to realise their vision. They do not question the underlying assumption that the system they are dealing with is based on order.

Joseph can make a list of what the party needs to be a success. Afterwards (only) he can say whether the party *was* a success. Was it a success because he followed his list? This is called retrospective coherence. Rule-based epistemologies often confuse correlation with causality. You can confuse simulation with prediction just as you can confuse correlation with causality. People who believe that may try to do systems analysis and then to convert their analysis into heuristics.

How do birds fly south for the winter? They call the airport for a weather report, file a flight plan, hold a union meeting to discuss rules, then agree

an order of precedence for who will take off first. *NOT!* :-)

- Humans are not ants. Only autists make rational decisions [Niall: *I* make rational decisions. But I've known managers who didn't. But I though they *were* autists :-)]. Humans make decisions based on first-fit to past experience and then rationalise that (this statement comes from research published by Gary Klein). [Niall: my managers are more apt to complain that I seek the best fit decision when first fit would do. :-)]
- Humans have multiple identities even within one system (and we coexist in several systems). Stephane is a professor in two universities, does research, has a family, etc. He has multiple roles in each of these systems and the role priorities can change unpredictably.
- We also tend to impute intention when none exists: "I just made a mistake but he did that on purpose."
- We can evolve into malicious gossips, passing on bad news faster than good. This makes it hard to collect best practice stories; maybe it would be more in line with human nature to collect worst practice stories.

(Q. best practice is good? Discussion. Simple and effective versus simplistic is the issue. Joseph certainly sees a role for noting best practice.)

Many cultures have a tradition of passing on worst stories. For example, the Sufi culture write 'Nasrudin' stories. If a Sufi does something silly he writes a story in which legendary Nasrudin did whatever he in fact did - arrive at JFK the day his visa expired or whatever. We have the same; it's called Dilbert. Any consultant arriving at a new client looks to see whether Dilberts are pinned up and what Dilbert cartoons are there.

So what does all this philosophy have to do with SCRUM? Joseph showed a primitive artefact from the last century called a PERT chart which assumed you knew all about your project in advance. Actual software development is like driving: continual observation and correction (tending to cluncky over-action and over-correction when you are new to it).

"Driving out inefficiencies increases vulnerability to new threats as the adaptive mechanism of the complex system is lost" David Snowden. [Niall: I prefer Edmund Burke's 18th century prose: "A state without the means of some change is without the means of its own conservation."]

Joseph picked up a water bottle to represent the timeline of a project. "If I open this, it really will be the waterfall model." The worst point to discover you screwed up is at the very end; the best point is the very beginning.

A team can work very quickly in SCRUM but you cannot control them, something some managers do not like. Joseph showed various guru quotes on this, including one from Watts Humphries (CMM's inventor) admitting the impossibility of pre-specification. Agile is the art of the possible.

Babatunde Ogunake (chief chemist at DuPont) studied which management methods handle systems of various complexity. He found that defined processes decline rapidly in effectiveness as complexity increases.

So what is SCRUM. It has just three roles:

- SCRUM master: establish practice and rules.
- Product owner
- Team member

In relation to tasks, it also classifies people into chickens and pigs. (From the joke about the chicken and pig who plan to open a restaurant. The chicken suggests calling it 'Ham and Eggs'. "No way", replies the pig. "I'd be committed. You'd only be involved.") In meetings, only pigs can speak. We've all been in meetings where a chicken is talking on and on while the pigs want to get back to work.

SCRUM has three different types of meetings:

- Planning meeting: the product owner meets with the development team at the start of every Sprint iteration.
- Daily meeting: always in a neutral meeting room. *It is not a coffee break.* No drinks may be brought into the room so *it starts on time*. Everyone *stands* and answers three questions only:
  - — What did you do since last meeting?
  - — What will you do till the next meeting?
  - — Are there any problems blocking your progress?

That is *all* you do. Joseph has seen teams of 15 get through this in 10 minutes. (There may then be 20 minutes of several bilateral discussions in the coffee room afterwards and that's OK.)

SCRUM master training is one thing Joseph does. (How do you deal with the guy who always blames someone else for his problems? How do you deal with the guy who always wants to help someone?)

The further ahead things are the less you plan them. A task estimate is like the label on food. It has a shelf life, so should have a sell-by date which is one or two iterations long, after which your estimate is worthless (because the basis on which you made the estimate has changed, because in two iterations you learned something more about the system and if you didn't you should be in another job).

SCRUM teams are self organising. Seven plus or minus two is a good estimate of how many people can be in a single conversation round a table (if people are to engage on equal terms). People commit to doing tasks and can do what they need to get the task done (Grace Hopper: 'It is better to beg forgiveness than ask permission.' JP: working code is the best excuse.)

The Product Backlog is a list of things you think you need to do (functional decomposition). The Sprint backlog is what you plan to do in the current iteration (task decomposition). You may also have what you plan to do in the next iteration and for the release; that's it. Further off have no 'expected to be done by' date.

The standard sprint is a month, i.e. 30 days, i.e. 20 working days. Joseph personally prefers two weeks. Use the period that works for you. How long can you develop before you must make a decision based on what you've done. How long can your manager leave a team alone before they feel they must exercise control.

You do *not* ever extend the period. When it closes, what you've done is what you have. All else goes back into the backlog for consideration at the next planning meeting.

Estimates are given in burn-down times. A burn-down time is how much time you think remains to be done in a task. You estimate it will take 8 hours and work on it for 8 hours, then at the next meeting you estimate it will take 2 hours. The burn-down time is 2 hours.

SCRUM fits well with XP. Joseph briefly ran through his XP coding cycles (TDD, refactor and pair-programming), team cycles (40-hour week), etc.

Q. You've told us some SCRUM best practices but before you told us there is no solution to complex systems and no best practices? Joseph drew four domains of complexity: Simple (elements weakly connected, top-down arranged), Complicated (elements interact strongly so the order is networked, not hierarchical or otherwise simple and repeatable), Complex (strong connections between agents with no initial order, only an emergent order) and Chaotic (weak links everywhere). Complex systems are resilient to change whereas Chaotic systems are subject to the butterfly effect and can be very sensitive to input. Using techniques from the ordered realm in the unordered realm is very wasteful. Using techniques from the unordered realm in the ordered realm forfeits chances for optimising and thence achieving scalability.

Q. We like to think of software development as a learning process. How does this relate to that idea? Software development is indeed a learning experience. An estimate has a shelf-life precisely because of this.

Q. How do you deal with those who want an answers on cost and time? That takes longer to explain in full than SCRUM but he can gives some suggestions. If I ask you for an estimate you can reply with a number and a confidence estimate for that number. A fixed price project's bottom-line cost has much to do with politics and company goals but little with the work. Marketing and finance people are happy with this because they are accustomed to having 'how good do I feel about this' estimates on their decisions, e.g. how important is it to win this customer?

(Of course, many managers have read-only memories.)

Cooking with SUnit, Joseph Pelrine
This is an Intermediate level talk. On the one hand, it assumes you've written tests and know what assert: and deny: mean. On the other hand, it will not talk of CrossProcessTestCase or CompetingResource or SharedSetUp or any similar subtlety.

SUnit: never has so little code changed software development so much.

Ralph Johnson got people together at the first Camp Smalltalk to prepare ANSI compatibility tests and they promptly found the tests did not run the same on all dialects, never mind ANSI. So Sames, Jeff O'Dell and Joseph began SUnit 2.6 and 2.7 to be dialect neutral. Later, Alan Knight kept asking him for Glorp things and so he added some stuff to SUnit which few other people use or indeed understand. That's what this talk is about.

(All his demos are in VisualAge because that is what his current client is using. No dialect preference is implied.)

JUnit is a big fat baroque testing framework for Java whose only affinity with SUnit is that Kent Beck worked on both. People use JUnit as the basis for ports which is why the first version of NUnit was so bloated (later ex-Smalltalkers slimmed it). Joseph did steal something from JUnit:

```
self assert:...description: 'what does failure mean'.
self deny:... description: 'what does failure mean'.
```

Joseph ran `testLeapYearDates` (and had the demo hiccough - it ran - he'd opened test runner before loading tests :-). He showed getting 'Assertion failed', thence debugger and work back to what failed, then redoing with a description that told him which year failed. Descriptions can often save you having to go to the debugger before fixing your test. (In Rosetta, his description string actually printed out the class definition so when his load failed for lack of a class he could create it instantly.)

[Niall: descriptions also document your tests; text in a description string is more useful than text in a comment.]

```
self assert: ... description: ... resumable: aBoolean
```

By running his leap year test with `resumable: true` and a meaningful description string, Joseph sees that 2004, 2008, 2012, etc., all fail the 'February has 28 days' test - perhaps this tells him something. :-)

The Glorp group needed TestResources. Each test runs setUp and tearDown. So a sequence of tests is

```
setUp ... tearDown setUp ..... tearDown setUp ...
```

and some connections just cannot take being hit that often. Moreover, some connections take very long to set up. Moreover some things are essential for tests but you expect them to be invariant during the test so the resetting has no value. Joseph created a simple resource that just wrote to the transcript (`TestResource subclass: #MyFirstResource`). To use it, he must tell all the test cases that need it.

```
MyTestCase class>>resources
  ^Array with: MyFirstResource
```

He showed running MyTestCase alone and also as part of a group of tests. (Vassili's "Do what I mean" utility saved him repeatedly - 'Run this one in the TestRunner' and so on.) He set up a TestResource to capture the

TestResult object.

```
MyTestCase>>testDummy
  self assert: true.
MyTestCase>>run: aResult
  MyTestResource current result: aResult
```

He then made it print to the transcript on tear down, so running tests had the transcript showing the overall number run, passed, failed, errored.

(Joseph noted that Jeff O'Dell's SUnitBrowser for VA, excellent in many ways, does things with test resources that means this would not have the effect intended if run in it. I mentioned my extensions to it.)

Often, his clients don't want to set up databases so he bosses out data and similar simple solutions. He does not want to test on his live data so he has a StoreStateTestResource that he can capture data into. He always reloads on setUp because he might have changed the data. When he reloads this resource, he runs tests to check the data is the same.

```
StoreStateTestResource>>setUp
  self reload.
```

He noticed that, when developing tests, failures tended to be either solved in 5 seconds from the description string or after 10 minutes in the debugger. So, when working with IntelligentViews (Jan Schuemmer et al.), he put together a TimeDelayedTestResource to delay resource tearDown. He ran tests including one with this resource. They ran, then we waited five seconds before the Transcript showed the resource tearing down.

```
TimeDelayedTestResource>>current
  self stopTearDownProcess.
  super current

TimeDelayedTestResource>>reset
  self ...
  super reset
```

If one had failed, he'd have had time to investigate before losing the connection which would make debugging state left by the test harder.

There are many tools that it would be nice to wrap in SUnit. ExtensibleTestCase lets you define a different behaviour for a test depending whether you are running it or debugging it. If your test is comparing two strings, you don't want to see the debugger first thing when you hit debug, you want to see a string comparison tool. He ran a failing test and debugged it and it popped up his string comparison browser. The first implementation was complex; he has rewritten it to be simpler.

He showed a Smalllint test case where debug brings up the Smalllint browser. There is no test* method. SUnit method `allTestSelectors` that recovers those methods it thinks are tests. SmalllintAbstractTestCase and its subclasses override this to return the applications (packages, whatever) that should be tested. It also sets which rules should be run.

Q. Why not just run Smalllint? Automation; Joseph has SUnit running outside of images in builds. It's very convenient to have all the checks he wants to make being reported to one place.

Joseph then started reflecting on where SUnit should go from here.

1) He is thinking of dropping the TestRunner. In the latest versions of VW, VA and Dolphin, almost no one uses them. Joseph is integrating the description strings into hover help in Jeff's SUnitBrowser so you don't even have to open the test to see the error description.

2) He has an SUnit3.2 in beta. Should he just ship that and then resume work on SUnit4?

SUnit4 has standard logging mechanism based on Toothpick. He looked at Log4J which is a bad Java open source implementation of some good ideas. His first implementation was called SmallLog and was very bloated like its Java inspirer. It was refactored to Chopstick and thence to Toothpick, getting smaller and smaller (these names are Vassili's; any smaller and they'll have to call it Splinter). The logging is event driven and can log to files, email, transcript; with just error, with timestamps, whatever.

He also wants to refactor the SUnit core to use events to signal running and results, etc. He wants to make TestError and TestFailure into objects in their own right, not just instances of Exception. Working on 3.0 and 3.1 taught him about compatibility layers.

SUnit2. Joseph has talked to Travis who has very good ideas. SUnit2 was built for one dialect; Joseph want to take those ideas that are cross-dialect. There is also the question of what's in the core and what is add-ons. Are resumable test failures an add-on.

Joseph is back and 3.2 and 4 will appear (and then he'll make arrangements to pass on stewardship).

Go to http://www.metaprog.com/ESUG for some old images, the Smalltalk Knights of the Square Bracket logo, the 'extreme programming exploited' book cover, etc.

Q. (Hernan) They have made some changes in SUnit. When there is no description string, a description string of the line of failing smalltalk code is shown - it is better than just 'Assertion failed'. They have tests marked 'for future - we don't support this yet'. They have a different way of running Smalllint in SUnit. (They have also done something with signalling which Joseph liked but on which my notes are inadequate.)

Q. Why are TestResources so misunderstood? No manual yet, he's written the first ten pages (available on his web site).

Plan to write book, 80 pages, all proceeds to ESUG, heavy hitters wanted to help write and to be part of SUnit team. Also role for reviewers (who

should not be heavy hitters).

Cincom Smalltalk Business, Suzanne Fortman, Cincom
She started with Digitalk VSE way back when. Since January 2005 she has worked for Cincom. Tom Nies recently said 'If Smalltalk is doing this well without our doing anything with it, what could we achieve if we actually pushed it?" So they wanted to beef up their marketing and Suzanne's name kept cropping up in discussions of who could help. So, she's back.

Suzanne toured North America in the last six months. They love Smalltalk. They need more visibility of it, primarily in the vertical markets. They need their managers to be reading about it. So Suzanne's aim is Visibility, Visibility, Visibility. We must talk outside the Smalltalk community. Tell Java people that Smalltalk is good, not that Java sucks.

Suzanne likes the mission statement of ESUG. If she does her job right, all the students in the other session will have jobs. This is her first ESUG so most faces are new but she's delighted that Giorgio Ferraris is also saying 'All these new faces!" and that's great. She knows lots of people who say "I love smalltalk but I have to work with Java/.Net/, ...". The students here want to know if they will be able to work in this cool language they're learning.

She mentioned what she did over lunch at a conference and someone said, "I thought Smalltalk was dead"; she replied, "You have three divisions in your organisation running Smalltalk applications." This kind of thing was happening even in Cincom, which is a large organisation where some lines have no real knowledge of what Cincom does with Smalltalk. So she has begun marketing Smalltalk within Cincom, mentioning customers. This has led to Cincom people asking whether they should use Smalltalk.

Lots of customers say, "What can I tell my boss to say why we should use Smalltalk." She aims to create a climate in which if people cannot talk publicly about what they do with Smalltalk they can at least talk about it within their own organisation. And that will mean that next time they go to their decision makers, it will not be "Why do we still use Smalltalk?"

Suzanne needs the community's help to put the apps written in Cincom Smalltalk out to the vertical markets. You are not the decision makers. Some other guy signs the check and he's reading a different magazine from you. Kapital and JPMC are cool for them, OOCL is cool for them, and there are people here from both those projects but the decision makers don't read the ESUG report. [Alas :-/)]

She wants to build the Smalltalk community, not just Cincom Smalltalk. Building the community will build her share of it and we are too small a community to do anything else.

Because Smalltalk gives competitive advantage, she often hears, "I can't talk about it". Suzanne does not say, "Use Smalltalk", when she visits companies. She says, "You have a problem; I have a solution. By the way,

it's in Smalltalk". Customers are not interested in being told to go to this website where there's a list of apps written in Smalltalk; they'll tell *you* where to go. Give them the info, don't make them search for it.

Lots of people this week have told her about great apps written in Cincom Smalltalk; let's market these. These apps were written to fill needs. This can be talked about. We must start talking outside the Smalltalk community. We must talk in the language decision makers understand. "I have a problem. I need a solution. I know that many solutions have been written in Cincom Smalltalk."

Resources: the community has people saying I love Smalltalk and I can't find jobs and the customers are saying I find Smalltalk valuable but I'm worried I won't be able to find Smalltalkers. (Also "Smalltalkers are more expensive" - but they are more productive and they're not more expensive, e.g. Lucas Renglii is still at university and at NetStyle.)

Cincom now has an add-ons and applications programme, aimed to help them make it possible to get the success studies out.

So what can we do?

(Reinout) Joseph Pelrine's talk mentioned success rates of projects versus price. Can we publicise that? Suzanne's been back in Smalltalk for 7 months and is aiming to get that research.

(Niall) We don't know what magazines and conferences decision makers are interested in? If you tell us (web site, whatever), that may influence us.

(Mike Roberts) Universities don't mention Smalltalk, nor do geeks. He did audio and music and knew audio drivers were written in C so that's what he looked at. Smalltalk had a good programme in the old days and it has dwindled. Now they will restart. Ralph Johnson has told her lets start with high-schools: get them young. They will do something in this. The kids will still know Java. We must make them aware of Smalltalk.

Q. People are very helpful in the Smalltalk community about giving links and web stuff. We need more things like SqueakSource where you can find out about things written in Smalltalk? Yes, she wants to have one place, not be told go here, go there, and to start pushing data out.

She played a game with a fellow-passenger: he started noting, "They use our product", of the firms they passed so she had, "They use, and them, and them, and ...". Him: "This is no fun; you have all the cool customers."

Q. Often here "We need industry-standard solution" (meaning Java). How do we tell them that Smalltalk is an industry standard in the sense that lots of industry solutions use it? Yes, to many people 'standard' means "something I hear about all the time", not "something that works all the time". (Be aware: her programme activities will be aimed at those people, not at us.)

Q(Roel) He wants to be able to tell the students and the university people that students will be able to use this language in industry.

Q(Francisco) We could get leverage by working with others and committed groups? You're sitting where Petr sat at Smalltalk Solutions when he asked about working with Java. With mergers and acquisitions, people must work with other systems. (James) You don't have to make Smalltalk universal; you have to make it universally regarded as an elegant and effective solution. (Niall) use Georg Heeg's examples of Smalltalk being good at making 3rd parties talk to each other. (Suzanne) I'm looking at having tracks in Linux and elsewhere, so STIC will look at rolling Smalltalk Solutions into a larger conference or similar.

Suzanne wants to create the 'me too' mentality with the applications and add-ons programme. She's excited about Cincom Smalltalk these days and about Instantiations and what they will do. Tom Nies has mentioned Smalltalk in the last few interviews he has had (which is good internally).

Teaching with Squeak, Rite Freudenberg, University of Magdeburg
(This talk was guaranteed to run to time because Maja d'Hondt had arranged that the workmen next door would resume drilling at coffee time.)

Rita has spent her last year teaching teachers to use Etoys and also children and students. Her talk is about the differences she has seen teaching Etoys to these three groups.

Children: at her son's school, she and another teacher taught a group of five children aged nine. First you must explain and then give the laptops; if the laptops are open you will never get the kids attention. Except for this, it is easy to motivate the kids to attend lessons on computers and robots. You must go slowly. You must have things in the native language; great that Etoys is available all in German. Many Etoys explain themselves.

Drawing and geometry is a good start point. Their first exercise was to tell them to draw a Santa Claus house. It made them think about where to put the lines and angles. It was hard to get them to end the lesson; they wanted to carry on. Everyone wanted help so they could have done with 5 teachers for the 5 children. Children try anything so get themselves into all sorts of unforeseen situations, then ask you to get them out of it.

You must have simple tasks. There are no computer science classes for children so they borrowed books from the university. It took her two hours each lesson to collect the books and computers and take it over to the school. (It's a lucky German elementary school has two computers.)

Students: she taught various courses, some for girls, some for boys (i.e. also for boys, I think, not just for boys). The girls course was introduction to computing and involved drawing/painting and animation with computers. They had bugs trying to escape from mazes, bugs replicating themselves, etc. The students had all chosen the course so were motivated.

A school course for 16-year-old girls (a get-interested-in-computers course): painting and animation was interesting.

A ten week EU project for 8 students was to go to firms and do micro-controller programming, voluntarily for 3 hours from 17:00 after school. She found drawing only occupied one hour (plus boys draw fight scenes, not flowers, which was less motivating for her :-). so she integrated micro-control work to do a traffic light control project.

Older students can read documentation, including documentation in English (very useful for reusing examples on the Squeak page; she translated the first example to get them started, then let them read the rest in English). Squeak animations are certainly suitable for getting girls to take an interest. They get interested and become willing to try more complex tasks.

Teachers: these teachers were seeking an extra qualification to teach computer science along with what they were teaching at high schools. Thus they visited the university once a week to study, over and above their weekly workload. She showed the squeaker DVD and they liked it but saw no relevance to their teaching tasks as the children on the DVD were much younger. They were not enthused by the idea of the gravity example for their own use. They could not see from the DVD what Squeak would let them do. She was surprised by this as the DVD had motivated her but her teacher pupils were dismissive; they only saw that the children were younger. So she has the problem; how to make these teachers see the power of Squeak.

They also saw the English documentation as a problem. This is not just the older teachers; she found even teachers in training were demotivated by the thought of having to handle English documentation. Typical reactions were along the lines of: “Would you like to use it in your classes?” “No, we have no time for this because we have guidelines for every subject. We could use it for Computer Science but there we must just teach computer science” (and Squeak is about using computers *for* things). You can’t find much about using computers in other subjects in the guidelines. So somehow she has to find how to integrate Squeak-use into the existing guidelines. (Germany has 16 states each with its own education system so maybe she’ll find one.)

There is one exception: the project week. Once or twice a year, the teacher is allowed to escape from the guidelines and teach something they plan for a week. So maybe they could provide a Squeak package oriented to this.

She has one power; she controls the course and can say what the teachers must do to get their certification so she plans to make them do a one-week Squeak course as part of it.

“It’s hard but we must use the teachers we have.” :-/)

There are also the non-public schools (i.e. not state owned: what in the UK

would be called public schools :-). They are also compelled to regard the guidelines but have more freedom. She has two motivated teachers from such schools teaching 11 year-olds. There is a project where you connect a temperature measurement device to the USB port and use Tweak in Squeak to sense the information. They can connect Squeak to Fisher-Technik.

Q. Stephane has seen the same thing with his wife who is a teacher. The “so what?” factor does occur. It can be helped by providing more info on what is the goal of this example, what do the children learn from it.

Fun with Programming, Stephane Ducasse
Etoys is not enough. Stephane wants to teach computer science. If Squeak is good for teaching then it can be used for that. What is a program, a class, a method, etc. The book uses OO terms throughout but not ‘late-binding’ and suchlike as that would be too complex.

Stephane’s book is for novices of all ages, and those who are teaching them (parents, teachers, whatever). There are no expectations on the background knowledge of the readers. Stephane’s wife (a teacher) proof-read it for usability. Her aim was to be able to reuse just the diagrams and pictures in her classroom. (Visit the web site for sample chapters and code, etc.)

He starts with simple graphical manipulation and concepts; difference between absolute and relative angles, etc. Then loops: why, what are they, etc., used in staircase-drawing examples. Then he presents some problems. Start coding squares within squares, pyramids, etc. Drawing uses a turtle and users tell it to `go` (draws) or `jump` (move to new position without drawing).

Another example is a bug wandering around the screen. All scripts in this chapter are 5 lines long and let the bug look for food, look for the way out, look for friendly environments, etc. This teaches controlling the bug by direct manipulation, then by scripting, finally by methods and abstraction. Later maze-escape examples teach path following and recursion.

He is rewriting everything in tweak because when he had a large multiple turtle example in Morphic the semaphores everywhere sometimes locked up; it was too complex.

His standard presentation screen has the simulation display and a very minimal Smalltalk browser where people can define methods and classes. His syntax highlighter has been customised to help users. He showed what he did with children at the last conference he attended. He wrote a script to draw a staircase.

```
r2d2 := Bot new.
4 timesRepeat: [r2d2 ...
```

He showed an environment of coloured squares with a robot moving from square to square.

Stephane’s wife teaches computer science at the French school in Berne to

11 - 15 years old and was his first customer. “I recently started a course with 7th-graders [13 years old] with Stephane’s book -- they love it” Klaus Fuller, Germany.

He has one user (an elderly teacher) who uses the environment to create pictures.

He wrote 900 pages and 300 are in his book. So if it sells, there are likely to be sequels. Details are on the squeak wiki.

Q. How do you tell children that `4 timesRepeat:` is how to loop? Stephane’s wife finds that is much better than loops in other languages. The children were soon doing 2500 timesRepeat: with variable handling that caused it to look slow so he moved the variable stuff a little later so they met it when they knew enough not to do that.

Scratch, Mike Reuger
Scratch is based on Etoys and is used in computer clubs. It is visual programming of scripts: drag-drop, etc. in LHS pane to control visuals in RHS pane. You can have several elements and step them (at different rates). Much effort has gone into the sound part of it. It is downloadable.

Benefits and Risks of Service-Oriented Architectures, Andy Berry
Web services let your systems and your customers’ systems exchange messages over the web. Service-Oriented Architectures could convert your isolated applications into an architecture and bring you closer to your customers and your suppliers. Andy thinks that web services will take off, so your architecture will be future proof.

However there is the risk of being an early adopter if web services fail. Fortunately, it is easy to find small aspects of your apps that offer specific benefits from being made into web services. Andy recommends having a mentor and choosing your product with care. With your prototype, get buy-in from the company/client.

Web services are well implemented in several Smalltalk dialects. It took Andy one day to add the google service to his Smalltalk app. Just do it. [See Alison Dixon’s talk and Kirk Blackburn’s talk in my ESUG 2002 and Smalltalk Solutions 2002 reports.]

So what’s next, beyond SOA? It would nice if we just paid for the services we used.

Q/. What do the commercial products you mentioned provide? Basically, they provide ready-built adapters for certain applications. Thus they do half of the work for you.

Q. Pay as you go will go the way of Java’s wild claims? Yes, it’s a risk but Andy believes it will go further. It *is* a risk and that’s why he says start small.

Q. Is this market anywhere near developed enough to be commoditised? If not, will the result be charges you cannot predict or collect? The services metaphor only works when you have a commodity and this seems nowhere near commoditised. How will it benefit business to give out control of their architecture when there is no clear way to call for a service and there is no market so you can't tell what to expect when you call for another? (Andy and Reinout) To people who use the service, it can easily be commoditised but the people who provide it are an issue. Discuss more off-line.

Environments and Tools

OmniBase, David Gorisec

OmniBase is a multi-user persistency system for Smalltalk, not a database in the usual sense. It aims to let you save single objects, objects in transactions, etc., but otherwise have the directness of image-saving in making objects available to you.

It aims to be cross-dialect so no GUI code. It runs on Dolphin (most up-to-date because that's what he uses.) It has been ported to VW, Squeak, VAST and ST/X by other Smalltalkers and these are maintained by him and updated from time to time on request. (If you have a need, contact him and offer to help.) Porting to Linux has problems with file-locking as it calls directly to the file-system, not via sockets, and somehow seems to be depending on Windows semantics [CS work seemed to find the cause; see CS report above]. OmniBase is used at 150 sites. It was made an open-source project recently.

OmniBase has multi-version concurrency. You see the snapshot of the DB at the point in time when your transaction started. Each update creates a new object with its own version number.

OmniBase has a basic schema migration mechanism. It has the obvious semantics for when instVars are added, removed or reordered and offers an ODBUndefinedObject for cases when deleting classes creates a need for it.

```
db := OmniBase createOn: 'c:\myDB'"new"
db := OmniBase openOn: 'c:\myDB'"existing"
```

Cycle is start transaction, get, create or change persistent objects, commit. Transactions can be used explicitly or implicitly by associating a transaction to the active process. A typical use is to start a transaction when the user opens a GUI and commit it when they press OK.

```
[OmniBase root"implicit"
  at: EMP001' put: Employee new]
    evaluateAndCommitIn: db newTransaction

txn:= db newTransaction.
txn root at: EMP001' put: Employee new.
...
```

The root object is a dictionary by default but you can change it to whatever you want. Only what can be reached from the root is safe from garbage

collection (persistence by reachability).

```
OmniBase root
  at: ‘misc’
  put: (OrderedCollection newPersistent
        ...)
...
coll := OmniBase root at: ‘misc’.
coll add: ....
coll markDirty.
```

You must `markDirty` to tell OmniBase the object has changed.

A cluster is how OmniBase understands a complex object that is the root of a graph of objects you will always want to be fetched as a unit. Clusters are serialised as a contiguous stream of bytes and stored together in OmniBase under a unique id. Thus clustering choices will affect performance. He showed an example from his web page of storing a collection of strings as a cluster or as a set of strings or as the strings being clusters. This affects uniqueness. Strings as clusters ensures that the string is recovered uniquely via references but if a clustered collection is recovered and the unclustered string is recovered via another root you will have two copies of the string. OmniBase uses `isIdenticalTo:` to check identity in cases where you cannot tell whether OmniBase will return the real object or a proxy to it.

There are various issues with `markDirty`. If you add to a collection, the object added has not changed, only the collection. Generally, parents need to be told when their children change. You could add a trigger to tell all parents when their children change or you could serialise all objects on committing and compare them to see what has changed. You could also make objects immutable on load and change that when changing. He has not implemented any of these as yet, just done experiments. He expects to use exceptions when changing to handle this in the next generation of OmniBase for Dolphin and VW.

OmniBase indexes work much like Smalltalk dictionaries but Smalltalk dictionaries don’t perform well past certain sizes [Niall: 10,000 entries?]. There is also the problem that Smalltalk hashing does not handle object versioning. A later version of the object will have a different hash. This causes serious problems with parallel updates. He therefore has added a b-tree dictionary, with better large size performance and safe parallel update behaviour. An object automatically becomes a persistent cluster when added to a b-tree.

```
index := OmniBase root at: ‘misc’.
...
index keysFromDo: ...
index transactionsDo:
  [:txn :eachEmployee |
  self processEmployee: eachEmployee].
...
```

A btree dictionary can also act as a cursor with `getFirst`, `getLast`, `getNext`, etc. The sequence is the sort order of the keys.

OmniBase also has a full text search support utility, added recently. You can add a declarative model on top to support more SQL-like queries.

You get the object as it was when your db transaction started. To update your view you must restart your transaction. You can use object locks to prevent anyone else changing the object while your transaction is running (only write-locks, no read-locks). Locking is fast as it uses file-locks. Dictionary key-locking is slower as it must change things in the file but it may be overall faster to lock a b-tree if you are doing massive updates to it.

Be aware that b-tree dictionaries are versioned objects too, so it is wise to preserve the order in which you update it in your application.

Users wanted on-line backup; they now have this. There are no DB administrator tools; you must call Smalltalk methods (so can easily write a GUI if you want). There is no on-line garbage collection (usually not an issue for users with today's cheap memory). The case of a b-tree growing towards the 32-bit limit can be handled via the b-tree reorganisation code.

Info is at wiki.gorisek.com. Downloads are at www.gorisek.com.

Q. Performance and size limits? 4G limit on each index size file and 2G limit on each container. Speed is very model and app dependent. (Usual demo hiccough: he tried to demo performance but the internet connection to the projection machine had been lost.) 16bit (~65,000) users limit.

Q. Why have your 150 sites used OmniBase instead of e.g. Oracle? He thinks because they prefer to use Smalltalk rather than embedding SQL in their code. There are Seaside sites, e.g. Argentinian army uses it. Maybe also users like the fact that there is no licence-oriented user limit.

Q. VW limitations? There is a concurrency limitation. VW file-locking under Linux is not completely the same as in Windows. Dolphin is Windows-only so for VW they wrote wrapper code to simulate the behaviour but it seems to have an issue. With one image it is no problem but if you have two images concurrently writing to the same b-tree you can see a problem. [As noted above, Camp Smalltalk work seemed to have made progress solving this; see the Camp Smalltalk report above.]

Q. You use it in your commercial work? They use OmniBase-RDB which they have not yet had time to make available.

GemStone 64 bit, Adriaan van Os, Soops

GemStone is a privately-owned company with 100 employees, 200+ client companies. Their products include GemStone/S, Facets (GemStone/J) and GemFile (distributed caching and continuous data stream querying). Their Smalltalk market is growing and the customers are very loyal (90%+ renewals). GemStone 6.1 (currently 6.1.4) runs on a range of platforms. Their customer feedback has told them to keep GemStone/S current (on the latest platforms and OSs) and to keep it stable. The next release (6.1.5; due late 2005 or early 2006) will fix bugs and support AIX5.3 and Solaris 10.

However, 32 bits are limiting; 64 bits allows much larger object caches. The shared page cache size rises from a best of 3.75GB on SUN for 32 bits to 16 terabytes in GemStone/64. The maximum object count rises from 1 billion to billion in the first release (still 32 bit object pointers) but 1 trillion (2^{40}) in the next release as it will have 64-bit object pointers.

The 64bit project was funded by OOCL, a shipping company who are a major customer. The projects started in 2003 and will complete in 2007. It has 3 phases Ashland, Bend and Corvallis (no, Adriaan cannot explain these names :-). Ashland aims to improve performance, Bend to improve scalability, while Corvallis will address tuning.

Ashland delivers a VM that is twice as fast, 16,384GB shared page caches, better garbage collection (keeps a not-connected set, collects large objects better) and an online backup capability. The server runs on Sun Solaris 2.9 and HP HPUX 11.11 (on PA RISC). Clients run on Win2k and XP for VW (7.3 and 5i.1) and VA (5.5.2 and 6.0.2). (Alfred Wullschleger has also got GemStone/64 client running in VW3, which he uses in Swiss National Bank; their performance limits are all from GemStone so they have not ported forward from VW3 yet.) It was released in Beta at the end of last year and fully at the end of March.

Ashland 1.1 was released in Beta mid-year; full release in due in early September. It adds GemConnect for Oracle, epochal GC, soft references and `System continueTransaction` (and runs on AIX 5.3L).

One drawback is that you need to know how much memory you will use in your session and exceeding that kills it. Customers are not happy about that so maybe they will change it.

Bend will do 64 bit object ids so giving 1 trillion objects. Page size will rise from 8k to 16k to match this. The SmallInteger range will be expanded to +/- 2^{60} and there will be SmallFloat and a special date-time class (unique dates). It will offer indexes on IdentitySet and IdentityBag. It will support GemEnterprise/SMF and GemBuilder for Java, and will run on Linux. Bend is due out in Beta at the end of this year and fully next Spring.

Corvallis will give a faster VM, multi-threaded GC, faster tranlog replay and restore, etc. Its beta release is scheduled for end 2006, full in mid-2007.

GemStone has 1 customer in production and two more will deploy this year. Soops is one of 8 customers implementing a proof-of-concept system. Adriaan showed a list of 15 things where performance impacted Soops' users. He has run benchmarks for all of them, doing first time and second time (caches populated) checks for each. He showed the raw out-of-the-box upgrade benchmarks. The 64 bit first time always equalled or beat the 32 bit second time. On direct comparisons it was noticably faster: some factors were 6-to-1.

One drawback is the GemStone now copies objects from the shared page cache to the line session so if you only do one thing with that object, that

is slower.

Q.(Alfred) The shared page cache is so complex that two runs can give very varying results. How consistent were these ratios? Fairly consistent; we did several runs and did see variations as you say. (Discussion with an OOCL guy raised the point of tracking CPU time in such cases, seeing how much of the page was read, etc. If I understood aright, OOCL people running benchmarks have done these checks.)

Q.(Niall) The memory limit; why? GC in Gem has always been expensive so they tried to eliminate it with this memory limit. There has been lot of opposition from customers to this Java-like limitation and Adriaan hopes this will be reverted. (OOCL guy confirmed this and discussed the technicalities in some detail. If you take enough memory to guard against this then you will anyway not encounter the overflow problems that are the only issue for which this is a solution.)

VASmalltalk Going Forward, Mark Johnson, Instantiations
My Smalltalk Solutions 2005 report covers this in great detail so here I just note additional information.

Mark showed ‘the wheel of reincarnation’ showing how ‘the Smalltalk companies they have been involved in have been merging and splitting off since 1984. Instantiations is now offering the VASmalltalk product. Instantiations is in many programmes (IBM PartnerWorld for Developer and PartnerWorld for Software, Ready for Rational and Ready for WebSphere). More impressively to me, they are a member of STIC.

IBM wanted to get out of the VASmalltalk business. (Since that lets Instantiations pick up the torch and carry it, Mark is not sorry. Eric is passionate about Smalltalk and has been hoping for this opportunity for years.) In IBM-speak, Instantiations is part of their longer term, “staged transformation” strategy, where IBM recognise (sotto voce) that the stages might be so very long that customers never do actually move away from Smalltalk. However it is part of the contract and obligation that previews of migration tools (Synchrony and etc.) are included in these talks.

VASmalltalk 7 is based on VASmalltalk 6.0.3, fully backward compatible but with Instantiations products added (VA Assist and WidgetsKit, plus GF/ST as a goodie). VA Smalltalk is an Instantiations product, not an IBM product, and Instantiations supports it.

Mark then spoke of the road map. He stressed they welcomed emails from customers on their long-term needs. They will improve DB interfaces (IBM’s interest were limited to their own DBs). There will be a 7.5 release next summer and 8.0 the year after that.

Pricing New $6995, upgrade-from-IBM-supported $1495, upgrade from unsupported $1995 (also from other Smalltalks). For consultants, academic and ISVs, there is a greatly reduced rate.

IBM 6.0.3 has only just been delivered so that delayed the VASmalltalk7 release correspondingly. Release Candidate 1 will be available for download by the end of today (in the US). VASmalltalk release for windows will be released end-next-week. Linux, AIX and Solaris will be out by the end of the month. For info, visit www.instantiations.com or email sales@instantiations.com (for sales info), vast@instantiations.com (seen by Eric and whole team) for general info.

Q. Macs? They have no stated intent to provide a Mac VM. If there is a business case, they'd consider it. You may notice that they have dropped OS/2 and HPUX which IBM were supported. They must have an adequate business case to support a VM.

Q. Embedded development? IBM never sought to position it so. They do support it on MVS and IBM have committed to continuing to market, sell and support the VM on MVS (not to enhance it beyond keeping it current with latest MVS). Instantiations are committed to being fully compatible with the MVS version.

Q.(Roel) We are a university with IBM deal to get IBM tools free for use in teaching; can we get the same deal? Talk to us, we will be flexible.

He then fired up VASmalltalk7, not to demo, just to show us it was there.

The Squeak VM: Exploring Garbage Collection; a view from 10,000 metres to the bits, John McIntosh, Corporate Consulting
John is the maintainer of the Squeak Macintosh VM and works with Ian Piumarta and Tim Rowledge on the Squeak VM generally. He also works on the Sophie/TK4 project.

Having spent years looking at pretty graphs of GC info in VW and elsewhere, he remarked that it would be nice to be able to see the same information in Squeak. He then of course found that he was the obvious person to do that, making the VM collector have a memory policy that can change its behaviour, as in VW.

The original Squeak aim was to GC in less than 10ms so that Squeak could play sounds. So they implemented a copying compacting GC which copies and compacts from free space up to the end. The downside was that a full GC became expensive.

The other thing they did was to use direct OOPs. At load time they do bit reversal (big/little endian).

You allocate objects in young space; how big should it be, when do you stop and do an incremental GC. So periodically it does, based on values set in the late 80s or early 90s; maybe these values are no longer appropriate. :-) When you GC, you decide whether to tenure objects: do you expect them to live forever?

The garbage collector is triggered at a number of points, one of which is

after 4000 objects, which took less than 10milliseconds years ago and is now done in microseconds.

Roots point at objects from the remembered table. ‘Mark’ finds dead roots and ‘sweep’ compacts, so the dead objects are all moved together, creating free space. If there’s a problem you will find it fast; the VM will attempt to refer to somewhere in the middle of another object and instantly crash.

A pointer is an unsigned integer. Slang returns a signed integer by default. Thus on a 4Gig machine you cannot go over 2Gig without getting into signed versus unsigned issues and the VM crashes. Work is ongoing to fix it. The 64 bit machine semi-fixed it so you could address 63 bits before seeing the problem. If you ask for a Gig of memory, the OS may give it to you over the 2Gig boundary and then the VM crashes. How many have seen that (several hands).

Past the boundary, objects get tenured. Question: is that boundary set sensibly for today’s machines. A full GC can shrink YoungSpace. The remembered table is resized recursively (not exactly but it means Squeak will simply get slower, not crash if it turns out that the RT is too small, unlike VW in the case where it needs to resize the remembered table and demands more memory from the OS and the OS refuses). This was done because Squeak wanted to be able to run if there were no underlying OS.

If you allocate a big collection in old space then place new references in it, Squeak can get slow as incremental GC scans every slot in the collection. The workaround is to allocate the collection then do a full GC, promoting the new collection slots into old-space.

Free space is calculated after the remembered table and that can cause an issue (if the RT needs free space but what you just did filled it).

You can fail to finalise young weak objects that are referenced by WeakArrays living in old space, so not finalised until a full GC. This has been fixed in April 2005, being ported to all platforms in Summer 2005. However, Seaside used weak objects for session data and found that if you had 40,000 objects you scanned all 40,000 looking for the one to finalise. They switched to a different pattern.

Squeak used to raise dialogs re the lowspace threshold. With current machine times the user gets 0.3 of a second to decide how to respond.

In 2004, John discovered that when you approach the end of memory you try to increase memory but you accelerate your GCs so as you approach the boundary you would do 500-600 GCs before reaching the boundary and increasing the memory. This became obvious in progress dialogs that were hanging briefly at a certain point. John changed the memory bound logic to imitate VW’s: you can now grow to 16Mb before becoming aggressive.

This summer John added another 120 values to what the VM GC collects. He showed diagrams of GC when starting up Croquet, which takes the

image up to 200Mb, comparing before and after changing to let it grow to 16Mb before becoming aggressive. Croquet started 8 seconds faster but still took 500 seconds. He then looked at marking, reducing the amount done by forcing tenuring if the rate of iteration in the marking is high, and saved another 40 second clock time.

Of course the optimum solution is to rewrite the logic. (Does anyone have a million to spend?)

Q. Did they (or should we) consider multi-generation? Way back when, they saw a two-generation GC would do for their requirement. They decided that was the simplest thing that could possibly work. It works, and would be fine with rewritten logic.

Q. Specific tuning? He showed an example of actively changing the GC parameters for a Seaside app. The result was a 15% improvement (167 seconds v 190 seconds for a given benchmark test of tasks).

John's slides list some added GC commands and new work planned. Time prevented him from talking about the forwarding logic but his slides are on the website.

Q. Any plans to introduce parallelism in Squeak? Do you have a million to spend. Seriously, you need to fund a couple of people for a while. This will have to wait until a real funded customer appears. Parallel algorithms are hard to do. VW looks parallel but is really incremental. The OldSpace collector marks 20,000 and then sweeps them and so on. Some Java implementations claim to be multithreaded but people say their performance is dismal when stressed, which probably indicates that the algorithms do not work well.

Q. (Mike) Where do you hold these VM stats? I just write them to a file and then produce these charts from Excel. You could write them to a socket and do remote diagnostics.

Q. (Mike) Effect of collecting on collection? He has set it up to capture data every tenth of a second without (he believes) affecting the application.

Q. Finalisation problem on fonts? Objects were dead in OldSpace for a long time but not finalised till a full garbage collect occurred which might not be for an hour. One case was a program that failed to keep a strong ref to font-rendered stuff so it lived for ages in OldSpace and rendered characters fine then good GC was done and suddenly it ran 50 times slower. Fix was to keep that string reference (done a week ago he thinks).

John stressed that any Squeak Mac problems he is happy to hear about. Core dumps are acceptable, especially with full descriptions of what was being done when they happened.

Design Discussion

Collections in Meta-Data Frameworks

I presented the problem of handling the collection hierarchy in metadata frameworks.

A typical metadata framework will have the means to build graphs of objects and to walk these graphs for various purposes. These graphs are built from instances of subclasses of a metadata class (call it 'Frame' for discussion purposes). This class may have a class instance variable (call it 'meta' for discussion purposes) which holds metadata information. This information usually includes a dictionary whose keys are instance variable names and whose values are objects that control the code that walks the graphs: is the instvar persisted, what values can be added to it, etc. All objects in a graph, except possibly for some leaf nodes, are of kind Frame.

InstVars of subclasses of Frame are controlled by their class' metadata (call such controlled instVars 'slots' for discussion purposes). Class Frame itself may have instVars that are used in constructing and walking graphs; for example, the graphs may have a tree structure defined by a 'parent' instVar of Frame. Frame's instVars are part of the framework-defining code, so are exempt from the metadata rules that control the slot instVars of subclasses.

The problem I wanted to discuss arises when you want to use collections in the slots of your metadata framework graphs. The graph-walking code expects every object it finds in a slot to understand the whole meta-data protocol, for which it requires Frame's class-side and instance-side state as well as Frame's methods.

- A common solution is to clone the whole collection class hierarchy as a subhierarchy of Frame. This is not good. Typically, the initial clone replicates only the classes thought to be needed. Later, others are implemented hastily and poorly (as the initial lot may have been). The two hierarchies get out of synch as the Smalltalk dialect upgrades, but there is rarely time to port this to the metadata duplicate. Divergence causes irritating protocol differences and further cloning of code. Poor or old cloned implementations cause performance problems.
- A solution I have used is to reuse the `Object subclass: #Class` pattern. Instead of cloning the collection class hierarchy under Frame, clone Frame under each concrete subclass of the collection hierarchy that the meta-data framework wishes to use. Put any collection class methods that need overriding (e.g. `copyEmpty`) into an extension of the cloned class. This arrangement is a kind of poor-man's mixin. It lets vendor upgrades slot in much more naturally. It minimises real code cloning; since class Frame is under your control, you know about any changes to it and simply replicate them.

Discussion raised several solutions:

- A true mixin implementation exists in Squeak; it takes 5 lines and should port easily to other dialects if not already done. Use this with Niall's solution.

- Terry Raymond has implemented Traits with state in VW. Use this to extend the collection classes (or trivial subclasses of them, as in Niall's solution) with meta-data abilities.
- Give Frame a single subclass CollectionFrame that encapsulates a collection. Give CollectionFrame the Roel / Blaine Buxton behaviour that a DNU is forwarded to the collection but also (if understood there) is statically compiled into a delegation forwarder.

I like the first solution. The second is good if the Traits implementation is performant. The third is good provided delegation is the right answer in general. It adds a message send to all metadata framework collection handling so has a slight performance impact.

Collection Evaluation Protocol
Someone wanted `#(1 2 3) sum` to return `6`, `#(1 2 3) text` to return `'123'`. Reify so have CollHandler text: myColl, CollHandler sum: myColl. You could give your collection subclass a type and use DNU plus `Type subclass: IntegerType` to which you forward. He wants at the moment one collection type and many types of objects in collections. Their current solution is `GCollection subclass: #IntegerGCollection` and `GCollection new add: 1` causes an empty collection to be constructed and then to `become:` an `IntegerGCollection`.

Reinout and Niall suggest dispatch to first element with default values for empty collections. Roel and others recommended reifying the messages to objects; it will be easier in the long run.

Design heuristics
Hernan Wilkinson has some rules he uses while designing and would welcome review of.

- Use immutable objects as much as possible
- Validate all new objects when created (c.f. contrasting case of VW date where you can change the year to anything just by resetting it)
- Wrap old objects in current objects rather than changing them

Reinout remarked that he uses flyweight for dates so these are good rules for these situations. You can use these rules in maximum table sharing situations. Katerina mentioned monads, Roel gave the book reference. As soon as you need to share, you need to synchronise and then the model becomes complex. Hashing is about storing things based on a value (must be strictly nested hierarchical model). Get a ref to the same object via flyweight. Hernan noted that a later deal is not the same as earlier deal. These rules are not just for flyweight identity patterns.

Christian Haider said these rules simplify systems when applied to objects that interface between subsystems: they make testing the two sides of the interface very easy. He uses them and will give a talk on it in a later year.

Issue of validating when the context is needed, or when e.g. you create three objects in a collection or in the slots of another object and only then can sensibly validate them.

Equality in Serialisation

Lucas Renglii would like a way to implement = when there are no real natural values to compare to. His WebDescriptions live in a dictionary where they are the keys. Identity works fine but if he stores them to disc and restore them then he must have =. Reinout suggested interning your objects when you read them in after unmarshalling them from disc.

```
Set>>intern: anObject
```

Suppose the description objects consist of two strings and have been sent out on disc. The two strings will be mapped to their unique strings in the set and so the Description will be uniquely identified. This only works with acyclic structures. If this will not work, e.g. because you have changed a label, you must use identity.

Niall: use model location. Read them in, accepting they will be different briefly, and then walk the model, unifying any descriptions that are pointed at by identical model objects.

Immutables and Copy-on-Write

Petr had a copy-on-write issue. He has one million ordered collections sharing an immutable object. Reinout: there is a special parcel that makes it possible to share immutable data bits, designed by GemStone and given to Cincom. Ask Eliot for it. Be aware it is experimental: Reinout has found that the debugger will lock up more often than before with it loaded.

Protecting self

Petr also wanted to catch references to some kind of protected object, so that self was always accessed through some wrapper. You could use `yourself` or method wrappers or subclass via a lightweight subclass (Hernan suggested this, c.f. John Brant's 'Unique Instances' paper).

Lastly we reviewed the discussion. It was thought very useful. It was suggested that someone should chair and determine who speaks.

ESUG Innovation and Academic Track

ESUG Innovation Award Demos

During the beer tasting, the eight entrants demoed at tables scattered throughout the hall, after which we dropped our ballots in the box giving our choices for first, second and third-best apps. (I note the person demoing when I visited each table.) As I was wandering from table to table with others, joining demos half-way through, etc., I could not take notes on my laptop so the notes below are from memory.

Promoter, Noury Bouraqadi

Promoter is a Seaside app that lets you register your site with search engines. It is easy to drive so users (like me :-) who know nothing about registering with search engines can make it work.

WikiDoc, David Gorisec

This is a documentation system. To all the usual wiki capabilities it adds a range of useful features: support for cross-references, etc.

Intensional Views, Andy Kellens
(See Kim Mens' talk for a detailed overview of this system.)

When you browse an individual method or class in a classification, the StarBrowser shows a Refactoring Browser with it selected. I suggested to Andy that when a whole classification was selected, it should show a (spawnable) browser on the environment containing just that classification's methods and/or classes. This in turn could be the basis for using the StarBrowser as a tool for composing environments on which to open a RefactoringBrowser or RewriteRuleEditor. Andy agreed that this made sense and noted the idea for future work.

We also discussed saving extensional classifications built by users while doing tasks. Classes and methods are easy to save. Instances may make less sense to save and reconstruct. A user might be willing to live with saving the classes of instances or some other simplified memory-trigger of them.

RoelTyper, Roel Wuyts
This year's winner. See Roel's talk for details.

SqueakSource, Lucas Renglii, netstyle.ch
SqueakSource has some 350 projects, probably indicating a thousand users. It provides a web interface to Monticello. People and projects register themselves, gaining keys they paste into Monticello to get access.

BottomLine, Jim Robertson, Cincom
The posting tool for Jim's blog application, now a stand-alone tool usable off-line and with WithStyle-empowered WYSIWYG editing. Jim also presented the BottomFeeder and Blog apps. While demoing the referrers-tracking system on his Blog, a delighted Jim suddenly discovered that he had made Feedster's 500 list today (at position 335)! From that moment, I think he hardly cared whether he won the award or not; he had just won something far more valuable to a blogger. :-)

From small beginnings this has grown to a very impressive system, and not just technically. If Jim had thought for a year, he could not have made a better choice than this suite of apps for something to build that would empower a product manager. It's a communication tool. It gives him the valuable experience of being a customer of his own product. It is a Smalltalk product that does something that vast numbers of people who've never heard of Smalltalk want. It gets compared to rival apps built in other languages in all the web-geek magazines, consistently scoring better on supporting even more standards variants, etc. It plugs Jim into the growing world of blogging, and because that world is exploding and he got into it in what was still its relatively early days, that gives leverage.

Of course, Jim didn't think for a year. He just likes to rant and has the self-discipline to rant constructively, so was a natural to notice the blog-world early and choose that for his application.

Moose, Tudor Girbe
Moose is a code analysis system and metrics tool implemented in Smalltalk, integrated with the StarBrowser, able to analyse Smalltalk, Java, C/C++ and COBOL. It presents the structure of the software graphically. Classes are shown as rectangles, larger if they have many methods, tinted to show few or many lines of code, coloured to show how a class evolved over time (extracting information from Store).

AmlTalk, Jean-Francois Perrot
This Seaside-based system allows ecologists to define ecosystems and simulate effects, interacting with them.

ESUG Research Track

Open Aspects, Robert Hirshfield and Stefan Hanenberg
Robert and Stefan have extended the AspectS system. In standard aspect systems, the base model plus the aspect model produce the overall model which is then compiled. In AspectS, an Aspect class might have a method returning the result of

```
BeforeAfterAdvice>>qualifier:pointcut:beforeBlock:
```

(all taking blocks as parameters). You `install` / `uninstall` an aspect, just as you would a method wrapper, to modify / restore the base model.

If you subclass or add a method after installing an aspect at a pointcut, the old implementation of AspectS, in common with many other aspect implementations, would fail to instrument the new code (so in the example in his slides, `mouseEnter` and `mouseLeave` would not be caught because they were not present at weaving time). Unweaving is similarly not done when needed. OpenAspects fixes these. Another benefit is that changes to aspects after they are installed (e.g. to add a counter) take effect in weaved systems.

Robert then looked at adding components to weaved systems. Most systems require intervention in such cases. OpenAspect supports policies to determine if components are weaved, left unweaved, or prompt the whole system being unweaved. Removing components similarly is policy controlled: do nothing, withdraw aspect from whole system, or prevent removal. Changing the aspect pointcut expression is also policy controlled. OpenAspect uses a dynamic conditional model composition of the base model, the aspect model and the adaptation model.

Implementation: The weaver takes the BlockContext, the AdviceQualifier and the Receiver's class and selector. Thence it populates the method dictionary with method wrappers that execute the advice behaviour. Previously, every installation of an aspect executed the pointcut expression which placed the method wrapper in the system. Now we remember where these wrappers were placed in the Adaptation and monitor (via system change events) differences between the current pointcut evaluation and that we obtained at weave-time. Any difference prompts a policy-controlled corrective action such as prevent, unweave or allow.

They have also improved the weaver.

Q. How to decide what policy to use? Application dependent. Default should be system always matches current pointcut.

Q(Niall) is security one application? That was the original motivation of the work. Prevent key code being uninstrumented, ensure the system remained secure as it evolves, etc.

Towards Unified Aspect-Oriented Programming, Noury Boraqadi, Djamel Seriai, Gabriel Leblanc

Noury summarised aspect-oriented programming as a technique of coding orthogonal properties (functional, logging, security, etc.) separately, the code being integrated ('weaved') by the aspect system. The application core is the unaspected functional code. Aspects crosscut these core classes by altering the application structure (add new classes and methods: static cross-cutting) or execution flow (change e.g. message dispatch: dynamic cross-cutting).

AspectJ has serious limitations. They provide different constructs for the two kinds which causes unwanted complexity and also seriously obstructs reuse. It also fails to handle all possible conflicts between aspects.

Noury explained mixin-based inheritance, an alternative to multiple-inheritance that avoids the code duplication of single inheritance and the problems of multiple inheritance. His implementation generates implicit classes into a Smalltalk single inheritance tree. The developer sets the order of mixins which becomes the order in which implicit classes are inserted into the single inheritance tree.

A reflective language gives access to its own semantics. The program handles the base level, the evaluator handles the meta-level. A meta-object gives access to base classes. He showed a 'noury' instance of class Person with method `sayHello`, and instance logMetaObject(#sayHello) of LogMetaObject recording the execution of `sayHello` on instance 'noury' of Person.

With this terminology established, he explained two Aspects X and Y, with Y having one Mixin and X two. For each base level class, the weaver applies the mixins. Thus base class C has mixins X3 and Y2, and 'C meta' has Mixin X2 and Mixin Y1. Instances c1 and c2 of C have metaobjects 'c meta 1' and 'c meta 2', both instances of 'C meta'. (Looking at the slides' diagrams while reading this paragraph may help. :-)

The weaver puts static crosscutting at the meta level and dynamic crosscutting at the base level.These unified aspects are application-independent. Weaving evaluates preWeaving scripts, does the standard linking of mixins to classes and then runs some postWeaving scripts. Because the two kinds of crosscutting use the same approach at base and meta level, reuse is as easy (much like moving code from class to instance side). Mixin conflicts are handled by the ordering of the mixins.

They need to study whether evaluating all pre/post-weaving scripts before/after mixing mixins is sufficient or needs to be more configurable. They have built a framework for weaving; they must marry it with an expressive language for weaving. They should look at persisting choices across application rebuilds. They want to think about finer-grained weaving at the method and instance variable level.

Q. Why mixins, not just extensions? Two reasons: you may need to change the class structure (i.e. add instVars) and you may want to handle conflicts by ordering; extensions cannot be ordered.

Q. AspectJ versus your system? Noury has examples of how reusing even simple aspects quickly becomes complex in AspectJ.

Q. (Remark: some people in Bonn are trying to extend AspectJ with a logic language to solve the reusability issues.) Granularity? Tom Mans has a very good paper on mixins, encapsulation and granularity.

Q. Traits work suggests that ordered composition is not enough for 'classical' use of mixins? Alex is working on inserting state into Traits whereupon Noury may switch to using it. Until then, he must use Mixins to insert state.

Q. Scripts? Standard smalltalk code.

Inter-Language Reflection, Kris Gybels, Maja D'Hont, Stephane Ducasse, Roel Wuyts

Inter-language reflection is a combination of traditional reflection and linguistic symbiosis, the latter meaning that two programs can manipulate each other as data. Work has already been done on each of these at Proglab, e.g. in SOUL. The new work applies them to maintaining causal connection.

The motivation is that ProgLab has been studying writing declarative programs and meta-programs for some years. There is no need for the base language manipulated by the tool to be the same as that of the declarative meta-program. Tyruba is a logic programming language for reasoning about Java. It parses Java programs into a set of logic facts. You can then write rules to extract more information, e.g. to recognise a double-dispatching pattern. Because Tyruba is not Java, there is no causal connection between the modelling program and the base program, unlike in Smalltalk where the two are the same.

Their goal is to maintain the causal connection while still using different programs for base and meta representation, and ideally to let either manipulate the other.

In SOUL, causal connection was maintained by rules that exploited Smalltalk's reflection. He showed the code, which had a key piece of reflection-exploiting Smalltalk embedded in its logic rules. Thus SOUL and Smalltalk are each individually reflective and you have a protocol

mapping of meta-representations between the two.

SOUL's model was applied in ProgLab's tools RBCL (for C++) and Agara (for Java). Agara is a prototype-based language unlike Java. He showed code installing a listener on a Java object in Agara.

Maja has worked on applying logic rules in SOUL, for how customer loyalty is mapped to discounts, to a Smalltalk application for managing purchases. She has Smalltalk code (to discover if the customer has a store charge card) within her logic rules.

To achieve the same generally, they need Java objects to appear in Agara and conversely. This is done by a common meta-level language that has two meta-level interpretations (one for Java, one for Agara) and a protocol for mapping between them. However message sends in Java are dispatched on an argument's static type, unlike in Agara. They dealt with this by letting Agara methods include argument types in their name. A second issue is that travelling to Java from Agara and back should return the unwrapped meta-object, not a double-wrapped one.

In SOUL, Smalltalk objects are unified if they are equal in Smalltalk's semantics. SOUL is implemented in Smalltalk, which therefore acts as the common meta-level but this can cause confusion. In Java, you don't have access to the full meta-level so you must first wrap the base-level object. This wrapper maps base-level Java objects to meta-level Agara operations (i.e. you get a message send to an object, not `object send: #message`.

Q. Relationship with CORBA, .Net? CORBA and .Net choose one language to be dominant and all others must map to this. It must also be done on the base level since they have no idea of reasoning about meta-level differences.

Q. Java's limits mean you in practice make Agara dominant? Yes, Agara to Java is a map down from meta to base. The clear separation in Agara between base and meta keeps the overall mapping clear.

Q. The Piccola language (maps from Java) uses optimisations; could your approach reason about these? Discussion.

Q. Writing business logic in a logic language is attractive; how would the idea scale? Ask Maja. The work aimed to improve the SOUL-Smalltalk linguistic symbiosis so if later you decide your logic will be better in Smalltalk, it is easy to take the rule to a method without rewriting anywhere it is invoked.

Runtime Bytecode Transformation for Smalltalk, Stephane Ducasse, Eric Tanter, Markus Denker

Transforming bytecodes can let you adapt your system to e.g. a new platform, trace it for debugging or add new language features. Smalltalk lets you add methods and classes at run time but you have no means of changing a method body other than overwriting the whole. ByteSurgeon

lets you change parts of methods. It runs in Squeak and should be easy to port.

Changing bytecode means

- you don't need to change the VM
- you don't need the source
- you could operate on other languages that compile down to Smalltalk
- it has better performance by minimising what is changed to just part of a method: their benchmarks show excellent performance.

He showed various examples. The simplest was just to add code to the send, e.g.

```
(ExampleClass>>#aMethod) instrumentSend:
  [:send | Transcript show: send selector printString]
```

You can also `insertBefore:`, `insertAfter:` and `replace:`. So using `insertAfter:` counts only sends that actually complete. You can concatenate strings to construct code but it is ugly so they wrote a simple quasi-quoting method `insertAfter:...using: aNameDictionary` to get the bindings of variables in the method.

ByteSurgeon provides meta-variables to let you access vars on the stack, e.g. the receiver is accessed in

```
instrumentSend:
  [:s |
  s insertAfter: 'Logger logSendTo: <meta: #receiver>']
```

Smalltalk has scanner/parser sending AST to a code generator. Squeak has a two-phase code generator that translates the ST to a simpler intermediate representation from which code is generated (this IR could be the same in VW and Squeak). They worked on the IR, which was much simpler than the AST would have been.

MethodWrapper with ByteSurgeon is 6 times slower to install but 40 times faster to execute.

Marcus showed a simple meta object protocol, written in ten lines of code (see slides).

ByteSurgeon has shown that bytecode can be usable but ASTs have some nice properties; they want to review the level at which they act. A project to implement a very fine grained meta-object protocol is ongoing. A newly started project is the OmniscientDebugger.

Q. Make method wrapper available with same public protocol (for apps where you want to trace without hitting performance)? Can do.

A New Object-Oriented Model of the Gregorian Calendar, Hernan Wilkinson, Maximo Prieb, Luciano Romeo
Hernan teaches Smalltalk at the University of Buenos Aires and also works

for Mercap. Time is key in many domains. The Gregorian calendar is the one used in the financial domain. The irregularities of the Gregorian calendar (month lengths, leap years) cause many problems. You need to manage time precedences, lengths, and subintervals. You need to distinguish kinds of time: working days is the most obvious. He demoed by executing various time expression examples.

The existing applications have problems. Smalltalk80 uses symbols for days (so #monday > #friday). In VA, everything is a calendar and it's very hard to see what you are dealing with. (Java is worse; he read the whole documentation for the Java calendar class and could not find how to create a date!) Squeak is best but uses too few too powerful classes: for example, you can ask today how many days it has.

Their metaphor starts with time's hierarchic granularity: GregorianYears have GregorianMonths have GregorianDays have DateTimes. These are all subclasses of PointInTime, a subclass of IntervalAwareMagnitude. GregorianLeapYear is a different class to GregorianNonLeapYear and that is how they handle irregularity. DateTimes simply encapsulate a date and a time.

Some time are circular, e.g. the months go from January to December thence to January and so round again.

Q. Days in February in leap year? The instance of the month February double-dispatches to its year.

Measurements model distances between time. Measurement has an amount and a unit. He selected various expressions to demonstrate.

```
(January distanceTo: February) = 1 month.
(January 2005 distanceTo: February 2005) = 1 month.
1 year + 6 months = 18 months.
5 days + 3 weeks = 26 days.
1 day + 1 hour = 25 hours.
```

By contrast, `3 months + 5 days` returns a measurement of these two incommensurable entities. `0.1/(1 year) = 0.0333333 months.` Their base calendar can then be subclassed to add more units such as Decade.

They have MCPInterval. (Q. Why not TimeInterval?) They model the beginningOfTime and theEndOfTime which are very important (but he has not the time to explain why :-).

```
holidays := TimeLineView
  addDayRule: Saturday;
  addDayRule: Sunday;
  add: August first;
  ...
```

They can filter points on timelines, apply them to intervals, etc.

They need to expand their Timespan protocol. There is no singleton pattern yet; they know they will need that in GemStone.

Q. Kent's paper on multiple timelines to handle the time-related errors and corrections and putting-retrospectively-in-force of corrections that real systems need? Yes, multiple timelines is something they will have to model.

Q. January special? Yes, the circularity is better modelled with a special class.

Q. Locate problems: e.g. first day of week, time zone? You can set which day of the week is first. They have not yet completed TimeZones; should a day compare equal to a day in a different TimeZone or not?

Q. Beginning of time? It is a transfinite, so will always return itself to operations, as for Infinity.

Q. Available? They are considering making it open.

Towards a Taxonomy of Unit Tests, Markus Galli, Oscar Nierstrasz, Software Composition Group, Bern
Markov worked for 6 years in OS Smalltalk programming, where it annoyed him to have two browsers, SUnit and the class browser where the code being tested was displayed. More seriously, the test is easy to write but the scenario can be hard to set up. His goal is to make it easier to navigate from tests to methods, and to compose tests into test scenarios.

Romain Robs wrote a Squeak BrowseUnit tool that lets you navigate from a method to all unit tests that call it. It is good to have but will often show tests in which your method is being used as a given, not really being tested.

He starts by classifying tests into those focusing on a single method and those which do not. The former split into those that don't expect exceptions and those that do, and he splits the former into one-method tests and what he calls test suites [Niall: seemed a non-standard usage of this term to me] that have many asserts. 53% of Squeak's tests check exactly one call of a method under test. The latter he splits into optimistic and pessimistic tests, based on whether they seem to expect success or not.

His multiple-method commands split into those that apply the same scenario to a range of commands, and the others

Multi-faceted test suite: Constraint tests look at the interplay of methods, no one being the main subject.

Cascaded tests reuse the results of one test in another (e.g. test encoding works, then that decoding returns original).

He noted that tests can return results [Niall: mine often do]; VOID has no meaning in Smalltalk.

He showed a five-pane Squeak browser of methods and the tests calling them. (Delay here while he tried to reset the screen to have a small enough

resolution to show it.)

Q. Use dynamic coverage? He does (important point)!!! Thus his browser is showing tests that exercise a method, not just that call it at top level.

Co-evolving Code and Design with Intensional Views, Kim Mens, Andy Kellens, Frederic Pluquet and Roel Wuyts
The tools were demoed in the awards track. This talk focuses on concepts. Intensional views and relations aim to document the high-level source code structure in an active way, i.e. you can run the intensions against your code. They did a case study: document initial version of system, rerun after small evolution of one month, rerun again after large evolution of one year.

Typically when you want to change a system you want to focus on a subset of all application code. Roel's StarBrowser lets you define intensional and extensional classifications. Suppose a bank application has the idea of BankCards and BankAccounts. You can put all subclasses of BankAccount in an extensional view or define an intensional view of it (more robust). SOUL (see talk above) can be used to define intensional views. You can define different definitions that you think equivalent of a given intensional view and check they are extensionally consistent (i.e. that they find the same contents).

Your views will be related. All Account-changing methods should call persistence methods for example. This can be expressed as a logic relation using the standard quantifiers (for-all, there-exists, plus some they added: most, a-few, etc.)

Their toolset integrates seamlessly with VW7.3, using the StarBrowser, SOUL and Moose. Tools let you define your classification relationships, see which relationships break (like a unit tester). Colour-coding sorts derived, user-defined and broken.

SmallWiki was their case study.

- A manual code review, recording all interesting classifications they noticed, had 15 views, 15 nesting views and 17 relations (and revealed some small bugs, inconsistent protocol uses, etc.).
- After a month, there were several red lines. Some were just renamed classes or new classes. Sometimes exceptions they had flagged were fixed and so now the rule showed red because the exception was now in the view.
- After a year, it was much the same, probably indicating that SmallWiki was stable and well tested.

It showed them what had changed.

Why use SOUL? He showed an example where the SOUL was 3 lines and the Smalltalk was 17 lines. Mixing the languages worked best.

They are thinking of using line thickness to convey information, e.g. how important is this relationship. They want to mine source code for views.

They want to understand how scalable the tool is; will it be better to look at subsystems of a large system?

Q. Can I version my views in Store? Yes they are stored in class methods, so can be put in any SCM system.

Q. Can I put views in views to handle my large system? Yes, he thinks that can be done. You'll need mining to handle a large system.

Q. How to mine manually; do you have any rules? They are thinking about mining so have not yet written up a methodology.

Q. (Niall) There is an alternative to mining source code. Test-driven coding helps the user code faster up-front and incidentally builds up a refactoring suite. Similarly, the StarBrowser lets the user grab classes, methods, objects, tests, etc. while solving some task. If you provided an easy way to save such extensional collections, or parts of them, under memorable names, the user would be grateful: a month later, when doing a similar task, they could recover the classes and methods they found relevant last time from the StarBrowser, not from their memory. Meanwhile, you would have a set of saved collections that would be the basis for intensional analysis. So what do you think of mining user actions? Yes, some work has been done by Andy Murphy and others (using their tools, which are similar).

Microprints: a Pixel-based semantically-rich Visualisation of Methods, Romain Robbes, Michele Lanza and Stephane Ducasse
Romain considered human limitations to visualisation techniques. A picture may be worth a thousand words but when starting to look at an unknown code base, a UML diagram is often worth rather fewer. Syntax highlighting can also be used and Moose provides a variety of polymetric views using size shape, connections, etc., to convey much data.

Humans impose limits. Colours cannot be too close to each other or too numerous. Past a certain point, too many annotations (strikout, underline, etc.) start confusing rather than helping. The microprint approach assumes that it is easier to merge information rather than to sort it out. Thus you should offer many code views (variable access, control flow, object interactions, code coverage, etc.), not a single much-annotated view. Since all the views now risk becoming excessive, you have a main view and many stacked reduced-size views.

He showed a browser of the main pane and views stacked in the RHS code pane. The stacked views were massively reduced in size but colour-coded and preserved the method's format shape so some understanding of where the colours were in the main method was possible (c.f. thumbnails).

Romain showed some example. The lazy initialization pattern can be easily spotted in the microprint, as can ordinary initialization, early guard clause, and (probable) pure utility method.

Alternatively you can look at microprints from CodeCrawler (usual demo

hiccough; not on right screen to invoke at first), seeing the microprints within CodeCrawler boxes. This can help to spot duplicated patterns or maybe code.

Implementation: microprints use the RBParser, the RB's AST and their visitors. Each microprint is a colour-to-mapping marker. Markers are subclasses of RBProgramNodeVisitor that mark nodes matching their criteria with the appropriate marker. Sometimes a node will match two marking criteria so they must have a priority. It is easy to create new markers and `self mark: a Node` or `self mark: aNode value: v` (the latter capturing an integer metric).

Q(Joseph) Visualisation needs standard code formatting. Do you do this? They keep the method's format but do not reformat at present. Joseph learned from Kent to reformat code when mastering an unfamiliar code base.

Q(Christian) Easier to merge than to sort; is this true? Stephane has asked this question of himself.

Q. Markers definable only for individual nodes or for patterns of nodes? That is possible (RB nodes know their adjacent nodes) but it has not been done. You can also find larger patterns using the Rewrite framework and he has coded the means to do this.

Q(Niall) Agree with Christian in general but there will be tasks where the side view of e.g. variable access will be valuable. If it's an optional non-default code tool in the Refactoring Browser, we'll find what it's good for in time.

Other Discussions

Mercap uses single-access merge integration procedure; you have to get the token and then you can merge. This resembles Lifeware's remotely-accessed integration machine; only one person can share to it at a time.

John McIntosh described recent work for a client in Ottawa. His client had 300Mb images; John realised that perm-saving the image would reduce the amount of fresh daily loading from 6 million objects to some 300,000, by doing some timestamp comparison. This saved 5% of startup time which was worth doing. The client stripped their image because ParcPlace had required them to do so years ago; they were pleased to start just hiding the development tools (until the developers needed them to debug a production problem). The client uses the old trading software from integral solutions (the company that turned itself into a fossil by obeying its investors' and managers' decision to switch to Java; see my ESUG2000 report).

Christian Haider has been contacted by Romax, who were pleasantly surprised by his not-too-alarming estimate of what a port from VSE to VW Pollock would require.

Andy Berry is working as a technical architect.

Alfred Wulshlegel has back-ported code to allow GemStone 64 to run with his VW3 image. Swiss National Bank's performance limits are all on the GemStone side, not on the VW side, so they feel no need to upgrade from VW3 yet; other tasks take priority.

Ian Prince, like many, was forced out of Smalltalk in the mid-90's. He got back into it by discovering Smalltalk wikis, and then Seaside.

Through pairing with Avi in Camp Smalltalk, Mike Roberts has gained insight into Seaside idioms. He will use it for some internal webapps, and present on it internally.

Maximal Usage of Tree Structure, Alfred Wulshlegel, SNB
Alfred gave me an overview of a tree structures utility used in many places in his system. The system defines forms saying what data is of interested and how structured. Messages come from banks supplying this data: state of current accounts, deposits, etc. The archive has 1.5 million messages.

Open message to add rows. One observation is one coordinate tuple with typed value (types are number - no float/int/... distinction - enumeration and date). He showed a Form with 80 observation positions (defined data values) in its matrix and an example message with 36. They will not accept messages that do not match the data model defs (i.e. the defined positions).

The Message object is the basis (and was defined long ago). Alfred brought up a new tree (folder-like icons). The second tree was a tree of forms; double-clicking opens a set editor. This has two panes: above has time intervals; below, when interval selected, you get the data for that period e.g. what form must banks use for reporting during a given time period. The row between varies according to type of data e.g. showing columns.

The system knows nothing about the economic meaning of the data it manages: a number is just a number and a tree node is just a tree node.

A second dimension of data is who is obliged to supply data (all Swiss banks). You can expand an interval to show what has changed in a time period (how many banks there are, which have been added, which have disappeared, and then can see what those banks were). All things are set-based so users never have to learn new ideas of how to browse.

Third dimension: when must information be supplied. Legal requirements keep that simple (20th of month, every 3 months, biannual, etc.) plus some special ones (e.g. end-April three years after 1998).

These are put together in erhebung (inquiry), of who, when, what (definition) (date, subject, form). He showed a period change; 6 month reporting changed to 3 month reporting.

Subject is implemented as a tree structure (composite pattern). He showed the tree structure and the form generated from it (laid out as a typical form). Tree structure elements know whether to display themselves or their

contents. Paging is done so that a form can cover multiple pages (to lay out all elements well). There are also other browsers to see the definition of each element of the three in the RHS. He selected one leaf element and showed the dynamically-constructed form for just that element. Colour coding shows the types of elements, and icons show e.g. whether they can be temporalised (vertical bar) or are temporal (horizontal bar). The periods of invariance are shown (for tree and for selected subtree). You can supply a date and see the tree at a given date.

He showed a node that could be added to (red colour) and clicked to add. It provided the list of what node types could be added to this type. He tried to add one to trigger a warning ('instance already supplied of that one-only type'). Shadow icon means 'structure can be duplicated'.

Trees can control the modification of trees into other trees. He opened the configuration editor (only for power users). The tree of this editor contains a range of tree-homomorphic structures. He opened one of three elements that were extracted from the tree. Thus it both presents certain attributes and adds data to those attributes. A typical example would be exporting to a relational database; the homomorphism selects the elements to export, the names of the tables to export them to and the grouping, etc. Generating SQL is then trivial.

From time to time the customer wants to change the form, not just for new banks but for all banks, and they want to look at past data in the new form. Banks vary in how much data they must supply and therefore what tree they have; large banks must report much, smaller banks need report less so have a simpler tree. The tree defines each bank's reporting requirement (the law classifies banks to define what they must report). The application in which he first developed this utility (for the Swiss police) had 150 templates. For banks there is essentially just one but some subtrees don't apply to small banks so are removed from the trees of individual banks.

Delta trees morph trees by applying a user-created delta node that transforms all trees by adding specified new elements (or deleting - rarely used but done for removing titles). They use names (associations of affinities), not direct pointers, and the transformation nodes hold the names of the nodes they will transform and are underneath their clones in tree. The presence of a node in a delta tree tells it that all trees with that node will be transformed. The children know which node to transform by name reference, not by being children, but having them there makes them easy to find, so it is easier to leave them in that structure.

He then showed using a tree to define what a tree was, thence generating a form, thence creating a tree. They have used this to define which configuration was in production by extracting the current production to the form the configurer was editing. Their configurer has said it was a huge relief to *know* that he was working from the current production state.

Alfred opened the nodes field layout form and edited it (based on a grid of 80 character positions per line and changeable defaults on when to have a

new line for new elements). Trees have RSSets wrapping them (see Alfred's talk at ESUG2002) so you have security on them.

Next he looked at trees which hold rules to verify forms. The system automatically generates a tree that applies known rules to form elements. He added a program node and thence created a tree with a data-defining node, an 'exists' operator node, and a severity node. The resulting form had a text field where you could say what the meaning of the rule was. He then generated a test tree. This added a node to the rule tree that could be run to show whether the tree was correctly constructed. The test tree can then run over real data of the appropriate time period to verify the rule is satisfied.

The final type combines subject info (a homomorphic tree) plus test tree to create a report.

To Do

- Ask Noury for mixin code in VW.
- Discuss full RB and Envy integration in VASmalltalk7 with Instantiations.

Conclusions

My seventh ESUG and a lot of fun. Having a large room that doubled for Camp Smalltalk and for coffee breaks, demos, etc., meant that Camp Smalltalk went well.

---

* End of Document *
