Smalltalk Conferences between June and September 2008

This document contains my reports of

- the Smalltalk Solutions conference in Reno June 18 - 21, 2008
- the ESUG conference in Amsterdam, August 25 - 29, 2008 (and brief info of the Camp Smalltalk 23rd - 24th and Seaside Sprint 29th - 31st)
- the VASmalltalk User Group conference in Frankfurt, September 23, 2008

I have combined my three conference reports into a single document. They follow in the order in which conferences occurred. An initial section 'Shared Keynotes' gives two talks that were given at both ESUG and Smalltalk Solutions.

Style

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

Author's Disclaimer and Acknowledgements

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

Shared Keynotes

Gilad's talk on Newspeak and Georg's talk on Bach were given at both Smalltalk Solutions and ESUG. A single write-up combining material and questions from both conferences tells the story better than two write-ups.

How to find the Bach house in Cöthen, Georg Heeg
(This combines the talks Georg gave at Smalltalk Solutions and at ESUG. Questions asked and answers given at either venue are included. Strictly, Georg's talk was not billed as a keynote in either conference, but it presents Smalltalk in a context of much wider interest, so I think it is worth highlighting here.)

Dinosaur bones have been found in Cöthen, and evidence of stone age occupation. More recently it was the capital of the independent state of Anhalt in Germany; there are city records going back 900 years. It was also the location of the second largest ESUG (in 2004, eclipsed only by this year in Amsterdam).

Johan Sebastian Bach worked for 7 years in Cöthen, during which time he wrote 'The Well-Tempered Clavier' among many other works: Georg played us an excerpt from the 10th piece. Prince Leopold of Anhalt was the reason Bach lived in Cöthen for many years. (Unlike those princes who just had parties every day and spent all their money) he was very musical, invited Bach to the city, founded societies for the study of the German language and so on.

Even during his lifetime, Bach began to lose popularity and only 100 years later did the superb quality of his music begin to be recognised. Thus there are many open questions about his life, including where he lived in Cöthen. Today the city wants to improve its tourist trade by finding all it can about Bach's time there. Their first question was, 'Where did he live?' They got EU funding for a suitably long-winded title (Feasibility study for ....) and asked for bidders; Georg Heeg won. The bidding started in 23 March 2006; Georg Heeg completed the task on 15th February of this year.

They did it in 4 phases as an agile project. They knew from the start that if they used the same approach as all the historians had tried - looking for a piece of paper with an address on it - they would fail: if such a paper existed, it would have been found. Instead, they intended to use exclusion as their proof method (c.f. the computer proof of the four-colour problem).

Phase 1 was drill down to brainstorm and identify key issues. Phase 2 was locating, scanning and OCRing all the old material they could find in archives, libraries, old bookshops, eBay. 736 documents in gothic, latin and handwriting fonts were scanned. For example, they copied all the city property records for 100 years. Phase 3 built the semantic network. There were two existing semantic network products written in Smalltalk. K-Infinity (see my ESUG reports in 2001 and 2002) is used by the police, publishers, etc. It had no interface to word. Atlas TI is an analysis tool for non-numeric data, but is not extensible (it easily could be but the company only sell the packaged software). Neither of these could be fitted exactly to their problem without some work.

They therefore created a specific semantic network solution GHBachNets. It provided direct connections between the automatic procedures and the human decisions. The UI was a web browser, using Seaside on VW and GemStone. Two phone calls to Monty improved performance from hours to seconds. The coder and researcher worked closely so UI ideas were implemented quickly.

The first thing they discovered was that the 18th century public treasurer was either a fraudster or very bad at arithmetic; they suspect the former but cannot pursue it as he's been dead for over 2 centuries. They also found lots of simple writing errors. To do the searching, they went back to Thorsten's JavaSpektrum tree implementation, porting it from VW to GemStone.

They soon realised that a semantic network concept of an idea or a Notion (Begriff in German) was naturally a class so they made it a subclass of ClassDescription (at first of Behavior but they found they needed the

subclass' abilities as well). Thus Notion (Begriff) is a sister class to Class and Metaclass (only Smalltalk can do this!). Special Notions are Values and Relations between two or more Notions. GemStone does not allow this extension of its metaclass so they had a mapping from a description of a network to the real network in VW.

Next they needed TemporalObjects. "Bach lived in this house" is not useful information without knowing from when to when he lived there. They created a time framework that could handle vague and precise dates and historical ways of dating. For hundreds of years, tax registers showed who owned a house and how much they had paid (but not where the house was located). They had old maps to locate things and they had a princely homage accession record which was a few years too early, but useful because it lists who rented as well as who owned. Thus they had a tax register of householders in 1710, and in 1765, and of those houses with the right to brew beer (most useful data - and doubtless a much-prized right at the time, too :-). He showed one page of the thick one-per-year book of which they handled 100 years-worth of books. Georg showed some of this information in an Excel table for a single home: house built in 1738, first tax record in 1741, relationships, graphics, etc.

All this was imported from Excel via COM connect (the VW7.6. implementation was driven by this project's needs).

Cöthen was in the postal system. Bach wrote from Cöthen to the city of Erfurt (120 miles, which cost 2 groschen which was about 10 dollars in those days); Georg showed a copy of the letter. As was typical for that time, it listed the title, profession and position of recipient and sender in great detail, but not the street address let alone a house number. Half of the houses in Cöthen were owned by the Prince and rented; the rest were owned mostly by their residents. They swiftly concluded that Bach neither owned a house nor rented from the Prince; those records are complete. The oldest city map dates from 1730, seven years after Bach left (and conveniently is oriented north, not always the case in those days). By comparing with Google earth, you can see that streets today are very similar to what they were then. A second map of 1778 also showed little change and was easier to work with. While Bach lived in Cöthen, the city was enlarged and its walls rebuilt to enclose a larger area. They started to build the new city wall on 24th February 1719.

We know that Bach must have had a spacious home. He had a big family and students lived with him, and he needed four big waggons for his goods when he moved (this useful fact is recorded in a humble newspaper of the time). They looked at all the houses that had been proposed. They also looked for any other homes that could fit.

The first candidate (Burgstasse 11) was built in 1721 (too late) and was too small (luckily, as it has now been knocked down and is a space between two very modern houses). The second was simply too small. A third one, Holzmark 12, is mentioned in a list of 1855 with the names Bach and Erhardt. However it had six or seven families living in several apartments

within it - not at all ideal for music practice all the time. Stifstrasse 11 has a sign ‘probable Bach home’ and famous historians have backed the claim but it is not so; the project systematically proved their arguments were wrong, and were a classic case of historians quoting each other so that an ‘astonishing’ document found by one such historian was in fact already well known and not persuasive. Essentially, a historian called Hartung just published a guess.

That left three homes, all owned by Johannes Lautsch. One was a shop, which does not fit music practice. Another was seventh in the property tax list (2 thalers and 9 groschen ~ $265), so one of the largest homes in Cöthen. There is evidence that Bach complained that the rattle of a water mill next to his home disturbed his music - he could only compose music with one speed. There is a water mill 100 yards from this house. Its previous occupant, Stricker, who was Bach’s predecessor as cappelle maister, had his rent paid by the prince and we know that Bach’s rent, also paid by the prince, was the same as that of the prior occupant. This looks like Bach’s first residence.

They also wanted to find where he lived next. Before 1719, Wallstrasse was an area of gardens and a gold and silver thread factory. On 25 February 1719 Prince Leopold decided to expand the city: he ordered Wallstrasse and Shulstrasse to be built as streets, and ruled they would be tax-exempt until all houses were built. It is not only in modern times that laws cause the reverse of what the government expected. This rule, intended to encourage building, had the opposite effect: the people who moved to the new streets had a strong reason to delay their completion. Only when the prince’s successor, his brother August Ludwig, ruled that only three years of tax exemption would be allowed were the remaining houses built.

A tax-exempt house Wallstrasse 25/26 was built by Johannes Andreas Lautsch. The house only appeared in the tax register in 1730 but it was built in 1719, in the middle of the time when Bach lived in Cöthen. (A claim it was built in 1712 was recognised as a misreading of 9 for 2 in an old document.) They concluded that Bach was the first renter of this house. When his landlord built a new house, Bach moved to it. Georg was very pleased to note that it was only two doors away from his own house. (A later owner of this house was Erhardt, which is where the 1855 confusion occurred)

He demoed (in Safari; they use some clever stuff for tooltips that not even Firefox supports yet). He showed the various lists (literature, names, etc.) and hints, thence navigated to schematics of the houses in their locations in Cöthen, plus tables of data on the houses.

Q. Were you surprised to find you lived so close to the second house? Yes. (As a side effect of the work, they also discovered who built the house that was on the site of their own house before the current building which is 1911 art deco. The builder was a doctor and Georg knows where he did his thesis, who he married, how he built the excellent garden, and that he was mayor of Cöthen at one point)

Q. Will you find the holy grail next? If we're offered a contract to do so. :-)

Q. How many people were employed? 30 part-time (this omits some of the data collectors). There were 6-800 pages of handwritten text to scan and the rest was printed in gothic or latin font.

Andrés mentioned Silverman, a friend of Bach who talked to the Italian inventor of the piano and built an early piano. Maybe Cöthen can get even more tourists if a connection can be discovered. Georg searched and found some 50 documents mentioning Silverman in their data.

Georg played some more music from 'The Well-Tempered Clavier' to close.

(Eliot) Is it Koethen (the spelling in all the ESUG 2004 docs) or Cöthen, the spelling in Georg's talk? The most correct spelling is with K (the word comes from a Slavic language) but in the nineteenth century, or when latinising, it would be spelt with C. The oe is just the way of anglicising an umlaut.

Q(Christian) Why were you unable to use the existing products (K-Infinity and Atlas TI)? We needed to customise to get our data in and we had to map between numeric and non-numeric data; those were the two reasons.

Q. Reaction of historians? Some say, "Yes, we've found it!" Others say, "We do not accept your methodology; you did not find the piece of paper we were looking for."

Tampering with Perfection: From Smalltalk to Newspeak; Evolving Smalltalk for the Age of the Net, Gilad Bracha, Cadence Design Systems
(This records material from the talk Gilad gave at Smalltalk Solutions and the one he gave at ESUG, including questions and answers from both.)

Gilad wants a language where everything is a message send. Not almost everything as in Smalltalk, but everything.

Relevant changes since 1984 include multi-cores and this web thingy that Al Gore invented or maybe it wasn't quite that way :-). We've learned that security matters, that modularity is crucial and that ecosystems matter (no he's not referring to global warming, that's enough of the Al Gore jokes) i.e. how a program fits into its environment.

Smalltalk was defined to be open and malleable so it is not that secure. the various, non-standard unsatisfactory solutions for modularity around today are also not that secure. Interoperability: why would you want to go outside Smalltalk? Sometimes there are good reasons, e.g. to get something current Smalltalk solutions do not do or do not do well.

Newspeak is a message-based language with no static state in which classes can nest.

Message-based: isn't Smalltalk? Well, Alan Kay says 'Smalltalk should have been message-oriented." The idea of Newspeak is that *everything* is a message. In `t := Array new: n.` only `new:` is a message send. If we rewrite so everything is, it would look like

```
self t: (self Array new: self n).
```

Self went this way but decided to have implicit receivers, so the above becomes `t: (Array new n).` That only works because of various constraints. Instead, Newspeak writes `t:: Array new: n`. Here `::` is a message send but with lower precedence so you can skip the brackets. With this, no code cares what class implementation you have chosen. In Smalltalk, most code does not care but your subclasses care; rework the superclass and you must refactor your subclasses. Here, you are talking to self so no code cares. Even the class itself does not care, i.e. its method code does not care.

Implications: access control starts to matter. In Smalltalk, all messages are public and you encapsulate by putting things in instvars or blocks. Thus Newspeak must apply some private/protected/public rules to messages that are replacing non-messages in Smalltalk.

Global state is a bad thing for many reasons. Gilad focused on one: security. Newspeak is not a secure language: it is a basis for building secure systems. If a party does not want gate crashers then either the person on the door can check if the person arriving is on the guest list or else they can ask them to show their invitation. In the programming domain, the latter approach is called capability in terms and maps well to object-oriented systems (described in a good paper by Miller in 2006).

Thus the object-graph provides the authority but this means no static state and thus no ambient authority. For example, you may be handed an object that knows how to open a given file; you are not possessed of ambient authority to open that file.

In Smalltalk, there's a bigger problem. You can look at an object, inspect its methods, compile new on the fly. This is one of the key things that makes Smalltalk wonderful but of course it makes it hard to argue you have a secure system. Newspeak uses the idea of mirrors that Self introduced (not for reasons of security but it works well for security). Mirrors are objects that reflect other objects but, like a real mirror, give no inevitable ability to change the real object whose reflection you see. Mirrors act as capabilities for reflection. They let you control what people can do and so you can design the security you need. Mirrors are also useful for deployment.

There is no static state, so how do classes share state: they use nested classes (as in Beta; forget Java's nested classes: this is not that!). In Newspeak, classes can nest. Gilad thinks this is an oversight in Smalltalk: if you push Smalltalk's semantic ideas to their logical end, he believes that this idea falls out automatically and is a powerful enabler of many things.

He opened a Squeak image in which an implementation of Newspeak was running. (The browsers are Vassili's; see his talk in the Smalltalk Solutions conference.) He opened a class AlternatingParser that was nested within CombinatorialParser. (It showed the syntax. Newspeak has a syntax, not just a file-out format. :-)) It had four slots (in vertical bars, like temps in Smalltalk). You can only access the slots via accessors (`pfun` slot means `pfun`, `pfun:` accessors are automatically and inevitably present).

```
either: pf1 or: pf2 = {
  self assert: (pf1 isKindOf: BlockContext)
  pfunc: pf1.
  qfunc: pf2.
}
```

where pf1 is actually an accessor call to pf1 and likewise for pf2.

The enclosing class also had slots. These had initialisers and an = sign meaning they are read only, i.e. there will be no setter available so, short of reflection manipulation, you cannot change these values.

```
BlockContext = platform BlockContext
OrderedCollection =
              platform Collections OrderedCollection
Error = platform Error
```

Q. These are message sends? The above are not expressions but they are compiled into message sends.

A top-level class has no global scope; you must get its scope from its superclass or get things explicitly. (They could have a global scope provided it had no state but in fact they see no need for it.) Thus slot BlockContext gets initialized by sending a message to platform which returns a BlockContext class. Here we have a true modular structure. This class' methods have no access to the outside world except through parameters so if you refuse to pass it an object that e.g. has the capability to open a certain file, it cannot contrive to do so. Thus every object runs in a sandbox all the time.

You could assign platform to a slot named platform, calling platform OrderedCollection or whatever all over your code and be back in Java's situation of fully-qualified state, but they are training their users - few in number at the moment - not to do this.

Q. So you need to know what classes you will call beforehand? The end system will have tool support to add these as you type a class. For the moment, you have to go back and add it as you code.

Q. Add new classes on the fly? Yes, if you have the right mirror.

Because there is no state, all the code in CombinatorialParser is innately reentrant. We can also have other implementations that do things differently e.g. a more complicated and larger memory footprint but faster

implementation. Since all classes are accessed via message sends, plugging in different implementations never meets hard-coded obstacles. Every module has its own sandbox; it can only call what it is given.

The classes are created lazily; the first time you send the message CombinatorialParser that is when the class CombinatorialParser is created. He showed where SequentialParser was overridden between super and subclasses. CombinatorialParser can be extended as a subclass of SuperCombinatorialParser, whose slot is defined by

```
SuperCombinatorialParser = super CombinatorialParser
```

The message Object is the same: it is defined in Object and can be overridden in subclasses (Object exists in the bases system but any subclass' Object would be created when first called).

```
WrappingParser = CombinatorialParser
```

says WrappingParser's superclass is defined by (the send to itself of the message) CombinatorialParser and so on up to Object. These classes are computed lazily and cached. Thus we have the class hierarchy inheritance.

The point is that because a class sends its superclass name to itself, it can override its superclass method and redefine itself. A class has no idea who its superclass really is: it is just a message send which can have different implementors at different times like any other message. Thus every class must be defined as a mixin and can be used as a mixin; it can be in different hierarchies in different contexts.

Q(Christian) What about tools; who sees all the objects as opposed to a scoped subset? The appropriate mirrors would allow it; most users should not have those mirrors.

Now we can understand nested classes. Classes are nested per instance of the class, not per class. (Needless to say, this is not the way it is in Java !) There are backpointers from a nested class to its instances and the instances know about their enclosing objects. Understanding this is key to understanding how method lookup actually works in Newspeak.

In the absence of an explicit receiver, we send a message to the activation record of the send. Accessors get sent to the implicit receiver self but a message like BlockContext is sent to the slot which is inherited from the enclosing object, so we have to know AlternatingParser and its enclosing object. Method lookup goes up the receiver inheritance chain as in classical Smalltalk but at each point in that chain it goes up the lexical scope chain, going to the next super only if no match is found in that lexical chain. The lexical scope chain starts from the activation record and thence looks in the local class, then the local class' superclass and so on, eventually reaching the local class' Object. Then we go up the receiver inheritance chain: the superclass looks in itself, then its local superclass and so gets to its Object. This continues till an implementor is found or until DNU is reached.

Although this sounds complicated, that is just the complexity of explaining

it. When programming, it is obvious because as you program the lexical scope is right there. (Vassili compared it to the difficulty of explaining to a newbie Smalltalker how super lookup worked as against the ease of using super in code and understanding what the effect would be.) A necessary consequence of this scheme is that the machine has to know the enclosing object to figure all this out.

Explicit receivers are different: they do not walk the lexical scope. Suppose we were explicitly calling `self Dictionary`. If Dictionary were not defined in this inheritance tree, we would walk to the top of the inheritance chain of implicit receiver self - *without* walking the side chains of lexical scope - ending on DNU if it is not found. The effect is that changes in supers do not get captured, unlike Self or Beta.

Module definitions are objects. An instantiated module is deeply immutable in the module's `main:` method. The mirrors you are assigned will be the ones you get.

So modules are well isolated; so, how do we hook them together? He showed a slide of an Object literal (which they have not implemented yet; they do these in Smalltalk workspaces today):

```
Object
  private class MyApp platform: p args: args {,...}
  public main: platform {
    MyApp platform: platform
          args: platform commandLineArgs ...
```

This is like C but without pointers back to loads of places so it is sandboxed. Thus you link your modules at runtime when you start up. His slides show an example for CombinatorialParser, with grammar, parser AST and other Newspeak libraries linked explicitly via the modules' factory methods.

Interoperability: don't you just love explaining primitives to non-Smalltalkers. Everything is a message so are primitives messages? Well, sort of; it can sometimes be hard to say what the receiver is. Newspeak has no primitives. Instead you send a message to the VM object reified via a mirror (so not everyone can invoke every primitive). In many languages, primitives are just foreign calls; in Java, for example, you call C to access some VM features. This is wrong: these ideas are distinct. The VM is not just another language. (If you do things right, the VM will ultimately not even be written in another language.) Your primitives should know all about your data. In Smalltalk, it is the other way round: calling foreign functions is usually a variant of the primitive syntax, often an ugly one since it was an afterthought.

In Newspeak, you send a message to an alien object, e.g. a DLL (Niall: pretty scary alien that one :-)). Specific APIs may map to particular 'planets' from which groups of aliens may come. Thus they can do e.g. portable native GUIs. (Current Smalltalks either emulate or do something very low-level, e.g. Squeak's bitmap, or are very specific to the dialect and the host OS.)

This lets them run portable native GUIs; he showed a vista GUI ("it runs because the licence happened to fall into my hands" but is a little flakey).

The team is 4 people (Peter Ahe, Vassili Bykov, Yaron Kashai and Gilad) plus Eliot Miranda (emeritus). If they had the planned current team of 5, that would still be few people to do all this. The main thing they lack is libraries; until they are there, Newspeak is embedded in Squeak for better or worse. Squeak was a great place to start. Newspeak no longer runs on a vanilla Squeak VM. There will be some tweaks to syntax and semantics. They aim for public release under the Apache 2.0 licence.

Q. Versioning? See his talk on YouTube. In the long term, he aims to rid the world of versioning.

Q. Syntax of Newspeak is extensible? No; he is not a fan of extensible syntax. He will stick with Smalltalk where its syntax adds something and yield to 'marketing' pressure where it does not (e.g. character literals). Newspeak is the only project he knows of that actually uses its parser-combinator library for its language, as opposed to just writing papers on it.

Q. Developer's doing maintenance may need to know what the superclass is? Well, Smalltalk does not tell you what a message send will bind to. A Fortran programmer would probably say 'I need to know'. No, they need not to know. Q. How do I know what messages my superclass implements? Same answer: you don't 'know' what messages the return of a message send understands; this merely puts super in the same state.

Q. Funding? It's been easy to convince people to fund this and open-source it afterwards. It addresses issues that people care about and where Smalltalk has weaknesses (and other things have great weaknesses).

Q. Security? Security has to be there from day 1 or it is never there. It is your machine you connect to the internet. This runs on Squeak so of course is not secure but it is an architecture which makes it possible to build systems that are secure.

The above is how they solve issues that were known in 1980 but not addressed by Smalltalk. The world has also changed since then; now we have multi-cores and the internet. Actors (known in 1972) and functional programming are a natural fit to multi-core.

(Q. Have you thought of E's concurrency model? The project has not yet actually *done* anything on this. Gilad expects what they do to be more E-like than Erlang-like, and to be actor- and FP-focused. Newspeak is already very near FP, differing only in areas like initialization and similar.)

In the age of the net, everyone talks about cloud computing. Clouds are vapour and, having reached the end of what has been implemented so far, he is adding his bit of vapour at the moment. Cloud computing is about delivering software and maintaining it over the net. Javascript is the assembly language of this idea and the web browser is its excuse for an

operating system. Using these is like going back to 1970s timesharing or later X-terminals.

Web apps have their downsides. The system software has to be local, UI is the web browser (!), the network had better be reliable, fast and cheap, and session expired is the reboot of this world. Gilad does not like rebooting. As he sees it, if you need to reboot please be discrete: surely you can do it in 300ms, below his threshold of perception. Instead, his phone now reboots and his television is starting to reboot!

The cloud model means the ISP is providing your computing power so they are driven to session-expiration and similar low-level computing tricks to do things like save their electricity bill (yes, seriously; talk to people in this business). There are very good reasons for pursuing solutions that use your local computing power. But the good thing is that users cannot go to Google Earth and say, I liked your 2003 web page so you must maintain that till the end of time. The great thing is, users don't even realise they can't ask for this. The best solutions are when the users can't even formulate the stupid question.

Maintaining the software and the data on the server provides audit trail. Having modularity gives you well-defined units of deployment, helps hot-swapping over the net. Some people in the Smalltalk world are already doing things of this kind and Gilad aims to make it reasonably pleasant.

State of Newspeak today: the language will evolve and may become less Smalltalk-like in some syntax: it will definitely keep keywords but might throw in some curly braces. The implementation is incomplete, especially the libraries. They don't yet have a native GUI on Unix.

Generally, they find a good synergy between Newspeak's elements: message-based programming, object capability, modularisation and virtual classes / mixins, mirror-based reflection.

Q.Syntax changes: make them settable? It is a possibility. Gilad does not trust the user to change the syntax but might allow consistent syntax schemes with the user choosing between them. He shudders to think what Ruby programmers would make of it if left to themselves. :-)

Q. Lookup approach: are messages distinguished? Messages may be automatically defined from a slot or by the user in various ways. If you have separate namespaces for different kinds of message instead of having a single lookup and namespace, well, Gilad spent 10 years in a language like that. :-/)

Q. Lexical scope understandability post-hoc when methods have been added to superclasses? Beta and Newspeak both find it tricky to explain the mechanism briefly but Beta found that it did what users wanted and he believes Newspeak will find the same. Ask him again when he has a million users.

Q(Georg) class extensions? Eliot liked them too; Gilad does not. See his blog post 'Monkey Patching'. he does not expect everyone in this hall to agree with him. He sees class extensions as tending to create classes with many methods and noone responsible for coordinating them so it becomes harder to know what will happen when you load. Of course, the system is open-source and people can change things.

Q(Colin Putney) Implicit receivers versus explicit receivers? In regular dispatch, the receiver is fixed. In implicit receiver, lookup must determine from the lexical chain who the receiver will be. It is not done for some specific set of reasons; it is done almost all the time for almost everything. They also have super send and sometimes you send to another object but implicit is the norm. Thus lexical gets priority because lexical is what you can see. If there really is a conflict and you really want inherited, you write self or whatever

Q. Performance aims? Their baseline aim is Squeak performance. Naive implementations of implicit send lookup can hit performance; they have now optimised that. Super cannot be bound and Squeak does not do a good job of their implicit super (no Smalltalk would, he thinks).

Q(Georg) What is the big picture? Making it easy to write applications that run locally but can easily be updated over the web. Gilad got people to pay him to do this by concentrating on that. Beyond that, it will be open-source and where it goes he knoweth not.

Smalltalk Solutions 2008, Reno, 18 - 21 June 2008

Seattle being a well-placed hub for travelling from Heathrow to Reno, I visited friends in Seattle the day before the conference. On Wednesday, Sue dropped me at the airport for my morning flight and drove off to her work as a coastal engineer - after which I realised that the small pouch with my cards, my money, my passport, and all means to identify me was still in the car with her. Having no means to reach her and get it back before my flight, I presented my self to the homeland security desk as a mad Englishman who forgets things and asked them to take pity on me - which they generously did. I was sent through 'special search' and for all I know now figure on some database, but I got to Reno. Travis kindly guaranteed my bill to a doubtful hotel check-in lady so I got a room (and Sue faxed my passport details to the hotel the next day to reassure them). Adriaan and Sytske lent me some cash. Luckily, I had arranged to fly back to Seattle and see my friends again en route home, so reacquiring my passport before attempting to leave the US was easy. All in all, it was the most painless 'arrive at airport without identification' experience I can imagine, but I shall avoid repeating the experiment.

Summary of Presentations

I have sorted the talks I attended into various categories:

- Keynotes
- Experience Reports

- Tools and Process
- Aida and Seaside
- BoFs and the Coding Contest

after which I describe Other Discussions, note some Follow-up Actions and give my overall Conclusions.

As each afternoon had two tutorial tracks and two talk tracks in parallel, I could not attend, still less report on, all I wished to see. (Some choices forced by the schedule were painful, but luckily James Foster's go-at-your-own-pace GLASS tutorial allowed me to attend it *and* Vassili's talk *and* Georg's talk). James Robertson's blog posts cover some talks I missed, as do those of other bloggers, and recordings of the talks will be posted. Talk slides and video can be reached from the STIC website http://www.stic.st.

Keynotes

Interfaces without Tools, Vassili Bykov, Cadence Design Systems

(This talk follows naturally from Gilad's in the Shared Keynotes section above, so is put here in 'Keynotes' even though strictly speaking it wasn't.)

Vassili started by doing something magical which he did not explain (it would take half the talk), so opened something that looked like the standard 5-pane browser.

```
browser := BrazilClassBrowser new open
morphicDesktop := browser window desktop.
windowDesktop := ...
```

He moved the browser from the Morphic desktop to the Windows desktop. He showed that while its state was preserved from what he had selected in Morphic, the browser widgets were now true Windows widgets. He then did:

```
windowsDesktop remove: browser.
morphicDesktop add: browser.
```

It reappeared in Morphic and he opened halos and showed doing the things you can do in Morphic and not in Windows.

He calls the widget framework Brazil because he feels the movie has many references that could easily be applied to software, and is in keeping with the (Niall: slightly unfortunate, to my mind) 1984-theme Newspeak name. Morphic is just one of the native libraries from the point of view of Brazil.

That was all on Brazil. The talk was about Hopscotch. The Squeak browser in is quite like the first one created in '76. The structure of objects in Smalltalk is 4 levels deep (categories, classes, protocols, methods) so the browser did not change much because it was a good fit to this.

In Newspeak, classes nest. Thus you need an N-pane browser similar to TrailBlazer and suchlike; he sees this as a sub-optimal solution. More seriously, the browser shows you one method at a time but exactly while you are coding one method is when you need to look at other methods.

The hopscotch browser deliberately looks very like a web browser. It has buttons along the top left, a search box at top right and then panels for Navigation, Recently Visited entities and Did You Know.

He went to the demo package for this talk and created a class: click button, see template, edit. He went to other places then returned and saw his class template edited as he had left it. The created class showed its name in a list with number of methods and number of subclasses tab-listed to its right.

He clicked on a class and showed its comment, definition, instance methods and class methods listed in vertically-arranged panels. Some ghost methods - subclassResponsibility in super class indicated method was needed - were shown.

He jumped to the Collection class and showed the methods being annotated (one by one, took a few second for them all) with icons / numbers for how many senders each had and similar. There is an icon for whether there is an overridden definition; click on it and you see the local implementors view.

Vassili likes using just one window and navigating around and back to see the views you want.

All this has been standard Smalltalk. He then went to a Newspeak class StS2008Demo and showed in the definition how he was importing and renaming.

```
Slots
Presenter = platform Hopscotch HPresenter
...
```

He then got panels for File and its subclass Folder under the StS2008Demo definition panel and added methods to these two classes to enable the StS2008Demo to show files of demo information. He created a demo object and sent it one of these methods (`exampleDirectoryTree`). He briefly presented the Presenter/Subject paradigm: Presenter is like an application, Subject like an implementor of an application.

He created a class FileSubject and noted there were no ghost classes because we did not yet know its superclass; that can be bound later in Newspeak than in Smalltalk. He then created FilePresenter and gave it a method definition that called the non-existent method file, for which another panel promptly appeared. (Nothing of this is modal: Vassili can go elsewhere and return to find it as he left it.) He was now working on two methods at once, their code stacked vertically in his single window. He saved and opened a window that showed the FilePresenter on the top of directory tree, i.e. on `/`.

In the definition method he then added

```
...
  row: {
    label: 'Name'.
    label: file name}
...
```

and showed the file presenter window redisplaying to show Name: /. (Add on package adds {} array constructor). A bar of colour on the right of a panel indicates when things are being edited (i.e. dirty in DB terms). He made the FilePresenter clickable. He added

```
private Color = platform Graphics Color
```

Alas, Newspeak is still work in progress and so he had to reboot (Sorry, Gilad) to make this binding visible. Now he can make the FilePresenter use colour. Georg asked whether that was Newspeak or Smalltalk. Objectwise it was both, one messaging the other; syntax-wise it was Newspeak.

He added 'inspect me' and 'browse me' buttons to the FilePresenter. The latter demoed getting a reflection mirror using `sendUp navigatorDo:...` where `sendUp` walks the hierarchy of widgets. He then combined these to show an inspector when collapsed and a browser when expanded.

Thus we no longer have a tool. We have Presenters within Presenters that reflect the structure of the domain in the structure of the window internals.

Thanks to Peter Ahe, Gilad Bracha, Eliot Miranda (emeritus) and Bob Westergaard.

Q(James Savidge) Newspeak version control? Vassili is not really the person to answer that.

Q. Return from method and return from block? Just like Smalltalk.

Seaside: Your Next Web Framework and **Persistence Solutions for Seaside, Randal Schwartz, Stonehenge Consulting**
Randal gave the first of these two talks two months ago to 7,500 people in Brazil. They gave him a 3-day Smalltalk conference as a result,. So he's repeating this talk to us, though many of us know all about Seaside, to show ways of explaining to outsiders why Seaside is good.

Randal likes Smalltalk because it is very simple to learn: the entire syntax can be shown in 2 minutes and explained in 20. All the libraries are open to you to show how to code and to extend. Smalltalk has been around for 20-30 years so it has extensive, mature, robust material.

It has very mature debugging. The walkback stack is alive: it's not like doing an autopsy on a dead body on a table. When you mix that in with the web framework you get intra-hit debugging. Fix and resume your web session. Randal was interviewed on web radio two weeks ago and as James put it later, "You could hear the guy's jaw dropping" as Randal explained this. (After seeing Dale's talk on GLASS, Randal realised that he could also persist these errors and e.g. debug errors that happened last night.)

Smalltalk has good unit testing. The Smalltalk community invented eXtreme Programming.

Seaside uses Smalltalk for the templating language and this is very

expressive. (At this point he would show examples but he'll assume this audience knows Smalltalk.) You paint on a canvas with a brush using objects, written in ordinary readable text, from which you naturally generate sane HTML. Smalltalk refactoring tools also make improving your code easy.

You have a choice of vendors. He reviewed them. People used to Ruby or Perl find this a good point.

Team development environments are Smalltalk-aware, so powerful. Squeak and GemStone have Monticello, Cincom has Store, VASmalltalk has Envy. They can all file in and out if you want to use files but you won't.

Continuations: Seaside maintains the inter-hit state transparently. You write your app as if the web is uninvolved.

```
self firstPage.
[self validate] whileFalse: [self secondPage].
self thirdPage.

html anchor
  callback: [self exit];
  with: 'exit'.

html textInput
  callback: [:value | e := value];
  with: 'exit'.
```

This handles the back button. It provides you with callbacks. It saves you form having to name all your parameters.

Persistence (more in following talk): you can use object-relational mapper frameworks of which GLORP is the most obvious. Alternatively you can save as objects with Magma (watch this tree and save when I change it), Squeak reference streams, or of course GemStone for very easy, very scalable, very mature pure OO persistence.

Seaside has a large community doing active development, lots of add-ons like Scriptaculous and a free hosting site seasidehosting.st.

Get this talk from MethodsAndMessages.vox.com and give it at bar camps or web user groups.

That ended his first presentation. Then he spoke on persistence in Smalltalk. Randal only recently reentered commercial Smalltalk after many years (but he first executed a doIt in 1981). His first Seaside app for hire was deployed last month. He has also done a lot of Seaside advocacy (which is fun but does not pay as well).

Randal then talked about licensing. He has nothing against the GPL or LGPL but be aware that it can block anyone else's use because subclassing could be regarded as derivative work. However it is a non-issue for web servers which you are not distributing. (Niall: but you must then be careful *never* to distribute to anyone who *might* want to exploit their GPL rights.)

His perl experience is that with friendlier licences he can arrange to write a system, give back some code but leave the key parts with the customer. Alan mentioned that Glorp, which is under the LGPL, explicitly 'clarified' how Smalltalk code should be handled, excluding such ideas.

Persistence requires you to understand both reachability and migration. Over time, classes acquire new behaviour and instance variables and also change these. Do we tell everyone to re-login to our website or do we take people with us. Some persistence schemes deal with these; others ignore it. Persistence means dirty objects; do we have to notify or will our solution handle it automatically.

Thus we have various strategies in Smalltalk. Saving the image is easiest and of course gives us most problems re the above issues. We can serialize objects in and out. We can access a non-Smalltalk database directly or through an OR mapping layer. Finally, we can run in an OO database, persistent from the start.

Randal showed the simplest code to save the image every 5 minutes. You could be saving an image you could not restart so it's better to save it with the timestamp in the name. This is good enough for e.g. a small blog server; few people will be furious if their last five minutes of blog typing has to be reentered when the occasional crash catches them.

Serialization is simple if file-based. Randal also knows a couple of server-based solutions, better for clustering.

```
(ReferenceStream newFileName: 'blog')
  nextPut: self allPosts.
...
allPosts := (ReferenceStream oldFileName: 'blog'
  contents.
```

The above is from Ramon. Avi also told him about storing part of the live image on the disk. Avi is using this for DabbleDB but Randal looked for documentation and failed to find much. If you see Avi, ask him how it works (and tell Randal).

SIXX converts objects to and from XML via `asSIXXString`. Randal is no fan of XML but observes that you can use code very like the above:

```
(SIXXWriteStream newFileName: 'blog')
  nextPut: self allPosts.
...
allPosts := (SIXXReadStream oldFileName: 'blog'
  contents.
```

Be aware that SIXX streams do not inherit from the main stream hierarchy and only have some of the stream API methods implemented.

OmniBase does not load into Squeak 3.9 or 3.10 and he could load it but not make it run in 3.8 so perhaps it is no longer maintained. He liked its general multi-transaction design but disliked that all dirty objects had to be explicitly marked. Unless it is being maintained, it had best be left.

MinniStore is incompatible post Squeak 3.4 and the tests fail on 3.8. Its SqueakMap entry teases you with cool things like 'English-like query language' and 'multiple indexes' but don't go there till someone tells you its running again.

Server-Base solutions are more complex to configure but also more scalable. The two main players here are GOODS and Magma. Goods is language-neutral; he even found a CPAN Perl binding called Pogo. You have to configure a server in C++, not ideal. Persistence is by reachability and it notices dirty objects automatically. All this is good but commits may conflict in which case exceptions are raised and objects get updated in conflicting ways, which concerns him.

Magma is local single-user or clustered multi-user, is ACID and handles large collections. It does live class migration with some tools. However it seems to be a one-man development project (others use it). It cannot be vacuumed to reclaim old storage while clients are connected, so this blocks 24 x 7 use. He likes its transparency for dirtying objects. Commit is just reaching the end of the commit block, rollback via raising an exception.

There is a package, by someone whose name he forgets, that combines Magma and Seaside by adding WAMagmaConfiguration. So your configuration pages have an addition screen to configure server location, connection type (single, shared, pool). So except for the 24 x 7 and one-man operation, he likes this best.

There are also interfaces for SQL DBs.

SQLite is indeed light and fast. It is small enough that you can embed SQLite in your app but the Smalltalk wrapper does not do that; it uses FFI to talk to native SQLite lib. But he cannot load it into Squeak 3.8, 3.9 or 3.10. (It works fine on VW and Jim's WebVelocity demo uses SQLite.) He's pretty sure it is a small problem, possibly due to changed version of Monticello. You connect to SQLite library, which opens a file for you, and talk to your connection. However SQLite does not handle placeholders in SQL so you must be clever to escape them correctly.

PostgreSQL is similar to SQLite but much more powerful. It handles placeholders, streaming results (i.e. getting a large result not all at once), and events. It is mature and well built but is now slightly out of date and he had to monkey patch it once for a customer.

ODBC he's heard of and no more.

OR mappers are Glorp, and ROE. Glorp is a very large and uses names that might conflict with other apps in non-namespaced Smalltalks. Its test suite is large and educates you in it. A Glorp descriptor says what the table contains, what objects participate and how to map them. It is very flexible, almost too much so (needs moderate amount of work to do ultra-simple things). Glorp lets you do really interesting complex mappings e.g. row values can determine which objects were returned (e.g. return Manager if

manager bit set, Employee otherwise). He creates a session object from an accessor object from a connection object from a login object - all lazy and it all works behind the scenes. It does the trace and reachability and seeing dirty objects stuff.

Cincom then decided to add an ActiveRecord pattern. This will build a simple starter mapping for Glorp and then you modify as desired. The ActiveRecord may not work on all legacy databases but should be fine for 80% of cases.

Avi provided ROE. It is under-documented but has tests. It lets you write SQL queries as Smalltalk objects using a DNU trapper. It talks to PostgreSQL and works but appears to have been abandoned.

The easiest way to do persistence is to do nothing at all. GemStone is this and see Dale's talk, Dale's excellent blog and all the info GemStone provides.

Q(Georg) Some things alive, some dead; what is the common factor? Randal feels there has to be a critical mass of users and more than one developer. In other languages, e.g. perl, he sees these as the essential features. To get these two, it has to solve real issues adequately and be well-enough designed to let others work on it.

Q(Niall) Your first talk mentioned which points played well to Ruby and Perl people; what points block such people? Mostly they relate to Smalltalk. "How do you deal with that stupid image". He replies, "Try it for a week and you'll never go back." Performance, that ancient myth, is often raised; you just say, that is out of date (demo it). "If I start doing this I'll not be hired anywhere because there are no Smalltalk jobs." However the enthusiasm about continuations and debugging intra-hit gets people past this. He taught a 3-day perl course and in spare time at the end offered a seaside half-day. At the end people asked him, "Why did we spend the last three days learning perl?"

Q.(James Savidge) Niall and I are doing work on the Smalltalk database so point them at it and maybe they'll think there are some ST jobs.

Go to seaside.gemstone.com to locate Dale's blog.

Q(James) Does anyone ask about security? The only questions are about Denial of Service attacks using old session states.

Q(Colin) OR mapping is hard - he wrote one as his first Smalltalk project - and so it is natural that many have been abandoned.

Implementing Programming Languages for Fun and Profit with OMeta, Allessandro (Alex) Worth, Viewpoints Research Institute
Alan Kay heads Viewpoints Research. Alex works there on the 'Steps towards the Reinvention of Programming' project which seeks to build a whole personal computing environment in 20,000 lines of code. That is

almost an order of magnitude smaller than Squeak (don't mention the 40 million lines of Windows XP). The aim is to put people in charge: noone can understand 40 million loc but 20,000 lines is a 400-page book; a person can own that. Another aim is didactic: university students could study it and learn about architecture. They could even learn it instead of Java. :-)

STEPS cares about code size and understandability. Choice of language can affect that a lot. So what language should STEPS choose, and should it be just one or several for the different part of the system? They don't know the answer so they must experiment. However it takes a lot of time to implement a programming language, taking time from experimenting with it once built. Another problem is that traditional programming languages are big and they only allow 20kloc for the whole thing; the programming language had better not take too much of that. OMeta is a way of implementing programming languages that is quick and small.

He introduced his examples with an 'Ometa Inside' logo complete with tune. His first example was an Ometa/Squeak implementation that generated Javascript: 300 lines with parser and all else. He showed javascripts controlling a morph and a lego game.

Ian Piumarta is developing COLA, a new STEPS-oriented environment. His slide showed some things done in 350 lines of code running on Sun's Lively Kernel. A Javascript program in a text morph evaluated on the fly to generate a spiral pattern.

They also looked at Prolog ideas, but with syntax focused on non-programmers. Alan Kay suggested Toylog, a front-end for Prolog that is written in very English style, implemented in 70 lines of OMeta:

```
Homer is Bart's father.
Marge is Bart's mother.
x is y's parents if x is y's father or x is y's mother.
Bart's parents? -> Homer, Marge
x is y's grandfather if x is z's father and z is y's
father or z is y's mother.
```

etc. The Squeak-based program that runs it lets you ask such questions and shouts the answers in excerpts from the Simpsons, complete with cartoons.

Alex thinks the computing community splits into programmers and people using computers who are not programmers. A goal of OMeta is to create good domain-specific languages; these come from people who need them, not from programming languages. Another goal of OMeta is to make your apps scriptable by end-users without their having to learn Smalltalk, let alone any more ugly language.

So much for motivation and goal. Next he presented OMeta in detail. Traditional programming language implementations (Squeak is an exception) use a kind of batman utility belt of specific tools for specific tasks: lex, yacc, AST transformation and suchlike tools are used to do specific things. This cannot work for their case. In OMeta, lexical analysis of characters to tokens, parsing tokens to trees, folding parse trees to parse

trees and generating code from parse trees are all examples of the same thing: pattern matching. Using pattern matching for everything makes it simpler to build, easier to learn, and more extensible.

Other languages have pattern matching: what does OMeta offer. ML-style pattern matching is great for tree rewriting and poor for lexing and parsing. OMeta uses a Parsing Expression Grammar (PEG) [Ford 04] which is a good framework for writing recursive descent parsers. A recursive descent parser's 'or' is ordered. This eliminates ambiguities and makes the parser easy to understand (no SHIFT REDUCE conflicts). Backtracking and lookahead work better and we get semantic predicates. (?[x == y] means make sure x is equal to y; if not, fail this thread).

He then showed BNF for a language (called MyLang) defined using OMeta

```
dig ::= $0 | ... | $9:d => [d digitValue]
num ::= <num>:n <dig>:d => [n * 10 + d]
expr ::= <expr>:e $_ <num>:n => [{#plus. e. n}] | <num>
```

PEGs operate on streams of characters. OMeta extends that to operate on streams of objects. `anything` matches any one object; shorthands to avoid using it everywhere let them write `'hello'` strings, `#ans` symbols, etc. He showed an example:

```
eval ::= {#plus <eval>:x <eval>:y} => ...
{#plus. {#plus. 1. 2}. 3} -> 6
```

He extended MyLang to MyLang++ that understands minus as well using standard OO super

```
expr ::= stuff to do - instead of + | <super expr>
```

This is an example of what he said about extensibility. We use the OO stuff we know to extend. He then showed parametrised rules looking very like Smalltalk blocks

```
range :a :b ::= ...
```

You can have higher-order rules. You find yourself writing stuff like

```
formals ::= <name> ($, <name>)*
args ::= <expr> ($, <expr>)*
```

over and over so you'd rather write a `listOfP` higher order rule instead of defining comma separated lists in longhand all over the place.

If we have an OMeta parser and a JSParser and only single inheritance, how do we write an OMetaJSParser. Inheriting from either forces you to clone the features of the other and later we'll change the other and our clone will be out of synch. Inherit from both? But we may have clashes when the same name has different meanings in the two languages. So instead they offer foreign rule invocation. You can lend your input stream to another grammar: "I can't parse this; you try". This lets you compose grammars.

There are several OMetas: OMeta/Squeak (and someone ported it to OMeta/VisualWorks), OMeta/COLA, OMeta/JS, and others are being built. Each one has syntax fitted to its host: OMeta lists in OmetaJS look

like Javascript lists.

Takashi Yamamiya has done stuff in Smalltalk. He used Squeak to write a Javascript executing web page and this inspired Alex to write OMetaJS. Javascript is not Smalltalk but it is late bound, has properly closed first-class functions and so on. Above all, it is everywhere. You can treat it as an assembly language: you don't have to program in it, just target it for code generation. (Douglas Crockford's book 'Javascript: the good parts' gives excellent insight into these issues.)

He opened a browser on the OMeta/JS web page. It reminded me a little of Carl Gundel's RunBasic pages. You type Javascript into the pane and evaluate it as if the pane were a smalltalk-style workspace. He sees no need to waste such a neat idea on Javascript so has some indirection between the code you type and the code that gets executed. He made the printIt, doIt invocations pass the string to the translator which is the OMeta and JS union he discussed above. He wrote a trivial OMeta rule to show it matching. He then went to pages with successive developments of a simple language definition and used them to recognise expressions, to evaluate them, to compile them, etc.

He then showed a library of 70 lines of code written in Javascript that implements the semantics of Prolog. However to use it you must write very ugly code (he showed the code for his Simpson family definition and it was indeed horrible). With OMeta, he took 12 lines to write an incomplete parser for Prolog that put a pretty face in front of this stuff.

He then showed a Logo OMeta program drawing a spiral on the page. Then he showed a source-to-source Javascript compiler. This is just an identity compiler of course, taking a JS program to a JS program but it is a start point for writing extensions to Javascript. He added a `say()` feature to Javascript, made the window use the new language and showed it handling `say('Hello')`, then changed it back and showed it reporting an error.

Alan Kay taught a course on program design. Alex prepared an example of language bootstrapping for this course. It bootstrapped a language like OMeta, defined a compiler for the language in the language, showed the new compiler parsing correctly, then recompiled the compiler in the new language and so completely bootstrapped the language. The students liked this and some, for their projects, extended OMeta and tried things out.

His last example was called Etude. Some students in the class, first year grads who had never written a compiler before, wrote a programming language to describe music, generating midi files from 'code'. He compiled the code for happy birthday and (after the usual demo hiccough; he was so near the end I almost thought he wouldn't have one) it played.

He visited a friend with a Wii and discovered that it has a web browser (Opera) and OMetaJS works there too. Javascript really is everywhere. :-)

See the DLS2007 paper or visit

- OMeta mailing list: http://vpri.org/mailman/listinfo/ometa
- OMeta JS Wiki: http://jarret.cs.ucla.edu/ometa-js

Q. Where did the name OMeta come from? From Meta2, a 10-page paper that changed his life.

Q. How up-to-date is the VW implementation of OMeta? (Jim Robertson) The latest publish to Store is May 27th. (Alex) that is recent but Squeak and will have the latest and greatest soon. Masashi Owosama is helping.

Q(Georg) relation to general parsing work / research, specifically declare-before-used variable declaration? This is more expressive than many things used today for industrial strength applications. It is more powerful than context-free grammars. Variable declaration in Javascript must handle its complex scope rules. He handles this in OMeta by having visitors that find and decorate the parse tree in multiple passes over the same data.

Q. Tutorial on the web? Yes: google Ometa/JS 2.0 and find the tutorial on the website.

Javascript has a bigger grammar than Smalltalk. He had written Smalltalk and JS parsers before. This time he implemented a translator from Smalltalk to Javascript. His first workspace understood Smalltalk and then he extended that in OMeta to understand Javascript.

Experience Reports

Porting VW5i/Envy to VW7/Store, Tom Hawker, OOCL

OOCL are a shipping company (see my report of Tom's talk at StS2006 for a detailed description of its domain). The port is ongoing and has reached the point of its core being handed to QA for acceptance testing.

The application is over 10 years old and was originally ported from Cobol to VW3/Envy thence to VW5i/Envy. The rich client is very complicated and has 100 of screens used modally and non-modally. The tasks that define bookings and eventually produce bills of lading are very complex. They have 2000 users on-line at a time.

They want to port to get back on supported software, to get performance benefits they have noticed in VW7, and to get refactoring tools and other benefits. However there is a big cost: they expect it will take 18 months total elapsed time from the three staff (all at the talk) that work on the port. Some costs are also benefits. They have doubled the number of tests in order to make the port safe.

They had to resurrect from old emails the directions to old websites where documentation actually resides. Some release guides had useful hints and they used the refactoring browser rewrite tool to find and change code that used VW5i patterns to code that used the equivalent VW7 patterns. They wrote a utility, called Topping Lift (Niall: I deduce someone on the project has a yacht), to move code from Envy to Store (StoreBridge was useless).

Smalltalk lets you look at a city skyline, raise it two inches, replace all the subsoil, drop it down again and have it look the same. Their goal was to have no changes in either application and server code and except for things like immutability, found by Refactoring Browser: this goal was achieved.

A Harry Potter book mentions a cursed book you can't stop reading; at times they feared they would suffer the curse of unputdownable code. There was no OSKit upgrade or equivalent (or suitable GemKit then, alas Paul's work was not then available). Envy is not Store and breaking Envy users of Envy habits is also hard. Murphy's law was a concern: throughout the code, they discovered that Murphy had been involved in its location. Column sets were modified in place to make specific windows do things the application needed and porting forward broke that.

They knew that a rewrite of GemKit was inevitable given its state when they started. OOCL has a policy not to use open-source software, so they were obliged to rewrite GemKit from scratch. They went to a memory-based comparison model and reduced a six-hour build to less than one hour. They added heavy instrumentation which slows it but lets them study what they are doing. They had to map interface behaviour. He showed slides of the architecture of their GemKit implementation, and a collage of its windows. The VW-side drives the Refactoring Browser to change the image. The windows show hierarchies of changes.

Store as it is does not fit their new agile/incremental development process, which aims to manage parallel disparate developments. However blessings were easy to change to match their development process. They then made it possible to label bundles with a tag or group of tags; prerequisites only match for tags specified or derived from the development. They found Store admin groups too primitive to control this mechanism so rolled their own re who was allowed to add or use tags. A slide showed all the changes to Store they made. Another showed a collage of windows: the four on the left were standard store windows with their additions, the others new.

The port has taken 9 months elapsed so far (not continuously worked on, they have other things to do). The server code works fine, their frameworks are sadly broken, their GemKit does one-third of what they want it to do.

Lessons learned. Noone follows their own development processes: the guidelines say use the framework this way but people use it that way, so it breaks when you port it. Envy does atomic loads that saves you from many pre-req errors. Store does not and working out why a load failed is not easy. Topping Lift was written to solve this for them.

Vendor issues. Store did not come from Cincom, it came from Andersen consulting, so he has no inhibitions about being frank on its having some poor, rigid code and absent or unhelpful documentation.

Murphy and his children have bigger class libraries than Jim Robertson and are impervious to bullets.

Q. Can you make your work on GemKit, etc., visible to the community, given that open-sourcing is not an option? Originally, OSing was the plan but Hong-Kong may not be open to the idea. They may be able have GemStone or Cincom cast a vague protective shield over its appearing in the Open repository or similar.

Using User Changes, Leandro Caniglia, Valeria Murcia, Caesar Systems
(This talk was also presented at ESUG; this write-up merges information and questions from both presentations.)

Leandro presented but explained the paper is in fact almost entirely the work of his wife and colleague Valeria with occasional help from him. The smalltalk change system has a few stable commands (add class, method, ...). A living application has many, changing commands.

In their system a composite pane contains a wrapper round its model object. When it sends a message to its model object, the wrapper logs the message and passes it on. To the user and the programmer, this mechanism is transparent. Only messages that change the model are logged.

The wrapper logs by creating a Change object which reifies the message. The Change validates itself; in debugging mode, the user gets a warning if this fails. Validated changes get added to the appropriate change collection, which can be written to a file for replay.

Changes know their timestamp, author, message selector, receiver name and arguments. Validation checks that these are non-nil and non-empty, that receiver names resolve to an object that responds to the selector, and that the appropriate number of arguments for the selector are provided. Their mechanism for naming objects uses names like `a:b2:c2` to trace from root object `a` to `b2` and thence to `c2`.

ModelObjectWrapper has nil superclass. It knows the object it wraps and its changeLog. Their first thought was to have a wrapper class for each class in their GUI but this merely duplicated their existing class hierarchy. They soon realised they only needed one object that recognised all messages in the GUI and sent them all to the wrapped object.

```
doesNotUnderstand: aMessage
  | selector |
  selector := aMessage selector.
  self shouldBuildMethodFor: selector)
    ifTrue: [self buildMethodFor: selector]
    ifFalse: [aMessage receiver: wrappee].
  ^aMessage perform
```

When a new message is first generated, the programmer is prompted to tell the system whether this message changes the model or not (if the question makes sense; they keep a list of many messages for which there is no need to ask). If the user says it does, the wrapper acquires a new message, e.g.

```
oneArgSelector: arg
  changeLog newChange
    receiver: wrappee
    command: #oneArgSelector:
    argument: arg.
^wrappee oneArgSelector: arg
```

He showed this in the browser for the `renameTo:` method.

The system therefore has a single ModelObjectWrapper class, a single Change class, a single UserChange class and a single ChangeValidator class. Most methods are automatically generated so the framework is very lightweight, development-wise.

Their interest in changes of course is not to log them but to replay them. Leandro restarted their system with an empty project, showed it was empty, reran the changes and showed it had the same set of objects.

Changes filed-in are always validated (they are coming from a file that might have been edited, corrupted or whatever) and the user is always warned if validation fails. They send the replied mechanism to the wrapper, not the receiver, to replicate the precise behaviour, including the logging, that created the change originally.

Having outlined the system, Leandro discussed applications. They can replay their work whenever needed. They have auditing; you can see who did what, and also what changed recently in a model. You can right-click on an object and get the list of commands sent to that object. You can go to any part of this history and replay the change; this can be used as a kind of local undo/redo. He showed entering some data on a form and accepting, then changing his mind, getting the changes and altering the input data.

You can use the change log as a scripting mechanism. The same approach gives you demo scripts and tutorials (and trouble-free conference talks :-).

You can overcome backward-compatibility problems. If your model will not load in your new system, try replaying its change log in the new system. You can often fix the problems and complete this sequence.

Merging changes is easier than merging projects. If a user calls you with a problem, and you take two days to solve it, the user will also have changed their model during that time. So instead of sending them your altered model, you send them your change set for the fix.

Support: you can send the user the changes that show how you tried to reproduce a bug. They can send you the changes that elicit a bug, or just their recent changes when they don't know what they did to elicit a bug. Once you have a script that shows a bug, it can be reused as a regression test, and it will help you write a test for that bug. Scripts can also teach the system to new programmers. Users can share patterns, ideas, etc., by sending the changes for an example of them.

You can count the number of commands or (by multiplying by command length) the number of keystrokes a user needs to achieve a given state. This is a useful metric to advertise for a productive Smalltalk system. You can also look for confusions or bad practices in your users.

Changes were once Smalltalk's CM system. You can still use changes to combine several people's work on a model. Database accesses that did not commit can be recovered from their change set. Work done at home can be added to the central repository.

These are general uses. They have also uses specific to their system. Scenarios can be captured as change sets and written out for later re-examination; these change sets are much smaller than keeping a copy of the whole project in the database.

Decision Analysis is a growing area of interest in their domain. Users can create a base model and explore scenarios, keeping the change set for each scenario. You can then copy the base model and apply a scenario change set. Each scenario is a child of its base since if the base model changes, each scenario will see those changes. This can be repeated to create a decision tree of changes to the model, in which each parent change set has changes common to all of its children who also have their own changes. He opened a tree browser in their system to see a tree, annotated with the value of interest.

They also do Monte carlo simulations.

This was inspired by the Smalltalk change log, Dan Ingall's Squeak event recorder and Valeria's own work on that to include Morphic wrappers long ago.

Q. Changes can be applied quickly (by design)? Their validations ensure this.

Q(Christian) changes separated syntactically? Yes via Smalltalk ! chunk separator.

Using Opentalk in an Unexpected Way, Giorgio Ferraris,
OpenTalk makes communication between two Smalltalk images easy. It also does other things but Smalltalk-to-Smalltalk is what Giorgio will talk about today. OpenTalk is in your VW image today. It is mature and robust. It is a tool you have in your pocket to us. Giorgio heard talks about OpenTalk and often thought "must use that" but never had occasion to do so. Until ...

He had a monolithic client-server application built in the last three years to support tour operators. It runs on Windows and Linux in VW and was sold as "this will run *wherever* you want." Then a customer needed to connect an AS400 database and this needed ODBC. In Windows, it was not a problem: they built the connection and it ran fine. But then the customer wanted it to run on Linux. ODBC on Linux? It's a nightmare!

They had to find a solution *quickly*. The customer had the app and had paid for it. Giorgio was called on the Saturday by a most unhappy tester.

Can we connect from a Windows machine where ODBC is running to the application running in Linux. Web service was not adequate. Then they thought of OpenTalk.

OpenTalk is easy to use, Giorgio opened two images and demoed a simple OpenTalk example.

```
serverBroker := RequestBroker newStstTcpAtPort 4242.
serverBroker start.
serverBroker objectAdaptor
  export: Transcript
  oid: #transcript.
```

and the same in the other image for the client

```
clientBroker := RequestBroker newStstTcpAtPort 4243.
clientBroker start.
clientBroker remoteObjectToHost...
```

His experience: OpenTalk was easy; almost too easy. Because you can get it working in seconds, it is easy to forget something it would have been better to handle.

He showed his domain model. DbConnection was the abstract superclass of the specific database connections. HOPOdbcConnection was the ODBCconcrete class and SqlTranslator generated the SQL query from the systems request. The right place for OpenTalk was between the two of them. In two hours, they had the new system running.

However everything was running in 2 hours so a relieved Giorgio forgot many things. Giorgio was running 2 images on the same PC. The developers are in Italy, the customer is in Genoa and Giorgio is at home so errors cost delay as these different locations and times got back about them.

When you are on the same PC and environment, you may not notice you are not handling things correctly (see ‘too easy’ comment above). You can pass an unexpected component e.g. a system resource, that you should not. You cannot see latency issues. So he moved to two PCs. On the same local network, the latency is still low but now sending a local ODBC handler to the other operating system will raise an error. He now saw that not all methods in the monolithic system were still valid on a distributed system. A proxy was needed to filter methods appropriately.

Next he moved to two machines with different OS, Windows and Linux. Now filenames were being written in Linux’ way and sent to Windows; the slash was wrong. The final try was to run over a slow connection. This exposed latency problems that needed optimisation. If you run smoothly over an internet connection, the task is complete.

They have a DbEngine for each database they support. The SqlTranslator needed information from the DbEngine. OpenTalk sends references not

values except for simple types (string, number, ...) so this generated much flow across Opentalk just to get data that was always the same for ODBC. Passing this data by value was the answer but it was hard to find how to make that change in OpenTalk's documentation and he found it in a presentation slide.

OpenTalk holds *weak* pointers to published objects, so remember: you must hold a reference yourself; don't expect OpenTalk to hold it. Giorgio refactored away an instVar he no longer needed and could not understand why his system now ran fine for 10 minutes or 5 or 2 and then suddenly died (when the GC took an object he needed).

A TCP/IP port allows one client only. When the customer ran many clients on the same Linux box, they needed to loop up to the first free port. (Giorgio's slide shows the code he wrote to handle that.)

When you have a hammer, everything looks like a nail. After getting OpenTalk running in 2 hours, he started using it everywhere. He uses it for two reasons: for scalability and because of necessity. He has other systems where he is being forced to move to distributed systems.

They had a system running on Linux that wanted to add data (e.g. specific addresses) to MSWord documents. They created an MSWord server image on Windows talking by OpenTalk to the Linux system. He demoed this, loading COM and his package for MSWord document handling into one image, while another, without this, made requests for information from MSWord documents. The local image used class side calls but Opentalk prefers instances so slight changes to the API were helpful. He added the new method `openVisible` ('visible' so we can see it; you can also do this in background without having Word documents popping up on the screen).

He now showed how COM raised a warning when a weakly-referenced instance dies (alas, ODBC does not warn). Then he went to the client, filed-in a dictionary with some address values, got the remote reference and tried to call it; it has vanished so we saw the debug window in the client complaining that the server was denying all knowledge. Then Giorgio went back to the server and made sure he had an explicit reference on the server side so it did not vanish. Then it worked.

A hotel supplier they supported only has a COM interface; their web-service interface is still being built. So they used OpenTalk to let the Linux system talk to that interface.

Now they find they want to build an architecture on OpenTalk to facilitate their uses.

Conclusion: OpenTalk is well built and scalable. Documentation could be better but Martin Kobetic was always there for them.

Q(James Savidge) Any rules of thumb for when to pass by value, when by reference? No, we just let performance drive us. If the performance bugged

us, we considered switching to value passing, otherwise we accepted OpenTalk's default behaviour.

Q. Need to redesign for distribution? Not really, In the ODBC task they found places where the system was ill-designed and rework helped but that was not OpenTalk specific.

ControlWORKS, James Savidge, Adventa
(Impromptu talk offered by James after a BoF.) James demoed ControlWORKS 4.3 running on VW7.4. (Because their customers are large they like carefully staged upgrades, so tend to be running a version or two behind.) James had worked in the wafer fabrication domain and wanted to keep working in Smalltalk. He was therefore pleased to get a job with Adventa some two years ago, after two years coding in Objective Forth. He was hired to work on their related ProcessWORKS product but the trend of work that customers needed led him to ControlWORKS. He will be visiting a customer on Monday to show, amongst other things, the configuration comparator work he has recently done.

The main image controls other images run on embedded machines running Windows or VxWorks. Their TMC image can control the robot and the interfaces to the central chamber. The same image with different parameters does the process monitors. He started one of each on different ports. The ControlWorks UI opened. The interface it uses is to a considerable extent required to be what it is by industry standards. (Industry wants operators to be able to move from machine to machine without having to handle different interfaces.)

Users must login; this is mainly for capability: different users can do different things, especially can or cannot do dangerous things. They also have interlock, rules in the scheduling engine that are there not to allow things but to prevent things. You don't want to replace one chemical in the chamber with another immediately afterwards if the two make an explosive combination.

(The usual demo hiccough occurred at this stage. Everything appeared started but the control image was not doing anything. At first, James shutdown all the modules and started over but it emerged that parallax on the demo machine was the problem; it was doing an imperfect job of emulating the appropriate OS. He had to reboot it and stop displaying to the big screen; instead we clustered round his machine. Later, he had to use two mouse devices, one to move the mouse arrow, the other to get right-click menus.

Because we were clustered round his machine, I could not take my usual notes at the time. These were typed up from memory afterwards and may contain errors. See user report on ControlWorks in my Smalltalk Solutions 2005 report.)

James took us through several features of the app. An impressive graphical widget embedded in the screen to manage a cluster shows its circle of

chambers with their valves, the robot moving wafers between chambers and the input and output wafer stacks. The robots, the values to the chambers, etc., are controlled by the scheduling engine, whose efficiency has immense financial significance. A stack of wafers input to a cluster might contain 25 wafers. If the scheduler allows the cluster to process two extra wafers in an hour, that could mean $500,000 extra earned by the end-user of this system.

The system must manage the drift of the machine from its initial behaviour as it gradually gets 'dirty' (understand this in a very relative sense) with use, and must schedule the various maintenance processes that restore it to pristine state. At a given point in this cycle, the exact times and settings to do various things will differ. It must also manage the tolerances that the various analogue processes involved can endure.

One of the key features is user-customisation. Users may create subclasses and connect the rules to them, etc. These configurations can become complex. James showed recent work on a comparator that lets both them and their user see the differences between configurations: all differences or just the important ones as they step by step see examples of and exclude types of difference that are not of concern.

All in all, it was a powerful, well-elaborated system. It is well-placed in its chosen domain and could probably be applied in others, wafer-fabrication not being the only task using robots, complex processes and schedules.

Tools and Process

VASmalltalk 8.0 and Beyond, John O'Keefe, Instantiations

I've incorporated all the material, discussions and questions from John's StS and ESUG presentations into his most recent one at Frankfurt.

GemKit, Paul Baumann, Intercontinental Exchange

GemKit was built by GemStone consultants for their customers. Later it was open-sourced at Camp Smalltalk. Paul has worked on it and version 4 was released immediately after this presentation.

His demo was of updating VW code from an empty GS/S database. He opened a vanilla VW image with GemStone client loaded (GBS). He loaded the GemKit code (connect to repository, load the version, very straightforward). GBCManagement does the main work. SystemUser represents the users in the GemStone dictionary. Each user can have their code maintained separately. Some extensions that GemStone needs (and that Paul will get added to the GemStone base release later) are supplied

Normally a dialog would lead you through installation but his demo database has GemKit installed. He opened the comparison browser and showed how the 'exclude similar' button lets the user ignore white-space differences. Whenever there is a difference, the browser shows 2 panes, one for the database and one for the image, with colour-coding of the differences. His demo had 6,000 pieces of code to compare (GemKit seemed fast and responsive, given this number).

He updated the code (there were few changes since the demo image and the database were well aligned) and the globals (many changes). Some VW prompters (e.g. for do you want to capitalise Globals coming from GemStone with lower-case names) are not yet hidden by Paul.

GemKit uses the original GSS class definition method. Later, he will allow exploitation of VW's ability to add attributes to classes.

He then went to an application database with code. He showed producing a patch that can apply a set of changes repeatedly. He first compared the code in the package (116 differences) then compared all the code in the Globals namespace (many more differences). A specific application would categorise (e.g. as GemKit, PerformanceProfiling, ...) and you update against specific packages rather than just update everything into the Globals namespaces unsorted (when you would be unable to distinguish general extensions from application-specific code and so on).

He then used the compare tool to examine specific changes and do specific updates. Then he logged in as DataCurator and looked at AllClasses. He reverted a couple of things that had been changed to be capitalised (because he had OKed VW prompters which in fact you would not OK) and otherwise the 'SystemUser' and similar code was largely unchanged, with new code in 'Managers' and elsewhere. In VW, he showed that the changes where he took the GemStone code were also in the appropriate packages.

If you do not commit, you lose your changes in GemStone but they remain in VW, which might sometimes be what you wish but usually of course you will commit.

When you get used to using GemKit you may well want to edit the bundle specification to e.g. move Globals out of it or otherwise associate your specific changes to your application bundles rather than GemKit. This is an appropriate part of adapting GemKit to your application's use of it.

He showed comparison of 26,000 definitions between GemStone and VW with an unprepped shared page cache. It took a few seconds: maybe as many as ten. You can reformat to make the code appear to match, format-wise, in both panes and scroll both panes together.

So far, he had showed developer tools. You can also file-out the entire symbol dictionary to be filed into GemStone via a topaz script by a database administrator. He showed generating a patch of differences and filing-out those (he demoed this for a package but you would more often do it for the entire bundle).

If you have a package with several integration-ready changes from several developers, all in different bundles, then they can be hard to find. A tool (the ReleaseAssistant) finds all such changes that are under a selected bundle and lets you select and merge to produce an overall set of changes to apply. He demoed using the release assistant to see changes from another developer, create an integration build of all the changes, apply comments

that will only be set on his own changes, etc. The tool warns if you include a change that the merge tool would exclude under 'exclude similar'.

Paul then showed how, if you have been lagging, you can apply your changes to the latest build. The setting 'make changes on the latest build' does this. He loaded a previous version, selected a method that was changed and demoed this.

Many people have two paths - a release path and an ongoing development path - and if they make fix changes in their release path they want to apply them to the development. The merge tool lets you apply changes in path to another path, with ignoring of white-space differences, etc.

Q(Dennis) Exception-handling? GemKit has approaches to handling exceptions and subclasses.

Q(Dennis) I have 3000 identical non-GUI subclasses? GemKit had a feature called mirroring which has not yet been ported to this Store-oriented version but could be got to work.

Q(Niall) Some methods that are naturally different (e.g. DNU has different implementation in GemStone and VW)? This would be handled by putting the 'don't make the same' code in a package or packages that you would know not to publish to/from GemStone.

Q(Dennis) I have 20 databases to be updated every night? Paul thought Dennis' existing process solution for this would probably use GemKit within the overall process rather than be replaced at top level by GemKit.

Q. Method deletion? Comparison tells you and will remove (old GemKit had an ugly 'remove everything and reapply': this has been fixed).

Q. Production users? Usually, only developers use the tool. The artefacts button is customised for apps; at Intercontinental Exchange they show a dialog to produce paths and etc. and where they were saved so production users find the patches where they expect and run them when appropriate.

Monticello, Colin Putney
(I had to leave before Colin's talk was due but he kindly demoed for me during lunch on the last day. Read these notes in conjunction with my 2006 Smalltalk Solutions report on Monticello.)

The UI is what is new. The Monticello 1 interface looked at the whole image at once and all the repositories you knew about: you had to set up repositories for each package manually. The new UI project ties together all the bits that go with Pier or Seaside or OmniBrowser or Monticello itself. All the slices, all the repositories are grouped and you save a snapshot to all the repositories (that you can see: you can synch with the rest later). When you load, you load from any repository where what you want can be found (checked in the order you added them at the moment; later he may add a strategy e.g. to take from local repository if available).

The next task is a much better merge tool. The merge command is 'include' to make clear that its semantics differ from what you expect in Store or Envy.

Italics mean an item is loaded, bold that it is not in the currently loaded item's history. The tool shows all different elements, in bold if conflicting.

The layout of the tool is diagonal: the bottom left pane shows the code in your image; the top right pane shows the code in the version being compared. This is unlike all other merge tools I know, but in all those other tools I find it is easy to get confused about which of the two lower panes corresponds to which of the two versions you are viewing. You can resolve in various ways and see immediate unbolding / whatever. Resolution code appears in the bottom right pane. You can edit the method there and save it; if you go elsewhere and come back, your edits are still there but not yet in the image. This is useful when you need a third option but must apply things atomically. You then apply all to image and snapshot to write to the repository. Applied resolutions are marked dirty so you know to save.

Normally, resolving and publishing decides the history; that your choice is preferred to its rival when merging is known to the repository. (However you can also apply a resolution without enforcing the 'my choice is the right choice' record. You can apply to image without creating history.) When merging later to a third version then if a method or class definition version in it superseded the version you chose the tool knows that and offers that as the preferred choice - which you can change but usually you will take it.

So far, this has been simply improvements to version 1's features. The project configuration slice is a new feature: Projects themselves are versionable. Colin distinguishes a project's repositories and its line-up: the slice of the project's contents that are loaded, the versions that are loaded. Colin demoed loading a changeset slice.

Documentation is to be written. He wants to write both a user guide and the technical definition. He is porting to GemStone. He also wants to port VW. Release will be soon.

Automating Smalltalk Builds with Cruise Control, Randy Coulman, Key Technology

This is half a how-to talk, half a report of Randy's experience. Key Technology build big food-sorting machines (see Smalltalk Solutions 2006 talks by Travis and Randy). 60% of their code base is in Smalltalk and it all runs on Linux. Builds should be automated because people make mistakes, especially when under pressure to meet a release deadline, which is just when you will be doing the build.

Continuous integration is an XP value. Every time you make a change, send it to an integration machine that adds it to the main build. It was introduced in Envy but proved hard to translate to more conventional tools in which branching and merging are heavier tasks. Thus a tool for CI is

needed. Key has multiple languages and their CI tool gives them an .iso image (of their C, C++ FastScript, Smalltalk, Graphics files, etc.) that they can burn on a CD and ship at the end of every check-in. They've been using ANT for 6 years. ANT drives CruiseControl.

He opened a VW image, published a couple of packages (Key do not use bundles) and showed the automated build. It starts with a quiet period when the system things "he's just published so maybe he's still publishing", after which it kicks off the verbose script. He opened his mail and saw his build, including a warning that a remote tester that he cannot see at this site was not reached for some tests. (They archive these emails for years back, sometimes useful for reconstructing when things were done.)

CruiseControl was an open-source project written in Java released by ThoughtWorks back in 2001. It has a pluggable architecture. It has a build loop, a JSP-based recording application (he opened a web page and showed his project). He thanked Arden for publishing James' build scripts which have been very useful. He went over them carefully (James: "They work just as well as my code." Randy: "That's why I went over them very carefully." :-)) and a dashboard (newer; he's unsure whether he likes it or not since the information is in a more attractive format but possibly less usable) that shows when your builds last passed.

Travis built a Smalltalk system which Randy changed to work with cruise control for working with the rest of their code. CruiseControl 2.7.2 has Store support: Randy wrote it and the project added it the same day he emailed it to them.

The build loop checks for changes, builds ("compile", run tests, create debian packages, etc.) and publishes the results. Their C, C++ usually has make files so "compile" is calling them. For them, publishing the results is an email; you can make lights flash if you want to. They configure via XML; he showed the XML file of the build instructions.

The raw XML has a lot of duplication and there are ways to reduce this using preconfigured XML entities or whole projects. He showed the refactored file that used these. Two pages reduce to 12 lines or so.

CruiseControl lets you do remote control via Java Management Extensions (JMX). The UI is ugly but it lets you force builds, or pause the build while you put something it needs in place or do some maintenance without which it would fail. He forced a build and showed the email.

For Smalltalk builds he tried to follow the standard Cruise Control expectations. He checks Store for changes (StoreForGlorp made this much easier than it would have been in standard Store). A version regex exploits their versioning standards to find what it needs. A file is used to cache the Store change information so he does not have to refind it at each point in the process. It is ordered by dependency, so it is a valid load order. The search is with reference to a particular timestamp.

He showed their prerequisite graph tool (thanks to Martin and Travis for cleaning it up) which is in the OR.

A TestLogger package runs the SUnitToo tests (or it could run SUnit of course). It outputs in the format that JUnit uses because CruiseControl knows that. Then they run Fitnesse tests (see his tutorial).

His slides show useful links, including his blog for more information on all the above.

Q. Determining which version to load? We load the latest that matches the version regex. You can set a minimum blessing level as well.

Q(Peter H-M) Stripping? It can and they do a little of it. They build on a base image from which stuff they do not need are mostly stripped anyway.

Travis is using some of this stuff and extending it to use bundles.

Q(Peter H-M) SUnitToo? Travis' variant of SUnit that holds tokens, not whole test cases, so avoiding delaying GC.

As an experiment, he built a variant that ran all his tests every time he accepted and he likes it better than he thought he would.

AIDA and Seaside

AIDA/Scribo: a powerful CMS at your fingertips, Janko Mivsek, Eranova

Sadly, Janko could not be here so Martin Rueger gave the presentation. AIDA is not Seaside but it is a powerful web framework with many capabilities. Scribo can be compared with Pier. It is a content management system built on AIDA. Janko tries to have *everything* is in Smalltalk, even things like proxy path handling. It uses AIDA's strengths: an MVC model, REST-style interface, built-in security, components and AJAX. The REST-style bookmarked URLS are important for CMS systems.

Scribo is well suited to blogs, wikis and complex document-style sites. Developers can use it to do many things. AIDA runs on Squeak and VisualWorks. Scribo is mainly exercised in Squeak today but should work in VW no problem. Scribo was inspired by a prior system called BiArt.

Components in Scribo are called scriblets. The core of Scribo is a Document. Versioning is built-in and many versioning schemes are supported. The rest URLs make it possible to interact with the versioning programmatically and/or webwise if needed.

Documents have lifecycles. Scribo provides a range of states e.g. pending, redo, approved, released, obsolete, so an organisation can realise its process in the CMS. Folder is a subclass of document, Documents have chapters, horizontal and external links, etc.

Scribo has good multilingual support, mapping user language to display language and locale.

Scribo is in alpha currently. Plugins are generic Wiki with subclasses Blog (very complete) and Website (less complete at the moment, so much like its Wiki superclass).

Scriblets, components embeddable into the text, are how you extend Scribo. There are predefined scriblets like Gallery and Table of Contents, and developers write custom scriblets for what else you need.

Aida grew out of legally-valid-archiving work done by Janko years ago.

See Scribo on the web at nico.bioskop.fr (blog), Squeak Project Manager, the BiArt/ISO quality management system, www.swazoo.org, www.aidaweb.si and www.nets.si (a commercial website that shows templating scriblets).

Future work: Janko will also do work in VW and his version of GLASS: GemStone, Linux, Aida, Squeak and Smalltalk. There will be more plugins and more scriblet sites.

Michael then demoed, launching the server in Squeak then going to his web browser. He logged in. Yesterday he had to set all his permissions since that is not automatic yet; that will be one of the beta things. He went to the blog settings page and then wrote a blog post (using the same rich text editor that other systems use).

Then he went to the wiki. It uses the same text editor, lets you add links using a typical smalltalk wiki syntax (WikiWorks IIRC: links are written [a link title>PageNameOrURL]. He created a page with an image. The settings page lets you configure the virtual host and other things that other systems would leave to Apache but Janko lets you do in Smalltalk. You can still do it in Apache instead if you wish but his replacing of the A in GLASS has reason. Other settings let you control access rights and see statistics for your site. Thus you get a server with everything in it, all very easy to find. Martin feels that is where this stuff shines.

Q.Is this for platform applications or for general web stuff i.e. more static stuff? Rob Rothwell explained that it has AJAX built in and Nicholas Petton has been working on Scriptaculous.

Q. Seaside scriblets? Janko is thinking of it. The different component models of the two systems mean a framework would be needed.

Why Smalltalk? A Healthcare Perspective on Creating Internal Domain Specific Languages, Rob Rothwell, Fairfield Medical Center
Rob Rothwell is not so much a programmer as a user who users computers to solve his problems. Janko helped him a lot to get a web framework on his app and he found he'd agreed to present.

50% - 60% of a hospital's income comes from medicare and medicaid. They lose $3-5 million per year in their emergency room since people go there instead of visiting their doctor. They would like to save money.

All the advances in medical care means increasing specialisation. The result is less holistic medicine. This in turn leads to complaints and this leads to government regulation: prove you gave the heart attack patient an aspirin within 24 hours. This is called 'transparency' but it is a lot of work for the hospital and it is 'voluntary' but income will depend on your having tick marks instead of red Xs here, there and everywhere. Then the insurance company decides to use the government's scheme but with differences. So he has to generate that information as well.

Having achieved this, he has to send the data through a vendor, not directly (legal requirement lest he fudged his data) so 3 months elapse iterating this data through the vendor and then they learn they failed, which is too late.

How did he get into this? Well when he left the army he wanted to find somewhere he could make a difference and this is where he arrived.

Everyone in the hospital is very IT challenged. This is not a criticism; it is the situation. Times are written down on forms and they do not match the times in the system and a project is needed just to match these for some data.

All the above is an opportunity: large IT companies supply monolithic systems that are too rigid here and too customisable there and people in the hospital use 10% of it at most because they have not time to learn the rest or it does not really match their needs.

Example: he tried to measure the cost of a surgery case. To know the cost you have to see what was used and know what each item cost. They collected all the wrappers and tried to match them to the names: an item can be called an adaptive dressing by one person and a non-occlusive dressing by another and the database has a third name that matches neither of these anyway. Is this a 1 inch needle or a 1.5 inch needle? People run out of the room and get new items quickly; if time allows they scan it, but if the patient is in a serious state they're too hurried so you get the barcodes from the waste bin.

One provider provides barcode scanning. It takes many screens and mouseclicks to get from the front screen to where the barcode can be scanned. Then you must find the right keyboard wedge to scan and ensure you're on the right place on the screen. This is hopeless both for the technically-challenged workforce and for the timescale.

People collect all this data and put it into Word, from which they copy it into Excel, then put it into Word and send it to someone who then puts it into Excel and so on and on ... That is how this reporting ends up. As for development staff, there's him and another staffer, and a guy who must be given very clear instructions.

So why does he want a domain specific language? Because he has a set of very domain-specific problems.

They must report that antibiotics were given 24 hours before surgery and discontinued 24 hours after surgery. Start time for surgery might be defined as when the surgeon entered the room in one scheme, as when the first incision was made in another, etc. The time window might be 24 hours in one situation, 48 hours in another, etc.

Excel is the most prevalent business domain-specific language. When the name of a column is changed, mapping between data does not work so well but it is customisable by the users. Everyone else cares that two patients got the same care but they want their specific nuanced data on that specific patient.

Smalltalk is a generic domain specific language. He can use it as is and customise it bit by bit to his needs. He needs transitional data storage. Healthcare has 30 years-worth of systems from which he must extract data (without bringing them down because they're still using them on live - "and we'd like them to stay that way" - patients) and unify it. To him, a data warehouse is just another data source because he's not going to get everything he needs into it anytime soon.

A web interface can be made visible near to the person who has the data. A fat client is also possible. Dabble DB would be interesting if it could be brought in-site; they are not going to put medical data on an external server.

Rob believes that documentation tools for healthcare could be better but will always be poor. Rob wants a nurse to see a screen that look like a patient so she clicks on an elbow and sees what elbow info to record. Ordinary people need extraordinary tools; that's why he came here, to meet the people building such tools. People need to program. "Change your process to match my software" is what everyone says but it never works.

Smalltalks collection classes alone save him. Someone comes to the ICU then gets sick and goes elsewhere then comes back to the ICU and so on. It is hard to query a DB to get the intervening times matching this process. He can write a simple program on collections in Smalltalk to explore that.

ODBC is a must in Healthcare because someone has always bought someone else's custom system and you have to suck out the info.

Thus his phase 1 task was a generic web-based data extraction tool. Phase 2 was preloading all the info; don't make people add the patient number *and* their name. Phase 3 is knowing whether the data will pass or fail against the government's schema, against the schema used by Leapfrog (a healthcare company), etc. He implements this by subclassing general measures to capture that Leapfrog has 48 hours where the government has 24 and so on.

He then demoed his phase 1 system running on Aida. He has a Data

Abstraction Page Designer in which he can add checkboxes ("Checkboxes are real big in healthcare") and menu designers. The result is all in the database and refactorable; you can rename headings and suchlike. Next he needs to bind these dynamic things to his changing requirements, Next year they will have 70 new core measures on which they will have to report "voluntarily, so we get paid".

The one good thing is that the numbers are finite. Congestive heart patients, 300 per month, are probably their largest single category.

"We don't face a health care problem in the US; we face a health business problem - socialist beliefs with capitalist payment."

Q.(Georg) How did you meet Janko? Rob was exploring web issues and Janko was superb at answering his questions.

Everything in healthcare IT is based on, 'Is it free', 'Is it cheap', although somehow they can always hire 3 people to type data whenever the IT is lacking.

GLASS: Share Everything, Dale Heinrichs, GemStone
After a few minutes, the screen projector was persuaded to take notice of Dale's machine and he opened a Firefox and a shell.

Avi's blogpost on GemStone architecture and Ruby got a comment saying 'Yeah these shared caches are OK but how do you do share nothing? That's a technology that's been around since 1994'. :-) (Later the comment got edited or deleted.) Share nothing could be characterised as 'Hit the database every time you need anything.'

GemStone/S persists every reachable object. In Seaside, they extended the framework to do an abort when an HTTP request is received and a commit before an HTTP response is sent. Thus users get the behaviour they expect without having to do anything more than they do already. There is no need to embed transaction logic in their application.

Scalability for Seaside means being able to handle more requests by just adding more resources. The aim is to go from a single app to Apache running multiple Apps without making any changes to the application.

Every object in GemStone has an oop (an integer) and objects are on pages. Thus object 1 on page 1 may have a reference to object 3 which lives on page 2 (real oops are a bit larger of course :-). The shared page cache means multiple VMs can use the same pages. An object graph can be partially loaded (for the very large models possible in GemStone, models must be, as the entire database will never load into memory at any one time). He showed object 1 being loaded with its reference to object 3 which if used would then be loaded ('faulted in' in GemStone's parlance).

Object tables map ids to actual locations on pages. A VM can load an object table (call it object table 1) and then makes changes and commit

them, so one VM can point at an object table (call it object table 2) while another points at object table 3, each valid different entire views of the database. The changed object 1 can be written to page 3. The old object 1 on page 1 s invalidated when no VM's object table is still pointing at it.

Commits can conflict of course and the loser in the commit race must handle that. In Seaside, if the automatic commit before the HTTP response fails then they resend the HTTP request. This aborts, restoring the state when the request first arrives, and so the request gets the same behaviour as before. In effect, it is just rearranging the times of the responses and will normally result in users getting times that do not overlap. Of course, at very high volumes, or with users who are having very long transactions, you may reach a situation where you want to provide application logic for commits. Every 3 minutes, the maintenance VM checks for sessions that have exceeded the Seaside timeout. It also does GC.

The Seaside data is shared across all VMs so you do not need to use session affinity; any available VM can handle the next Seaside request and will have all the data it needs to do so.

Three months ago, they changed the tools environment to make auto-commit the default. In development, every accept commits. The standard log is big when you are running 5 VMs and even bigger when you are running 100, so Dale added an RcQueue object log with a Seaside interface focused on the data a Seasider will want to see.

Dale then opened the tools to demo debugging. GLASS likes the Omni-Browser tools since, unlike GBS, all the objects live in GemStone and the tools act directly on lists of them, not on replicates of them into the client.

Next there was the standard demo hiccough: Dale found that his demo counter was incrementing when he decremented due to work he was doing this morning and forgot to revert, so decided to use fixing that for his demo. He set a breakpoint, saw a standard Seaside walkback, went to the GemStone transcript window in the tools, clicked on the debugger, selected the appropriate context to debug (only one in this demo) and saw the debugger. The code can be saved in the debugger but hit refresh and you still see the old code; things like this are why this is still in beta.

He then showed the object log in the web browser. It showed which VM handled the request, the oop of the object, the continuation (reified and persisted) and the process, etc. When they debug, they persist the process so when you hit resume it finds that process and resumes it. Thus you don't have to think about whether you have 2 VMs, 8 VMs or whatever, you just get the process you want.

He did a halt and showed that his drop-down list on the debug button had two items, the halt and the prior breakpoint (which no longer had the up-arrow because you can debug it but you no longer can resume it, the infrastructure for it being no longer around).

Seaside is stateful and can generate a lot of state while you are running. Both he and Colin Putney independently decided that 2.9 is going to reduce the amount of saved session state.

GemStone 3 will allow non-tranlogged objects since tranlogging is done for recovery and session state is not wanted for recovery. This will happen automatically when session data is put in a different global dictionary. Questions raised the point that if users could see this global dictionary they would find uses for it.

In 3.0, exception handling will be ANSI standard. 3.0 has plans for Native methods and Foreign Function Interface; these are also interesting to GLASS. Sharding to address performance, there being a limit to how fast a commit can be done, will be looked at.

Q. VMWare appliance? That is their preferred way of distributing GLASS. They are working to extend the installation possibilities but for now that is what works most easily so they recommend it.

Q. Non-Squeak client for the tools? If you have VW or VA and GBS, then you can do much the same thing. They have not yet ported the Squeak tools to VW and VA, but on the other hand perhaps you will prefer to use WebVelocity or suchlike. Moving the OmniBrowser into GBS is being considered and certainly can be done technically.

Monty pointed out that a Seaside app developed elsewhere can be deployed to GemStone easily - one customer did this in literally one minute. GLASS' aim is to create a pool of Seaside-competent programmers who will then work on larger apps and drive the growth of this market.

Q(Paul Baumann) when working on this kind of thing a while back Joe Boscancas found some cases where you wanted a request that did abort but never committed. Do you have that? If no objects are written, that is the effect you get. That's why Dale is working in 2.9 to reduce unnecessarily-saved session state.

Q. SeaBreeze, WebVelocity, GLASS: how do they tie together? James and Dale both answered. They are all written on the same base Seaside framework. Genuine write once, run anywhere.

Building a Seaside Application with GLASS, James Foster, GemStone
James explained some features of the GLASS licence. If you go over 4 Gb, your server will shutdown. You can contact GemStone and get a one-week GLASS licence for 8Gb. During that week you will either realise that overflowing 4Gb was an accident and slim your DB under the limit again or else realise it was the natural growth of your app, whose revenues should therefore also have grown to the point where arranging a standard licence with support is both affordable for you and the right thing to do.

I went to Vassili's talk, then arrived and paired (by prior arrangement) with Werner Wild. Werner and the others had installed the GemTools software

from the DVD James supplied. Those who had VMWare ran a local database, the others got user info and connected to James server machine. Everyone brought up Seaside, saw the counter example, added a breakpoint and saw the debugger, and connected to GemTools. At that point, some network problems had to be sorted out. Conveniently for me, I arrived just as they had been fixed, with Werner and Angela having GemTools back up and logged in, the rest doing this; work resumed.

James switched to a more demo-oriented presentation, with those who had the VMware able to work locally, the rest observing, and everyone got started again on the exercises. James started servers for the course participants; each participant's account ran with its own set of source code so each user has their own view of source code. Thus we again saw the counter app with its halos and the usual behaviour.

Note that Squeak is not always 100% reliable. GemTools will occasionally get UI errors and lockups. When it happens (it has happened to James every hour or so at times) just close Squeak and restart. It may be a GemTools issue or it may be Squeak or a combination. Gemstone commits every time you save a method so you will not lose anything except your latest edit.

The next exercise was the mini-calendar in the test suite. A walkback had been inserted into it and we had to fix it. The walkback appeared in the debugger. Werner selected 'remote debug' and at first could not see it in the drop-down on the Gemstone transcript widget but in fact he had just been too fast (large course class all running on small local server meant you sometimes needed to allow enough seconds). He fixed in the debugger but then could not proceed because the problem was a bad receiver (code was trying to treat the integer 6 as if it were the month of June) so we had to drop that session go back to the tests and run.

After looking for WaMiniCalendar in Squeak and not finding it James explained to us that we should look in GemStone; select the GemStone Transcript and go from there. Squeak is not a host for Seaside, or a programming environment, as far as GLASS is concerned. It is simply a source of GemStone windows. (BTW, see James' Dolphin equivalent). Thus we fixed and resumed.

Then the server / connection crashed again and James restarted. It crashed repeatedly. Eventually it dawned on us that our attempted fix of the method

```
year
^ self year
```

was likely to cause stack overflow and crash the server, after which we fixed the problem instead of making it worse (the WaMiniCalendar was initializing `month` to an integer instead of a Month). In the Squeak base time classes `Date today month` returns a Month object. In the GemStone base classes, the same code returns a SmallInteger, thus making it obvious where the bug comes from; the raw port from Squeak to GemStone of the mini-calendar hit this base class discrepancy.

We opened the chasing browser (hit escape when a class is selected gives you that menu). Next we went Seaside web page -> tools -> object log and browsed all our crashes and other activity.

Now we were ready to create our first Seaside component: a web counter to monitor the huge hit rate our amazing app will soon be receiving. A few methods added and our counter could render all the content it could detect. While doing this, Squeak locked up for us and we restarted it. As James had promised, all our work was saved in GemStone so we simply had to open a browser on our new class and continue work. After a couple of hiccoughs, we managed to execute and commit the `registerApplication:` call for it and see the number zero.

Seaside Tutorial, James Robertson, Cincom
James showed the counter, the simplest Seaside app that could possibly work. This very simple class shows several Seaside values in a nutshell; no marshalling and unmarshalling of state; instead callbacks give a fat-client GUI feel. He set a breakpoint and walked through the debugger. Refactoring is far easier than if the logic were scattered across true code, code in templates in files, and etc. The workflow is a lot easier to follow.

WAComponent's subclass WACounter overrides `renderContentOn:` to paint the canvas. Set `canBeRoot` and `registerAsApplication:` on the class side to define your top-level component.

First exercise: create a component that does 'Hello world'. Werner and I noticed the guessing of protocols in vw7.6; it notices when the method is in a protocol on a superclass or elsewhere.

We then loaded a domain model for a blog, plus some posts, and then built a blog server to show it on the web.

After building the blog website we then added another entry point. If the user cancels when invited to login or is not allowed to post, we want to show them the blog view. Otherwise, we will offer them the chance to post to the blog.We needed a form

```
html form:
  [...
  html textInput on: #username of: self user.
  ...].
```

and a call. Call is like invoking a new UI.

(We managed unintentionally to set our BlogViewUI as the top level component instead of the standard Seaside, probably by setting it in the configuration.)

Lastly we discussed styles.

BoFs and Contest

Smalltalk Coding Contest

The first round of the Smalltalk coding contest is described in detail at http://www.cincomsmalltalk.com/userblogs/niall/blogView. The winner was Rajesh Jayaprakash (who was using the contest to learn Seaside and had not used it much beforehand!).

As Rajesh was unable to get to Reno this year, we invited him to come next year and organised a second round in which contestants had four hours to help Christopher Columbus rework what were columns and what were values in the spreadsheet he had prepared for the Queen of Spain. Martin McClure won, Peter Hugosson-Miller came second. Martin could be seen afterwards beaming as he mastered his new iPod touch. I understand his wife was also delighted; she will now finally be able to get her hands on the one she won in a promotion some months ago. :-)

STIC meeting, Georg Heeg

STIC's main task for the last few years has been to organise this conference. Georg was elected executive director a year ago. It is a most honourable job, i.e. it is not paid. (Suzanne: "You've been not paid for a year. We've been not paid for much longer than that.")

STIC has worked with Gartner to get an appropriate positioning of Smalltalk and that report is now extant. Georg has found this one of the nicest Smalltalk Solutions for several years and thanked Suzanne, Joy and all the people who made it so. James is their webmaster and STIC's website is now much improved.

Monty thanked Suzanne and Alan and Joy for all their hard work, and Niall for the contest. John O'Keefe echoed that; several people have commented to him that the hotel is a great facility.

Suzanne discussed the initiative of 2006 and 2007 of sharing Smalltalk Solutions with Linux world and Network world. It did give us visibility. Suzanne started hearing from old customers. Unfortunately, they changed their direction dramatically from Linux world to IT360 and while we tried that for a year it was not such a good match so Suzanne and Joy decided to take the conference back. This location with its one-stop shopping, free shuttle and OK price was great. Nevertheless, she feels the buddying to other conference has paid off. Niall agreed and stressed that we should be prepared to combine with other conferences again on occasion.

James mentioned that some East Coasters will be glad of moving around (i.e. nearer them). Alan and Suzanne said Seaside should of course be near a beach. Suzanne noted that Reno was not easy to get to for some but when you got there you did not need to leave the property and that is a good point for Smalltalkers.

Discussion of what conference we might buddy up to; a Ruby conference, an Agile conferences (but they tend to be large). Eric has suggested if we cannot buddy to the Ruby conference could we be in the same property or

down the street (and it would also give Eric less travelling to do :-).

One customer has hired 15 Smalltalkers in the last two years. The customer had decided to rewrite in Java but now backed off after a few years of unhappy experience. ObjectStudio is also seeing a lot of growth.

Q. Numbers? Suzanne noted that publishing Cincom financial data has been a sensitive issue in the past. Someone mentioned that having the ability to navigate to those numbers, even if they were not too obvious to random browsers. If management do quick research on Smalltalk they see that IBM sold VA to Instantiations and Dolphin went under. Numbers would help. Suzanne noted this message and said she would see what could be done to put info on their site and the STIC site.

James pointed out that GemStone, Cincom and Instantiations are all investing in Seaside, something that is not focused on the existing customer base; this is proof of something. Monty noted that GemStone has seen their Seaside efforts have paid off promotionally and privately. They would not publish data on specific customers but could work with STIC to provide an industry measure.

Q. Publicise Cincom use of Smalltalk? Suzanne noted the request.

Like Cincom and GemStone, Instantiations is a privately-held company. However they can confirm they have had substantial revenue growth. Instantiations did work with Gartner to provide info recently. Cincom also worked with Gartner and that will be a route to get info into the domain. Arden noted that Gartner has moved Smalltalk from elderly to mature.

Q(Thierry) He was very pleased that vendors are open to SNBs which previously he had felt not. John OK noted that IBM of course was not interested in SNBs whereas Instantiations attitude is much more positive.

Seaside BoF
Carl Gundel demonstrated Run Basic. He showed a hangman program written by someone with no prior experience of the web whatever.

They have sold several hundred copies since the start of this year. Their forum has 200 members. Users are very happy with it. Their other product makes them regular money but it is not unique; it competes with other products. By contrast, this app is unique. They could not have done it without Seaside. It was seeing what other people had done with Seaside that inspired them to try it.

He has not yet tried any of the AJAX stuff. He wants to use it but feels he does not fully understand the examples. There is no STUG in Boston and Seaside users in Vancouver are a bit far to visit.

When he started this, a frequent question was ‘Why are you creating a web basic?” His best customer asked that before she saw it. After she saw it, she saw the answer.

Peter remarked that he met Seaside for the first time yesterday and liked it.

Colin Putney has OmniBrowser running on the web in Seaside. His start page is one button but will become something like the VW launcher. It shows a Mac-like icon list and one of these opens the OmniBrowser, or would except for a DNU. As we were overtime and the GemStone BoF was ready to go, he decided to complete the demo later.

GemStone BoF, Norm Green, Martin McClure, Monty Williams
Norm presented the GemStone 64 server product roadmap. Martin talked about GBS and Monty outlined the Ruby Maglib work they have done.

The 32 bit product is being wound down as almost all their customers have expressed intent to move to 64. They have just released 6.3.0 and 6.3.1 is due by the end of this month; it's in QA now. He hopes this will be the last 32 bit release for a while (but probably not the last). The current 64 V1 release is 1.2.5. Major customers have moved to it and others will shortly. The next V2 release is scheduled for October 3rd.

They've added Solaris 10 and HP Integrity (Itanium 2) to their supported server platforms. There is also an Apple Mac (Leopard, Intel) non-production version available to customers who ask for it.

Gemstone 64 2.3.0 will have PersistentSharedCounters (like shared counters) for apps that need to generate new ids and similar. The protocol is the same as SharedCounter with larger range (signed 64 bit) and atomic persistence. StatMonitor can be run without running a GemStone and monitor any process and all processes owned by a given user id. All the host system statistics can now be accessed from Smalltalk. GsFile can now read and write compressed files (uses gzip library).

A user-profile can now be made read-only programmatically by sending `disableCommits` to it (so e.g. could respond to improper operation by preventing user committing it).

It will support direct I/O, bypassing the Unix buffer cache, a performance win for certain cases.

GemStone 64 V3 is still in the planning stage. It will have native code support, which can give 2x speed-up, and multi-threading support for tranlog replay, so restarting will be faster. DLL access will also be added.

Various other things might be added. Multi-thread global GC (single-thread takes weeks for 3 billion objects and the offline is better but can still take days). They know how to do this. Several other operations would benefit from multi-threading. They would like SSL socket support, SNMP, LDAP authentication, Web service classes and automatic load balancing across several shared page caches. They have customers who run large pools of Gems across several caches and there is no easy way to decide which cache a Gem should be started on; the system knows which is least loaded so could allocate sensibly.

Q. Proposed date for 3.0? Norm's proposed guess is the end of this year and he is sure it will be released earlier in 2009 than this date.

Martin McClure then spoke about GBS. Martin's slides were yellow with blue background on his screen but orange with black background on the projector. Fortunately, they were still readable. Since the last Smalltalk Solutions BoF they have released 7.1.2 (bugfix and support for VW7.5) and 7.2, which had performance improvements, configuration changes and single round trip. It is only supported on GS 64 servers at version 2.2 and later. It reduces several multi-trip operations to single-trip operations in forwarder sends (it also avoids updating unchanged things) and in remote evaluation (`evaluate:` calls). It also used to be the case that you chose lazy versus immediate faulting; since the new single round trip eliminates the other trips where lazy faulting would piggy back, they've eliminated this feature since immediate faulting will always be faster.

This year, in GBS 7.2.1, they are working on performance. On Monday, they issued a release candidate that is twice as fast as their old benchmarks, and this is older 32 case compared with newer 64 case, so handling 30% more data in the benchmark. This is the fastest GBS they've ever released. He's hoping that some performance techniques users have tried in the past can be discarded now that GBS can replicate > 100,000 objects per second. They got this performance by replacing C memory access (5 times slower than accessing a byte array in VW) with pointers in object space to avoid that primitive. The other technique was to streamline the code.

In 7.2.1 they only support single trip so it only works against Gemstone 64 2.2 and later. They will then release one where multiple round trip is reenabled so you can get the other speed-ups in 7.2.1 against older servers. Eric will work on completing the latest VASmalltalk GBS when he returns from this conference and that will appear soon (yes, they know it is late).

Other future features. Concurrent traversal of buffers (get first buffer, ask for second buffer while processing first) uses 105% or 110% of current time so with multiple CPUs it will reduce the elapsed time.

Forwarding replicates that mix and match which methods they forward and which they handle locally is coded but needs tests before it can be released. Uses include DNU forwarding (if the client doesn't understand this, try the server) and one-time forwarding.

Q. Relation to uncached forwarder? Uncached forwarders are an abomination. That some people use them is what drove him to work on ForwardingReplicates. It is a far cleaner implementation of the same intent. He will give people time to migrate from them before eliminating them. (And if anyone thinks they use them in a way that ForwardingReplicates does not support, please tell him.)

Q(Angela) When released? Before the weather turns cold again.

Q(Angela) Supporting VASmalltalk 8.0? A GBS release has about 100

supported configurations so they are keen to limit it to what users use. If 8.0 is out a few weeks before their release they will support it as well as earlier VASmalltalk versions. VW7.6 will be supported by GBS 7.2.1

Monty then talked about Maglib. Avi suggested that Ruby is close to Smalltalk and that Gemstone would make a good Ruby VM, existing ones being not so robust. Monty went to their conference and counted that there were a lot more of them than us but they are a lot less sophisticated: noone here need consider giving up Smalltalk for Ruby.

Monty showed the 3.0 VM running Ruby and another window running a standard Ruby VM. He had to load stuff just to let the standard VM compute a 3000 size fibonnacci without crashing. He started a big fibonnacci in this and then opened topaz on the other and started Maglib. In less than a minute they had passed the slowly scrolling standard VMs benchmark and reached completion.

People in the Ruby world thought what they and Avi were showing was magic whereas it was very ordinary to Smalltalkers. There were a large number of people who didn't get it - thought it was 'hung up on performance' - but some of the cleverer ones were very impressed. However there was also the fact that Ruby apps are very Rails oriented with the ActiveRecord well embedded, so dropping SQL for another persistence approach will be a stretch for them.

3.0 uses the same VM for Gemstone/S and Maglib.

Kent gave a keynote to 2000 people talking about 20 years of Smalltalk and unlike a Java conference the general message is very pro-Smalltalk. "Work with the Smalltalk people because they know what they're doing and you don't." It was nice to be at a conference where they thought Smalltalkers walked on water (correction from Randal Schwarts: "Smalltalkers don't walk on water; they send a message to water to walk on itself.) However Monty noted that one thing Rubyists seem to care about is to be able at any time to hit a button and reload all their code from text files, so the image is something they find weird.

Q. If Smalltalkers walk on water, do Rubyists plan to become Smalltalkers? These people came from Java or Perl. What Ruby gives them is a great improvement on what they have. If they tried to become Smalltalkers right away, their heads would explode. Chad Fowler gave a talk in Toronto where he said the innovation of Ruby was that it was not innovative; people can go to it easily. At the conference, someone asked if Ruby and Smalltalk can run in the same VM and the answer yes was taken very positively. If they can get over the 'have all code in files' issue they will be home free, but that is a big hill for them to climb.

Q(James) Is it that people who sign the paychecks are the hold up? Those people know that traditional ways of building web apps don't scale. However Rails apps have their persistence in Oracle so the manager can sleep at night. The decision to use GemStone persistence as well as the

GemStone VM will be *their* hill, to climb.

Other Discussions

Adriaan and I dined with John on Thursday and Ed on Saturday. The IT market in the US grew by 8% in the past year; Instantiations saw more than twice that growth in Smalltalk.

CS13 and ESUG 16, Amsterdam, August 23rd - 29th, 2008

For the first time since Southampton in 2000, I was in the same country as ESUG before it started.

Summary of Projects and Talks

I give the Camp Smalltalk 13 projects summary, then the ESUG activities reports (including the awards presentations and ceremony). Next I summarise the conference talks, sorted into various categories:

- Applications, Frameworks and Experience Reports
- Development Tools and Techniques
- VMs and Smalltalk Environments
- Aida and Seaside
- Modelling Tools and Techniques

followed by the 10-minute talk track and Other Discussions. Talk slides are reachable from http://ww.esug.org.

Camp Smalltalk 13

Camp Smalltalk 13 ran for Saturday and Sunday before the conference, and during the conference breaks, afternoons and some evenings of the five conference days. More than 40 people attended it. There was much activity in the 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.

The Custom Refactorings and Rewrite Editor Usability Project
Michael Prasse worked on a tree widget for the class pane in the VW RB. TreeModel(SequencableCollection)>>includes: only reports true if the collection contains the item; if the collection contains an object whose children contain it, it returns false. HierarchyNavigatorPart>>fillInState: gets the hierarchy class of a new class, sets the widget to display it and its parent path, and then select:in: calls includes: on the list while mapping object root names to objects - and fails to find the class its state demands. He worked on a fix for this subtle interaction.

Adriaan van Os and I ported the Extract with Holes refactoring to VA, then looked at making its dialog a code tool as an alternative mode of providing refactoring UI. Reinout gave us user's feedback.

PostgreSQL EXDI
Bruce Badger and Michael Prasse worked on an issue with nested queries

in the PostgreSQL EXDI.

Seaside Applications
Dale Heinrichs, Lukas Renggli, Adrian Lienhard and several others worked on Pier and Magritte. Work was also done on the Seaside 2.8.3 release. (2.9 by end year is hoped for, not certain.)

COLA 86_64 code generator
The title says it all: Martin McClure worked on an 86_64 generator for Ian Piumarta's COLA. In the early stages, when the generated code was seriously rough, he had a hard time persuading the debugger to open on it until he realised it needed to be given two addresses; with only one, it was unsure where the function might end (or whether?).

Amelia
Primary school teaching tools using Squeak and Croquet were worked on by Filipe from Portugal and others.

Moose
Tudor Girba and quite a few others worked on a range of features. One was making the IC framework (Seaside bar charts and similar) less dependent on VW-specific features so it could be ported to other Seaside-using dialects.

SqueakNOS (no operating system)
Leandro and Valeria were among those working on this. Sadly, Gerardo Richarte's talk on it conflicted with Alfred Wullschleger's and I missed it.

Other Projects
Breakpoint logging in VW.

ESUG Activities Reports

Introduction to CWI, Paul Klint
CWI was founded 1946 as Mathematics department, broadened to other disciplines including software engineering. He summarised language development LISP, Algol 68 (remember that; I programmed in it once - once was enough :-)), and in 1972 Smalltalk appeared. Static typing was fashionable for a long time. Python appeared in 1991(Guido van Rossum started it as a project there and still visits CWI regularly). CWI researches these and many other languages. Currently, they do much work on meta-programming: analysing old programs (e.g. in Cobol - he hastened to assure us noone at CWI actually writes any Cobol :-), transforming them, etc.

They do much research on parsing. There are still problems in the parsing field: compose two LALR context free grammars - it is not guaranteed that the result is LALR so you need generalised context-free grammar parsing. He showed screenshots of some of their tools.

Conference Welcome and ESUG Activities Overview, Stephane Ducasse, Noury Bouraqadi
Stephane thanked the sponsors (see their logos on http://www.esug.org/conferences/16thinternationalsmalltalkjointconference2008); he is really happy to receive a request for sponsorship from someone he has not asked. He also thanked the local organisers: Adriaan van Os, Rob Vens, Mathieu van Echtelt and all those who helped. This is the largest ESUG conference ever: 172 participants (max 150 on any day) 17 student volunteers, 43 talks (occasionally 2 parallel tracks), 22 award competitors and 4 books to buy.

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

ESUG sponsors Smalltalk in various ways:

- ESUG can sponsor presentations of Smalltalk, i.e. pay travel expenses, etc. For example, Stephane gave a lecture at Turino, the temple of type theory, and they were positive about Smalltalk (noone left and he was told that students walk out if the lecture bores them). They also had an RMLL'08 booth. ESUG offers material for giving Smalltalk lectures.
- Via the Summer of Code, ESUG sponsors students to do projects. This year there are two projects (one started, one about to start)
- ESUG sponsors free Seaside hosting (handled by netstyle.ch).
- If you get a Smalltalk article printed in a magazine, ESUG will give you 100 euro.
- ESUG helps students who move to Smalltalk groups. They also help people who go to conferences (will pay 150 euro and get ESUG on slides): one this year was rated 'best talk in conference'.

If you want to do any of this ask: they will evaluate and let you know if they will sponsor. They also sponsor projects: DrGeo (maths teaching for kids) has scripting in Smalltalk (its scripting was in scheme) and was sponsored by ESUG.

Attending ESUG is how you sponsor all this. Next year, they are thinking of holding ESUG in Brest or Barcelona. Any interested would-be local organisers please contact them. Noury thanked us for attending the conference, especially MediaGenix who sent 13 people.

Lastly, a new board member is wanted: ask Stephane what work you would be signing up for.

Presenting at ESUG, Tudor Girba, www.tudorgirba.com
Tudor used to be extremely nervous about speaking in public but had to, so prepared this talk. He started with a hilarious, very well acted, 'how not to' demo: Use large fonts, have six bullet points per slide, left align your text and put your logo on the right, delimit zones on slides so people know where things are, have a nice footer with your name (in case people forget

who you are) and the slide number (so people follow progress) but don't have the total number of slides ... .

The talk is not your slides. If all you will do is read your slides, why did you not just stay at home and put them on the web. You come to ESUG to tell a story beyond what is on your slides. Know your audience, then choose the message: not 'a message' but 'the' message, not two, not one and a half, just one. After 3 years of research, Tudor wrote a 200 page PhD to defend one sentence. Think about what you remember from last year's ESUG. Your audience will remember one message at best so aim for that, not more.

What were slides called in the past (transparencies, foils, ...?) Slides were once called 'visual aids'. 'Slides' is the name of a solution, not of the problem. He compared a traffic stop sign with a slide of description of what the sign means. The description is detailed but the slide is effective. Not all details are important: he described a fish store whose sign 'we sell fresh fish here' was gradually slimmed down to a logo of a fish. Then Tudor finally added 'fresh' back into the logo; you *can* strip too far, but you are far more likely not to strip far enough.

Open your slide: see Click here to add Title, click here to add bullet points. So you add a title and one bullet point. Then you add more because just one bullet point looks ugly on a slide. Consider having a blank slide and just put your one point on it. Then have several slides each with one point. By using a 64-point font, Tudor ensures he cannot put too much on his slides.

There was a time when people just talked, without visual aids. Slides are technology; don't let them get in the way. It is normal to be nervous but do not let that make you turn to the technology instead of to your story. Relax, walk out from behind the lectern (have your remote control).

Tudor recommended the www.presentationen.com blog. And what was his message: presenting is storytelling.

(Later in the conference, it was noted that one thing presenters should always try and know how to do is how to make their font larger or smaller.)

Smalltalk Awards Ceremony, Noury Bouraqadi
(Happily, a rumour that the wine for the awards ceremony had not arrived proved groundless.) There were 6 entries at Kothen in 2004, 9 at Brussels in 2005, 11 in Prague, 15 in Lugano and 20 this year. Noury thanked all the entrants for an impressive array of applications. All Smalltalk code (and related code, e.g. a Smalltalk VM) is eligible, whether used commercially or for research, and whether written by academics, by students or by commercial programmers, provided it is separable from its background system. Prepare your software for next year!

The entrants made two minute presentations. The winners were:

- 1st prize (500 euros): Dr Geo

- 2nd prize (300 euros): SeaBreeze
- 3rd prize (200 euros): iSqueak

Books
Squeak by Example was on-sale for 10 euros, Andrés' books on Hashing Functions and on Smalltalk Mentoring for 15 euros, and the Seaside book for 18.

Farewell and Next Year, Stephane Ducasse
On Friday, Stephane led a discussion on how the conference had been.

Reinout thought the best talk (which was also the shortest one) was Rob Vens' talk on exploratory modelling; please put it on YouTube.

Having a full program is good but the days were very busy so it would be helpful to have session chair to say: keep to time, to announce e.g. whether, if time runs short, we have a short lunch.

When there were two tracks it occupied both rooms; can we always have a break-out area as well? (Noted; only late on was it realised we would have to have two tracks in some sessions.)

The first prize went to Dr Geo. Its creator was a C++ programmer a few years ago till Stephane paired with him in a session. We can all do this.

This autumn, Rob Vens is organising two seminars for students leaving university to get them interested in Smalltalk.(gosmalltalk.nl is the Dutch users group). Next ESUG, we would like to have a talk on how to train Smalltalkers.

Where next? One proposal is Brest: low cost, good flights from Paris, Asterix lives there :-). Many rooms in student accommodation with internet connection are just across the street from the venue. And there are also good hotels and restaurants. The very first ESUG was in Brest. Barcelona is the other possible site.

ESUG has principles, not rules. Ask them, give them proposals and they will decide. The need more *active* people in the board. How will they develop a new president. Maybe in 3 years Stephane will force us to find a new president. (Eliot: when you have gray hair, you can resign. :-)

Applications, Frameworks and Experience Reports

WideStrings and utf-8, Philippe Marschall
Philippe sent ubercool (i.e. ü, u with umlaut) and saw it displayed in browser looking anything but cool (tilde-Afibercool). So what is happening here?

ISO-8859-15 (Latin-9) is the same as ISO-8859-1 (Latin-0) except it has the euro. However it does not work in eastern Europe let alone Asia. So people decided to solve the problem once for all: this complete solution is unicode.

Unicode currently defines 100,000 characters (Philippe says ‘currently’ because solving a problem once for all time is an ongoing job). It has a huge number of what we would call characters and it also has a lot of ‘wingdings style characters, white rabbit symbols and suchlike. Unicode is effectively a 22-bit character set. ASCII is contained in latin 1 which is contained in Unicode: all characters are included with the same index. Unicode is defined in terms of code points: a code point is an atom of text. [Niall: and this leads on to glyphs but he did not go into that level of detail.]

Philippe use an analogy between Integer and String representations. An Integers is a SmallInteger or a LargeInteger and we talk about the range which these two classes cover, not endianness. Similarly a String is either a ByteString (ISO-8859-1) or a WideString (Unicode ISO-8859-1) and we talk about the character set, not the encoding.

Unicode is a 22-bit character set but a SmallInteger in Squeak is 30 bits so you have 8 bits left over to have fun with. The #leadingChar uses these for some extra info, e.g. the language and some presentation data. This info is taken into account or masked out as required/desired when reading the character.

So how do you know whether a unicode character is a letter, a number, a white rabbit or other shape, or what? You have to know unicode’s rules. Fußball is a German word (German German; Swiss German does not have it). The ß character is called sharp-S (it’s a curly-B-like character for anyone reading this without the appropriate character sets). It only exists in lowercase, having no uppercase implementation. (Georg promptly corrected him: 4 weeks ago it was given an uppercase representation and that made a big noise in Germany! Philippe was fascinated. However, as what he was about to say illustrates a general point, it remained of interest to continue with the explanation.) If you do `Fußball asUppercase` it returns FUSSBALL: your string got wider.

Character ordering is also locale-dependent; A umlaut goes before A in German but after it in Swedish. “Some characters are more equal than others”; unicode defines degrees of equality: does e acute equal e? You can also compose characters: the umlaut character plus the a character equals the a-umlaut character.

Encodings are isomorphic mappings from characters to bytes. For ASCII and Latin-1, one byte is one character. For wider character sets, the mapping may not be to just one byte for one character.

There are (at least) 2 ways to encode a 32-bit integer: big endian or little endian. In UTF-32, both ways are allowed. You also have a lot of zero bytes (16r00) and code written in C doesn’t like zero bytes. UTF-16 has additional problems (little endian / big-endian). UTF-8 tries to avoid these problems by using less space for the western world (the rest of you, tough luck!). It is 100% ASCII compatible on the binary layer. Everything beyond ASCII is multiple bytes: send umlaut-u 16rC3 16rBC in UTF8 and the browser thinks it is in latin1. WAKom does 1:1 direct mapping from

bytes to characters which works for latin-1 and nothing beyond. WAKomEncoded uses utf-8. It reproduces WideStrings which in Squeak are known for bad performance and bugs. Philippe nevertheless recommends it: if you find bugs please report them or they will never get fixed.

You must tell the browser what encoding you use. Set it in the http header and in html header tag, the latter being a nasty layer violation but nevertheless recommended. (And you have to come up with your code pages.)

In Seaside 2.8, subclass WASession>>charset which defaults to utf-8. It will do both of the above.

In 2.9, they realised that the character set is more application related than session related so they put it into the configuration.

If you want to test, don't use latin 1 text. Use hindi or something like that and test the whole round trip. Philippe strongly recommend Tim Bray's two articles on unicode.

WebTerminal: less code more RIA, Wouter Gazendam and Dirk Heijink, CosmoCows

WebTerminal is a product developed by the authors at CosmoCows (see Mathieu's talk). Mainframes were where big insurance companies were located, serving clients: the clients were completely powerless without the server. The next stage was the PC era. People had freedom to choose applications but the management of all this was a pain: you had 1000 PCs to upgrade, not one mainframe with a thousand terminals. Then we got GUIs, leading to rich desktop applications. The end users faced increased complexity. So did the creators of UIs and the writers of user manuals. Thus many developers adapted MVC to have a UI person in the team.

Finally, we are getting rich internet applications. Now the application has returned to the server.

WebTerminal is a client-server architecture, communicating by HTTP (S). They support IE6+, Firefox 2+, Safari most recent and Opera most recent. Customers on corporate networks have no control over what plugins are installed on their browsers. Therefore they cannot use solutions that need additional plugins; no Flash, no Smalltalk plugins and so on. The only thing they can use on the client side is Javascript. Most of these networks run behind firewalls, some of which disable long connections that are not exchanging information. Comet uses a long-lived connection in which a single client request then leads to several server pushes of data to the client over the same connection, but they cannot do that because the bad firewalls will kill the connection.

Static content is served otherwise (by Apache, say); only dynamic content is handled by the app.

The client-side terminal must render a rich user interface: input fields, images, buttons. The terminal must record how the user has interacted (clicked here, typed there) and at the appropriate moment they must reply to the server in an XMLHttpRequest. The server's response described how the UI must be updated and the terminal does that.

The client-side terminal has no application logic. The server does that. It maintains an MVC pattern for communicating with the client. They have a protocol, the delat protocol, for sending UI changes from server to client. It synchronises the right UI on the client side with the virtual UI that is computed but not actually displayed on the server side.

They only communicate significant changes. They ignore mouse and keyboard events. They synchronise high-level components (widgets), not bitmaps like the X Windows Systems. They have all the usual widgets: label, image, button, tree, groupbox, hyperlink, HTMLContainer and many more. He demoed and (had the usual demo hiccough - needed to refresh his session and) showed a tree widget. He then showed the messages exchanged between client and server when he changed a tree widget property. These are enough for administrative applications (they are not aiming at the games market).

They use Javascript, the Prototype library, and have implemented a few widgets via jQuery and Xinha.

Performance was a issue because they used a lot of DOM calls and on IE these can be very slow (one page took 30 secs to render on IE6 and < 2 seconds on Safari). They rewrote the rendering mechanism to avoid DOM. Instead they build an HTML representation and put that source in the DOM screen only at the last moment.

Support for CSS is poor in IE6 so they had to restrain their use of CSS. In some browsers, backspace is a keypress event but in others it is only a key-down event. This is hard to debug and some browsers are also very poor at debugging.

The architecture is pure MVC with slight terminology changes: Applications (c.f. Application Model) and PageSpecs are connected to WebWidget Builders, thence to WebWidgets.

Finally, he built a chat application in WebTerminal. He put the chat widgets into a simple grid layout, added the past posts widget and put them all in an overall panel widget. He showed the pageSpec:

```
PanelSpec new
  beVertical;
  addSpec:
    ((GridLayoutSpec new
      columnWidths:..;
      rowHeights: ...;
  specAt: 1 @ 1 put: (LabelSpec label: 'Name')
  specAt: 1 @ 2 put: ... aspect: #name ...
...
```

```
  addSpec:
    (RepeaterSpec new
      collectionAspect: #messages;
      spec: (LabelSpec new
               labelAspect: #yourself;
               yourself)
....
```

He saved the pageSpec method, then generated its aspects, name and nameHolder, text and textHolder. The messages must be the same for two users of the application so he did the simplest thing that could possibly work by making the instance-side delegate to the class-side to get messages from class instance variable.

Lastly, he told the system that Chat was a top-level application (an entry point), showed it, added an entry and (got the usual demo hiccough - he had not yet implemented the submit button) and had to tell the other client to refresh and so demoed.

Using User Changes, Leandro Caniglia, Valeria Murcia, Caesar Systems
This talk was also presented at Smalltalk Solutions; go there for my single write-up merging information and questions from both presentations.

GStreamer: media streaming in Squeak, John Macintosh
(Having media problems during a talk about media is tedious. John was going to show Sophie but the 600 x 800 projection screen does not really allow it.) John has been working on Sophie and on the iPhone Squeak VM (with financial help from ESUG; thanks) with Michael Rueger. They will release the code for it next week so you can see it in the iPhone simulator or, after paying Apple, in the iPhone itself.

GStreamer is a library for building graphs of media-handling components. It can be used and abused in a wide range of ways. Its code has good and bad features; e.g. mpeg codexes are not in the base and must be found by hunting on the internet. OGG is the only open-source framework for this John knows of. You can get GSTreamer on WIndows, Linux, etc., and has a reasonable licence (see the wiki).

The Squeak plugin is not a full implementation. The GStreamer class library is immense and their subset lets them decode and encode audio and video via some 100 primitive calls. It has been written for 32 bit; any 64 bit people here who would like to volunteer? (some ‘yes’es). It should be quite doable

Most code is in Slang; a very small amount is in C. There is an SUnit for each publicly-exposed API call. The architecture is subclassed from GStreamerObject (fortunately, the GStreamer library is fairly object-oriented). The plugin works by marrying a C object to a Smalltalk object so they must GC the C when they release the Smalltalk objects. However a pipeline of C objects will crash if all are released ineptly on its married Smalltalk object’s death; you must avoid the double-freeze problem.

Finalisation is when your code tells you that a Smalltalk object has died or is about to die, having gone out of scope. Thus a Smalltalk object married to a C object gets a message which prompts it to tell its C object to go away. (There is also resurrection, a horrifying procedure which indicates bad code.)

A GStreamer can be a source (microphone, camera) or a sink (e.g. display) or both (audio comes in, gets altered and is sent out again. GStreamer objects are referred to by name and there is a command that lists all the installed elements. An element is then put into a pipeline or bin. You must then hook it up to show how the data flows between the elements. An element has a pad, either static (always there) or dynamic (created at runtime). A volume element would have a static pad but the OGG would have a dynamic one.

A pad can be asked for its capabilities: it returns a mime stream of what it provides (source) or consumes (sink). He showed a video device returning its settings. You could tell a pad 'you only understand 640 x 480' to control elements. John showed an example of creating a tone generator on an ALSA platform (LSA means Linux Sound Architecture; what the A stands for he does not know). He created an element for a source data note 'audiodtestsrc' that plays tones:

```
audiodtestsrc := GStreamerElement
  elementfactorymake: 'audiodtestsrc'
  name: 'source'.
```

added a converter element and volume controller element, and finally a sink element to talk to the ALSA hardware. (One would of course have helper methods to hide code like this in real use.)

```
pipeLine := GStreamerPipeline name: 'My-Pipeline'.
result := pipeLine addElement: audiodtestsrc.
result := pipeLine addElement: audioconvert.
result := pipeLine addElement: audiovolume.
result := pipeLine addElement: audiosink.
result := GStreamerSystem default
  linkElementSrc: audiodtestsrc toDest: audioconvert.
result := GStreamerSystem default
  linkElementSrc: audioconvert toDest: audiovolume.
result := GStreamerSystem default
  linkElementSrc: audiovolume toDest: audiosink.
pipeLine setStateTo: #playing.
pipeLine setStateTo: #playing.
pipeLine setStateTo: #pause.
pipeLine setStateTo: #null.
pipeLine release.
```

The lowest level method `setKey:toStringValue:` connects to evil C typing which has to go on somewhere; double-dispatching hides it as best it can. See the gst-inspect Unix command to get info on an element: it is very helpful in deciding what capabilities an elements has. The

```
waitUntilErrororMessage:upToMilliseconds:
```

helper method returns on a range of conditions: end-of-stream, end-of-file, error message, 5000 milliseconds.

He then opened a Squeak image to show a more complex example: running a test to play 5 secs of audio from an OGG file recorded from the internet. The test quotes the unix command line equivalent. He ran it and produced 5 seconds of sound !

GStreamer lets you do complex video and audio streaming and push the responsibility of having it work down into the framework.

`requestForcallbackSignal:` is ugly magic. The demux wants to create pads but the base Squeak VM does not support callback to Smalltalk. Their special set up knows how to interpret the pad-added message to hook-up the pads.

John took the morphic mpeg player (written 8 years ago by John Muloney based on work by John) and rewrote it to use GStreamer. An OGG file can be audio or video or both and so they hang around for 5 secs waiting to see if it has video or errors and then assumes it is audio. He let us hear the audio, then opened a video (which he could rotate, etc., as usually in Squeak).

Q. Who keeps the timing; Squeak or what? They wait for the next frame and then blt it to Morphic. John uses a q object and a p thread. It is a waste of time for audio to go to Squeak and then just be given to the laptop's sound system so in this example he has the audio in this going directly to the audio output.

Squeak can be a source but there is still work to be done there (volunteers welcome).

Lastly, playbin and playbin2 aims to figure out all the complexity of arbitrary input and output. It works in some platforms for John but not on OLPC; it could be made to work.

Q 100 native primitives? Calls for each type, to get attributes for a tab, set them, etc.

Q. Use FFI instead of primitives? FFI is not offered on EToy environmnet which is one of the targets for this.

Heating Control System with Smalltalk, Alfred WullSchleger
Alfred retired in February and had time to address the problems in his home heating system. All control was handled by 230 volt switching of 4 relays for the outgoing flow (on/off) the mixer (open/close/off) and the gas valve (on/off). The system can know temperature going to the house (tv), the temperature returning to house (tr) and the earlier/later temperature going in and returning from the mixer (tvm, trm).

The house likes 30C < tv < 55C. The boiler temperature should never be < 40C (causes condensation, which causes corrosion). Boilers are small (< 10 litres) whereas radiators have 300 - 500 litre capacity. Thus the heat capacity of the radiators far exceeds that of the boiler, so you must be

careful when opening the mixer. It needs to rotate to just the right angle.

The start point was the commercial control unit built into the boiler: tklower and tkupper control when the gas valve is started and stopped. Since the mixer is very inert, tk rises fast and the burner stops soon. When the mixer opens tk falls fast and the burner starts again. He showed a graph of these rapid rises and falls (six peaks and troughs in less than 20 minutes, and this behaviour typically continues for 40 - 50 minutes after the system starts), with frequent drops below 40C. By contrast tr varies very slowly. Thus the boiler is incessantly switching on and off, stressing the gas valve and not meeting the 40C requirement.

(At this point, Alfred's slides tried to help us understand by not aligning with the screen; he was using Dirk's computer.)

He therefore worked on a new control unit that would meet tk > 40C except for the first 3-4 minutes after startup. His hardware was a standard PC plus USB (from Minilab1008 USB module with 8 AD and 2 DA analogue channels; later, he will move it to a suitable small component on the boiler) plus an interface he built himself to control the valves and temperature sensors. All control and UI was in Smalltalk. The (manufacturer-supplied) DLL for the Minilab1008 is pleasant to use.

He can control it remotely or though the house' LAN via sockets for remote operation. (Q(Noury) Plain sockets, not distributed framework such as OpenTalk? Alfred was accustomed to using sockets in this way in his work. He also has very small commands, just changing temperature bounds.)

He then made a simulator that was independent of hardware. He proposed 3 phases:

- early heat from cold boiler with tk < tmin, only tk is interesting,
- intermediate with tk > tmin, tv < tvdesired
- normal running with tk > tmin, tv ~ tvdesired, and tk should be as close to tv as possible

(of course, he only realised this structure of the problem after a period of experimentation). The minimum energy needed is given by the formula Emin = c*mass of water in radiators * (tvdesired - tv). When Emin is < 0, the house has a reserve of energy. When Emin is > 0, we do not switch off the gas valve. This means that the mixer must be controlled very finely to avoid overheating or underheating the boiler. This phase can last for a long time, hours perhaps. Only when Emin is <= 0 do we revert to the hysteresis-style control (similar to that of the commercial unit). While we have 0 > Emin > EminMaximumReserve we do this and if we exceed the maximum (10kJ) we switch off the valve hard, waiting until Emin has dropped significantly.

The mixing angle is measured from the steady movement of the mixer linearly from 0 to 90 degrees while the mixer opener is on. He always closes the mixer for 2 minutes (it needs 107 seconds to close) to close it and so get into a known situation. This causes tk to rise quickly. He then starts

to open the valve to a set value. After 900 seconds, the intermediate phase starts and tv is also monitored, causing a more subtle behaviour of the mixer, but the gas valve remains on.

He noted how tk moves significantly with tiny mixer motions. tv always shows a bump when the mixer moves (output is near the boiler so r bumps up and down as heat is supplied and then slightly more slowly is pushed into the house) while the return values move very slowly.

Finally we reach the normal running mode. The mixer can be left at 90 degrees when tk is well above tmin. He finds he can manage it between 17 and 70 degrees for much of operation. Closing the mixer for 10 minute periods when the gas valve is off conserves heat in the boiler.

When he started this work, the curves were not as clear as in his slides. They wandered all over the place. Fine control of the mixer was the key realisation. The start-up mode is more complex to handle than the steady state.

Starting work in February was a good choice as the external temperature rose and so he could study a range of states. However he must wait till next year to see if it saves him financially or not.

One safety issue is to avoid steam. He guaranteed this by a software-independent bimetal switch which turns off the gas valve when tkm>80C; the boiler came with a manufacturer device that switches off at 100C which he thought too risky.

He can also switch between his unit and the old unit by moving a single connector. It is clear that his system is much more stable, less fluctuating, than the commercial one. Smalltalk was a great environment to work in; he could sit by the boiler and make live changes fast (important when the boiler was heating up very fast) and switch to hand control of the mixer. He could also see when his simulation did and did not match what the real world did, so teaching him the scientific behaviour he was modelling.

Q(Niall) Commercial value? Yes, but having worked all his life in banking and similar he has few contacts. Anyone know any top-end boiler manufacturers?

Croquet/Cobalt: An Open Collaboration Architecture for Education, Robert Sheperd and Julian Lombardi, Duke University
Rob Sheperd has founded eduVerse (group in Amsterdam) and is talking to surfNet, seeking to interest them in Croquet. He wants to work with Squeakers here. He then called Julian Lombardi who gave the talk, his face in one screen, the slides in another.

Julian summarised the mouse, window and office metaphor that our computers use now, then gopher and the world-wide web. The world-wide web uses the metaphor of the document. It's good for that and poor at, for example, presenting talks like this. Google is a great search engine for

finding information you want to have. At his university, they introduced an online library catalog - and people complain they now cannot simply wander round the stacks discovering things. They have lost the context. If you want to show the context - the books on the shelf that has your book, the other people browsing that shelf - how do you do that.

The web works by replication of data. VirtualContext systems such as second life rely on this replication of data; they create a virtual world on the server. A client accessing that virtual world needs to ask incessantly about what is happening in that virtual world, unlike reading a web page. Many clients talking to a second life server creates a huge load; 20-30 people at most can be handled by a single server.

Croquet wants to make virtual worlds, and deep interaction in virtual worlds, available to millions. Companies that serve virtual worlds today have server-oriented financial models. Croquet/Cobalt aims to avoid the problem by replicating computation: each client has its own computation and synchronises only when things change, communicating with pier-pier mechanism called Teatime.

They snapshot the world from time to time: if a client switches off and later reconnects, they get the snapshot and later messages from any pier.

Croquet will be open-source under MIT. They are funded by the Andrew Mellon and the NSF.

He then demoed - and immediately lost signal. A quick reconnection sorted things. The demo is available to ran over the web

(There is also QuackForms, a commercial company, with several Croquet architects working there, that is commercialising some Croquet technology. He is focusing on open-source efforts.)

Pier-pier messages are timestamped and executed in order of timestamp for all participants. He showed two users moving a window within a window (usual Alice and Rabbit avatars). Collaboration comes naturally in this stateful system: Julian can edit text in a subwindow and so can the rabbit. Portals resemble hyperlinks: avatars walk though portals to other worlds. The rabbit walked through a portal, then created another world and entered it. Julian then picked up the world/portal and moved it to a third world.

Croquet can access remote applications., The rabbit launched the Squeak chess game (2D, projected on a screen in 3d croquet) and played it with Julian. Via VNC they could project many external applications from Windows or Linux or wherever, not just Squeak applications. He showed an avatar carrying around a screencast (of him lecturing, I think) projected into Croquet.

He flew around the helicopter and showed the demo of drawing a shark in 2-d, rotating it into 3d and making it swim.

Next he projected a Flash screen - in Croquet Flash it can be shown in perspective because it is projected into Croquet, even though naturally Flash only accepts isometric transformation.

Having demoed Croquet, he next showed Cobalt. Croquet has been under development for 4 years (depending how you count, he said: I'm sure it was doing some of what he showed in 2003). Croquet is a development environment. Cobalt will be an application. He dragged a jpeg image from the desktop to the cobalt world. Cobalt is making Croquet usable to end-users - the ones who will create virtual worlds. The Cobalt avatar was much improved over the basic Croquet avatars in terms of its movement and appearance (but he was much less characterful than the rabbit and Alice - just a typical PC figure).

They have Jabber (rendezvous server), heads-up display tool, terrain editor, etc. He added a turtle from a repository (Niall: clearly Croquet is turtles all the way down :-).

Q. Recent rewrite of OpenGL and deprecation of a number of functions will force rewrite on you? Yes, they will have to at some point but for the next year they are focused on getting this out the door. They are poorly funded and relying on a community-based effort.

They aim to release Cobalt/Croquet next summer. Croquet was a top-down project. It is not any more. There are 12 people actively involved in croquet development; he would welcome involvement from people here. Visit croquetconsortium.org for the roadmap. And things not on the roadmap are also getting done - a group in Malaysia is integrating FreeCAD which was not on the roadmap but they thought it needed to be done.

Madeo: a CAD tool for Reconfigurable Hardware, Loic Lagadec
Madeo is from the University of Brest. Refactoring, reuse and agile development also exist in real hardware. Co-processors can have dedicated and dynamic instruction sets. Reconfigurable hardware is a way to build these. He is looking at very hardware programming at a very low level: take two input signals, decide what output should they compute.

Q(Reinout) Static or dynamic? the idea is to change e.g. a portion of memory at a time while using memory.

The targets are FPGAs and the eFPGAs that are coming on the market. These could be used for implementing hardware VMs amongst many other things.

It is a very complex task to program these circuits. Loic seeks to apply OO methods to this task. He has been working in this field for ten years. Their first task was to build a framework to map high level code to circuits for one specific target (XC6200 from Xilinx), to learn all they could from that. They then generalised to more targets. He showed a screenshot of their system with circuit diagram displayed and an inspector open on a HardwareContext. He had various implicit assumptions in the first model

which use corrected.

He opened the system and selected a target architecture. A complex indented text description listed the functions and their interconnections of the architecture.

He opened a box array diagram; each box was a switching element. He moved to a diagram of connections and selected some functions. He opened a table of input combinations mapped to results. He built his choices, returned to the box array and placed his circuit on the board.

They use the meta-modelling tool from Alain Plantec. They use the visitor pattern in several places in the tool. The modules are pluggable to allow various routes in from application specs and out to EDIF, C-like code and other outputs. In this domain, these results must be validated by standard tools, so must be exportable to them, but they are Smalltalkers and also use SUnit.

Madeo has been used in several projects and is open to third party use. Development lessons he learned: meta-modelling is a cost killer. When exporting code, target your partner's reference language; treat partners as clients. Don't show partners that you work faster than they do; they will become suspicious. Say the next release is in 1 month, not in 1 week. Treat validation as very important: your code must still work with the latest release.

Q(Reinout) licence? Not yet clear but they are happy to send an image tomorrow ("Don't say tomorrow, say in 1 month" :-).

Q(Eliot) Optimise and download to chip or execute globally; how far have you got analysing that? (I lost some of the discussion: background noise) You want to map very high-level behaviour so can we have a semantic framework in which you could study the value of mapping a dictionary into associative memory? That is above what they are doing today.

Q(Eliot) Time to market in games is vital. Some functions have been using lazy functional programming languages. The cell is a coprocessor with very limited resources. Does the Smalltalk machine dynamically manage the coprocessors? Loic has seen talks on using FPGAs to manage resources dynamically. A student is working on it. Problems are to decouple memory access from computation and to build the pipelines.

Development Tools and Techniques

Exploratory Modelling, Rob Vens, www.robvens.nl, rvens@sogyo.nl

Rob has half-an-hour to describe what has taken him years to understand. He thought the best way to do it was just to do it. When he does xM with customers, he has to tell a story first, else they will stay in their own familiar mindset. Most businesses do not have a language to express themselves. They may think they have but in fact they automatically fall into business baby talk, "explaining things so simply, they'll understand".

Business has been living with IT for a long time and having a lot of problems so they talk as if they were computer technologists. So when not talking business baby talk, they talk the language they think Rob will understand.

Thus Rob tells them that the world they live in is the world he will mirror into their software. In a mirror, things are not exactly the same as in the real world. Mirrors turn some things upside-down or back-to-front. When you mirror the real world into the software world, things become alive and living beings become dead. Accounts and Products become alive and do things, whereas the accountants and product managers become incapacitated. In Disney, doors open themselves, teacups walk around. In the software world, products sell themselves, accounts transact themselves.

Domain-driven design is very important to the Smalltalk community. Dynamic languages are becoming popular, closures are becoming popular, but also domain-driven design are becoming popular. (Reinout: people model the solution domain, not the problem domain.)

The business owns their domain model, not (usually) the GUI, security, etc., technology components from which it is built. Technology components are there to synchronise the mirrored model with the real world. Good technology is invisible.

As well as this active-passive switch, domain models can exhibit time reversal. They are constantly moving. A principle Rob uses is that models are evolved by adding new components: the existing components do not change.

Bravely, Rob then asked the audience for a problem on which to show his approach. He got one: a company collects old metal in containers for recycling; the containers are distributed round the country, then moved to central locations and emptied, then moved back again. At the end of the year, 400 of their containers are missing. How can they track them?

Rob starting modelling at the end. That is a key rule in his approach: first ask, "What is the situation you want to end up in?" At the end of the year, you want to have all your containers in a location and you knowing what that location is. Rob opened a vanilla VW image and created a package for the model. (This never bothers customers - he just says he needs a place to keep the model.) He opened a workspace and did

```
Container new.
```

(This container may live for a few days or a few weeks while the modelling session changes its methods and instvars a lot - only Smalltalk can do this.) He sent `arriveAt:` to the container, got DNU, hit define, wrote as a proposal

```
arriveAt: aDestination
  self unloadAt: aDestination
```

leading to a discussion of whether the end-year destination is the recycling

or whether it is the return to the collection point after recycling. (Again, customers never complain about seeing the debugger). Rob stressed this time-reverse approach: he is working back from each end state to its predecessors.

He had a customer with a very complex problem domain. He extended the inspectors to let him drag-drop (important for customers). The customer's people were very pleased; they liked the sessions and found they understood their own problem domain much better than before.

He is rewriting his MyGold personal finance system using this technique to guide the rework and will publish papers on it.

Q. Eric Evan's book? It is interesting but very technology-oriented.

He built an XMI exporter so models constructed this way can be exported (and imported to C# in this customer's case).

What Smalltalk can Learn from Java, Philippe Marschall
At the end of this talk he expects people to say, "we could build something better than this." So do it!

JMX monitors and manages images. Images do go bad in production on the server. In general it happens slowly: memory problems, too many sessions, whatever.

In the past, Philippe just used a work space: some `allInstance` checks, state checks and so on. This does not scale well, is not automatic, etc.

You can build your own: you need to convince your manager, write it, debug it, and maintain it - forever. It is yours and you alone use it. Seaside won't use it to tell you how many sessions there are. Glorp won't use it to tell you its cache size. If you hack into Seaside, Glorp and so on to use it, you must maintain your changes to those base products.

JMX lets you query values, invoke some operations and receive notifications. There are in-image and ex-image APIs. He opened the external UI and viewed data on an image: how much time was spent in GC, in JITing, memory state, etc., as graphs and data. Another widget showed a list of web sessions with their age and state, the database sessions likewise, and much other information. He exercised the app and looked at changed values.

Your Swiss manufacturer does not call the moon face and other aspects of your expensive watch 'features'; they call them complications. Features are bad: they cause bloat and confusion. However, just as Swiss watch manufacturers discovered they needed 'features' to sell their watches, so we find that features are good: they sell your software. So you want to let people take the features they want. Plugins are one way of doing this. Plugins are good - provided they work together. Will your Squeak image still work after you load this package? Module systems break your

application into feature sets and also make it easier for you to add small features.

OSGi is a module system used by Eclipse RCP, GlassFish v3 and smaller application servers like Sling, etc. Application servers that supplied all their features as standard could take 20 minutes to start up. GlassFish v3 starts (in a second) as a core that can only serve files, with much else loadable.

Maven 2 is a build tool. It aims to work in 4GL declarative style. Try releasing more than a dozen Seaside subprojects to SqueakMaps and Universes. It is not automated and you have to script a lot of stuff. We should make an object model of a project and publish it, not script.

POM lets project builds inherit from other projects. A Pier subproject, for example, would inherit a lot of stuff from a general Pier project. He showed the dependency graph of Pier Blog. It needs RSRSS and Pier-Seaside, which needs Pier-Model and Magritte-Seaside. Pier-Model depends on Magritte-Model, Pier-Seaside on Seaside.

Dependencies are transitive, may be optional, may be only for tests, for development or for deployment. All this data can be generated into a project's home page and other tools.

ANT was mentioned in discussion. But you should not use it in general. JMX and OSGi are not good. Java has an unerring gift for doing things wrong so we should not aim to be compatible with them or imitate them. We should aim to model abstractly the problem these solve, then build that abstraction.

Q. How long to build these? In a week, you could have Maven working, then extend it.

Q(Tim) Some Java sites have build servers? You check something in and it gets built and tests run. SqueakMap is littered with things which will not even load.

Advanced Techniques for building Testing Tools, Andrés Valloud, Cincom

Andrés thanked Stephane for mentioning his books then recapped SUnit. Tests pass (true), fail (false) or raise errors. Andrés had to write a test tool for hash functions. Hash functions analyse a particular dataset with a particular hash function and produce a set of statistics, which is rather richer than true, false, error. What object caused a particular hash value to appear. What collisions occurred.

Thus your results are a collection of small integers. He therefore reified the evaluation context. The Distribution sends `resultsInTheContextOf:` to the EvaluationContext which holds a hash bucket dictionary.

He also had a hierarchy of datasets. He showed the hierarchy drop-down in

the tool, a standard class hierarchy. However some data sets are repetitive sequences, some are sequences of 4 letters, some are Dutch words as against English words, and so on. These do not share much code so have no common superclass but this commonality of type is important for the tests: does a given function do well or badly on given types of dataset. There are also ad-hoc datasets intensionally-defined by e.g. allInstances or a collect: called on another dataset or whatever and these also have no natural identifying superclass. Lastly, he sometimes wanted to handle the hashed result collections as input data.

He therefore created AbstractDataCollection whose instances behave like the classes he wishes existed for these non-class datasets. Polymorphism lets classes be objects so it also lets objects be classes. :-) This has method `subclasses` that returns these dataset 'classes', while the actual class that the dataset is an instance of is returned by `classToImpersonate`, and LeafDatasetMetaclass>>new just returns the dataset.

He returned to his tool and showed the ad-hoc dataset hierarchy with its not-classes. He opened an inspector on the Smalltalk dictionary and adopted it as another dataset in his tool upon which to do hashing tests.

Andrés wrote various extensions to SUnit for validation and benchmarks. He therefore needed to extract all hard-coded references to e.g. TestSuite in methods so they could be subclassed. He also extended `runCase:` to handle the greater range of exceptions. He did not want to call `on:do:` over and over again. He instead double-dispatches on exception handlers.

```
on: ...
do: [:ex | ex occuredForTestcase: aTestCase
              inTheContextOf: self]
```

(self is a TestResult). Thus he can add validation and benchmark SUnit failures without overriding code.

When he wrote his book he had a lot of string matching checks (7 * 56 lines output for a typical test) which was very tedious. He was running essentially the same test for many pairs of patterns and matching result strings. For benchmarks he wrote multiple setUpCaseA1, setUpCaseA2, etc., methods with running the test meaning setUpCaseA1, run, tearDown, setUpCaseA2, run, tearDown. These also set whether failure meant a little slower, 10 times slower, etc. He also created modifications of the basic tests via e.g.

```
self setUpCaseA1.
self makeDataUppercase.
```

but the more he extended SUnit the more he was concerned about incompatible SUnit mods (for example, lots of people do logging extensions to SUnit and he felt his work would be incompatible with theirs).

He therefore created a new framework: Assessments. He asked why we have to create an instance of TestCase. He thought that having a separate test for each run in a given context was like giving someone a questionnaire

with a separate sheet of paper for each question instead of one sheet for all questions. An Assessment is a single instance that refers to a large number of Checks (equivalent of TestCases) on an AssessmentEvaluationContext. This has a result policy which returns a given Assessment result; the policy handles whatever logging you want.

The Assessment Result has results of type A, results of type B and so on and knows which checks returned type A results, which returned type B and so on (so basic SUnit would have pass, fail and error as its types, his validation and benchmark would add two more types, etc.). The `classificationTag` for any result is its class. A Check has a receiver, selector and arguments, and it also has an executionPolicy. Run is one policy: get the result and don't show the debugger. Debug is another: who cares what the result is, just show me the debugger. Thus he has a set of polymorphic exception handlers and the Check knows how it should run. So he can have execution policies for SUnitVM, for SUnitToo, for SUnit. (Someone's computer or phone played a sad tune at just this point. Andrés picked that up as, 'Well, if you are in SUnit, indeed that is sad for you. :-)

He maps SUnitToo exceptions into Assessments exceptions; receive an exception, raise another. However he still has SUnit TestCase subclasses and SUnitToo (different) TestCase subclasses. He therefore uses the Metaclass trick he showed above to see them all and call them all from his Assessments UI via

```
pretendToBe: self testCaseRoot from: someSuperclass
```

He opened an Assessments evaluator and ran some Assessments tests. Then he expanded the AbstractSUnitTooBridge, ran some tests, found a failure and debugged: the debugger opens in a stack where you only see SUnitToo stuff; no Assessments stuff gets in the way.

Q. This is available, licensed? Yes it's available, feel free to use it. He will discuss licensing with Stephane then add an MIT or similar licence to allow maximum use and minimum inconvenience.

Starting fresh every morning, rebuilding a development image every day, Yann Monclair, JPMC

Yann went to university in Brest where he got hooked on Smalltalk. Kapital was released in 1995 and now has 22,000 classes (base VW 7.3 plus Envy is just 2,200 of these). Each development cycle changes 5,000+ classes (last cycle change ~ 7,000). The production image is 90Mb, the development image is 120Mb.

A small team can resynchronise their code bases easily. A multi-site, three timezone team needs a process. 60 - 150 classes change every day. Every day, 25 change sets of 5-8 classes are applied. On average, there will be a class change conflict every week. There is also duplication of work if you do not resynchronise images. A large change can be hard to remerge (he speaks from experience). It is better to decompose your changes into small units and release them day by day.

Prerequisites: the Kapital rule is to make Envy happy. Envy by nature is very grumpy and complains a lot.

A final reason for rebuilding your image every day is to discover unknown dependencies, much better than `ifDefinedDo: [...]` which is a recipe for hard-to-track silent problems.

Envy holds code in Applications and Configuration Maps. It also versions classes and methods, its unit of granularity. To build the Kapital image, they load the top-level map, which loads everything else (or maybe sometimes fails and carries on having told you on the rapidly-scrolling past Transcript). Then they validate the build. They run SUnit-style code-driven tests to verify specific methods work. Then the do data-driven end-to-end testing and checking that the results are as expected. This is called the SmokeTest. These smoke tests include deep scope tests - compute today's value of every product in Kapital - and wide scope tests - verify in detail some functions for selected trades only.

It takes 40 minutes to build an image and then 2 hours to validate it.

The Kapital launch command gets today's image and loads your current unreleased working code into it, ensuring that if some specific action is needed to make your image work, it's absence will be revealed.

Q(Arden) Development or production? Development: production release process is formal in a bank!

Saving an image, common in Smalltalk generally, is doable but not often done in Kapital. They prefer saving code to Envy and loading new each day.

Breaking the build will happen. The first thing to do is identify the issue: failing tests, uncompleted build, successful build but will not load. Tests can fail for good reasons - someone released a test as I released my code - or for understandable reasons - not enough time to run all tests before rushing out this fix.

Uncompleted build: you find the offending code change. Most often it is prerequisite issue; a method not yet introduced or a class or variable not yet declared.

Successful build but failed load means that Envy does not load all the way. It got so far, raised a warning e.g. it tried to do a code override which Envy will not tolerate), then does the save and testing anyway.

If the build failed, that means that yesterday's code changes are not validated and today's are beginning to accumulate so you must fix as fast as possible.

Failures can be because your method is not yet there (e.g. an initializer uses the `with:with:with:with:with:with:` extension, not yet loaded) or

clashing code because Kapital loads class versions, following Envy.

Q.Make Envy do reduced conflict merging of class definitions and method defs rather than whole classes? Yann preferred to keep Envy happy as is.

Q.How do developers get their code into the fresh image? The developers put their modified class images into a system (called FDPs) and they load their current one(s) into the fresh images.

Smalltalk Standards Report, Bruce Badger
I only caught the end of this. See http://smalltalk.gnu.org/wiki, http://lists.openskills.org/ and click on ANSI-Smalltalk.

There was discussion of tests. Do we need tests for every STEP: Bruce is recasting the ANSI-Smalltalk syntax as BNF and that would be a review or social use thing rather than a testable thing. But generally, yes.

Q(Dale) The rubber meets the road when a STEP is implemented and the vendors say, "We support that step." Bruce agreed.

Syslog, Bruce Badger
RFC 3164 is a BSD 'standard' for logging. "For the self-aware organic units..." says the spec but no it is for computers, not us or aliens, specifically for sockets. Messages are 1024 octets in length with PRI, header and message. The Device sends messages, the Collector receives them and stores them and a Relay will filter and route them (and acts as a Collector and a Device as well).

OskSyslog implements this (in VW), is in the public store, and can be used by anyone. He typed logger hello, world on the command line and saw it echoed. Then he did the same in OskSyslog. Nothing happened because by far the majority of syslog servers are not listening on UDP by default. They mostly listen on local *nix sockets. So when developing this stuff, by aware you will need to add a -r and restart the logger.He then executed the Smalltalk code - and met the usual demo hiccough - he was sending it to the wrong IP. Finally, it echoed.

OpenSkills runs many http servers on Gems and they all get attacked with bogus requests and these get logged to local log files and it is a lot of work to review them. Now he can access any syslog device. You can log through TCP instead of UDP, (at lower throughput, of course) but ask yourself, if I cannot afford to lose a message or three, am I really doing logging?

This listens on port 514 so you must run as root or use iptable to redirect traffic from 514 ti a port with number > 1024.

Only one syslog server can listen on a port at a time so you must switch off your machines syslog server on that port in order for your OskSyslog to take its place. He did this and saw some messages on the Transcript.

A Collector receives messages and does something with them: writes them

to a file or a database or counts them or whatever.

There are heaps of tools out there for analysing syslog messages. Syslog can be added to Toothpick or whatever as a source or as a sink, so Toothpick can do the logging and syslog then converts these to let you do extra-image things with the logged events using all these tools.

VMs and Smalltalk Environments

Keynote: Cog Back to the Future part 2, Eliot Miranda

The motive for seeking a better VM is better server performance and better user experience in virtual world apps and similar.

Why called cog? Well it is a small cog in Squeak, it can mean cognomen, it suggests a logo (and it let him show the Honda Accord 'when things just work' film; afterwards he showed the follow-up film on how it was made: six months of trying to make it work - which Eliot felt was exactly like a programming experience.)

The VM is being built in baby steps: block closures solved (so goodbye BlockContext)

```
tempVector := Array new: 1 ... do:
                [... tempVector at:...
```

is easier than

```
| next |
...do: [:each | next := each binaryBlock...
^next
```

It is easier to keep the 'tempVector' around than manage that the temp 'next' can outlive the method that declares it and must have closure refs.

The task is to do a VW-style JIT. He has added closureOrNil.

```
MethodContext
  instvars: 'method ... closureOrNil receiver'
```

There are five new byte codes. Eliot uses a trick to have 2 opcodes in one byte code for pushNewArrayOfSize:/pushConsArrayWithElements: and four other combinations.

The image wants to use the context for everything: Seaside, the debugger, etc. Mapping the context to the stack is complex when you terminateTo: and don't want to figure out the context-stack sizes for all the stuff you're throwing away. Andreas suggested to him do an interpreter with the same context organisation as the JIT as a first stage. It may well coexist in the final system because then you do not have to JIT huge methods: those can be given to the interpreter.

A stack frame (see slide) records the caller, receiver, caller context, method, flags and so on. A JIT knows the receiver and numArgs at compile time but an interpreter does not, so the interpreter has to fetch the numArgs and subtract along to get an arg. Thus in the interpreter argument access is slower.

This is implemented and has underwhelming performance: some benchmarks are 10% faster, occasionally 68% faster but the end-user experience is not improved (maybe even poorer).

Eliot has hacked together a VM profiling tool which they have lacked in Squeak for a long time and he will develop it further.

Eliot hopes to have an x86 fast JIT in April 2009. It will have a two-word object header to allow the same representation in 32 and 64. It takes six instructions to construct an arbitrary 64bit literal, needed to load a class, so in VW 64bit, Eliot stored a 20bit index into a table of 2^{20} classes. In Squeak, he will use a 3 byte index to 2^{24} table. Except when method lookup must traverse the class hierarchy, the two-word object header will suffice. A class' id hash is its classTableIndex so you no longer lookup because its hash is its index. Eliot would love other systems to use this.

Newspeak will have a different bytecode set from Squeak so he would like to make the decoding of bytecodes not a big switch statement as in conventional VMs but a table of functions so the bytecode set can be pluggable, with pluggable generators. This is what Eliot means by an open JIT and he hopes it will be used by Croquet and Newspeak as well as Squeak.

The target is good floating point performance: AOStA - adaptive optimisation - has been renamed Speculative Inlining - SIStA - since it is optimising optimistically with guards. He will reify the PIC state, count the conditional branches... The PICs provide type information that an image-level optimiser can use to e.g. just add two SmallIntegers without worrying about type and overflow, just do `at: aSmallInteger` without checking. This causes you to create much bigger basic blocks between sends. Thus it may be that low-level VM optimisers may be able to optimise these large blocks; recently, he met the guys at Apple who do LLVM and they agreed.

Q. LLVM is C++ technology? Eliot _thinks_ if the separation is done right it will be usable.

The next stage, the quick (not just fast) JIT will be 2010 or 2011, earlier if you will help. :-)

In questions, Eliot explained how he find it satisfying to figure out the shortest possible sequence of instructions - like playing solitaire. The difference between Squeak and Dolphin when doing fibonnacci is 5, and Dolphin is a really well-written interpreter. Between Squeak and VW it is a factor of 20.

Eliot is very focused on where he wants to reach - a solid fast VM for Quack and the Squeak community. He is not doing research.

Q. Multi-core? There is an emerging consensus that conventional multi-threading programming does not scale; people are looking for alternatives.

Squeak is looking at hydra: many images, one per core, with fast inter-process communication. Eliot takes from this the decision that it makes no sense for him to try and solve the multi-threading problem in the VM. He will instead make sure his VM plays well with hydra.

This is the first product Eliot has been able to blog about as he works on it. He enjoys doing so and the comments he gets. He will discuss offline with Pharo re getting stuff used early.

Cincom Smalltalk: Present, Future & Smalltalk Advocacy, Thomas Arden, Cincom

In Spring, Cincom released VW7.6 aimed at application developers, ObjectStudio 8.1 for business analysts systems and a maintenance release of OS 7. OS8.1 is the only Vista-certified Smalltalk (vista-certification needed a lot of work; they had to write a new installer). It supports Seaside (as VW does of course).

VW76 has new-look refactoring browsers. It supports Seaside, OpenTalk for Seaside, the Glorp O-R mapping framework (GLORP) and MySQL. Hashing is much improved. VW76 also improves the OSX VM greatly (he sees a lot of Macs in the room so this is good news and you can rely on continuing effective support on Mac platform). When they cancelled Widgetry, they nevertheless moved over the grid widget and some other stuff.

Where is Cincom going? Arden's aim is to provide improvements but always ensure that if a change forces customers to port then it makes their effort worth it. Thus the emphasis is on incremental change and backward compatibility where reasonable. It is also about making noticeable improvements in areas that you can show, or at least explain, to managers.

Cincom wants to grow Smalltalk. Continued investment helps that on itself, as does advocacy. Embracing new exciting technologies like Seaside (and adding to them: Web Velocity).

They get ideas of where to go next from Customers, internally (engineers, the star team, marketing) and the community. Arden asked for '9 for 09': three things to fix, refine or replace, three things to add, three things to remove (and then he added three areas to innovate).The results were:

- make the UI look modern.
- Store performance in large installations needs to improve. Store also wants more robust merging and better config management. [Niall: I assume this means line-ups.]
- Fonts need improving and more internationalisation is wanted so customers can penetrate more markets. They are looking at a CLDR solution.
- VMasDLL lets VW be a (better) part of a larger solution. The C interface speed and tools should also be improve.

Multi-core CPUs are becoming common so we must leverage them; this

really resonates with customers. Some solutions are easy, some are hard and controversial.

What made it: CLDR, 64bit VM, Graphics (Cairo), GUI Infrastructure, Graphics Designers.

Q. Reinout? Look nice and be usable can differ, especially when you employ professional graphic designers; he speaks from experience. The debugger icons have changed and had to be relearned. Arden notes the point; it has not started yet.

OS8 will have a GUI update too, and the classic modelling tool will be enhanced.

What else made it: shadow-loading, atomic loading, DLLCC speed, high-load thread safety when communicating with sockets, etc. They will also introduce posix compliance on delays (e.g. handle when long process goes past daylight saving change deadline).

Multi-core made the list: something will be done there.

Advocacy: one of their aims is giving current customers information to help them justify using Smalltalk. They give customers slides and info so they can argue it is a safe powerful choice. Another is to let past users know it is safe to return.

Perception of Smalltalk is changing. He showed a Gartner group report (not in readable-size font, sorry). Prior to this December 2007 report Smalltalk was listed as 'elderly' (not a good rating: means, "flee from this"). Gartner has now listed Smalltalk as 'mature': (means, "it is safe, move up to the current release"). Big organisations use Gartner. He is also very pleased with the number of people at this conference.

There is a whole new generation of managers and developers who do not know where Smalltalk came from. Now that Smalltalk is having its second surge, we must tell them. Look at the computer in front of you: its mouse, icons, drag-drop. It was all invented at Xerox Parc in Smalltalk 'The Technology that invented the future'. The office metaphor (desktop, files, etc., also came out of Xerox Parc). It was invented by a group that reinvented the language every two years.

Q. Stephane noted that while this is impressive to people who care about history, some modern Python developer often says, "yeah, great but that's history man. What's the relevance today?" Arden suggested emphasising that it did not just invent one thing but all the things that developer uses, and that a modern version of it is around today.

Smalltalk is a general-purpose high-productivity tool. It is portable, accessible (full source code etc.) and fun (which for managers means better employee retention). It is also an education: many developers will say, "It was not till I learned Smalltalk that I really learned OO."

"Smalltalk was ahead of its time". Smalltalk still *is* ahead of its time. Other languages borrowed the obvious things from Smalltalk. They missed the subtle things. Steve Jobs visited Xerox Parc (quote is from 'Triumph of the nerds'): "They showed me 3 things: I was so blinded by the first I ignored the other two" (one of which was Smalltalk). Others, like Steve, are so blinded by raw OO they miss other things in Smalltalk. They miss closures: other languages add control structures; Smalltalk can add any it needs. Smalltalk reflection is first class and so allows incremental exploration of it. The syntax is a jewel in Smalltalk: simple, consistent, expressive, robust. Alone of computer languages it was designed to be easy to read, which is why it looks unlike other languages. Arden showed my Memo example. If you are talking to an audience that may be disposed to think Smalltalk syntax hard to read, show them a purpose-of-talk slide in English:

```
Memo
  to: Ruby user group
  from: Niall Ross
  date: 2 sep 2008.

Smalltalk syntax
  is easy to read;
  is not weird.
```

Then show them the same slide in Smalltalk:

```
Memo
  to: 'Ruby user group'
  from: 'Niall Ross'
  date: (2 sep: 2008).

Smalltalk syntax
  isEasyToRead: true;
  isWeird: false.
```

Having been able to parse and understand the first text easily (because they knew it was English, not code) they will have a hard time claiming they cannot parse and understand the second, and you can explain Smalltalk's use of colon, semi-colon and full stop (period) easily by referring to their analogous meanings in the English version.

Developers in other languages hate having to work in the debugger. For Smalltalkers, it's fun. And snapshotting the image is like putting your notebook to sleep.

Cincom is seeing significant Smalltalk growth: customers are buying new licences and new customers are appearing.

He ran out of time so could not say much about Web Velocity: VW + Seaside + Glorp + ... It is not targeted at you: it is targeted at a sister department in your company (or wherever) who want to write web apps but are not developers. Use this to grow Smalltalk in your organisation. Changing languages prompts a lot of debate: choosing a web framework causes much less.

Gemstone, Martin McClure, Gemstone
Martin is not enthusiastic about the Gemstone product because he works there; he works there (for 10 years) because he is enthusiastic about the product. Gemstone is a Smalltalk implementation. Most of it is the same as any other Smalltalk. Martin will talk about what is different.

GemStone is a server smalltalk. You deploy by having a web browser talk via Seaside to a GemStone server or having a client Smalltalk talk via GBS to the server.

It is multi-user. Image-saving is a single-user model of persistence. GemStone is as-if you could save your changes and have them be merged with other users' changes. This is done in Transactions: `begin` updates your view of the shared image ('repository'); you now see all the changes committed to that point. You will not now see further committed changes. This property - read consistency - is only per select statement in many RDBs but in GemStone you will keep your consistent view until you decide to update it. `abort` discards your changes and updates your view as begin did. `commit` merges your changes and updates your view to that merged result.

Merging always creates the possibility of conflict. Optimistic concurrency detects conflict at commit time and fails the commit, letting you handle the result (typically by aborting the transaction and redoing the work). Pessimistic concurrency gets locks to objects. You are now sure you can commit changes to that object.

Gemstone's units of conflict are usually objects - two changes to the same object is a conflict. Logically, changing an employee's address and salary do not conflict but will be treated as a physical conflict by GemStone and happens rarely enough that it does not matter. However adding two new employees to the same collection may happen often, so GemStone offers reduced conflict classes, usually collection classes, named Rc... .

Not all objects are persistent. Objects that are reached from the Transient roots (which are the session state array and the stack) will be transient: they will live in your VM but noone else's (just as standard Smalltalk GC collects objects not reachable from the root(s) of persistence such as the Squeak system dictionary). Persistent objects are those reachable from the root of persistence, whose name is AllUsers.

Most Smalltalks are limited by memory. GemStone is not. It can handle billions of objects, hundreds of gigabytes of data, collections with millions of objects (so they offer an indexing and querying system).

GemStone is not an object database. It *includes* an object database but it *is* a Smalltalk implementation. Because the OODB part is so tightly bound, it is much easier to use than an RDB (which is why he changed his talk title; at first it was 'GemStone for dummies' but in fact it's the RDBs that are for dummies :-).

Migration in a Smalltalk image when you change a class takes a few hundred milliseconds. In GemStone that can be a much longer time; too long to do always immediately. So GemStone allows you to have versions of classes and migrate between them at your schedule. ClassHistory is an ordered collection of classes that lets `aFoo(1) isKindOf: Foo(2)` return true.

Q(Christian) Can you live with ten versions of the class? Yes. If you add an instance variable, you may migrate all your instances to have it and then add code that uses it. Or you may have the old instances' class return a default value where the new method is a true accessor.

Multi-user Smalltalk has to address security. A VM must access the repository just as an ordinary Smalltalk must access the image, so repositories have login and password. Some 10 privileges can be assigned in various combinations to various users: can alter other users' passwords, can save methods, etc. They have object access policies (actually called 'segments' but ignore this name; it confuses everyone, so they are thinking of changing it). There are read and write permissions, like UNIX permissions, where read can do anything that does not attempt to assign to an instVar.

Non-namespaced Smalltalks resolve names in the Smalltalk system dictionary. GemStone has multiple system dictionaries which are held in a symbol list. Bindings are searched for in each dictionary in the list in order. Each user has a symbol dictionary (so an object can be hidden from a user by removing its dictionary from their list but this is a mild form of security; they may be able to get a reference to the object in some other way). In the old days, Gemstone repositories were heavyweight things and not every developer had their own, raising issues of making the symbol list suit all users. Today, machines are more powerful and every developer has their own repository just as they have their own image in other Smalltalks. The main use of symbol lists is namespacing.

GemStone is not entirely headless. He executed code in VW to popup a GemStone window from the COLA project that Ian Piumarta is doing. It did simple UI stuff - type in box, click button, etc.

Q? SQL tools on a GemStone database? SQL assumes your data is relational. It is hard to do something generic, though you could certainly do something on specific data. GemStone is 25 years old and has not needed it yet. GemConnect lets you connect to other SQL databases. (Bruce Badger: the PostgreSQL drivers have been ported to GemStone so you can emit SQL to dump data to PostgreSQL.)

Q. Speed v. RDB? In SQL the tables have (generally) fixed size rows and use foreign keys to do lookups. A multiple join will be effortlessly beaten by a GemStone pointer-following for most object graphs. The OR mapping layer will also add some performance cost. A recent customer post said that their Gemstone system was doing 700 transactions per second and their RDB was doing 70 transactions per second on 3-4 times the hardware.

However you can always construct RDB-oriented examples which would show them ahead.

MagLev: Ruby that Scales, Monty Williams, GemStone
Monty is one of the founders of GemStone; he's been there since 1982. He wants to get Smalltalk technology into as many hands as possible. To some Smalltalkers, this talk should be called Ruby with Scales - how does it help Smalltalk.

MagLev is a way to run Ruby code in a 64-bit GemStone/S VM. So why should we care? Well, because it runs Smalltalk too. The more people who run Smalltalk the better for all; there are a lot of rubyists. Monty went to RailsComp in May this year. Obe Fernandes CEO of HashRocket gave a talk about the worst rails code he'd ever seen (includes 3000-line methods) and stated, "If you really want to know how to make stuff better, find someone who has been doing Smalltalk for the last ten years." There were 2500 people at that conference.

Why should Rubyists care about MagLev? Because it is a lot of things that Ruby isn't. MagLev is fast (Ruby is not as slow as people say, but...). It is stable. It is scalable (there are some large Ruby apps on twitter, but ...). It is distributed. It is persistent. It replaces a bunch of components such as mongrels that you would see in a Ruby app with a single thing whose components all work together. Above all, it is turtles all the way down. A user saw a 15 times speed up going from Rails to MagLev.

We should each befriend 100 Rubyists and show them Seaside.

He showed Journal On Software (well known blog) quotes from two clients about GemStone's superior performance. A shipping company managed 70 transactions in Java on three times as much the h/w as managed ten times as many transactions in GemStone (see February 16th 2008 post).

He showed the MagLev architecture: it is the GemStone/S architecture plus some MagLev smalltalk extensions, then on top of that the usual Ruby (Ruby core library, Ruby gems, etc.). They are trying to rewrite more Ruby in Ruby (actually, in Smalltalk) to slim this.

He showed a side-by-side demo. He fired-up, opened a Squeak-tools view and showed a GemStone browser with Ruby stuff in it. A demo to keep rubyists happy types into the topaz interface: he typed Smalltalk code, then Gemstone code. At the Rails conference, some people were simply (and some were literally :-)) incredulous about what he did next: he ran the YAR benchmarks that Rubyists use to compare side by side. Normally he starts the Ruby then starts the stone of the MagLev to let the Ruby have a bit more time and help reduce their incredulity (but not enough; some frankly asserted MagLev couldn't be actually doing anything to finish so fast; they could not believe that they could build such a system in 100 days).

He then ran the short YAR again on the Ruby side (if he ran the long one it would take the whole talk) and then went to the MagLev and ran the long

one which took very little longer than the short one which took almost no time at all.

Q(Stephane) Method dispatch; Ruby can add methods to individual objects, make methods private? MagLev does not do method privacy. They handle object methods by adding lightweight singleton classes.

Q(Stephane) if the Ruby people have a fast virtual machine, will it bring people to Smalltalk? Only consultants and book writers make money in Ruby. They say they are trapped in the web ghetto by stability and scalability issues.

Q(Reinout) will there be a Ruby calling Smalltalk, Smalltalk calling Ruby? We have not decided. It is not technically hard to do. Ruby people like the idea of hit a button and reload all your classes, so will they excuse Smalltalk classes from that. (Follow-up Seaside on Rails? Ugh!)

Ruby has thousands of tests that are the best spec for what Ruby is (there is no real language spec). They hope to give it to 10+ companies a limited private beta and circa RubyComp (November 4th) a public beta. Pivotal labs are redoing twitter; they are talking to them.

VASmalltalk 8.0 and Beyond, John O'Keefe, Instantiations
I've incorporated all the material, discussions and questions from John's StS and ESUG presentations into his most recent one at Frankfurt.

A Moribund Smalltalk still alive and kicking: The APIS VisualSmalltalk IDE, Thomas Brey, Heiko Wagner, Jan Kaiser, Andreas Rosenberg
Jan summarised the history of VSE. Digitalk published Methods in 1984. He showed a screenshot of what it looked like then ("Now we understand why Squeak looks the way it does"). In 86, they published Smalltalk V. In 1995, ParcPlace and Digitalk merged; just before they did, Digitalk published VSE 3.1. No release has occurred since so it has been a challenge to keep it running on newer platforms.

In 1999, VSE's rights were sold to Cincom to support it and to Seagull who own the code. Cincom released VSE3.2 as a maintenance release. Other help was viral mailing lists. Raimondo in Argentina and others shared bug fixes and tips.

VSE has no web technology features, no common controls, no unicode, no multithreading support. (Discussion with Eliot, Georg and others: S# does what Strongtalk does which is have one thread running at a time and callouts to others. Smalltalk MT does have multithreading support and the old ObjectStudio also has it but no more. Lesser Smalltalk also does it. But it looks like VSE is hardly unique in not having it.)

Since many users have modified the base classes to deal with making it run on recent platforms and with the above, integrating fixes between users can be hard.

APIS provides Risk and Quality management tools. He showed some screens, oriented towards dividing systems into components and thinking about how these components can fail, how these failures could interact with each other, etc.

They added a refactoring browser and a unit test browser. They tweaked their code pane to do code highlighting, formatting, autocomplete (similar to Vassili's class search box). There is also in-place renaming of temps: rename the var in the temp list and that is a rename refactoring applied to the code. (If the method is in an incomplete state so that it cannot do a refactoring, it silently fails to do the rename.) They have a configurable formatter. These are not unusual in modern Smalltalks but it is of interest that a such an old Smalltalk can look so modern.

Their Java Standard Edition 6.0 integration was developed independently from JNIPort and JavaConnect. The used proprietary APIS enhancements and some VM enhancements. It creates wrapper classes from .jar files and lets you use the classes in Smalltalk, calling the Java machine to process whatever calls the Java received. They find it useful to be able to use Java libraries to get many features (e.g. Unicode) and integrations (to frameworks, databases, etc.) that are not in VSE.

They use JNI for communication,. The Smalltalk VM had to add IEEE-754 floats and 64 bit integers.You create a Smalltalk SSL from any .jar file. You can view, edit and compile Java methods and classes within the Smalltalk IDE.

He opened a tree browser on a file with some Java classes, imported them and checked for consistency (and refs in these classes to others not available) and exported an SSL created from these. He called it and 'hello world' appeared on the transcript. It could also say hello from named objects.

He then opened a more complex example. It took a few seconds to open and then showed a long list of icon-distinguished (for public/private) Swing classes. He named a package to hold them. Running consistency would indicate references to other classes in other packages that you would need to import.

He then opened a hyperbolic application browser. He had created the SSL previously so just had to install it and (deal with usual demo hiccough by also loading the Smalltalk class that calls it and)

```
HyperbolicApplicationBrowser new open
```

opened Java UI code using Smalltalk data in a Smalltalk window. It ran perfectly. He clicked and explored in the Smalltalk class browser.

APIS bought the code from Seagull, so are able to do these VM changes and so on.

Thomas Brey and Heiko Wagner did most of the work.

Q. Typing? The datatypes are matched: Java string to string and so on. They are wrapped, not replaced by Smalltalk objects. The VM does the translation.

Q(Dan Poon and Reinout) Performance? Pretty good; the code runs on the Java VM and of course there is the wrapper overhead.

Q. Smalltalk or Java? If we can do it in Smalltalk, we do it in Smalltalk. If a library for e.g. XML parsing exists and is tested and robust, we use the Java.

Q(Dab) VM mods? Mostly for the type conversion.

Aida and Seaside

Aida, Janko Misvek, Eranova

Alan Kay suggested in 1997 that every object should have a URL. Janko knows noone else who has implemented that. `anObject printString` is basic to Smalltalk. `anObject printWebPage` is basic to Aida.

Q(Stephane) Recursion? Primitives? On the web you can go from one page to a second page and back to the first page; so when navigating a graph of objects. We do not have URLs for primitives.

Aida supplies starter applications: Blog, Wiki, general Site. It runs on Swazoo over VW.

He showed some Aida websites. (Some old sites are hard to put into Smalltalk because they do not have clean CSS/content separation.) Dirk Verleysen did a Belgian football team site. Nicholas Petton did a TalkingSmalltalk site and Blog plugin. He also created a Smalltalk project management system. BiArt is a commercial quality management system supplied by Eranova (see Janko's talk last year) and Scribo from this product is open-source. eLogis is another process management system they supplied. The Slovenian gas pipeline operator also uses them for their site.

Scriblets are embedded components. The Aida slogan is an edelweiss flower. It implies that their system is very resistant to harsh conditions and that they cover the web from sea level right to the top.

Aida/Web 6.0 and Aida/Scribo 1.0 have just been released. It runs on Squeak, GemStone/GLASS, VisualWorks, and Dolphin. They plan to run on Smalltalk/X (will be supported in 1 month) and GNU Smalltalk (gets them closer to the Linux community).

There are 60+ on mailing list, 2 core developers, tutorials in French and Spanish (and English).

In future, they will support tree-like navigation as well as graph-like. Major domains are graph-like but tree-like is good for yes/no dialogs and similar decision workflows. Thus graphs will have tree subgraphs. This will be their way of doing some things for which Seaside would use continuations.

Aida MVC separation usually puts actions into separate methods but they are thinking seriously of `onSubmitDo: [self observee save]` i.e. allowing block callbacks, avoided before for fear of breaking MVC and getting spaghetti code. Thus they may allow verification code but not general code in such blocks.

Internationalisation is important: they have Japanese in their group. The simple solution, largely in place, is to have the same web page structure with different content for the different languages. In-place translation is being thought of. Translations can be saved in methods.

They are producing a book (in French at first, in English soon) and making some old Squeak sites (SqueakMap) look more modern to make things more accessible to non-Smalltalkers.

Magritte Blitz, Lukas Renggli
This talk told us what Magritte is not, what Magritte is and what Magritte can do, then demoed.

Magritte is not trying to model UML. It does not use code generation, just visitors that walk the classes. And above all it has no fundamental link to Seaside. a Plugin generates Seaside UI from Magritte descriptions but that is just one application.

Magritte is an enhancement of Smalltalk's reflection capabilities to define an instVar as being e.g. a String with a label, etc. Magritte is extensible and described in itself. Magritte editors can be used to edit Magritte descriptions to alter e.g. a web application on the fly. It runs on VW, GemStone, Squeak, Pharo and most recently GNU.

Magritte can build viewers, editors, validators and reporters (the Seaside plugin provides all these for Seaside). In Smalltalk, you must write initialize methods or you can instead use Magritte to define initial values for your objects. You can use Magritte to decide what to copy and what to omit when copying a graph of objects. It can be used to document your classes. It can be used for DBs, XML and query generation, JSON. And it can be used for end-user customisation.

Lukas opened Pier and edited it using Magritte web-rendered edit page. Then he opened Squeak browser and showed a very similar looking form generated by Magritte using the Squeak renderer instead of the Seaside renderer.

He then added a 'ToDo' field, again using a Magritte-rendered form (back in the web) and gave it a label, set its options, etc., and saved. Then he edited the page again and had the ToDo field appear in various different ways.

Q(Christian) connecting between attributes? That is the weakest part of Magritte. There is some code but he also finds he needs to add glue code by hand.

Q. The new state for the ToDo field goes where? In a dictionary.

Q(Niall) Visitor subclassing, delegating? The edit layout that was the same between Morphic and Seaside is a general layout superclass with Morphic and Seaside subclasses. No more general subclassing or delegating pattern has been needed so far.

Hands-on Pier, Tudor Girba
(Tudor's slides still had bullets but he apologies for that.) The Pier CMS is easy, cool and free. Lukas Renggli built Smallwiki, then Magritte and then he built Pier as an application of Seaside and Magritte. Others then joined in, including Tudor. He is speaking because he uses Pier and because Pier *is* cool and *is* free; making such products more used is good for Smalltalk.

He opened a Safari browser on Pier. It shows the title of the whole app, some text and the Seaside toolbar. He browsed around: the usual info, licence data plus blog - and you can login (admin, pier).

More buttons appear: _Environment _User Management and so on. He changed the title and saw it twice - on the current page and as part of the wrapping environment page - the metapage. He went to the environment page: it showed normal html for the overall page. 'class title class' means that the title of the current page appears at the top. Instead he put 'logo' and then got a red link (no logo yet) and clicked on it. He is prompted to define the link type which he made a file, selected his logo file and saw the logo.

However it is not clickable. Pier uses an extension of the swiki syntax. He converted logo to *+logo+>root* and now clicking the logo takes you to the root of all pages.

Thus he can edit normal pages and administrator pages.

Web UIs are often rather clumsy in forcing you to use the mouse all the time so he did CTL-E (was in edit page) changed some text and pressed CTL-S and there he saw it. (This works for Pier whatever web browsers you are viewing it in, but may use different keys on some platforms.)

On any page, he can do CTL-D (edit design) to be editing the environment page. CTL-D also shows the CSS file. He made `.a` have colour green, showed it, then edited it back and redisplayed (no change) and did refresh (saw change); that is Seaside 2.8.2 extreme caching in action.

The structure of Pier is inspired by the Unix file system. There is a root and a hierarchy. This if you go to CSS, you see ...root/..../css. So which environment does a given page get. The root page has settings environment and environment/css. Go to the information page and press CTL-D and you see nothing: the page inherits from its parent (the root). Add CSS and your page will show the inherited root CSS overridden by whatever you have added. The environment of the environment is also environment.

He made a blog post, then decided he wanted a summary of the blog in a

main page. He added to the environment page, which is good practice - it makes an added element more manageable - but is not essential. He added a +postticker+ and added it as a component. There is a component for this and he set its values to show the last two posts and so on.

He does not want to see this on all pages, so he wants all other pages to have a different environment. He creates a new environment called _Main Environment (the underscore prefix is Tudor's convention to know what is a meta page and what is a normal page). The code in main environment points to things like logo and so on that should be in Main Environment but are not. He opened it and edited its html to have absolute paths for e.g. environment/logo.

However the much easier thing is to go to the old environment and say 'copy', changing the title to _Main Environment. When in a recursive page, look at the first part of your URL. That tells you where you are. He then made the other pages use _Environment, leaving the _Main Environment to show the postticker. This was done as an environment example. He could of course have added the postticker just to the main page. He did not merely because he wanted to use explicit html to place it and it is a pattern to keep that to environments and only use wiki-style markup in the normal pages.

Q(Thorsten) versioning? Click changes to see what you have done. You cannot see differences yet.

He browsed the Unix-stye permissions. He added the logo under environment which is owned by administrator and can only be viewed by the group. He wants everyone to see the logo so changed its 'other' permissions. (He had the usual demo hiccough; at first, he changed permissions to the wrong thing.)

[Niall: when presenting Pier, start with set up environment and show how it works, *then* do administrator. Stephane said the same. Presenting it in the order Tudor used was as if one presented Smalltalk from the metaclass diagram, and risked confusing the audience before they saw the power of the approach.]

Tudor showed examples: moose.unibe.ch and www.esug.org (has just moved to Pier). There will be a movie and documentation. There are components for Citezen (bibliography tool), Blueprint, LightBox, Randomiser (have you noticed that the Seaside slogan on www.seaside.st changes), Top Feed, Twitter, etc.: all these are listed on the main page.

Blueprint shows 24 lightly-shaded stripes, offering a grid for arranging your page. You can use these in layout. The environment can say where an item starts and how many it scrolls over. This example is a fixed size but they offer two fixed sizes and you could also make it scale it to the window.

A Pier Structure has children and subclasses Page and File. Structure knows its Contexts which relates to a Command (such as Edit and Login).

A Structure has a Kernel which knows about Persistency.

So far is general: a Structure has an environment which is a Page. Seaside is exploited by the fact that another subclass of Structure is Component which lets you add any Seaside application. Context subclasses to View which subclasses to Browse and Changes views. A Page has an environment Structure.

Persistency: you can use image persistency as he and Lukas do (cron job saves image regularly). You can also use File persistency or GemStone persistency.

Lastly, Pier on the iPhone ("Good" - "No, it's not good, it's cool; this is about making Smalltalk cool"). He opened the simulator, and showed how the pages had been laid out to make them useable. (He could not show the environment pages, probably because of a problem with the rights.) Walking round the pages was fairly straightforward. It took 100 lines of code to make it work on the iPhone. He also showed just browsing the pages as if on a web browser but the iPhone layout is more useful for a user making a quick edit to the site.

Q(Torsten) indexing PDF indexing when adding docs? Not by default.

Web Velocity, Jim Robertson, Cincom
Jim showed three pictures as metaphors of the developer without Smalltalk, the developer with Smalltalk and the developer with Web Velocity. The pictures were taken from the social event. (Yann was the 'developer without Smalltalk'; he assured me the appearance of his having raised his hand toward his mouth only to notice suddenly that it did not contain a glass of beer was a pure artefact of the moment at which Jim snapped the shot. :-)

Jim had a long discussion with Avi in 2004 explaining why Seaside was a bad idea. He has changed his mind since. :-)

Web Velocity tries to avoid the flow breaks that occur when you work in browser and debugger but present in Web browser.

He started WebVelocity in Firefox. The left side shows list of repositories: the Cincom Open Repository, Jim's local one on SQLLite, etc. Under that is a list of apps. The bottom left is a list of libraries; things implemented in Seaside that are not yet loaded; he can click one and see a lightbox with an expandable list of the blessing comments.

Next he went to the page that walks you through how to build your web velocity. The target audience is people who do not know Smalltalk, so they have made the documentation part of the environment.

Next James clicked to create an app, typed 'Hello World' for its title.

He saw a page with left list of items some in red (e.g. Overview was red

because we've not yet written some documentation for it, Databases was red because we've not configured any for it, etc.).

Jim got the standard demo hiccough at this point: "This worked last night". He worked on sorting it. James Foster turned off the network that could have been confusing the system and Jim got a new build but it still did not work. Eventually, after the talk, Jim worked out the issue: the combination of a particular new Firefox 3 version (3.01 IIRC) plus the specific network setup at CMI plus WebVelocity was causing the problem. It ran fine on Safari, Firefox 2 and other browsers. Meanwhile, the demo perforce had more discussion and screenshot than intended.

He opened the Hello app:

```
renderContentOn:
  super render..
  html text: ...
```

The browser shows him the syntax error highlighted in red. The Seaside walkback debugger opened, highlighting where the problem was. He can open an inspector in the debugger. He fixed and (would have but for the demo hiccough) resumed.

He loaded an existing app and looked at Glorp mappings to the database.

```
aTable createFieldname: 'id' type: ...
```

He opened a very basic blog app, provided as a suggested start point for users building blogs and similar system. He showed the drop-down list of components available and the methods for one.

WebVelocity is building migration code. He discussed `recreateTables` (again, demo hiccough meant he could not show it). Arden offered his computer, on which the build was working. Jim created a new application, then selected New Web Component from drop down. The main pane showed its instance and class methods. He clicked to invoke

```
renderContentOn: html
  html text: 'hello there'
```

He can open several methods in the page and not have to accept the first before editing the second. This is a non-modal method-editing system. He ran it, then wrote an error and attempted to run, thence saw the debugger in the web browser, fixed it and resumed.

Q. Are you eating your own dogfood? The Web Velocity website will be written in this. The Cincom Smalltalk site will be transition to it as well (but not the blogs as their code is stable already).

Q. Licence? There will be a non-commercial version and an online commercial version to make it easy to acquire.

Q. Headless server? Just type `visual.exe -nogui myimage.im`

Q. When you save, that saves all the methods? No, just the one whose context you are in when you save.

Lastly, he created a new application database.

Michael's blog shows WebVelocity movies, as does the main Smalltalk site.

Q(Christian) this will feed back into VW? Seaside 2.8 is in VW7.6. A lot of the pieces of WebVelocity will ship in the standard release. Database migration will be part of Glorp.

Q(Tim) WebVelocity final name (confusion with Java Velocity framework)? We've settled on WebVelocity as final name.

Q. How are databases set up? If you populate an empty database using the defaults, you get a pattern similar to Rails' active record. If the tables exist, it will read their metadata from the database and create the connection code.

Seaside Evolution; things you never knew you could do, Julian Fitzell
Julian is co-inventor of Seaside with Avi. Seaside has been through the Experimentation, Stabilisation, now Optimisation phases and Adoption is where we're going.

WebObjects was a good framework in Objective-C but moved to Java so Avi wrote IOWA in Ruby as a fairly close port.

```
class Main < Iowa::Component
def all_people
  Person.fetchAll()
...
```

and in the html

```
<ul oid="people"
  <li>[@person.firstName] [@person.lastName]
...
<?
people {
  item = person
  list = all_people
}
>
```

but it only reached version 0.4 because Iowa was released in January 2001 and in 2001 they discovered Smalltalk. Seaside: Squeak Enterprise Aubergines Server. Why aubergines? He wished we hadn't asked: Avi thought of it as a sound-alike to Java Enterprise Beans.

```
addBindingTo: template
  (template elementNamed: 'people')
    bind: #list toPath: 'allPeople';
    ...
```

and so Seaside 0.9 was born. He showed examples of conditional tags in

UML, and embedding code in HTML and so on; that was the architecture of Seaside 0.9 - there wasn't any. Everything depended on everything else - the session on the components on the ...

So they made a major rewrite to Seaside 2.x (called Borges, after a quote from 'The Garden of Forking Paths'). They found templates just got too big, too 'magical' and too hard to manage. (It got ported to Ruby, went in a different direction and is still around somewhere). Seaside 2.0 added callbacks for blocks and an html renderer. Seaside 2.3 moved the rendering to the top of the architecture layers so other things did not depend on it.

After 2.3 they got into the stabilisation phase. Initially all component state was backtracked and no session state was. Stabilisation reworked this. It also reworked the rendering to the Canvas API so no longer having all the combinations exponentially growing (button, anchor, anchorWithButton). At around this time, Julian got a full time job and stopped working on Seaside. Around 2005 Avi stopped working on Seaside in favour of using it (to build Dabble DB) and Lukas and Philippe took it over. They worked on optimisation and refactorings for flexibility, pluggability and so on.

Julian now feels we are entering the adoption phase with GLASS, Pier, seaBreeze, WebVelocity. Documentation remains weak but there are books and much on the web. To help documentation along, Julian will now discuss architecture, metaphors and pluggability aspects that are less known.

Slide of Seaside 2.9 architecture. Tests, examples and development tools can be loaded or not. The core is the render loop and the components that have them. Components use the Painter which uses the Canvas, and Components use the Session Management. It sits on Request Handling which converts Comanche or whatever requests to Seaside requests and vice versa using a Server Handler of the appropriate type.

Q. Why Seaside request? Mostly it is for portability, to use Seaside with any webserver.

The Painter is a place to hold html rendering code when you don't have session state. Earlier, there was a stateless component and he is thinking of putting it back. Components likewise do not have to use the render loop but normally will.

Q(Christian) Why do you make the html by way of a stream rather than as a tree of objects.

WAHtmlTreeDocument is an alternative canvas to WAHtmlStreamDocument but the latter is used for performance (and may not work as a replacement at the moment although it should). Most browsers will display stuff as it arrives so streaming gives the user a significantly different, faster feel.

The metaphor is:

```
main(){                                          WARenderLoopMain
  root = new root component
  while(1){                                       WARenderLoop
      render(root);                            WARenderContinuation
      process_callbacks(root);
      redirect();                              WARedirectContinuation
}}
```

The redirect is to a new page that shows HTML so if you refresh you are not resubmitting.

Julian listed some areas where they thought people would plugin but which have not been so used. Configuration was one. Another was custom error handling, e.g. to send an email to yourself or to save an image snapshot to explore the error later. A request Handler could implement REST API using the Canvas but ignoring other Seaside stuff. A session expiry handler could be done. Toolbar and Halo are also pluggable.

Configuration (in 2.9; 2.8 is different): subclass WASystemConfiguration, implement `describeOn:`. Optionally implement parents to inherit from other configurations. Then go to the config screen, add your config. Ensure your names are globally unique.

```
describeOn: config
  (config string: #myappSoapHost)
    label: 'SOAP Host';
    comment: 'My App SOAP server hostname'.
  (config list: #myappTheme)
    label: 'Theme';
    options: [self allThemes].
config at: #sessionClass put: MySession.
```

Similarly, to create your own Error handler, subclass WAErrorHandler, implement `handleError:`, optionally also `handleWarning:` and `internalError:`. Then select this error handler in your config.

```
handleError: anError
  session := WACurrentRequestContext session.
  self
    sendEmailForSession: session
    error: anError.
self forkAndSaveForDebugging.
```

A RequestHandler subclasses WAEntryPoint. It needs to override `handleRequest:`. (Optionally, implement your own configuration and add it as a parent in `defaultConfiguration`.) Add an instance of the subclass to a dispatcher.

Session expiry handler is similar: subclass WAExpiredSessionKeyHandler and override `handleRequest:` (to return a response).

Q(Annick) Any XForms implementation? Not that I know of.

Q. Scripts and CSS? SeaBreeze does that. In Seaside 0.9 parsing the templates needed modelling the entire HTML 0.4 spec which was brutal. Those syntaxes are large. (Niall: Michael did it in WithStyle.)

Q(Stephane) sometimes successful technologies become less successful as technology changes around it; any such things upcoming? Not that he sees, but the biggest risk is non-adoption. We must grow Seaside while people are watching.

Q(John O'Keefe). How do you decide when 2.9 is done? It will be done soon. Lukas and Philippe worked on 2.9 all the way through, he has just been in the last few months, so they will decide. They just have some bugs to fix.

Q(Annick) Browser-specific canvasses? They have avoided that and hope they can continue. Of course if you modelled CSS then you would have to.

Glass: share everything, Dale Heinrichs, GemStone
Dale has been working on Seaside in GemStone for 2 years. GLASS uses Monticello for source code control and OmniBrowser for development tools. It is easy to move.

"The Applicance" is VMWare configuration running an instance of GLASS with 3 seaside VMs, 1 Maintenance VM, and etc.

'Share nothing' could be characterised as 'Hit the database every time you need anything.' This is well suited to when the web server is stateless and the model is simple. Scaling is limited only by how much electricity you have (and h/w and ...)

Smalltalk is share everything: everything is in the image. GemStone/S handles very large images on multiple VMs; it lets you share everything with share-nothing scalability.

GLASS started with a single VM serving all sessions. After two months they dropped that in favour of shared session state: let any VM server any session. Finally they made _k and _s optional.

They abort start on session, commit if you changed during session. They use apache to round-robin requests to a number of VMs, with persistent session state meaning do not have to synch VM to session.

However every request changed persistent session state. 10 - 100 commit per session (10 million requests per day) will drives you to sophisticated hardware rather than commodity hardware. So he aimed to avoid saving 'unnecessary' session state. Seaside creates a continuation for every page (WARender, WARedirect, c.f. Julian's talk) but you may not need them to persist. By not doing so, he did an experiment and reach 7k sessions per second on a commodity hardware config (7 machines, 72 CPUs, 128 VMs) whereas if he saved every request he dropped back to 200 seconds.

The third option is session affinity. You could then use temporary memory. Two years ago, Dale thought, 'One VM per session? No way', but now he thinks, 'we could'. This is a two week old idea they have explored to ensure it has no killing drawbacks. It will be a strategy, chosen for its trade-offs.

Q(Eliot) Hydra? Gemstone is exactly like hydra. Hydra will minimise the memory footprint of multiple Squeak images. Gemstone does similar things, uses nmap and so on.

So you have 150 VMs running on 7 machines; how to debug? They autocommit: a saved method gets saved to all the VMs immediately to simulate the experience of working in a single VM environment. You would never want to look at 150 logs or 150 Transcripts for where your `show:` appeared so the object log acts as the transcript. Breakpoints likewise are distributed to all 150 VMs so any one of them can be the one getting the bug. Click on profiling: the VM that gets the request is the one that profiles.

He opened the ObjectLog in the web browser. The startup lists the VMs and the pid they start on. Entries have priorities so you can sort on these fields. The oop is shown (useful for saving your having to do == and -- checks). You click on objects to see more details.

He opened the counter app. and also opened the Squeak image that was connected to GemStone. Transcript show: 'Hello World' was executed in a VM in the appliance. The object log shows that 'Hello World'. The Transcript showed an array and so it was in the object log.

Q. The GemStone VM serving the Squeak image is running where? On the appliance.

He then opened a Seaside walkback. It has a remote debug link. Clicking on it creates a resumable continuation (showed it to us in the object log). He continued the continuation then went back to the walkback in the web browser and clicked the resume button. If he'd clicked it before it would just have told him he had not continued the continuation and so come back around but now it completes the web request. He looked in the object log and saw the same continuation, still there but now no longer saying it is a resumable continuation,.

So, in summary, when you reach the error handler it snaps off the continuation and saves it. A resume sticks the persisted process back in the continuation and resumes.

He toggled profile and in his web browser could see 'profiling' and the pid of the VM that got the profile request. He tried to bring up the profile and got '11 is too many retries' so this demo hiccough became a demo of debug. He had FastCGI errors; he'd had a timing error in the profiler that made it pick up a reference to a semaphore, which could not be persisted. The '11 is too many retries' is because they try 10 times.

He almost forgot (Christian reminded him) when he'd finished explaining this to bring up the profiler (by just doing back-button and retry, and it appeared). The profile tabulates the receiver class, the implementor and the data. You can step back from line to line in the call stack path. Clicking on the implementor shows it source code.

Q Licence? Size of DB 4Gb and a single server. That is free. The next step up is $7000 but they are looking at other models. They may break the steps into the two axes of number of hits and size of data: do you need more CPU, more shared memory or more disk? They may make more disk space cheaper than more CPUs.

Q. Tuning? Seaside is a known application which means they can work in the lab on what to tune and how to tune. A greenfields application would be harder because GemStone would not have that experience of the app, so they can supply an appliance with good tuning choices and good advice on what to look at.

Maglev and Gemstone 3.0 are the same product.

Seaside, Lukas Renggli
The seaside-dev list is for developers not users, people who contribute code. (So why are there 80 members? Perhaps just people interested to see what is coming.) They were using the Mantis bugtracker but they did not get enough bugs so switched to the Google bug tracker. 250 people per day are visiting seaside.st. 800 people are on the seaside list. Go there if you want to ask questions.

They had 10,000 downloads of the one click-image last year! It got a lot of attention from blogs and outside the Smalltalk community and was one peak on the graph. The other was their April 1st joke that got mentioned widely. (They pretended they wanted to port Seaside to Java.)

They are running on Squeak and Pharo (which Lukas uses for development). Seaside is also on VW, GemStone and on Dolphin (up-to-date there) and on VASmalltalk soon (see John's talk tomorrow morning). Lukas wants to thank Dale (Dale: "you guys make it real easy") and others.

Syntax for portability: do not use underscore. Do not use {1 2} brace arrays. GemStone also has array constructors and they don't use them either. Do not use ByteArrays [1 2 3]. Do not use variable bindings like Core.Object. Do not use GemStone selection blocks {:i | i.is.permanent}. Avoid `ifNotNil:` and `ifNotNilDo:` which are incompatible with other Smalltalks. Other syntax can be used. Pragmas, e.g. <javascript: 1.5> are OK (see seaside.st).

Do not compare collections, Collection>>= differs between dialects. Use `keysAndValuesDo:` not `withIndexDo:` Do not use `pairsDo:` since it is not dialect consistent. `1 to: aCollection size by: 2 do:` works everywhere.

Strings: Symbol is not a String in all dialects. String>>match: is totally broken in Squeak for UTF-1 and is not consistent generally. Converting via `asString` has the problem that VW breaks if you implement `asString` on object. `displayString` is also used so they now propose to use `toString`. (Bruce: Kapital uses `toString` already; Reinout: it is in a couple of libraries; suggest to use `asSeasideString`). They also avoid

IO since platforms are very different for file handling and so on.

Rather than make everyone remember that, they added new rules to Lint to target Seaside problems. Slime (Smalltalk Lint ...). These also address e.g.

```
html div with: 'Hello World; id: 'message'
```

`with:` is not the last to be sent; it should be.

```
html updater
  id: 'message';
  callback: [:r | html byte: ]
```

The callback should use `r` in the block, not `html` (the parameter is there for a reason; use it).

```
renderContentOn: html
  a := self call: WACounter new.
```

Do not call when rendering: we have tasks for that. Do not change component state when rendering or create components while rendering: refresh will invoke it.

Slime also looks at style issues: calling `div with:` instead of `div:`, or `div: [html text: ... ]`.

He showed the tests bar for 2.8 and earlier, then asked where are 2.9 tests: "Not enough space on this slide to fit them in.". They had 131 in 2.8 and are nearly at 400 already in 2.9.

In Squeak they use Monticello for versioning. Cincom reads the Squeak file-out into their image, similarly for other dialects. GemStone, by contrast, implements Monticello and Lukas strongly recommends that vendors implement Monticello2. It is the future of Smalltalk interchange. Dale agreed: whenever he saved to the repository, Lukas and Philippe would respond to his work without his having to push. The effort to port was one month to get Monticello working and one month making zip-file work. Lukas stressed Monticello 2 will be much easier.

They do not use Sport. They have a SeasidePlatformSupport package.

Q(Bruce) why? Discussion of some things they needed which were not in Sport which can then be moved to the standard. The aim of Sport is to kill Sport and get things move to the standard.

Only the core code should contain WA-prefixed classes so please others do not reuse this in your applications and in Vendor specific code.

They would like people to use Google bug tracker but if not please use a public bug tracker. Please use Monticello 2 to reconcile your changes to the main branch so they can see them.

They want to support multi-CPU. Squeak hydra project will be supporting this.

They want a better GUI.

The Seaside sprint followed the conference starting Friday afternoon and running on through Saturday to Sunday morning. The location was initially a cafe near the rail station but then moved to the new Amsterdam library which proved to be ideal.

Download the iPhone Pier from source.lukas-renggli.ch/isea (iSqueak came 3rd in the ESUG Smalltalk Awards).

He opened his iPhone simulator. It will run on any WebKit browser. Lukas browsed some methods and edited one.

Q.(Torsten) Documentation? Class comments are much improved.

Modelling Tools and Methods

MBA Smalltalk: to manage your objects, Mathieu van Echtelt, CosmoCows

In 2001, they started work in Squeak with pre-dot-com-crash plans that they found they had to revise. In 2002, they did consultancy work in Smalltalk for a US company, so remained solvent. In 2003-4, they started a product 'the ultimate contract and invoice management system'. They presented this in Brussels.

In summer 2004, they got a customer in a rather unusual way. A VAR who was aware of their system saw news reports about the Dutch fire brigade needing to manage their authorisation of whether businesses, e.g. pubs, were fire-regulation-compliant. Such regulation compliance has similarities to a contract between the pub and the fire brigade. They therefore visited to present their ideas - and discovered that the fire brigade had a much more urgent issue: they needed to manage training their people to ensure that e.g. first aid refresher course requirements were met and so on. Their product was a framework that could be transformed into solutions to this. From it, they built two business administration systems: 'Ready' and 'Status'. The created another company AGS to sell these. The more conservative name was better, as was the all-Dutch company website.

Ready is used by the fire brigade and heath care institutions to train and assign rewards for training. It is a web app. He logged in and showed some screens. 'Status' managed scrum-like sprints. He logged in, created a ticket, assigned the reporter and so on.

Mathieu created the name 'MBA Smalltalk' for this conference to explain the framework. A typical 'software street' has people responsible for Java client, for system code, for database and so on such that adding a date field can take a week. Their aim is to free the modellers who use MBA smalltalk from having to think about persistency, GC, etc.

In MBA Smalltalk, an editor lets users manipulate model description language, from which a generator creates Smalltalk code. (At first, they had an interpreter that ran model description language directly. However

the interpreter was too slow and they were successful in getting contracts to create models rather than selling the tool to managers to create their own models directly.) A renderer creates the web output from this, with styles,. Automatic tools aid configuration and line-ups, and make deployment a one-click process.

ESUG 2005 showed the first version of the system. The second version still had a web-based development UI. The third version, inspired by Squeak Morphic, let you go directly to code from web widgets. In the latest version they provided a Smalltalk UI for development and used Store as their repository. A fifth version is in development. The web output retains the morphic ability in development mode: widgets have task bars with 'inspect subject', 'pick slot', etc.

Their modified refactoring browser has a SchemaSource tool: left pane shows instance variables, right pane shows source. They use many pragmas.

He logged in on the web. In Smalltalk, he browsed a demo class EsugMenuBox. He created a contract allocating modules that included the ESUG module. He started with three empty schemas for person, organisation and report.

In Smalltalk he created a name method for Person with type and dataType pragmas, resynchronised and showed the string field on the web page. He then added more <propertyAt:put:> pragmas for #label, #isMandatory, #isVisible. Making it mandatory displays a red dot icon by the field.

Q(Stephane) Resynch? Generates Smalltalk code from pragmas.

Q(Stephane) Validation? Simple validation is done via pragmas. Complex ones (e.g. Dutch 'flex' contracts) are done in Smalltalk code.

Q(Christian) You mix presentation and data model elements? Yes. One typical working style is to set up screens and agree with customers, then evolve data model behind it. They find they can live without a model-view separation. What would that buy them except a doubling of the number of business concept objects from 600 (their current state) to 1200.

Properties can be assigned styles and these styles can be filtered.

(At this point he had the usual demo hiccough; he had stripped an existing system for demo purposes and needed something he had removed. He fixed in the debugger.)

He showed various slot types (checkboxes and so on) and the connection to low level process decisions. Collections are handled: a slot can be shown once or many times. He then showed code generation by putting a break in the debugger. Resynching calls addMethodPlansTo: aClassPlan which calls addMethodPlanTo: and so on. A hierarchy of builders handles generation of getters and setters, choices, collections, etc.

Q(Stephane) should we have a slot object, not just data? (There is a VW package that does slots.)

Fame: MetaModelling at Runtime, Adrian Kuhn, Univ of Berne
You can download the metamodelling framework FAME from smallwiki.unibe.ch/fame. Fame grew out of Moose. His talk example was to model an eternal beer store, i.e. one that need never be shut down to update it. Fame lets you update your metamodel at runtime. As your business changes, your metamodel must change. Today, the store sells beer. As it grows, it may start to sell speakers or cars or ... But at start, it sells beer. We model beers as objects, instances of a class Beer with slots name, alcoholic volume, price and size slots. This in turn is an instance of Fame-Class with slot properties. (He will call such metaclasses Fame-Class to distinguish them from Smalltalk host classes.) Fame-Classes are metaclasses, instances of metametaclasses. Our business is modelled by a model which conforms to the Fame metamodel which conforms to the Fame metametamodel. The metametamodel is hardwired into the host Smalltalk system and the same for all metamodels and models.

He showed their file format, which is a Smalltalk literal with the initial # character removed. (They started by using the Smalltalk parser and now use a dedicated parser so noone can inject executable code.)

Q. The alcohol content of the beer value increased in the last slide; is that a metamodel feature? No, it is because Adrian was drinking beer while doing the slides. :-)

Fame classes have corresponding Smalltalk classes. Pragma processing sets up the Fame classes from the Smalltalk classes. Refactoring modifies host classes when Fame class changes requires it. In the metamodel, a hardwired bootstrapping generates the Fame metamodel as instances of host Smalltalk classes.

Time to demo. He opened a Squeak image with the latest Fame version loaded.

```
t := FMTower new.
```

Exploring the tower opens a browser, with tree of model elements. Now we have a tower with only the topmost layer populated. He created a new tower to import the Beer model (Squeak-implementation is partially ported; in VW, you would reuse the tower you just created but a few Squeak things are to port; on the train coming here he was finishing this, fixing bugs, etc.; it is just done).

```
tower := FMTower new.
tower metamodel importString: ....
```

He browsed (and noticed another bug to fix: the annotation string replacement had mistaken pragma brackets for replacements; it works for all the other generated methods). He opened a Squeak browser on the HNK (HeiNeKen beer model) package.

Q. Methods have type pragmas. What happens when I add a string to a slot with type Number? Nothing at first since they do not generate any validation checks on setters. If you then verified the model, the wrong instance would report a mismatch with its Fame model.

FMManySlot handles two-way update of slot and back-reference-slot, thus modelling associations. He browsed the model data, adjusting alcoholic volume and price to please the audience.

Q. Change Smalltalk code: automatic update? Not in Squeak. Adrian has a VW version that does this.

Q. Map existing Smalltalk classes into Fame how? Set a pragma on the class side and on accessors that you wish to become slots. It can be easier to generate a skeleton, and read it in again.

Q. EMOF? They implemented it in Moose 3 years ago but found they never use it. EMOF has a metametamodel of ~ 30 classes instead of Fame's 4.

Fame can generate UIs, including for metamodels it did not know about at starting, since all ultimately works from the metametamodel.

Q.Extend at runtime; what happens to existing data? They do not handle migration of instances yet, You just get new slots being nil and so on. He has not yet had a client of Fame that requires more.

Using the Meta-Environment for Model-Driven Engineering, Tijs van der Storm

Tim works at CWI and also teaches at the University of Amsterdam. His meta-environment is a programming environment for languages, just as Squeak is a programming environment for Smalltalk. Algebraic Specification Formalism and Syntax Definition Formalism are two parts.

He explained his environment by making analogies with Smalltalk. In Smalltalk, everything is an object. In His ASF +SDF Meta-Environment (the MetaEnvironment) everything is a term, their name for an item of parsed source code. Terms are parse trees: Abstract Syntax Trees plus layout (i.e. comments, whitespace, etc.). He showed the (large :-) parse tree of

```
foo
  ^self
```

SDF is similar to EBNF (but its productions are reversed - historical reasons for this). It does GLR parsing of arbitrary context-free grammars. It offers disambiguation constructs since 'arbitrary context-free grammars' includes ambiguous grammars.

All Smalltalk computation occurs through messages. In the MetaEnvironment, all computation is transformation of parse trees by applying rewrite rules. The rules match ASF expressions and construct transformations of them. This is purely functional computing.

The language is the environment in Smalltalk. In the MetaEnvironment, languages are environment contracts. The environment knows about some specific languages such as one for errors, one for formatting, one for formatting. To add another, you must define it in SDF and define a GUI plugin that consumes it.

As Adrian explained in his talk, systems conform to models which conform to a metametamodel which also conforms to it(self).

He demoed a trivial language for markup called 'Waebric', writing expressions such as

```
module Hello

def main
  layout("Hello"){
    h1 "Hello World!";
    p "Home";
  }
end

def layout(title){
    head title title;
    body yield;
  }
end
```

and others for recursive menus and so on. He opened the tool set and browsed the graph of modules used to construct Waebric. An error pane lists issues with the grammar. He showed the parse tree of the code above, which was large and grew larger when he added a comment /* bla */ to the text. He ran and showed the HTML produced. The errors showed a non-XHTML 1.0 tag and the tool took him to its cause in the tree(main layout).

He showed the error syntax grammar. There is also a Java grammar, done by another group who use the SDF part of the tool.

Q(Reinout) Changes in last three years (when he used it and found it hard to use)? It is now more stable

Q(Niall) Comparison to OMeta: OMeta only parses a restricted class of grammars, thus avoiding ambiguity but forcing what Tijs feels is an awkward way of describing the grammar. This restriction also blocks applying OMeta to legacy cases that do not fit it. OMeta allows parsing actions in a dynamic language: the MetaEnvironment, as noted, restricts strictly to rewriting. The two are therefore aimed at their different domains.

Modelling and Mapping Tools, ObjectStudio, Dirk Verleysen, Cincom
(I missed Dirk's talk at ESUG; the following is an extract from a demo he gave me four months earlier.)

The ObjectStudio modelling tool has 3 explorers: design, use case and CRC.

Use Case: (re)name model then create Actors for your new model. An Actor can do use cases, e.g. an Administrator can enter new employees. A use case is just text saying what it does; it has no formalism greater than formatting this text. You can select items in the text and identify them to the tool as domain objects in the application. Other text could be made into a use case association e.g. 'check zip code' is another use case referenced in this one.

CRC explorer. You give objects responsibilities and collaborators, identifying more domain objects in the process. For example, a Company collaborates with Person in adding new employees.

Design Tool: this is the principle part of the tool. Domain objects appear as boxes and are defined: attributes / instance variables (an attribute can have various settings) and relationships: inheritance, aggregation and association. Detailed tabs let you specify an association's overall data and its LeftDetail and RightDetail (cardinality, traceability - can my far end walk back to me or not, etc.). A description tab lets you document the relationship's purpose.

Synchronise creates classes mapping these descriptions. If you are using the tool in OS7, you generate .cls files. From a list you select which kinds of methods are generated (accessors, initializers, releasers, events, drag-drop methods - called template*). The design browser then shows these classes (and methods but here there was the usual demo hiccough or it might be the tool needs resynch of model after generation, but that seems inappropriate). The design tool's diagram properties can be set to e.g. not show methods, and it can switch between various OO diagram notations. You can also edit the objects to highlight certain things, etc. Printing to .rtf gives you a basis for documentation.

Q.Georg (discussions) Old code, pre-exception-handling in OS, leads to code like `... onError: [^error].`

Changes to a method, whether made in debugger or class browser, now get reflected back in the model immediately you save the method. This works in some OS7 configs and is ported to OS8 along with the rest of this tool.

Dirk then built a UI for the model he had constructed. This was a straightforward putting of boxes on canvases and connecting to model elements.

Q(Georg) when Dirk was an OS consultant, what was the tool used for? They did the entire system for a customer in the tool and always kept it in memory so they could always print out the current state. These diagrams were used in discussion of what work was to be done. Eduard found it useful at project starting and after a six-month break in development.

ADvance shows two lines for binary relationship which makes large diagrams much harder to read. It is a much smaller thing than the OS tool with no synchronisation. Its sole virtue is that it is very lightweight and

unobtrusive; you can have a few diagrams if you want them and ignore it otherwise. ADvance is an aid to documentation and to brief design activities within an XP approach. A project that wants a full modelling solution will use the ObjectStudio tool, not ADvance.

Q(Georg) Porting to OS8; what was that like? The hard part was mainly testing. It is all UI so you have to create and assess them. Dirk also uncovered some bugs that were in 7 as well as 8.

Meta-Modelling Panel, Lukas Renggli, Mathieu van Echtelt, Adrian Kuhn, Tijs van der Storm

Reinout asked the panellists to summarise their systems in terms of round trip engineering. (One context for this question is fitting projects to legacy engineering.) Tijs has effectively no support for it and doubts the value of modifying generated code. Lenses are a way to formalise reversible transformations but they are harder to use and restrictive. Mathieu's system lets you change code and they do but the description will not change. Magritte is just objects in the image. (Niall: but you could map the ToDo field of the last talk back into an instvar and serialize its Magritte objects into Smalltalk code. Glorp ActiveRecord uses a similar approach to read database schema into mapping objects and then serialise them into code.)

I asked about collection classes (see my ESUG 2005 Design Discussions talk). Mathieu reimplements (some of) them, which is the standard but imperfect solution for this problem. FAME assume all collections are OrderedSets, so has a single subclass, the convenient solution (the clients have not complained yet): slots are a subclass of OrderedCollection that override `add:` and `remove:`. Magritte has a OneToMany description and uses the standard collection classes. Lukas has noted that complex relations are hard to handle, prompting much handcoding. This may be related to not having meta-enabled collection classes.

Christian played devil's advocate: it's better to write specific Smalltalk code; defend your use of meta systems. Mathieu would be two years of effort behind if he had had to write HTML and SQL instead of generating all that stuff. Yes his users must forgo hand-crafted interfaces but they are accepting this, perhaps because Mathieu's competitors are even worse or because they get something that works in a time they like at a cost they can pay. Very rapid changes are also possible because they change in one place and then the web UI and the database conform. Ten handcrafted pages are OK, 100 are not, 500 are so very much not. Magritte is not closed: handcraft all the Seaside pages you want and use Magritte for the rest.

Is Smalltalk special when it comes to implementing self-described system? Adrian has ported FAME to Python and Java. In Java, it cannot connect to classes, OK if you are a pure meta-model app, a problem if you want to mix meta and ordinary code. Magritte uses class extensions so would be hard to port to a language without them.

Show us your projects: ten-minutes presentations

A late afternoon session on Tuesday allowed a range of projects to make

10 minute presentations. Those not included in other sections are here.

Pharo: help people to invent their future, Stephane Ducasse
See www.pharo-project.org. They will rework the Squeak build to provide a clean and lean open-source Smalltalk, removing junk, sorting out dependencies, eliminating uninitializable old code and so on.

Simple Web Apps with HttpView2, Giovanni Corregida
This web app framework was built by Goran Krampe for the SqueakMap server in 2003. Noone else is using it.

```
HVHelloWorld>>default
  ^'HelloWorld'
```

and there you have a website. A small website is quicker to build in HV2 than in Seaside. It uses old-style Seaside-like methods `h1:`, etc. (now Seaside has moved to Canvas, Giovanni will move HV2 to canvas too).

It is HTTP-compliant: get, post, put, delete. Giovanni wants to evolve it to a RESTful web-services application (no point competing with Seaside).

See squeaksource.com/HttpView2.

Kerala, Alexandre Bergel
He wants to check architecture conformance, coverage, etc. He showed a rectangle-merging test and broke it down into various calls on rectangle, making up a history of calls on the tested rectangle. Thence he looked at a graph of unique class name references in his tests execution. He showed these graphs for several tests. He opened a Pier image and made the subclasses of a test known to Kerala. Then he ran tests and opened a graph of moments in the test, picking a colour map; each moment was shown in a column whose squares were coloured. “Please don’t ask me what these are good for; I don’t know.” :-)

Seaside XUL, Michael Davies
He opened Squeak and showed the usual appearance. They have written an application to give a client a more standard look and feel and have open-sourced this theme. The controls looked more Mac-like. Then he showed some other themes. You can tab between fields, see better where the current focus is, scroll via the mouse wheel without disturbing your focus which is also remembered when you switch between windows.

He showed a report-building application used for their clients, with drop-down lists, checkboxes and menus, all the familiar controls. He showed using it to edit a labelling file, with its barcode, routing text and so on, setting border colours and all the usual stuff. It is all done in morphic using various subclasses of ImageMorph to support translucency.

SeaBreeze, Karsten Kusche, Georg Heeg
SeaBreeze is the UI builder for Seaside. He opened a Seaside website. He clicked the toolbar editor link to open the left-hand-side widgets (also written in seaBreeze so you can edit the layout of the tool in itself). Select

a Seaside element, set its basic attributes, its CSS, its layout and its drag-drop behaviour. You can click to open a refactoring browser on a callback method. He showed making an element redraw without redrawing the page, making another element toggle, and using simple one-line method calls to get AJAX effects. It also shows a palette of elements you can add.

Q. On Squeak? Not yet, just VW.

Q. Cost? For non-commercial it will definitely be free. Commercial is TBD.

StakePoint
Stakepoint is a general purpose planning program using Glorp and Cairo in VW. The UI shows resources on the left, timelines on the right. Linear equations are solved in the image and a revised plan written in a single transaction to the database. You can drag-drop to add documents, emails, etc., directly to the tasks.

Q. Can you import MSDocument data? He can feed a task into an MSProject database.

Q. Why use this tool instead of MSProject? Many people can work together on the same document, thanks to the transactional behaviour.

Q(Bruce) Capture actual data from contractors actually doing the work? Not yet protected with capability-based login and etc., for that but yes it is multi-user so the contractor could be given access.

Demo can be downloaded from the ESUG website.

Nikolay
A new Squeak and Seaside-based disk has been released in Russia. One disk with Squeak, Seaside and content for a webserver is given to e.g. a school, installed on many computers and then updated over the internet.

Other Discussions

The UK smalltalk user group has a new website uksmalltalk.org, set up by Bruce Badger and Giovanni Corregida.

Georg announced that SeaBreeze will be available on the MIT licence. Heeg will also sell a supported commercial licence. They will publish SeaBreeze to the public store “as soon as the comments are written.”

Leandro announced the second Smalltalk conference in Argentina. It is free to attend! The coffee and cakes are also free! Last year they had 200 attendees. There will be three tracks, one for education, one for interest and one for research. Andrés Valloud will be running his mentoring course.

I talked to a GLASS user, Otto Behrens of FinWorks, who deal with retirement wealth management / insurance. Their system supports monthly investments into voluntary pension products, from employees whose

employers offer the schemes to the brokers who invest the monies. All parties get a central point in their system to save them from forever having to reconcile their data. The system displays info regarding unit trusts: classifications, trading currencies, fund domicile, etc. The FirstRand group has 3 entities: MomentumLife (80Gb Gemstone64 database), RMB International (does a similar business to FinWorks but working backward from employees not onward to monthly payments) and Rand Merchant Bank Treasury, whose system is similar to Kapital but is specifically for bond valuations. They use VisualWorks on Sybase. RMB International also run Seaside in VisualWorks for asset statements for RMB customers with large cash positions. FinWorks came out of RMB in the sense that Otto and others worked for RMB previously and FinWorks does work for them.

FinWorks have PDF generation from Seaside: they build the Seaside html page, then add special tags like page break, page number, landscape or portrait, then run a commercial tool (www.princexml.com, $3000 licence) that produces PDF from the page. They also generate pre-populated PDF forms this way which a user can fill in the rest of and have a customer sign. They are converting to Magritte: a pain here and there, but generally working and helping.

ESUG has prepared a PDF of a fold-out flyer describing Smalltalk. See it at daniel.cassou.free.nl and/or get copies from ESUG for conferences.

The perils of being the local organiser: Adriaan had to pay when Noury arrived with a large party of guests at a restaurant - and then discovered he had forgotten his wallet. :-)

CosmoCows are hiring. Good designers wanted, knowing Smalltalk or willing to learn it.

VASmalltalk Event, Frankfurt, September 23rd, 2008

Travel to Frankfurt was painless, despite my change being at the notorious Heathrow terminal 5. My hotel room could have slept 5 with ease - except that none of them could have slept while the overnight trains passed with incredible noise on the tracks outside. The forum's coffee and cakes were on a generous scale.

Summary of Projects and Talks

Joachim introduced himself and ObjectFabrik. ObjectFabrik is preparing Smalltalk training material and courses.

Instantiations Market Perspectives, Nicholas Gilman, Instantiations
Instantiations teamed up with Joachim two years ago and have found it a very effective working relationship. John came to Instantiations a year and a half ago after 40 years with IBM. (I told John he did not look old enough to have 40 years with IBM; he told me it was six months short of that. :-)

Instantiations is committed to Smalltalk. VASmalltalk is an important part of their business. They have sold new licences into all major country

markets in Europe this year and they expect to be using Smalltalk for many many years. Nick is responsible for the world outside North America but the overwhelming majority of his time is focused on Europe.

Q.(Christian) Size of market in Europe v. North America? The market is smaller but not significantly smaller. The US has some mega-installations (100 - 200+ seats) whereas Europe does not have that scale, but it is still a significant proportion. (Louis asked who they were; they are in the insurance market. Nick might be able to provide Louis with specific details for internal use only.)

If you want to influence the product, be the squeaky wheel (like NSF :-)). Louis mentioned discussions in the VA forum. Niall described how easy (for NSF) was being interviewed (by John O'Keefe) leading to an article that was placed by Marta; if it was as easy for Instantiations as it was for us then other users should talk to us today about it and maybe do the same.

Q. Young Smalltalk employees can be hard to find. Potsdam and Paderborn universities teach OO in Smalltalk but they use Squeak or VW. The questioner knew of a major real-time user of Smalltalk who went away from it because their Smalltalk team was aging and not renewing itself.

VASmalltalk and today's trends in IT, Joachim Tuchel, ObjectFabrik
IT today is complex, the more so because of the history a typical IT department will have. Old systems and databases have complex interactions. The idea of having only one database, only one OS and so on is often raised. In response to this idea, some IT shops have eliminated Smalltalk but elsewhere Smalltalk has survived, often because replacement projects have failed (after spending much money) for various reasons, some of them technical. Such experience teaches (some!) managers that business value is more important than having the 'right' (fashionable) technology.

After being neglected for several years, Smalltalk shops now find themselves with a lower budget than five years ago but told to continue providing, and usually also improving, business value. Integrating with other technologies is also a frequent requirement.

Presentation is another area of current interest. Today, there is a mix of sometimes going onto the web ("lets concentrate everything in the browser") and sometimes going back from it to rich clients again. IT shops want to avoid installing things on the client. Companies want a strong company-branded presence through a single portal. Global availability and platform neutrality are wanted.

Why do others not want to go to the web? The web limits interactions, can make complex tasks too challenging, raises the problem of how / where to save local data, etc. Rich clients give you platform-drag&drop, faster feedback and a generally snappier feel to the app.

Of course, a rich internet application is a possible best-of-both-worlds. You

can have active elements (e.g. sliders, drag&drop) without page reload. They combine client technologies (AJAX, Javascript, ActionScript, Adobe) and server technologies (Smalltalk, Java, Ruby). AJAX is an XML HTTP request that uses Java script to update the DOM of a page on the fly.

VASmalltalk offers various web interfaces. VAWebConnection is a legacy product (it was sort of a prototype for Java Server Faces). Server Smalltalk is a stronger and newer product. It is stable (ten years of use), scalable (multi-threaded, multiple images) and combines an HTTP/S server with Java-servlet-compliance on the client. VASmalltalk gives XML support, SAX and DOM-Parser and Smalltalk-XML mapping.

Q. Native multi-threading on Linux? Not supported on any platform for Smalltalk threads. Calls out from the VM, e.g. to a database, are OS-threaded but not in-image processing. A single image running multi-threading will significantly increase the complexity, so you are trading memory against complexity.

Q.(Christian) Air, XOOL, Silverlight? Seaside has projects to offer but nothing else is in the works. It is all XML so one can do it at that level.

Rich client platforms: Eclipse has a huge set of tools to let you drive SWT and suchlike GUI frameworks to build rich clients. VASmalltalk has been used to build rich clients for 10+ years. There are many parts and powerful builders (composition editor, WindowBuilder Pro). The result is a native application. However the default look and feel is now a bit old and OS2ish. We need new parts: pluggable toolbars (with tear-off), ribbons, etc. These abilities are there but are not as usable.

Smalltalk web start is Joachim's preferred 'most needed'. By 'web start', he means we should be able to have an image check on start-up for updates from a server and get the new image or the new code to load. Joachim would also like to see Windows CE/mobile and Mac OSX added to the platforms' list, but that is not an announcement!

Web services use HTTP POST for transport and XML messages for content in SOAP envelopes. Web services can easily become very complex. VASmalltalk's web services were provided in VA5.5 and have been improved in every version. You expose a Smalltalk object as a service.

RESTful web services are not the same as Web Services! REpresentational State Transfer is about creating a resource, not an operation, on a remote server. The URL is its unique name. It is (supposed to be - not every implementation actually is) totally stateless on the server side; cookies and all are managed by the client. A resource can be like a business object, e.g. a customer, a message, a flight booking, and, in more esoteric senses, a database transaction or a dataset. The mapping between your business objects and your resources may be direct or may group them into single XML messages (e.g. the customer and their address) or be more complex still. The basic create, read, update and delete operations are all

implemented by server smalltalk (it implements the whole standard). These operations have a usefully detailed set of result codes.

Why do we care? Web Services are quite complicated and RESTful web services are quite simple. RESTful also has some useful advanced features: if-modified-since, last-modified. Cache control is supported: have read-only objects or manage infrequently-changing objects. You can use a cursor-like accept-range: get the first 100 objects, then get the next 100, etc.

Usage is growing. The storage mechanism of Lotus notes has put a RESTful web service in front of it.

Joachim showed the code for his PRESTON client:

```
getResourceNamed: aURI {queryParameters: aDictionary}
postResource: anObject toResourceNamed: aURI
...
```

PRESTON provides proxies for hyperlinks and other useful features. He demoed, opening a browser on the Yahoo traffic service (alas, Frankfurt is not one of the supported cities) to explain it and then calling it from code. He used the normal `xmlMappingSpec:` shipped with VASmalltalk to map between his business object and the XML in the `queryParameters`. He brought up a simple UI on it showing the traffic situation in New York and Jacksonville (quite light at 05:50 their time). He set a breakpoint and showed the request going out with its query parameters, and the mapping spec. (A VASmalltalk goodie converts .xsd to a mapping spec. Joachim downloaded the .xsd from Yahoo - it's part of the overall message - copied and pasted it into the goodie and got his mapping spec.). He then used the Firefox plugin to show the structure of the message: a collection of results each of which is a traffic message ('Holland tunnel is blocked due to construction work' and suchlike).

Q. Why would a transaction be a resource? Suppose the customer is thinking of buying a flight? They may go on to buy, or search for another time, or stop and abandon the purchase. This is a business case transaction. It is worth knowing how to handle such things as resources.

He walked on through the browser and (made the screen bigger and :-)) opened the response object, looking at its header (an SstHttpResponseHeader). The status code of the response which came back is checked, so if it is not 200 OK it will throw an exception; thus you can have rich client behaviour around this. He walked the object in the 'dom' instVar, which was just an XML DOM as in any other language.

Thus all this is simple in VASmalltalk. The basic VASmalltalk is very robust and simple. PRESTON provides an API that looks like a standard RESTful API.

Q. (Louis) the XML mapping generation goodie will be in the next version of the product? (John O'Keefe) yes, and we will be doing more with this. (Joachim) downloadable (in two versions) from Instantiations' web site.

Joachim asked who uses Web Services. Three groups raised their hands for doing it in VASmalltalk and Christian was also doing Web Services in another dialect.

SST offers you a session manager which Joachim thinks useful even if RESTful purists think it should not be needed in a RESTful world. Using this and the other SST building blocks (e.g. naming service) Joachim built the PRESTON client. He then demoed another example of using it, called Todomatic, (a ToDo list manager). Serving a resource is serving an XML file and that is what a servlet does so the API is very similar; he inspected the PrestonServer and showed that its `applicationContext` object was just the same as for a servlet. He showed posting a Smalltalk object (a new user) to a resource (the list of users).

```
postResource: aUser toResourceNamed: usersURI
```

A put request updates a specific existing object. A post request adds a new resource to a list of resources and to access it later you must know its name, which will be returned by the response.

He then stopped the server and restarted it to show making the user list only support posting to it, as an example of controlling what operations were permitted. He showed how he built up the structure of resource managers. The system has a database connection where the todo data is stored and a session manager (useful if not strictly necessary). After he made the change, clients could no longer get the list of users though the server could still see them.

Q. A viable alternative to RMI? Yes. He mentioned a product (a database) that only offers a RESTful interface. You could use this to make it easier to interface with e.g. a Java app without needing an RMI library, avoiding RMI version issues.

Q(Niall) Can we browse this code to give us examples. Joachim is thinking about it; yes, he would like to make some of the code available.

Q(Christian) Collections? At ESUG in 2002, Alison Dixon gave a talk that noted the rules then needed (e.g. using Arrays not OrderedCollections). An issue is that a result needs a rootTag and if you just return a list you may not have it. This is a general issue, not specific to Smalltalk, so most public service providers have probably solved it, e.g. wrapping their returned list in a resultList-tagged object.

Most of these high-faluting web technologies we hear about today are just based on HTTP with XML and mapping these to objects on the server side makes them much easier to use. VASmalltalk is ready to handle all of them.

Portrait of an Agency System, Martin Elässer and Steffan Müller, Versicherungskammer Bayern

The talk was in German but the slides were in English. Their organisation, Versicherungskammer Bayern, is a company of Sparkassen Finanzgruppe. Their business is health insurance, life insurance and composite products

for individuals, institutions, etc. They handle 2.8 million claims a year (1500 per working hour). They pay out 17 million euros per day in insurance payments. They have many sales channels: brokers, agencies, savings banks and direct sales over the internet.

The are big in Germany and also have a branches in Hungary, the UK and elsewhere. They started in 1811 and have a long history of growth, mergers, etc. Currently they have just under 7000 employees. and have a recent history of steady growth, making them the prime insurers in Germany.

Martin then handed over to Steffan. Steffan uses Smalltalk to support all of this. Their system must serve customers, sell products, support acquisitions and provide a range of services to the sales and marketing departments.

The customer service function has no selling purpose. It must be a word processor, a scheduler and many many other things. The sales process needs these functions too and to manage the claims history etc. When the grow by acquiring the products of the new partner must be data-filled into the system and/or their existing systems liaised to by the system. Last but not least, sales and marketing need a help system to explain the data they are seeing, etc., etc.

The system was begun with some 50 employees of VkB, IBM, Inverso and external companies. It started in 1997 and went live in 1999 using VAST 4.0 and DB2, Lotus Notes as the CRM system, with C, Cobol and Java for some calculation modules. By 2008 they were on VASmalltalk 7.5 and had moved the calculation engine to an offline Java application to which they communicated via web services. Another Java app provided a web front-end. There were 4000 clients and over 1000 local servers.

The flow of data is complex (see slide). Customer and contract data is held in distinct back-office systems. Insurance contracts are automatically generated, logged in the DB and sent. He launched the Smalltalk application and walked round its various functions.

He also showed the Java app in a web browser (IE with a browser plugin so it looks like a native window) and walked round a standard submission in the browser, showing in fat client its effect on the Smalltalk side. The Java agent uses Java Server Pages and Java Beans in a Tomcat web server. It talks via web services to the VAST server. It has the usual behaviour: request a contract with bad data, see some fields in red appear in your browser, fix and carry on.

Customer relationship management is becoming more important and that is where Smalltalk effort will be put, so the Smalltalk UI is likely to remain and grow. The users like the powerful Smalltalk forms and Steffan certainly hopes they will remain. As for the web client, perhaps they will replace it with Seaside in a few years; they have hopes. :-)

VASmalltalk 8.0 and beyond, John O'Keefe, Instantiations
(This write-up includes material from, and questions asked at, Smalltalk Solutions 2008 and ESUG 2008.) John O'Keefe has a long history with Smalltalk. He first saw (Digitalk) Smalltalk in 1987 and was a founding member of the Smalltalk team at IBM. He was very glad to forge a relationship with Instantiations when IBM retired from Smalltalk a year and a half ago. He leads the development team at Raleigh, North Carolina. Instantiations' co-founders developed the first version of Smalltalk in 1984 at Tektronix. Instantiations has been born, bought, sold, born again and has always had a major Smalltalk focus.

For the last two years, Instantiations have maintained and sold VASmalltalk which was formerly VisualAge Smalltalk at IBM. They released VASmalltalk 7.5.0, 7.5.1 and 7.5.2 during that time, focused on tool integration and improvements to the product. Now in VASmalltalk 8.0 they are adding some key things and also taking some things away. Seaside and its friends appear. Browsers are enhanced. Web services are enhanced. After a long time in which the inherited documentation had fallen behind the product, they are now investing in new documentation and improving the documentation system. They have also made some small changes.

The 8.0 work had four main requirements from customers. Customers want Seaside. They say the browsers look tired. They want ANSI compatibility in full, which means ANSI exceptions (all else is already there) and internationalisation. Lastly, they want performance, which 8.0 has improved under the covers.

Last year in Lugano, John said they were looking at Seaside. Since then, they have ported both 2.8.2 and 2.9 but have now quiesced 2.8 in favour of 2.9 (John is grateful for the Seaside project's refactorings that make porting easier in 2.9.) Scriptaculous has also been ported. A Seaside porting layer has been developed and will be used to provide functionality in three ways:

- Parts of it will move into the base. They are generally useful.
- Other parts will move into a common porting layer shared with the RB and SUnit. They are useful whenever some Smalltalk utility is worth porting into VASmalltalk.
- Other parts will remain in a Seaside porting layer.

Q(Christian) Sport? John has worked on sport and talked to Bruce Badger. It was not aiming at exactly the same purpose so was not suited to their use.

Q(Joachim) use this (the third layer especially) to port other Squeak tools? It has only what Seaside and Scriptaculous needs, so would surely need additions, but could certainly be used as a base for such ports and John is thinking about doing so.

Continuations were (and are) the thorn in their side. One-shot continuations (i.e. simple Seaside call-answer protocol) are working today and beta code will be released as soon as these are robust. They also have the web inspector and debugger working and are working on the class browser. Full continuations need VM changes and these are under way.

Alas, VASmalltalk's process model is quite different to those of other vendors' Smalltalks and this is what has delayed full Seaside. Seaside 2.9-jf.183 is running. Seaside-Tests-Unit 2.9-pmm.156 run 95% green.

Q(Eliot) Architecture? The architecture will not be changed. The main need is to expose method contexts which at the moment can be read but not written, not through read only mechanisms but actually blocked in VM. Process copying is also blocked and must be enabled.

Toolbars and Halos and the Inspectors work. The browser is being developed; he showed a screenshot from his test system.

Browser look and feel: John showed the old, then the new with tabs. These tabs are native, not VA ones (and so they must accept platform dependencies). You will be able to choose standard browsers (1980 look), the VAAssist browsers (colour coded and features) and, based on the VAAssist browsers, these new ones.

These tabs are on the method pane but that will be replaced by a tabular form. You can switch between tabs without having to save the content: e.g. change a class method definition and a method that will use the changed definition at the same time. They want to make them dynamic (changing tab colour to show 'changed but not saved', 'content missing', etc.) using tab icon and colour. They want to make the method pane a sortable list, sorting on public/private or on method name or whatever. They will also offer a tabbed workspace; John showed it.

Seaside made John want to look at byte codes so a byte code browser will be available in 8.0. John showed it. (He will make it not *too* obvious how to turn it on lest geeks spend all their time in it. :-)

Web services are of interest to large enterprise customers (and few others?). They have customers with 10,000-50,000 bytes of WSDL in each of many files, containing multiple nested schemas. A new style of WSDL called doc literal wrapped has come in during the last two years. They will support the wrapped doc literal style in 8.0. Their insurance example will be greatly enhanced; it was supposed to show everything you would want to do with web services but did not and/or no longer does. They will offer documentation to show patterns for using them: there will be examples and a cookbook, giving step-by-step instructions. This will also explain debugging techniques; it is easy enough to do once you know how but there was no explanation of how. They will guide how to manage deployment descriptors, explaining how to set their configuration parameters and where they are stored.

The old web-presented VA documentation was ugly and the source for it has been unavailable for several years (long story). This is partly why the documentation has not changed since 6.0. They will completely revamp the documentation. The documentation server will go; they will instead use WebWorks. Search and all other features will work both locally and through the web, not just locally as it does at present. CSS will be used for

formatting. New PDFs will be written. John showed the appearance of a page in the prior system and in the new WebWorks system.

Q(Christian) You can select and execute code examples in this browser? No, you must copy and paste to a workspace. You can tell what changed? John is keen to have a when-last-updated field at the foot of the page. It is hard to show change bars in a document. There will be a what's-changed frontpiece but that will be a highlights summary, not a complete list.

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

Q(James Foster) do you have a conflict between Error class and global Error? No, Error class has been there for years. Discussion clarified the different thing James was thinking of. Seaside drove this; it makes extensive use of class-based exceptions.

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

Windows Themes are supported on XP (available as a patch now). There is full support for UTF-8 locales. Most Linux platforms now use UTF-8 is their default out-of-the-box so it was a problem even to install without it.

They will complete their support for UTF-8 locales.

OS/2 will *not* be supported in VASmalltalk 8. Instantiations has never formally supported OS/2 but till now has kept it running there. However 7.5.3 will be the last such version. John guarantees 8 will not run on OS/2.

Q(Louis) 7.5.3? That is the version with Windows Theme support. It was never actually released (they decided instead to focus on 8.0) but if you need Windows Theme support (or OS/2), email him and you'll get it.

A beta is *planned* for October 2008; John guarantees it will be the last day in October, or maybe October 35th. Release will be announced on the Instantiations' website. The *planned* date for general availability is three months after the beta which should be 4Q2008, but it might be December 35th. (John stressed that plans can change and do not usually change to be earlier.)

After 8.0, they will look at Seaside, Web Services, IDE Enhancements, Installation and other things (looks like the 8.0 list).

Seaside will be continued; if 2.9 is not code-complete when V8 ships, it will have a a backported 2.8 but they will certainly ship a 2.9 as soon as possible. Porting Magritte, Pier and RSS needs to be done and they may not

have all the resources to do it themselves. (In the past, people have remarked that Instantiations do not have a code repository. John is thinking of a Monticello-2 bridge so he can publish changes between an internal Envy repository and the outside world.) They will help people port Seaside add-ons. They themselves will keep Seaside and Scriptaculous up-to-date.

Web Service tools will be improved. An XML editor, better Smalltalk class-XML translation, etc.

Q(Vincent) will these changes apply to e.g TrailBlazer? (Question was asked earlier but deferred to now.) VASmalltalk has a plethora of browser: the base browsers, TrailBlazer, the Refactoring Browser, and VAAssist that sits on top of all of this. They will integrate all useful functions to a single browser (leaving the others there to use but putting their useful functions in the main branch browser). Tabs will let them steal useful features from other browsers, e.g. the version graphical relationships tool from TrailBlazer will become a tab. The other browsers will be deprecated, or at least quiesced: TrailBlazer will no longer be loaded automatically when you load the server workbench.

Consolidating the IDE branches was in last year's list but Seaside took their time plus they feel the changes they are doing in the browser will be a better base for the consolidation.

They ship tons of examples that people never find. They have the example launcher to expose a few of them; all the others will be exposed in it too.

Installation needs to be improved. This failed to make it into 8.0 so will be done in 8.x. Their installer works well on Windows, not quite so well on Vista and needs a little manual intervention on UNIX / Linux. They have a Smalltalk installer which is good and bad - they understand it but they have to maintain it. They must decide whether to evolve that or to go instead to using a commercial installer.

They will probably use Glorp as an OR mapping layer; their own framework ObjectExtender is so disused it is now a goodie.

They support many web APIs, some very old and creaky. They have deprecated Netscape server API and the IBM communication server API. That only leaves IIS or CGI, and CGI is slow, often because it is not persistent, so they will use FastCGI. TCP/IP v6 is not much demanded at the moment but they aim to be ready when it is (US government installations are starting to require it). Some Windows CommonControls need to be added (DatePicker for example). Windows services should not need a special start-up executable. (John confessed it was he who wrote it that way; he now sees they should be written in Smalltalk.) It needs two weeks of effort to do and will make Windows services more flexible to use and to debug, and easier to run as an app or a service.

Q(Adriaan) How do you port from Squeak to VA? John has developed a custom file-out package exporter. He is happy to make it available.

Q. Louis has spent time looking at deploying multiple images on one server. The load-balancer goodie on Instantiation's website is not industrial strength. Louis figured out how to do it with Apache but would have much liked documentation; will you add? Good idea; yes, we will.

Q. Experience migrating to 8.0? None as yet since noone has but now they have a new documentation mechanism they will update the migration guide. John is going through all the changes and noting what might be a problem. In fact Adriaan did some experiments porting to 8.0 and then to Seaside. Significant extensions have occurred and a class, IdentitySet, so where these are already added by an applications own base extensions these must be loaded. Niall suggested advising people on Envy behaviour when loading clashing code. Adriaan mentioned he had found it easier to get everything loaded by loading his application first then Seaside.

Q. AbtTimestamp change? The previous change had a severe performance impact so they rewrote it again.

Q. Seaside support needs VM tests? 95% of Seaside tests run with no VM changes but the last little bit, which is important - continuations and utf-8 - do require VM changes.

Q. If in future there is cog, would VA run on it? Eliot and John discussed that a few years ago. The VA VM is much larger than other dialects' VMs. There is a good deal of primitive function exposed that is not in other VMs. Eliot: "It is a small part of the problem but I hope I will provide enough function for you to consider it."

Q. IBM Eclipse work? John is pleased it happens. It reproduces some of the Black Knight project which never was released for various technical, legal and political reasons. John approves supporting Smalltalk availability in that environment. Would he personally like having a checkin/checkout environment in Smalltalk? He does not know. However Eric and John have agreed that Instantiations' website will have a link to the download for it.

Q. MQ and Tivoli? MQ series support remains. We've had requests for JMS support that we think we could do via MQ; that is the only MQ work envisioned in the immediate future.

Q Java Servlet interface? Yes, that remains. (John forgot to mention it in the list.)

John offered additional demos of Seaside during the breaks.

Seaside, Joachim Tuchel, ObjectFabrik
This talk did not aim to be a Seaside tutorial. After a short introduction to Seaside, Joachim demonstrated various aspects of using Seaside, especially the similarities to and differences from fat client. Moving from fat client to Seaside is an adventure but easier than any other technology for webifying fat clients that he knows.

Seaside was created by Avi Bryant and is now maintained by Lukas Renggli and Adrian Lienhardt and many others. He listed some Seaside sites (see slide) and recommended looking at CMSBox if you only visit one.

Seaside is different (Avi calls it the heretic framework). It is different not for the sake of being different but for the sake of being better. There are components, not web pages, no templates (you can have them if you want them but they will only constrict you), etc. Seaside is pure Smalltalk. You write Smalltalk to define your components and your control flow, and you debug in Smalltalk (can be interesting if you debug continuations but when Seaside is running as it should - which it usually does - you do not debug continuations).

Components subclass WAComponent. They hold state. They render themselves in `renderContentOn:`. They hold subcomponents in `children`. In the Java world, you may have to rewrite a component you reuse just to have a different name for the tag; not needed in Seaside. He showed the WAStoreCartView>>renderContentOn: method, explaining various items, the `html` canvas, the `div` tag (important for identifying things to CSS styling and to AJAX and other Javascript utilities), the `with:` block to define what goes in the div, the `cart` instance variable (components have state), the `anchor` brush which provides a `callback:` whose block will be executed when the user returns to the server. Thus we do not parse HTTP requests or handle Javascript directly or anything; we just write Smalltalk code.

Components generate valid XHTML only. CSS files style this. Ideally, the code is written by a developer and a web designer creates your CSS (you'll see an example of why a web designer is needed presently; Joachim did not have a designer for his demo. :-).

So much for 'what is Seaside'. Joachim then compared Seaside Components to VA Visual Parts. Parts execute their callbacks as soon as e.g. the button is clicked. Components respond when the refresh cycle is triggered so the callback cycle is longer. (Joachim repeated the standard warning about not calling `renderContentOn:` yourself; call `render:`).

Joachim compared `abtWhenPrimitive: #clicked perform: ...` to Seaside methods such as

```
html submitButton
  callback: ... ;
  with: 'Refresh'
"or"
html textInput
  callback: [:txt | self name: txt];
  value: self name.
"or short form of the above, using helper method,"
html textInput on: #name of: self.
```

Differences:

- in fat client, the typical feedback cycle is instant (type in a field, get validation) whereas in Seaside it is only on submit (submit page, get all fields validated)
- Seaside has no GUI builder out of the box; you generate your HTML and style it in CSS.

Q. Generate Seaside from composition editor (it can generate raw HTML)? Not in 8.0 or 8.1 and probably never because they find that people always want to change generated HTML. The composition editor never generated attractive pages and this is so for all such fat client generators because (Christian) the web model flow is not the same as the fat client's. (Also, a professional page will usually be designed by someone who uses a product like Adobe Photoshop anyway.)

Q. Can you draw widget connections a la composition editor? No you can't but code is faster.

In general you will not generate UIs that exactly resemble the fat client because to do that would need a div tag for every element in the UI. It is usually more natural to live with what HTML renderers do by default in some areas and accept that the UI looks a little different. There are two projects to build Seaside GUI editors. (One is SeaBreeze from Heeg which they have open-sourced under MIT with encouragement to port to other dialects.)

Tasks are subclasses of WATask and use `go` to control flow e.g. to ensure login is requested where needed.Components and Tasks can `call:` and `answer:`. The `call:` receiver component is replaced by the parameter component; when the latter receives `answer:` it returns a result (the parameter) and control to the caller. It is the calling component, not the whole page, that is replaced by the callee. A Seaside application has a task that calls components just as a fat client has MVC, so writing a Seaside application is very like writing a model GUI application.

AJAX is the most popular approach to writing rich internet applications (others are AIR, Silverlight, XOOL). AJAX lets you exchange a small amount of information between server and client to change a small part of a page. Script.aculo.us sits on top of Prototype and both are Javascript libraries. Prototype handles all the browser differences. Scriptaculous provides clever widgets and effects. Seaside wrappers them so that developers write Smalltalk and Javascript is 'rendered' in Smalltalk.

```
html updater
  id: 'myelement'"CHECK CORRECT NO ; HERE'
  callback: [:r | self renderNewStuffOn: r].
```

Then he demoed ("This is early beta: be prepared to see a debugger."). He opened a YahooTrafficComponent that rendered a table, an image and an internal table with some scriptaculous `html effect ...` code to show a drop down list appearing and disappearing below a button being clicked.

```
html evaluator
  id: #result;
  on: #renderEvaluatorOn: of: self.
```

He showed the very small amount of code required and then opened and demoed, invoking these effects, expanding drop-down lists, drag-dropping data from one component in the page to another, etc. He could not show the debugger in the browser (`answer:` has a bug at the moment), so instead showed the debugger on the server (opening it even quicker than he expected because he got the `answer:` bug the John is analysing at the moment whereby the result loses its stream every now and then :-).

Q. Does a developer of this need to know Javascript. Not to call it. Joachim advises understanding the Javascript flow of control. Instantiations will support Prototype and Scriptaculous? They will support ensuring you can call it from Smalltalk easily and will feed back any bugs found in it to its authors (but discovering a bug is unlikely, since it is in heavy use).

What must you change when moving from a fat client to a Seaside app. A fat client will manage persistency, presentation and business logic. In a web app, the browser has a small presentation layer of HTML and Javascript, and it is the server that has the presentation layer (serving the browser), the business logic and the persistence. Fat clients use their database to synchronise data; clients do not know about other clients, just about what the database tells them. In the web app, the server may have several copies of a persistent object, one per session. A web server will need connection pooling, parallel transactions, multi-threading and isolation in the image.

The situation is similar when you access other back-end systems such as CICS, host programs, etc. Your requests must not block the server, so if your current system has problems with that, look at replacing your current middleware with TCP/IP comms.

Since server round-trip validates at different times than the instant handling of fat client events, you must rework your validation model. In some fat clients, authentication is easy: 'If I can login to the database, I'm authorised for the whole system.' The server will want a separate authentication for itself as it uses connection pooling, so does not have a connection per user.

Server Smalltalk and Seaside can handle several hundred requests per second on a normal PC. Joachim is confident overall performance will be determined by the application.

Scheduling downtime for maintenance tasks (installing fixpacks, DB migration, etc.) is more important when you move to a web app. If your browsers talk through Apache to a number of Seaside images, you will want sticky sessions to ensure a user gets the same image through a session. Joachim sees no problem with these though he knows some people dislike them; he is unclear why.

Q. Security? Server Smalltalk handles HTTP/S but it may be wiser to let that be handled by Apache whose builders know a lot about it (Louis: or

you could buy a hardware load balancer, which will usually have security).

To start the project, use 3 or 4 developers to build a proof of concept using a few dialogs of your app (some easy and at least two of the most complex ones). Let it run for two months with fixed check points at 4 and 6 weeks and a hard 'that ends the prototype' in 8 weeks. Why convert at all? Well you reuses your Smalltalk code and skills while looking like all the other corporate apps and making your team's strengths visible.

Q. Security in Seaside specifically? Seaside avoids many issues that other frameworks face due to its non-RESTful URLs; an attacker cannot guess what the next URL would be from the one they get. Your page elements also have generated names. Thus you have security by obfuscation.

Seaside is fun: you can motivate your Smalltalk developers a lot if you give them Seaside.

Q(Christian) Seaside lacks documentation (your presentation adds to the documentation; thanks); will you add more? The Potsdam tutorial is good. Lukas and Adrian plan to write a book and a good deal is being recorded on the wiki as part of 2.9.

Closing Discussions

Why choose VASmalltalk? Test-drive it and see. (I mentioned Envy robustness; refactoring may be harder than in some other Smalltalk CM systems but it is a solid mature CM system.)

Q(Louis) grow support staff? Nick cannot make these decisions. They are pursuing a business case within the company for expanding and they hope and expect that it will be approved.

Q.(John O'Keefe and Joachim) anyone use VA Generator? noone. Anyone still on VAST? One or two; very few.

Q(Christian) next forum? Last year people said have a forum each year or when a new version is upcoming and now looked like the right time. They have not decided when the next one will be but they tend to think one per year. September is close to ESUG and autumn is the busiest time of year in Germany so maybe May is better (provided we do not clash with StS).

Other Discussions

Die Mobiliar's Smalltalk system is still going strong. It provides web services to Java front-ends. With luck, they may be able to experiment with a Seaside front-end soon.

A VA user wants to replace TopLink with Glorp.

An Irish company has web portal ERM product (mapping from web to independent back offices): Java web skin to Smalltalk system. They want to hire fresh Smalltalkers: see the smalltalk jobs database for contact info.

Conclusions

My tenth ESUG, my seventh Smalltalk Solutions and my second VASUG:

- Seaside just keeps growing
- Being respected by adherents of now-fashionable Ruby makes a pleasant change from being rejected by adherents of no-longer-fashionable Java.
- Smalltalk's second surge was much in evidence.

Written by Niall Ross (nfr@bigwig.net) of eXtremeMetaProgrammers Ltd

---

* End of Document *
