Smalltalk Solutions 2004, Seattle, 3 - 5 May 2004

I spent the preceding weekend with friends at Kirkland (north Seattle). The weather was good (unusually so, I deduced, when the guide on the boat tour we took spent much time assuring us that Seattle's reputation for rain was undeserved; but as I flew to Seattle from Glasgow, I can't talk). The boat halted in front of Bill Gates' house for several minutes while the guide described it in minute detail. I felt sorry for Bill; if you open your house to visitors you can close the gates when you choose, but he has a beautiful shore-front lawn - and boat trippers coming inshore to gawk at it when they please. (Perhaps it's a judgement on him for not using Smalltalk more.)

We visited the Space Tower on Tuesday night for an excellent meal, with excellent views and excellent conversation. I learnt more from George Bosworth about how .Net's users and market affect how and when one can get new features in, implementing them in the VM being usually the easiest part. Jim Robertson and I then walked back to the hotel, setting the world's problems to rights on the way.

Style

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

Author's Disclaimer and Acknowledgements

This report was written by Niall Ross of eXtremeMetaProgrammers Ltd. No view of any other project or organisation with which I am connected is expressed or implied. It is as accurate as my speed of typing in talks and my memory of them afterwards can make it; please send comments and corrections to nfr@bigwig.net. I thank all speakers and participants whose work gave me something to report, and the conference sponsors: Cincom, Gemstone, Knowledge Systems Corporation, Mission Software, Why Smalltalk, Reiling Consulting Corporation, IBM and Synchrony Systems.

Summary of Presentations

I have sorted the talks I attended into various categories:

- Web Development
- Applications and Experience Reports
- Frameworks
- Tools, Testing and Process
- How small can Smalltalk be?

after which I list Talks I Missed (those not summarised above), report on STIC and Vendor BOFs, describe Other Discussions, note some Follow-up Actions and give my overall Conclusions from the conference.

As there were often two and sometimes four parallel programme tracks, plus ad-hoc discussions, I could not attend, still less report on, half of what happened. (Some of the choices forced by the schedule were painful; should I listen to Michael Lucas on withStyle or get a tutorial on Seaside from Avi Bryant?) The talks' slides should be reachable from the conference website (http://www.smalltalksolutions.com/). For info on talks I missed, and others' take on some I caught, see Michael Lucas-Smith's blog, James Robertson's blog and John McIntosh' trip report: wiki.cs.uiuc.edu/CampSmalltalk/Smalltalk+Solutions+2004+Trip+report.

Opening: Allen Davis of KSC and STIC, and Alan Knight of Cincom

Allen thanked the conference sponsors, STIC, Alan Knight and Jason Jones. Alan remarked that it was good to see so many people here. He then introduced the first keynote speaker, explaining that Avi is just this guy, you know :-). Alan suggested Avi is an example of a general rule that the best smalltalkers are Canadians, Maybe he has a point, I thought; in my mind, I listed Canadian smalltalkers (perchance, list includes a certain Alan Knight? :-). Remind me to ask the impressive Australian contingent at this StS for their opinion (or remind me not to :-). A pleasant opening.

Exhibitors

In the past, Silvermark has shown me how to making Test Mentor support driving very application-specific widgets. We discussed following up on this. I also had useful discussions with the strong GemStone contingent.

Web Development

Keynote: Winning the Application Server Arms Race: Using Smalltalk to Redefine Web Development, Avi Bryant

Web apps are really awkward things. They are used because they are everywhere: you would not download individual clients to do all the things you do via web apps. Applets failed not because Java was not everywhere but because they were not the web. By the time applets came around, users had a mental model about the web that included the back-button; browser-based apps that were not web apps did not fit this model. A web app should feel more like a website than an application. Java failed as applets but had success on the server (message: server-side is a second chance).

Paul Graham sold an eCommerce store app, written in Lisp, to Yahoo for $43m; it was (and is) a core Yahoo app. Why did he write it in Lisp? "Because you can." Web app users don't care what it's written in. Lisp 'is a weird dialect of Smalltalk', i.e. it also has been around for a long time and is much more productive than ordinary languages. Paul could fix a bug in a running image while on the phone to someone reporting it; 'What bug?'

Why should the Smalltalk community be interested in Web development. Because we can. On the client side, rivals have an unfair advantage: Squeak looks funny. On the web, this does not matter so we can leverage the fair advantage we have. If you think web apps are awkward to use, you should try writing them!

Avi then talked about why web apps are hard to write and how Smalltalk can help in ways other platforms can't, creating a possible big win for us. A web app is

- user types URL

- that sends request to webserver

- server responds with an HTML doc

- doc gets rendered in your browser (text, buttons, links, ...)

- user reads, then invokes some operation in browser

- that produces another request

and so on. Thus web apps are stateless; each click on link is like a separate program. This is not like a typical client-server (e.g. VW client to GemStone server).

This is fine for really simple things. However, any real web app must have state. A typical eCommerce process might be

- get shipping address

- get billing address

- get payment info

- show confirmer

- get user acceptance

which builds up a lot of state. The traditional solution (many web apps use this) is that state gets shuttled back and forth in each response via hidden fields in forms. The billing request has the shipping address hidden on the page, etc. For developers, this is nasty; you must marshal this all the time. You spend much time parsing input, then do something very trivial, then spend much time parsing output. It's horribly tedious and can't be automated because each piece of state is different.

Global session state is a commonly-used bad half-solution to the above. A session is created to last from the start of the user's site interaction to the end. Thus you assign a unique session id and every request and response includes this session id. So you just send the new data and the session id instead of all the data. Now, suppose you get near to booking a flight, then use the back button to check something, spawn a new browser, then decide, that after all you will book the original flight. The global state now has the wrong session id and so the wrong data. (Generally, global state combined with threading is always a problem due to race conditions, etc.)

WebObjects (written in Objective C which is Smalltalk-like), arranged that each page instance within a session would have a unique id. That instance, helped by the overall session, handles the response and returns the id for the instance of the responded page. Thus if you backtrack, you get a fresh instance of the page, with a fresh id: you do not get confused. You don't have to marshal your state back to the user because these instances are creating themselves on the server and can use normal accessors to

communicate state between each other. This has problems but is a lot better and is a smalltalky way for approaching the problem; you are interacting with live objects, not marshalling state. (C.f. source code configuration management in Java - what directory is this in, what files - versus Smalltalk - my source is in this object.)

WebObjects does as much as you can do in ObjectiveC, or in Java (Apple ported it to Java when they bought Next; see also WebTapestry, the only other thing Avi knows that uses some of these ideas). However it does not address coupling, the next huge problem. Many web developers envision their site as a network in which this web page points to that one, which points to that other one, and likewise for the objects pointing at each other on the server. That's fine if you have a really rigid setUp but it's a problem if you need to rearrange thinks and a huge problem for evolving your system. One of Avi's early clients wanted a box office system where the main tasks had many similar steps - pick a show, pick a seat - but in different orders - selling a seat and exchanging seats did not start from the same place or go through the steps in the same order. The solution to this problem is continuations.

"Seaside is to other smalltalk web toolkits as Smalltalk is to most other OO languages; it's as simple as that.", Cees de Groot. This says it well. Seaside is not (just :-) better but better *in the same way* that Smalltalk is better, by using live objects not dead data.

Avi then started to build up the idea by refactoring a code example. When programming e.g. an eCommerce web app with a shipping address page, a billing address page, a payment details page and a confirmer, you could allow someone creating the component for one page to pass in the next component.

```
checkoutProcess
  ^ShippingAddress new next:
    (BillingAddress new next:
      (PaymentInfo new next:
        (ConfirmationPage new)))
```

Suppose you pass a block instead of an instance; then you can add conditional behaviour, and have the state as parameter.

```
checkoutProcess
  ^ShippingAddress new next:
    [:ship | BillingAddress new next:
      [:bill | PaymentInfo new next:
        [:pay | ConfirmationPage new
          shippingAddress: ship;
          billingAddress: bill;
          paymentInfo: pay;
          yourself]]]
```

This puts all the state in one place and reordering is easy but it looks odd. You are passing a block that tells you how you continue with the application (thus the name 'continuation'). You would prefer to put this behaviour in methods:

```
checkoutProcess
  ship := ShippingAddress new.
  bill := self call: BillingAddress new.
  pay := self call: PaymentInfo new.
  self call: (ConfirmationPage new
        shippingAddress: ship;
        billingAddress: bill;
        paymentInfo: pay;
        yourself)
```

Avi demoed this code running, including use of the back button. That rewinds up the method and forks off a new version that continues. Having the back button work as users expect is essential; users use it all the time. It works because `call:` makes a copy of the context stack. Avi demoed cloning the browser and going forward and back on the two tracks.

Q. how do you ensure only one state is right, if necessary? Seaside has a method `isolate`. A call of `self isolate` done e.g. after the user has confirmed one purchase, invalidates the other stacks. Avi demoed getting a 'this page has expired' by OKing one purchase after splitting the browser, then trying to OK the rival fork, in a `self isolate` case.

Q. Forward button works? Yes; Avi discussed the various cases. In this eCommerce demo, the forward button works before you submit and not after, as you would expect.

Q. space for these continuations? You configure to expire continuations for all but the ten most recent cases and that keeps footprint down. You are trading space for ease of development. In 95%+ of webapps you will write, it will be a good trade; maybe Yahoo would hesitate to use Seaside, but mostly you just add another server (by definition, you must be being successful to have the issue). Provided one user session is on one machine (standard anyway for webapps), you can add servers easily.

It is great that you can write the whole process in one method. It is a happy add-on that you get dynamic context across multiple requests and this helps exception handling. You can put an error handler around a high-level method, not around each request-response. Avi inserted an error and demoed how the developer sees an error on the page, hits the debug link, sees a debugger, fixes it, proceeds, and can still use the back button, etc.

Seaside lets you register objects as backtrackable so that this does not just work for temporaries; any objects you want to be rolled back will be rolled back (usually you use a stateholder wrapper for such objects). Session data conflict is impossible by design. (Obviously, any multi-user app using e.g. a shared database, must use distinct transactions, etc., and use standard techniques to handle that.)

In Lisp, Scheme has continuations and Common Lisp does not; it is a major at-the-VM-level choice. In Smalltalk, it is ten lines of code that you add. Avi showed the code; stuff all the temps, program counter and etc. onto an array. To restore, kill the current context, restore the saved one with all its values, and swap the context's sender, so you return to where the context

was captured (usually the self call: point). N.B. context termination does not handle `ensure:` perfectly; in Seaside it is desirable not to have ensures that go across multiple continuations (the interaction of continuation and ensure is an unsolved problem and is the reason Common Lisp does not have continuations, since it does have ensures).

The `call:` method stores the continuation and shows the component but avoids returning normally so it instead returns when the next page invokes the continuation. (Continuations hang onto compiled methods which hang onto literals but nevertheless they can be serialised, so they can be sent to another image to support the user from there if wanted.)

This is fine between pages; what about reuse within a page. Suppose a page has two addresses (e.g. shipping and billing address, or old and new address); we'd like to reuse two instances of that object on the page. The problem in the old approach is that duplicates of two keys must not occur so street must be street_1, street_2 and you're back to marshalling unnatural data. Seaside says, when you do this, don't give the fields a name, give them a block. The html is generated with the unique ids of these block objects so when the request comes in you just marry it to its block and execute. Avi showed Smalltalk code to programmatically create html.

```
employees do: [:each |
  html anchorWithAction: [self showEmployee: each]]
```

At first glance, programmatically generating html breaks the standard assumption that logic and presentation are supposed to be separated. However CSS is the right way the web community should achieve this separation; the Seaside approach is to generate very barebones html and use CSS for doing all your design. Seaside took halos from Squeak. You can toggle halos to show halos on all the elements of a page and you can select and do all sorts of stuff, including edit the CSS (and a web designer working with you can too). Avi demoed using this to convert multi-element control from tabs to side-lists. The Zen Garden (David Shea, Vancouver) is a site that shows how CSS can make the look change massively for the same barebones html. Thus the developers generate barebones abstract html, refactoring it to be productive, and the web designers then dress it with CSS. Today, Seaside puts CSS in component #style methods so it gets configure-controlled; Avi would like to progress to a CSS library.

Q. Suppose the alternatives are many; how do you handle that? You handle that on the call-back side; your app becomes event-based. (There are complex example applications people can study.)

Q.Porting? VW port is in Cincom current store. An old Dolphin version exists. VA and Smalltalk/X cannot use it because the stack cannot be copied. He does not know for other dialects.

Q. Documentation? There is not that much; attend the tutorial, watch the (very-active) mailing list so if you understand enough to ask a question I'll answer. The online tutorial is good (but only takes you so far).

Rendering and Editing with CSS in Smalltalk, Michael Lucas-Smith and Rowan Bunning

(Rowan motivated the withStyle work, then Michael described and demoed it.) Avi has described a super server-side solution. They offer the client-side companion to it. Web development has not moved on. One reason is because we're stuck with web browsers. For many web developers, you want to escape from that. However we have not yet seen convincing very usable XML solutions. Rowan has been looking for a good XML editor for a long time and has found little better than tree editors. However XML and web services are here.

We're stuck with very poor scripting (try dealing with a 13,000 line JavaScript app), Mozilla (bloated 20Mb downloads of 16,000,000 lines of C++), poor standards (little change from 2001). The browser widget pallet is poor, the interaction model is poor (tedious to iterate back and forth because of that field you forgot to fill in). Web browsers feel like a step back in evolution. "The web cripples usability.", Forrester, 2000 (but his X-internet proposal is years away from even being a prototype).

Flash-based UIs are flavour of the month, but ActionScript feels like Javascript+VB; ugh!!. Java applets, ActiveX, Mozilla and IE behaviours are all tedious. Their alternative is withStyle.

TwoFlower was an inspiration; showed it could be done. They want withStyle to be more than a rendering engine for XML; they want it to provide rich web applications with the same UI abilities you get in client-server app GUIs; drag-drop, instant sorting and calculation, data visualisation, etc. They aim to reinvent the browser paradigm, designed for applications. (So their immediate focus is Enterprise Applications.)

(At this point they had the usual demo hiccough; their whole presentation was being done in withStyle and a debugger popped up; they quickly fixed things. Cause: Rowan as a Mac user found having two buttons on his mouse challenging :-)

Michael started by showing ZenGarden (http://www.csszengarden.com.) running inside withStyle. Being able to render ZenGarden pages is a great challenge; they can render half of them at the moment. He explained that withStyle is an XML renderer, not a browser. Browser is just one of its many applications. He then introduced the demo. Styling XML with CSS is only the start. They aim for Pollock integration, for XForms (Mozilla people don't want to do them because it needs them to know 'too much') and for CSS-skinnable apps. CSS provides rules like, 'I want this information to be blue and always to the left of all else on the page.' XML is just information. They map XML into behaviour so they can do something with it.

They have built apps: a Web Browser, an XML editor, etc. They offer a developer program; you can sign the NDA and download the nightly build plus all tests (and test current state). They have 600+ tests, 5,200 methods and 35,000 lines of code, doing a long list of things (see slides).

Michael showed the web browser looking at www.wc3.org (front page only; others are all old), their first benchmark since that's where some of these specs come from; he pointed out two small divergences from what the page looks like in their own browser. Then he looked at ZenGarden; one of the pages has changed in the last week and showed a debugger. He then showed editing an XML document. The HTML knows what XML it comes from so you edit through HTML to the underlying XML. He showed the element-specific popup menus for editing the underlying XML, showing XML-specific, non-html things. (You can also edit the HTML directly if you wish.) WithStyle will plugin to IE and Mozilla.

He then demoed creating a slideshow. In the VW browser he added class FOO, methods slide, slide1, slide2 (XML-aware editor), then went to `resource://Foo/slides` as URL displayed in the browser. He then refactored to share common elements, created a Slide class and gave it behaviour (nextSlide, etc.) to attain the effect of a slideshow. The slides were now objects. He put them in a namespace (so they were recognised as URLs) and created a client FooClient class to be the root model object holding the slides. This displayed the first slide when shown but clicking had no effect. Michael added an XML event, tied to nextSlide.

```
<body.ev.type="click" ev.handler="clicked">
```

Now when he clicked anywhere on a slide, it advanced to the next slide.

Right now, you can use this at the webpage level. Seaside is a good server-side partner because Avi recommends simple HTML with rich CSS which is just what withStyle does. At present, they use straight XML. A 3-tier model (business logic server, serialise XML, client side withStyle) could be done. Visit www.softwarewithstyle.com for downloads, etc.

Further Demo of withStyle at later session
(Missed first few minutes.) It has 3 layers of API:

- basic

- if you want to do your own widgets

- if you want to do your own view

There is some information overload in the web notes which cover all three cases; to learn how to use withStyle, concentrate on the first style.

WithStyle can edit XML intelligently. It can see if it needs to split up the DOM tree if you e.g. apply italic across parts of adjacent nodes. Menus are built to let you edit a node at various levels, HTML, XML.

Q. MS word does not handle lists and nested lists properly; Windows loses ends of lists. Can you do better? Glad you asked, I have example (demoed). This is XML so you know where the list ends are. The engine is rendering engine; widgets within let you edit things.

You don't have to re-render everything every time. Version 4 of the rendering engine (which uses Pollock and is not yet released) is geared to re-render only those parts of the layout that changed.

Their classes subclass Pollock classes. They were key pushers for the double buffering and flicker elimination that Sames now does. Pollock does various CSS-required things for borders, etc., so they will piggyback from it. They use Pollock for the common UI functions of walking a display tree but have their own scribe for e.g. selecting because they have deeply-nested panes across which they select.

Michael threw some HTML into a workspace and added a Pollock list box in it (just `<pollock ListBox>`). Pollock would use code to get the items; they could provide the items as XML. CSS can then layout all these elements of the page as they wish. (Pollock layout rules are just classic GUI layout rules which have their place but they offer the chance to go beyond.)

Their XML editor is aimed at people who know what XML and CSS they use and have clients who want WYSIWYG word-processor style editing. They are not rivals for XMLspy and similar abstract XML tree-layout editing tools.

Combining with Seaside

• Seaside: get request, reply with an entire page

• Seaside + withStyle: get request, reply part of page

The combination can give much better response.

WithStyle also gives the choice of handling view XML or model XML. Avi is dubious about using model XML but it would allow fatter clients to then handle model XML in some way. (Browser vendors are currently ignoring the X-forms standard which advocates this approach.)

Avi: I do not see widespread *internet*, as opposed to intranet, adoption of VW end-user clients (nor does Michael), but for the *developers* of internet-targeted web apps to run withStyle makes perfect sense. Then Seaside can put out extra info such as halos but better (e.g. right-click and see what Seaside component it was rendered from). Michel demoed a simple 'where am I on the screen' monitor, right-click to inspect Smalltalk object, etc.

Smalltalk, XML on the Web, Michael Lucas-Smith, Wizard Information Services
Seven years ago, Wizard people met with a Java guy and an IBM Smalltalk guy and chose Smalltalk. Today, they get visits from IBM people and Java people who both tell them they should use Java, but they still use Smalltalk. Their aim was to build a core framework to support their products; OO but also pattern-oriented, emphasis on Naked Business Objects.

Their biggest success is Modis (Merged Audio/Visual Information System) used by the Library of Congress, The Oscars, Bundesarchiv (Germany), National Library of Norway and Screensound Australia who were their

first client. The system manages resources (e.g., 'This is a 50 year old tape so run it through acetate and do a cool check before you put it away or the next person who views it will break it).

Meta*WizDom is a content management and Business Transaction System. Their major customer is the Australian government. It manages 35 web sites, letting each site give the site-specifics look and feel to content that is often general.

Meta*Grants is a government granting system that helps both the ACT and Brisbane city council allocate grants with hopefully less waste than usual.

All Communities Online is an event-driven system that lets a group make simple web pages to run its activities.

So Wizard has a strong and profitable commitment to smalltalk.

Some of these apps are already web-enabled. The rest want to be. This talk is about how they can expose naked business objects via APIs. They simple need to create, read, update, delete and search (much simpler than Avi's domain). Their solution is 3-tier.

- The back end data: DBs, XML, file systems
- the business logic: smalltalk apps talking XML / HTTP to a web container (Apache Cocoon)
- the web container talks the XML / HTML to withStyle; it can also talk to other web browsers (SOAP / HTTP) and may connect to other clients

They use XML because it is easy to understand though alas not efficient. It is easy enough to read and debug *if you are a developer*. XSLT has one job, to transform XML into another structure, and it is well supported by Apache Cocoon and by withStyle. SOAP is XML with some tags, REST an alternative that they also support. Apache Cocoon is the publishing interface they use (the best piece of Java programming they've ever seen). WithStyle can talk XML straight back to the server, a very nice feature.

They use TopLink today but will move to GLORP. They use TopLink's 'privately owned' concept to decide what is serialised and what is taken only as an href of type 'resource', to be linked back into the system.

XML schema is barely machine readable and does not map well to OO. SOAP does not need it but XPath 2.0 does (and the industry thinks they want it). Generate it and regenerate it; never, never write it. He uses transformation rules and cardinality rules, called by exception handlers, to deal with tricky cases.

Their server translates SOAP into Smalltalk using XSLT. It then executes the Smalltalk and the result is cached. Cocoon gets XML from the server, transforms it via XSLT and caches the result. It holds session info and only validates user id and password via the server. Thus it is secure since people must access through Cocoon.

Michael demonstrated a VAST image linked to Apache Cocoon, logging into a demo web app, calling up information and showing work-flow-controlled page updating. Then he added simple behaviour (a 'hello world' popup) and showed the web browser triggering it the VA image.

A transformer turns XML into something else. A serialiser is the converse, turning an object into XML. Actions give behaviour to objects (e.g. update, refresh, save, cancel). Patterns define a configuration of transformer, serialiser, actions, etc., so clients can hit the server with radically different patterns of request. Rules allow you to find out what classes and instVars are visible and so choose which configuration to get according to your policy, e.g. policy might be to as much information as possible. These rules are in a mixture of Smalltalk and XML syntax.

Q. What do you do to put your Smalltalk app on the web? Not much: XML is generated for you, The XML transformer is in system for you. You can use withStyle to transform the look, actions, whatever. Think of it as a web-driven (GUI), command line (functionality) interface for your smalltalk application based on business objects being the central point and messages being sent to them. The responses are turned into XML and further transformed as you direct and web-displayed.

Applications and Experience Reports
Replacing Oracle with GemStone/S: The Agony and the Ecstasy, Joseph Bacanskas

Joe lives in Seattle. Typical weather for Seattle this time of year is rain; we have been so lucky during this conference.

For the last three years, Joe has worked at Washington Mutual Bank on their Smalltalk applications. WMB put their first Smalltalk application into production in 1996. It was for a leading application, was very successful and gave great competitive advantage to the bank, who decided Smalltalk was a good thing. They then decided to build Visual Banker, which is now in production and has 25,000 employees using it.

The bank had several non-Smalltalk applications that had good business value but would not scale. Joe was hired (as a consultant; more than half the bank including much of the management are consultants) to convert these to Smalltalk applications that would scale. Joe's app was a fraud prevention case management workflow application with 200 users, with a VisualAge Smalltalk front-end and an Oracle back-end. It went live in 1998/9 but had some problems. Users (WMB software development is mostly funded directly by users) decided to replace Oracle with GemStone because the cost of change of this system was too high, and the consultants skill base needed to maintain it were expensive. That was the business case for change; the technical case for change was that its maintainers could see clearly that the existing application was very brittle and buggy.

The three amigos who built it had three ideas. The first one said, use GemStone, but the bank would not then listen to him. The second was an eXtreme Programmer who wrote tests and his stuff worked. However the

third handled the domain and, alas, he just *knew* that 'domain objects should not have behaviour'. Thus they had several layers of meta-domain objects who had to know what to do for every domain object and every combination of domain objects. This third chap also disliked named instance variables because they were 'not flexible enough' so he provided dictionaries for every object instead. Perhaps you are beginning to guess why the system did not work well.

An additional contribution to the original system's problems arose when, late in development, it became plain that the connection network was too old and ill-managed to handle the planned load so frantic moving of behaviour from server to client occurred at the end of the cycle.

Joe told them that the one who said buy GemStone was right and a while later they hired him and told him to do that. Joe moved the client to run on GemBuilder, helped by a colleague who refactored it skilfully. There were a number of challenges with the database, most due to legacy maintenance.

Change management had to be done such that changes also went to Oracle as the legacy tracker. GemStone provides GemConnect to do this. However GemStone provides two ways for change control, a verbose but very controllable way or the meta-way, much less code to write but less easily tailorable. The bank had UI standards which required that 'list' and 'edit details' had to appear on the same UI page. Nothing gets committed from that page till the user hits 'Commit Case' so the user can add transient objects and delete them again; this made change control out to Oracle hard.

They killed huge numbers of stored Oracle procedures and reduced it to two connections. Their design ensured that anyone can read from Oracle but only GemStone could write to it (as it had to be as Oracle cannot update GemStone). Kent Beck says, that most object systems have three objects. That was true in their system: Case, Account and Customer summarises the key roots of their domain hierarchy. They designed a set of searchable collections based on these roots to handle access to domain objects.

They needed to get changes to the domain classes from GemStone to Oracle; GemConnect allowed them to define five (meta)classes to handle that. All the domain objects knew what tuples they would need to represent themselves in Oracle.

They had a good team but did not have a common understanding of how the application worked before, or of how to build application-to-GemStone interactions, nor did everyone have deep understanding of how GemStone worked. One guy would say 'It doesn't work that way? But it *should* work that way!' (I know how he felt but, as Joe said, one has to accept that it does not work that way.)

The bulk loading process was an issue. The fastest loading time of 60 million Oracle rows was 4.5 days (on a 1 Gig connection) but the longest available window was 3 days. His load had an oopHighWaterMark of 980,000; the limit is one million!!! 40 million of the 60 million rows were history events. Then they had to build all the objects from the tuples!!!

First, they used massive parallelism on the load to get it down to nine hours, plus they subsetted the data and allowed unarchiving data when someone requested really old stuff. Once they got it into production (10 months ago) they had lots of problems. They had design choices with subtle bugs (all to do with change notification, surprise, surprise :-). They used reduced conflict classes, normally very good but you must know how they work if you get clever with them. Joe peeked the RcQueue (he knew what a queue was; that was the problem) that updated Oracle, then said `queue next` but alas that might not be the one he peeked. This was very hard to debug and was found by a GemStone novice, fortunately at a time when Joe was on holiday or Joe would have talked the novice out of believing what he actually saw.

There was a major bug in GemStone indexing (fixed in 6.1) that was hard to work around.

That was the agony. The ecstasy is that now they can change behaviour on hours, not months. An emergency fix can be applied in real-time without client disruption (not quite so fast now because they were told if anything else got changed without management's change control procedures being fully followed, someone would be fired, but it's still a lot faster than before). Background tasks (reports) can be done on the server in minutes instead of degrading performance for hours. Business rule update used to lock-up the updating client for a long time; now it is dropped in a queue and is handled transparently. The 40,000 lines of code that used to handle the O/R mapping are now replaced by circa 1000 lines.

The users are a lot happier. They just rolled out a release that went like clockwork. They are moving lots of behaviour from clients to server which speeds the client; anything that can be in GemStone will be in GemStone. Lots of internal customers are now keen to see the system's information. They have been able to dispense with the full-time Oracle DBA.

Q. Why keep Oracle at all? We need to keep legacy data back for years, plus the Oracle DB was set up to do some reports it would have been tedious to redo.

John McIntosh did the reverse for an Atlanta-based client earlier this year, moving an application's database from GemStone to Oracle (change was for management reasons; there was no technical problem with GemStone).

The OpenSkills Skillsbase Project, Bruce Badger
Bruce is a smalltalk developer and database specialist. He is also president of OpenSkills, an international non-profit group looking at how to make money out of open source software by selling skills based on it.

The SkillsBase is a global non-profit resource that allows the identified members (identified via cross-signed keys) to promote their skills and be searched by those needing those skills. Bruce does Java for food but uses Smalltalk when he wants to deliver something that now handles 50 members in eight countries but wants to be able to grow, to survive being mentioned on slashdot, etc.

Bruce uses squid to cache static data from their server (thus scalable to massive hit rate) and to handle https and authentication. The backend is GemStone which will run Swazoo (i.e running inside GemStone - no external Apache or suchlike). Bruce joked about being the third speaker to tell people to look at zen garden (it must be good) but he's using XHTML and CSS to serve up raw developer-ugly pages at the moment, highlighting the key information. They serve HRXML as the external form of the member information.

For development they used VisualWorks (Swazoo is already in VW; they ported to GemStone). Analysing the problem was the major task. Building the model and SUnit tests in VW was easy, leaning how to use Swazoo was fairly easy, working out XHTML and CSS was hard, squid was easy with help (from Rob Collins, a squid developer). Porting Swazoo to GemStone was time consuming but would have been easy with the latest GBS release.

Bruce portability principles are not to change the system classes (I agree), to delegate from them instead of subclassing them (I disagree) and to use ANSI standard Smalltalk whenever there is a choice. Porting to GemStone revealed that there are utility method missing, e.g. `asReadStream` must be `ReadStream on:` (Q. why not add the missing utility methods extend in separate category? Good idea). There are also behavioural differences. `'Hello World' asByteArray` in VW prints 'Hello World' but in GS prints 'aByteArray'. Behaviour of `aStream upToEnd` differs (I've met this one between VW and VA as well :-/). Etc. (see slides).

Bruce demoed completely from scratch, loading his system into a pure startup VW image. Bruce uses a master bundle to capture all the version dependencies. He started the system and showed a web browser querying it, with bare-bones HTML (and fairly barebones CSS at the moment). Login uses standard HTTP request, not cookies, which is not ideal (logout by closing browser) but it is a standard. He then showed a hypothetical member adding their data and adjusting their visibility (often you want to tune your CV to the current job you're pursuing but not throw away all the other data).

Next he started up the GemStone version. The first line of the request is a `self abort`, which means 'teleport me to the current version of truth' in GemStone. GemStone preserves a read-consistent view of the database. Any request can go to any Gem; no state is held in a Gem, state is held in browser if needed (a very naive way of doing things but it works here.) The request creates a task object that is passed to whoever helps deal with it.

He inserted a breakpoint into GemStone and showed the URL failing to return. (Usual demo hiccough here; browser got a socket to make a connection, failed to get and now the socket is tied, must wait till expires.)

You can develop in the great VW development environment and then, with a single command. update GemStone, rerun the GemStone tests and carry on. The shared page cache mean that Gems are simultaneously very scalable and very space efficient; individual Gems only hold objects that have been changed.

Q. Why use GemStone? Because he could; non-profit organisation means you can use non-commercial licences. Also it avoided tedium of decomposition of complex objects into tables and rows.

Q. use VW XML parser? No; an OpenSkills person wrote one for interest (ordinarily, Bruce would have reused VW's but as an OpenSkills person wanted the experience of writing one, he was happy to get him involved.)

TAPDance: a system for maintaining multiple versions of software, Howard Ferch
Winnipeg is where they are based. They provide high-demand applications to various clients including local fire and ambulance departments (which I note would certainly motivate them to get it right :-). One of the systems they are replacing has (amazingly) not had an outage in seven years, which gives their users high standards.

In a dispatch centre, at 03:00, a 911 call wakes someone up. While they wake with a start, bang into a filing cabinet and try to prevent their phone set falling off, someone hysterical on the far end of the phone tries to tell them something. This call centre staffer does not need UI problems. Hence the tool aims to have a very simple and obvious UI, with unconfusable widgets and very clear subpanes. Some country stations are manned by long-serving staff who need UIs with no small fonts, have limited computer experience, etc., so they have specific requirements.

Cellphone providers will be required to report location in the future but today the dispatch centre does not even get a cellphone's number, let alone its location. Some address is entered and the tool has good matching capabilities. In Winnipeg, they can show the location of any address on the map. (Smalltalk is fast: VAST is very fast to draw the map in real time, faster than the C program they replace).

The dispatcher application has four screens, which it must update in real-time. One is a resource screen. Canada has very strong fire unions: if a truck has not got full complement of people it cannot go to a fire, thus the resource screen must show those trucks as red since they can only go to medical emergencies. (I found myself thinking that while Canadian Smalltalkers may be good, the country may have issues in other areas. :-)

Other screens handle map and incident data. In an urban environment, send lots of resources to a fire since if it gets out of control it will spread quickly and damagingly. In a country area, you send less and assume you'll have time to send more if needed. For regulatory reasons, they must use unreliable lines (state says they must use a private-to-fire-service line, so they use ISDN but supplier is not always able to maintain service).

Their goal was to avoid employing an expensive programmer to drag windows onto canvasses, probably in an idiosyncratic and inconsistent way. They have requirements analysts draw up the screens and then just get those screens. The put the GUI description in a database and build it dynamically when requested. They provide simple layout managers.

Howard showed various screens, which looked very readable and clear to me despite being untouched by human hands as far as appearance and layout was concerned. He showed various looks (including the one his daughter liked best: lots of pinks and purples and hand-writing-like fonts). He then contrasted the ambulance dispatcher look with the fire dispatcher look, showing different widget sizes (there are more ambulance incidents per unit time than there are fires, so needing different screen-space usage).

They assume that their database is relational and map to the forms via meta-data, so again needing no programmer work on specific code. Specific data has specific classes to manage it. TapDance runtime tracks all users of database records in active windows and provides general redisplay. They can override this in specific cases with a specific callback when performance requires it.

Howard then demoed, raising a hazardous material fire call against his own home and sending nine fire engines to help him. He clicked a button and it updated; 55 database transactions were needed to do the update, notifying all the stations, etc. His incident screen showed it being handled, his resource screen was depleted of nine engines, etc.

He then started the same program but with different meta-data, showing a resource duty roster application. Now they have, e.g. drag-drop abilities. These are enabled here because it is very helpful to drag resources to calendars. They were not enabled before because a just-woken dispatcher is more likely to drag-drop by accident than intentionally.

A typical application might have e.g. 65 windows and 35 database tables. They started by storing window definitions in the database but are now moving to defining them in Smalltalk.

Why Smalltalk: if you want to build a meta-data system, Smalltalk beats Java easily for ease and economy of programming it. They have one full-time programmer, two part-time and a tester. They have also used some coop students (speaker tends to see correlation between those students who like Smalltalk and those who are good programmers, but OK, maybe he has a slight bias :-). As another speaker mentioned, you spend time refactoring

after they've gone, so are the students cost effective? Hmm? They have replaced 600,000 lines of C, 5000 lines of Java and 38,000 lines of VA Smalltalk with 80,000 lines of VA Smalltalk in TAPDance.

3D CAD Framework for Smalltalk, A-S Koh

Askoh has been working on StCAD for over five years. In 1987, he worked on Motion Simulation in dynamic simulator system ADAMS. After he lost access to ADAMs, he tried various languages and in 1989 found Smalltalk. In 1994, he built OODS plus AutoCAD, which impressed ADAMS so much they bought it, replacing his simulator component with ADAMS. He released freeCAD in 1999.

FreeCAD lets you define 3D assemblies of 3D parts (simple rigid solids) connected by joints, constraints, dampers and special forces (e.g. position-dependent forces). You draw parts, run simulations, and then interrogate re what forces were exerted at specific locations, etc.

Askoh ran a demo of a steering joint example, with shock absorbers, springs, dampers, etc. He ran the demo, zoomed and rotated, etc. He then haloed the joints, selected one and called up plots of various behaviours (to display data, it uses the VW business graphics object kit, to which he has added zooming and other features).

An engineer who wants to explore an idea is the target audience. 80% of engineering ideas can be explored here to determine their viability before you buy some extremely expensive software for further analysis.

He showed how you would draw up an assembly. In a 3D view, he chose a plane (XY) to constrain the initial drawing and then extruding it to the Z dimension. He added various joints (circular, pin). All these joints are effectively expressing mathematical constraints graphically. He added a motion to that joint by customising a template, then set a step counter and animated, then got results plot for the joint.

Next, he added a second part, a wheel, again by adding a circle in a plane and extruding it. He assigned mass and suchlike details to it, applied some markers to points on it and set up a pin joint from his first model trapped inside the wheel. He resumed animation, to see how the wheel behaved, then added gravity and showed how it now behaved, then plotted torque.

There is no collision detection at the moment; he is working on it since engineering-wise it is useful. He would like to improve the UI (i.e. the displaying UI; the user's UI for driving it seemed fine to me, the displaying UI fine for an engineer but of course nothing like a game rendering engine or whatever).

A given state can be saved in a text form that is readable by a spreadsheet (where you could change the data by hand or by formula and reimport). He can also generate a POVray file for each step, letting you animate in POVray. Separate motion, shape, etc., files let you replace shapes in POVray yet keep the motion.

He then showed a 3-planetary orbit simulation, a JCB digger simulation, a collision example (puck on seesaw, using coefficient of restitution to determine collision behaviour.). He then showed an actual consulting job he did of a barge at sea, affected by waves, placing an oil platform on its supports using water ballast to raise and lower itself. Finally he showed a few classic engineering examples, spring system, cantilever beam and wobble pump air compressor (large example needed 15 parts and 20 joins).

StCAD is free for download on the web.

- The graphical code is basic and open source

- The geometric domain code is basic and open source. He is adding OpenGL graphics (using some stuff from Jun) and showed us a preview; better solids look.

- The numerical handling code is good and open source. He has well-tested powerful matrix algebra and solvers, differential equation solvers, and general Newton-Raphson. He also has a symbolic mathematics parser and differentiation.

- The multi-body handling code is advanced, and not open source

StCAD has been used for advanced motion simulation and is ready for solid modelling. Smalltalk made him very productive.

Q. Rendered by? Smalltalk graphics in everything I showed you (can use POVray as described).

Q. Who downloads? People in Boeing, US Military, Nasa and similar, plus many university downloads.

Q. Smalltalk is fast to code in; how fast is it to run? Fast enough for me. Simulating a full car would no doubt be 100 times faster in an up-to-date version of ADAMS than here but this is fine for real requirements.

Q. Visual Editing, e.g. select a part and resize? Yes but with a 'do in 2D then extrude to 3D' style as this is using a 2D package he has expanded to 3D but not seamlessly in all its capabilities.

SmallBars - a bar code library for Smalltalk, Dan Antion
Dan had two wholly unrelated needs to produce bar codes, for a friend who worked in retail and for his company. He did not want to mess with fonts. The first time he did this as an extension to TotallyObjects' Visibility product, which worked adequately but it really made no sense to do it there, so the second time he did it elsewhere. He never thought of issuing it as a goodie but later saw people ask for barcodes on comp.lang.smalltalk.

The library was developed in VAST (complete and tested as far as a $99 bar code reader is a complete test) and ported to Dolphin (basic port, untested - he's just discovered he can read bar codes on a screen so maybe he can test Dolphin without first writing a printer). It only handled linear bar codes (no 2D barcodes yet) and is generic (no other libraries needed).

The library returns bar codes as DeviceIndependentBitMaps. The library is free to use and port but do not publish the source and you cannot use it to write a barcode printing utility (due to royalty issues).

He supports (in order of genericity) Universal Product Codes, European Article Numbering, Numeric and Alpha-Numeric (this last was what Nuclear Insurers needed). He demoed what 'Smalltalk Solutions' looked like in all four of the alpha-numeric types (39, 93, extended 39, 128).

The barcode engine SsiAbstractBarCodeEngine does the basic drawing and rendering. Its subclasses let you add footnotes and other stuff. SsiAbstractBarCodeSymbology provides the 'character' table, interspace, etc. You call `standardCode39For: aString` and suchlike methods to create these, getting more detailed control via, e.g.

```
standardCode39For: aString dpi: aPoint height: hFloat
  font: aFontString metric: aBool.
```

Dan has only tested printing to his laser printer and to his screen but not to all devices and dpi's between. The DIBM sometimes has trailing white space, which he has fixed in UPC but not yet ported to all the others. He will do refactoring as time goes by. He would like to do 2D bar codes but the spec costs $1000 and his employers have not yet told him to do it. 1D specs are on the web (N.B. but not all correct). If you want the library, email him (antion@attglobal.net). He'll email you a 'conditions' email and when you accept that he'll email you the library.

Q. Why Smalltalk? The data goes into VA, plus existing VA methods for drawing rectangles and put encodings in dictionaries made it very easy to do. (Had there been a logic to bar code initialisations, it might also have been easier to implement that logic in Smalltalk, but in fact there was none, except for one code that used straight binary; he had to hardcode it.)

Porting to Dolphin took him ten hours, mostly rewriting the rendering piece which is different in Dolphin, and providing a few things not in Dolphin. The original probably took three or four weeks.

Dan deserves a prize for making the innately less-than-gripping subject of bar codes interesting, though his remark, "If you haven't seen these, you're not living", though literally true, overstates things implication-wise. :-)

Using SNMP for High-Performance Network Monitoring, Alex Pikovsky, Quallaby
Alex got Cincom's VW network clients project much the way Sames got Pollock; he was asked what needed doing in VW and rashly said, "network clients." Later, Quallaby asked him to build a high-performance network monitoring tool. The tool, Proviso, is a fat data pipe for collecting fault management, provisioning and billing data. Their own installation collects data from over 1 million managed objects, generating 100Gb of data files every hour. The Oracle data load rate is 1 million records per minute.

It is a ridiculously complex system doing a few ridiculously simple things. Why so complex: because of the vast configurability requirement. Data comes in many formats, SNMP, BULK, SQL, CORBA, SOAP, etc. SNMP is the most common. Each node has a Managed Information Base which has a schema that is a simple rooted hierarchy of data items. To get an item, you must supply the full hierarchic address. SNMP messages are very simple (get/set request, get response, getNext for table enumeration, trap for message initiated by managed object, e.g. an alarm).

Quallaby's tools must support new network devices, applications and metrics quickly. Their customers demand that it be very configurable. Their customers are users, not programmers. Hence they need a scripting language and are converging all those they use onto ECMAScript (JavaScript standardised by ECMA).

Their Smalltalk ECMAScript engine (Smee) is in the Cincom Open Repository. It uses the SmaCC parser and compiles to Smalltalk bytecodes. There is VW debugger support and 300+ test cases. A Smalltalk object can be exposed as a Smee object.

Their SNMP collector does network discovery. It find all the nodes by various ad-hoc methods and then finds out which have SNMP agents. Next it finds what managed objects (resources) there are on each agent, i.e. what Virtual Private Networks, circuits, interfaces, etc. This network discovery process can take hours to run. Then it does resource discovery for each resource it found (done infrequently, takes much work). Data collection runs every 5 - 15 minutes. It polls a very large number of resources (i.e. rows in Managed Information Base) to find the current state of each.

They try to optimise collecting this vast amount of data. You want to minimise network traffic. GetBulk usually looks best but if you are searching for disabled devices sparsely spread through a table then using an index to get the data is better. Users hated their old procedural language for specifying what data to collect; they wanted to give users an easy declarative way of saying what data they wanted, then compute the best plan to retrieve data. This sounds very like relational algebra: declare what data is wanted from various tables, then optimise the retrieval plan. Thus they created Smee-SQL. There is tight integration of Smalltalk, Smee and SQL. You can expose Smalltalk objects (e.g. statistics) to Smee-SQL.

They have to collect 15 formulae every 5 minutes against 50,000 resources. The optimisation strategy is to build the whole query set of objects, then start grouping. Almost every actual resource value is requested in more than one query, often in many, and to marshal is a cost, so they pre-marshal OIDs in requests and compare marshalled responses to not unmarshal.

Smalltalk in an Autonomous Underwater Vehicle, Jon Hylands
Jon machines some of the hardware for this, not just the software. His prototype is about the length of a forearm and consists of a PDA connected to other electronics all in a tube with a propeller and some steerable fins. The PDA runs Squeak and uses a standard RS232 to get data from and send commands to the hardware.

An AUV has no physical tether; lots of people use tethers and it greatly limits mobility. AUVs are used for scientific research, environmental modelling, surveys, cable laying, military uses, etc.

These things are hard to build and operate. Mars Rover is inaccessible but we can still talk to it. Radio stops working two feet underwater. Light is not good as visibility is often limited and also unreliable. GPS, to see where you are, is not available. The only thing that works is sound. However, under water, an acoustic modem has a very low baud rate and is unreliable. Acoustic modems are expensive (e.g. $20,000).

Underwater is a very nasty place to be. Water gets in everywhere. Salt water is very corrosive. Pressure increases linearly with depth. Expense increases faster than that with how deep you plan to go.

Bluefin Robotics (MIT spin-off) are one of the few companies that build AUVs in the world. Theirs needs a 300-foot ship to support it because the 10-12 foot AUV weighs 3-4000lbs and you need a crane that can lift that (so a large ship), plus repair facilities (and of course if you have a tethered item as well you need a huge drum for the tether).

ISE in Vancouver is another company that makes very large cable-laying AUV called Theseus (so called from the string Ariadne gave him in the myth). Konsberg Simrad (Germany) also make a very expensive survey AUV. Hydroid make a small configurable AUV (Remus). It can only go down to 100 feet but it only costs $250,000, a tenth of what most of the others cost; the navy like them and use them to remove mines (they use dolphins as well but dolphins are not that predictable and they don't scale; ten dolphins are much harder to handle and need a lot of joint training).

MicroSeeker aims to be clever enough not to need an acoustic modem and much cheaper. They did a pool test in Hamilton a fortnight ago (Jon showed the video) to verify that it was waterproof and slightly positively buoyant in the right orientation (which it was) and that it was fully self-directing (which it wasn't: will be when the motor generates sufficient torque). He needs to use a gear motor to produce sufficient torque at the right RPM. That requires battery changes and some electronics work.

The autonomous underwater vehicle is autonomous because it has software, in 32Mb of Ram and 32 Mb of flash. The autonomous controller, written in Squeak, decides where it goes and what it does. It has sets of layered finite state machines.

- mission layer: PoolStraightLineTestMission is a simple time-based mission

- navigation layer: this is a plugin; one changes navigators frequently during a mission. Navigators are waypoint-oriented, time-based, etc. navigators make decisions about how to get between waypoints, whether a sensor is detecting a target (e.g. a pinging target).

- Sensor layer: Sensors tell speed (from a paddle-wheel), heading pitch and roll (using a Honeywell compass), depth (water pressure; he plans to upgrade this), bottom sonar (electronics for this but transducers still TBD), leak detector (an aspirin; if it dissolves, we have a leak).

- Actuators, etc., can be bought from any hobby store.

He showed his simulator indicating the AUV's behaviour in a scenario with him adding some random behaviour (waves, etc.). Squeak was very good for creating these visual simulators; attitude screen, map screen, etc.

This summer he hopes to fix the hardware issues and complete the controller he is writing. Then he hopes to progress his next version of the design, much influenced by what he has learned from this one. He plans to move to an event and hierarchic mission controller design. He will need $60,000 funding to build this next version.

Currently, all the AUV companies roll their own software, the usual situation in a new field. He hopes to show better software.

Frameworks

Keynote: Smalltalk/V and .NET: A Comparison of Virtual Machines, George Bosworth, Microsoft

George worked at Digitalk. Since then he's been a refugee at Microsoft, but will always be a Smalltalk fanatic whatever he is working on at the time. He is one of dozens of architects who worked on the .Net Common Language Runtime. These dozens of architects had a wide variety of and past experience and viewpoint. George's personal view is what follows.

(N.B. George often gets acronyms and code names wrong - as do I - and the CLR has had lots of them: CLR, COM+, Project 42, Lightening, ...)

A commercial platform for languages and a commercial language toolset are not the same. The CLR is the former, Smalltalk/V the latter. The former enables divergence and differentiation: enabling that productively is the former's value proposition. The latter exploits uniformity by providing specific patterns for doing things so users can get on with higher-level thinking. Providing productive patterns is the latter's value proposition.

Smalltalk/V was a VM plus libraries, components and tools. The CLR is an execution engine, i.e. a VM. The VM provides a few more services than Smalltalk/V offered all those years ago (e.g. security, interoperability) but the main lists are very similar. The CLR also has base class libraries.

Unlike V, the CLR has CLS language interoperation rules and conventions. You define a subset so that, while everything C++ can do cannot be accessed for Eiffel, say, everything in the agreed subset can be exchanged. The CLR also has piles of specs, unlike V. It does not have lots of tools,

because there are lots of other products from Microsoft and others (VisualStudio and so on). Smalltalk/V tried to provide the tools out of the box whereas the CLR aims to have that provided by others.

It is not that one approach is better. These are different things giving rise to different views. The CLR builds things that run *on* it, so cares about hosting, environment and deep interoperability. V built things that ran *in* it. Some things, e.g. the garbage collector, differ for reasons of architects' different ideas, but mainly they differ because of these different intentions.

The basic building blocks look very similar, how things are loaded, how we track memory, how we produce executable code, how we provide services (including security but that service gets its tendrils into many places and must be thought about up front). There is more than one implementation of the CLI from MS:

- Desktop/Server (VS.Net. LongHorn)

- Rotor (open-sourced version)

- Compact f/w (PDAs, phones)

- the watch (very compact, for e.g. MSN-Direct-enabled wristwatches)

These implementations are very different in how they are built, though they share a few concepts, but they all are trying to implement the CLI and are programmable from VisualStudio. The rest of this talk is mostly about Desktop/Server.

The fundamental runtime control flow and execution model viewed abstractly of course look very reminiscent of Smalltalk. They have various JITs. D/S uses the standard JIT, Rotor uses econo JIT - because George wrote it and it is easier to understand, though not as fast :-). PreJit code generation is done at install time. Many think of these JITs as a compiler but that viewpoint leads you to all the wrong conclusions

An Assembly is the unit by which code is packaged and loaded. They have meta-data (the 'manifest'). George defines metadata as everything that isn't the instructions; CLR has *lots* of it (twice as much metadata as code in some files). Compiling captures versioning (I compiled this one today, that one yesterday, aims to help tracking dependencies.) When you find that a new version of an assembly does not work with one specific other assembly, versioning aims to be able to track that.

Security permissions are held by assemblies; methods can demand proof that permission is granted to the entire call chain (to enable, e.g. 'run this method from that one but only in context of this app'). Types are also named relative to an assembly; assemblies act as namespaces.

The compiler resolves all this: resolves method overloading using language-specific rules, adds casts needed, binds. It produces IL which can bind physically but usually it binds logically to virtual slots and methods, not direct offsets; this is much more resilient. The compiler emits meta-

data merged with the metadata it received and it emits IL code with meta-data tokens in it which will be just-in-time resolved to laid-out Vtables, etc. V usually bound more physically and invalidated on shape changes.

When a method is called for the first time, the code is JITted (and verified if needed). The tricky part is deciding when to resolve, not how. If a member ref appears, the type is loaded first, which will also load the classes it inherits from, find the member, do security checks (JITter may insert calls to security checks), and so eventually you call the code.

George then showed some slides to express this visually ("We have people at MS who are very good at producing slides"). You generate a PE file with header, tons of metadata, exception handling table, etc., and then send it to checking tools (e.g. PEVerify, which tries to tells whether a PE file is well formed, or NGEN) or to something that will actually run it (e.g. GAC, etc.). The CLR is written to be hosted inside a user application so it has a policy manager to apply policies to the code you would like to run. If permission is granted, the class loader builds a Vtable and class info for the code, which is then JITted and verified, thus reaching the state of being native code plus a GC table.

A PreJIT is not (or not usefully) to be thought of as an MSIL-to-Native Compiler. The PreJIT does pre-compile, pre-load and pre-layout that may or may not be able to run on the host. You validate, and then run if you can or revert to the normal pass. The PreJIT creates a special PE file (.exe or .dll) associated back to the IL assembly. It works by pretending to be going to run it (i.e. do PolicyManager stuff, JIT code) then caching it. Later you load it and do verification checks and then either fix-up ('not loaded at address I thought so apply offsets') or fail to fix-up (use normal pass).

The purpose of this is not to save JIT time (not a bottleneck) but to reduce the number of pages you have to fetch, which is the killer for startup time for apps. Thus very dense usage of the initial pages loaded, with whatever they do not need able to be left till later, is a key value. Also, if 1000 people login to a server and each start a CLR, that will be very slow unless most of the CLR pages can be shared across processes. This PreJIT aims to make it possible to identify truly shareable pages. This kind of thing is hard and Smalltalk/V was not looking at cross-process page sharing (I note that GemStone certainly has strategies for it today; other Smalltalks?)

Deployment models are a very big issue for MS since they do just occasionally (!!! :-) patch and they also ship upgrades (which may be dangerous as they cause use of product in new ways). Imagine getting a Watson-dump where you just have native memory with no data on where it comes from, what version, etc. Nirvana is *not* client-side generation of optimised code when the scale gets large. If you ever go down that path in Smalltalk think about that. George gets lots of Watson-dumps.

An application domain is a subprocess that holds a type hierarchy, separate from every other (logically separate; actually shares common types), where assemblies are loaded. Transparent proxies (which you may think of as in a global domain or in no domain as you wish) allow communication across Application Domains while maintaining referential integrity of memory.

Suppose you use a window handle in a way that is 'behaviour undefined' but works, and then later an upgrade means it no longer works. MS would try to say 'the application is broken' but some people (looking at Eliot :-) never accept this (Eliot causes several, being more inventive than most).

Some things have to be resilient across time; get the latest version, please. Some things are expected to be different across time; load the old stuff at this ref with this app. The need to manage both cases creates the demand for Application Domains.

Hosts want to apply policies. A server might not let you schedule threads. If the thread has a problem, what do you do: terminate process, hang, ...; the host's policy decides. Out of memory: what is your handler allowed to do? Can anyone ever allocate anything ever again? Policy is influenced by how common the fault is. Some servers uses Out-Of-Memory as the way to tune cache size: seeing more and more OOM, shrink cache; not seen one for a while, grow cache. Others are much less casual about it. The CLR is trying to live inside a range of hosts who have very different world views.

The two languages that had the biggest impact on CLR were VB and C++ (N.B. *not* C#, which is new and so tends to look at the CLR and surface what the CLR has, not do its own thing). C++ had lots of existing stuff. In general,

- Tools offer depth and inflict product churn.

- Libraries offer breadth and inflict versioning hell.

- Implementations offer guarantees and invariants, and inflict bug hell.

- Platforms offer real or de facto standards and inflict difficult-to-fix bugs/features.

An example: C++ has consts, MSIL has no const. The issue was what does const mean; an absolute guarantee or a usage convention that users can cast away? The latter seemed to be C++ view. The CLR is very concerned about absolute guarantees that always hold. Languages by contrast are concerned about patterns that warn when you step outside but do not prevent you.

Another example: strings that are read-only / immutable / share by ref.

- Can I change the string from my program? No.

- Will its physical bits ever change? Probably yes; e.g. you might change the locks on the object, or other tag bits. Some Unicode strings are weird but most are not, so it can help to have a bit to say, 'I've checked this one and it's not weird (so use the fast methods not the general ones)'. This bit is not writable by the programmer but it is writable by the underlying system.

So what was it that you wanted; programmer immutability or bit-identical sharability.

Q. Differences between Rotor and D/S? Rotor lacks lots of the specialised allocation helpers. The infrastructure for it is there so you could write any you need. Rotor is close enough to D/S that MS often tells customers. Look at Rotor, it will help you understand.

Q. Smalltalk's in-memory stability of objects is one of its strengths; we can get in there and look at them. This needs snapshotting of either an image or of object graphs. How does the CLR feel about this? The CLR languages do not today have a means of providing an external form of objects. They will be looking at this and it could be made to work, better than people think. However pure BOSS is brittle with respect to versioning so you must provide conversion and so on. It can be done (Smalltalk/V could export a BOSS file from 16 bit and load it into 32 bit, etc.). XAML is much less powerful than Smalltalk representation but is easier to give to a non-Smalltalk tool. The issue is the timeline: for how long and over how many different cases must this file be reusable.

(Dave Simmons) The issue is that in CLR you have to (have framework to) take responsibility for deciding what you serialize out and take back, unlike Smalltalk which snapshotted everything, thus avoiding the question. It can be done and the benefit is that you can merge two images. (Avi) It can be done half-way through running a computation, e.g. with a debugger up? (Dave) It could be done.

Q.(James Robertson), I fix my web app by testing locally, then just shove parcels (with shape changes) to the server and load them. What can the CLR do? In asp.net, it uses the statelessness of web apps to redo the page without problems. The state is held in a state store and if you want to change that you must handle it. (Jim) Change classes while running? Change layouts of existing instances? 'Edit and Continue' is due to be built to let you do some of that but it will be much less functional and robust than Smalltalk.

Q.(Heeg) What must Smalltalk do to expand? Work with the CLR, to leverage others' work. CLR does not give you all you want, alas, to run on it. (Eliot) Smalltalk is like this really successful pond scum that floats on everything. (I assume Eliot was speaking of the notorious Canadian pond weed, Elodea Canadiensis, perhaps confirming Alan's suggestion of successful things being Canadian - not the analogy I would have chosen. :-)

Q. What is missing from the CLR? The CLR does not have a tag type that goes right through it and that the tool chain does not fall over on. The anonymous reference stuff that is due to go in will help people implement Smalltalk on the CLR. Intermediate late binding is what Smalltalk wants and the CLR does not support th p.9(a)5but could.

Q. Need to work with languages that do expect (interface) things that Smalltalk does not? You can say what the set of fixed required things are (Smalltalk requires that DNU be always there). One part we depend upon, one part is dynamically looked for. For static CLR languages this pairing is (all, nothing). By defining some new ways of resolving we can do this.

Q. CLR assumes that in extending you will always be supporting reference types and value types; this has been a challenge for several languages and especially Smalltalk? It should be possible to describe at the boundary what the transformations are that you need. The same occurs in Python. If you are happy that noone can call this function from outside your language, you need do nothing, otherwise you must provide transformation meta-data.

Q. Do you need to make Smalltalk type-capable to be a first-class player in CLR? You must make it type-aware at the boundary via some marshalling approach (as has sometimes been done elsewhere in Smalltalk interop). Going further is up to you; whatever you want people outside Smalltalk-on-CLR to be able to call, you must marshal.

Q.(Eliot) Smalltalk/V knew the KISS principle whereas CLR has forgotten it. Eliot feels that MS has made a hell of its own creation and must live in that hell. If George had been at MS earlier, what would the CLR be like today? Operating Systems that are widely used by many people give you problems we could escape in ST/V. If you want to innovate, you don't want to be too successful. George almost was at MS earlier but he doubts that would have changed the CLR. The watch and the compact are built quite unlike the D/S. His Smalltalk-innovation hat says, "I'd like to get away with a few tricks", but his Watson-report-handling hat reminds him that half of them arise from third-party device drivers; you have to think hard about how users will abuse and misunderstand what you build. George is a Smalltalk fanatic so there are things he feels bad about. He would have loved to see an Smalltalk system on the CLR the day it was released but that would have needed certain things to be there. The art of product release lies in deciding what not to ship. Two-thirds of what George architected is not yet in the CLR. If one of those thirds were in there, the CLR would not yet have shipped. George believes that it was better that they shipped it.

The problems of adding things are illustrated by Generics (which are being added now). It is not hard to add Generics in the core code but how do you show a generic to a VB programmer. VisualStudio must have an answer to that. Until it does, you can add it but not use it. If you add it, the developer of the collection hierarchy is going to be very annoyed they cannot use it.

Introduction to State Replication Protocol (SRP), Paul Baumann
The choice between passing objects as binary or as XML is like the choice between seeing objects in an inspector or as text. XML has high portability of basic data types and can port higher-order types, but is inefficient in time and even more in space; binary has the reverse advantages and drawbacks. SRP seeks the best of both worlds.

SRP has been ported to Dolphin, GemStone, ObjectStudio, Smalltalk/X, Squeak, VisualAge, VSE and VisualWorks. It works on all these dialects from a single code base (held in Envy/VA, exported as parcels or whatever else is used by the other dialects).

Everything in SRP is a stream of unsigned integers (can be represented as ByteArrays or Strings). Simple parsing rules serialize and deserialise. SRP is architected in layers, each calling the next

- PlbStateReplicationProtocol: basic behaviour layer

- SrpPortableObjectsportabilityLayer)

- InterfaceLayers: a configuration provides an ObjectStream (the stream of objects to send). A Marshaller serializes and deserialises, using whatever StreamEncoding is required (reverse bit order, base 64, ...)

The configuration allows great flexibility in setting mapping rules, instantiation rules, etc. Convenience methods make it easy to control where you save (to a file, to a stream), etc.

A portal is a global variable that holds a dialect-defining value. Messages sent to it are dispatched to the dialect-specific method for that behaviour. Each subclass of SrpPortableBehavior can apply to one or more dialects. Their rules use a lot of respondsTo:, class name resolution, etc., to determine which behaviours apply to an image (only during initialisation, so the slowness is acceptable).

Paul launched a Dolphin image and showed a configuration that held some simple data types, references to the class, etc. He then showed a complex object (a method parse tree) for which he did not have the class defined in his image. He showed us what it looked like as SRP binary data (!!!), and how SRP handled loading its meta-state, the object that allowed us to load it despite lacking some of the classes it needed. The meta-state provides instvar names and default accessor behaviour so that, for example, a message sent to the object that only needed simple accessor behaviour of some missing class through which the message chain passed could work; mainly however, the meta-state reconciles the object to its classes on the two sides, and secondarily ensures the object can always be imported regardless of the state of the classes it needs.

After loading and saving various objects in Dolphin, Paul launched VA. In VA, the class absent from Dolphin was present so when he inspected the SRP form of the object it could print itself more meaningfully (by using the class' behaviour instead of having to rely on the meta-state's behaviour).

Paul's complex object was a Visual Age method which was long and used many datatypes. He dumped it as a text-encoded SRP and then in binary and compared the sizes (binary smaller). By using a meta-state table he can make it smaller still (no need to store meta-state with object) subject of course to having sent (or synched) the meta-state table between sender and receiver. Paul can use a general meta-state table or one customised to a particular object. The latter gives the greatest size efficiency; objects save in a very small footprint.

In VW, #become: is fast but in VA it is slower, so Paul does not use #become: in any dialect. Instead, SRP assigns a placeholder to indirect references precisely at those points where become-like behaviour may be needed later in the import of a complex object.

SRP has all the flexibility of Paul's earlier deep copy framework (and of course SRP can be used simply to deep copy). You can control serialisation and deserialisation behaviour very easily at the instance, class or inheritance level, map selected instVars to nil, etc. He used to have direct saving of binary to streams but removed it as such objects were not universally portable, conflicting with SRP's philosophy. Now it is gone SRP will always save objects with a behaviour that can reconstruct them.

There are many standard mappings, e.g

- (VA) EsString -> portable rep (SRP string) -> VW String

There are several ways to map a string; rules determine which is chosen preferentially. Objects can have postLoadActions that, for example, can use placeHolders to exchange objects.

Q. (Heeg) Applications? SRP has existed for 8 years. Some people have told Paul they use it.

Q. Comparison to BOSS? BOSS writes directly to a stream, so if the behaviour changes, you cannot load objects. Because BOSS loading uses `perform:`, data could e.g. send System exit, creating a security hole. SRP has no `perform:` so no uncontrolled execution of behaviour in import. SRP is very space efficient and faster than some raw binaries, slower than others. A object that is 10k in VA Swapper becomes 1.6 K in SRP and is handled at about the same speed. VW is faster than SRP.

Q. What rules are available? Class based, inheritance based, instance-based loading rules are all available. SRP version 3 also let you have class version number storing.

SRP is open source. Paul handed out SRP3.0 CDs.

Visibility-based Report Framework, Bob Nemec
Bob works for a hedge-fund trading system in Toronto. Their application has to handle a very large scale and must produce many reports. They needed the ability to configure reports quickly. They also wanted good diagnostic tools so if a report failed they would not have to start a full development image for a minor issue. They knew that doing everything in Smalltalk would give them access and flexibility. They decided to use their tools on top of Visibility, a Totally Objects' tool that lets you say, for example, that you want character X in location Y in style Z. (Bob regretted that David Pennington of Totally Objects was not able to be at this StS.)

Diagnostics: if something looks wrong in the final printed report, they want to see where along the path of creation it went wrong. The diagnostics show this (as hierarchies, colour-coded for stage, to show how something was constructed). This was very useful in development, but is little used now as the system is stable and reliable. Another diagnostic is a trace tool (who sent which method with what attributes). A third is a very verbose output trace that lets you see the context for why, e.g. you are going on to the next page unexpectedly; again, useful when developing the tool rather than now. They can also open explorers and inspectors on the final object in a report and drill down to what supplied it.

They think the 100% Smalltalk solution was useful, especially in development for diagnostics and refactoring. Diagnosing why something on a page is wrong when using external systems is horrible.

A sequence of students did the work with Bob refactoring the code between each student, then briefing the next student on where the tool was and where he wanted it to go.

Q. Size? They have 250,000 objects in their GemStone DB. Their hedge fund manages hundreds of active and thousands of inactive funds. Monthly, they take 6 minutes to generate a 200-page report. When it was done in Excel and external tools, people would be working 18-hour days as the monthly management meeting approached to pull the report together. Some reports are generated every day. A great many reports are viewed on line (every report *can* be viewed on line; this is doctrine).

Q. Any of this done on GemStone? Most is done on GemStone. (They have default font style and size tags that are enough for almost all reports.)

Q. Printing? They print to PDF because occasionally other fonts look odd. This is a known issue of VA assigning to printer fonts and/or screen fonts so that occasionally a given pixel size goes 'odd'. Their only issue with PDF is that they would like to set the output file beforehand, not be prompted during (PDF is just another device to Visibility). David is providing this ability. (Someone mention PDFFactory which will do this.)

Q. Report-oriented or data oriented? We provide both; some users are used to extracting a date-range from a larger report instead of selecting that date-range when requesting a report.

Making relational data first class, Avi Bryant
(Missed this but got a summary later.) Avi has created classes for relational algebra. You do your relational algebra work in Smalltalk, creating a network of objects (instances of these classes), then transform via visitor pattern and generate the SQL at last moment. You can pass a Smalltalk object representing your query around so that one routine narrows the search one way and another narrows it in another. You can defer your query till the latest possible moment and manage its relation to places where its result is wanted optimally. Avi sends a message accumulator into a block (the old TopLink trick) so does selects in a Smalltalky way.

Pollock: Into The Breach, Sames Shuster

Sames explained that the title referred to a bullet being in the breach (i.e. about to shoot, i.e. Pollock is about to be released) and was not a Shakespearean reference ('Once more into the breach, dear friends, or fill the wall up with our English dead.') as I had at first feared :-).

Pollock steals from VB, Delphi, Clipper, PowerBuilder, ObjectWindows, and all the Smalltalk dialects: VW, VA, Dolphin, ST/X, VSE, etc. Sames has invented nothing new in Pollock.

Sames' recruitment included an interview by Glen Krasner (author of 'Smalltalk: Bits of History, Words of Advice') circa VW5i.1 time. Asked what he would change, Sames said, "The GUI; it sucks.", adding that MVC was stupid. Then he realised Glen co-wrote the original MVC paper. A week later, he was hired and leading the GUI rewrite project. He had no prior experience of building GUI frameworks, but much of using them and understanding them (he always likes to know things from top to bottom).

Pollock could be an extremely disruptive technology if you're using VisualWorks. The aim of the talk is to reassure users that it will not be so disruptive and to make people aware of what disruption there will be. Pollock will go into production circa third quarter this year and should be out before end of year (but note that its release dates are disconnected from the November VW7.3 CD printing). As of now, widgets, XML and literal arrays, and code building frameworks are all done. Motif and MacOSX looks were finished on 27th April. Supporting all the features of the current framework is 80% done.

Needing a name to distinguish it from Pollock, Sames has christened the current framework Wrapper, because it has so many.

Pollock has 13,992 unit tests with 300,000 assertions. (I meant to ask Sames for a breakdown of these but was distracted. Many are standard behaviour tests. Some are public framework class name tests; since users will quote public framework classes in their code, the tests remind future maintainers not to rename them casually. And some reflect Sames philosophy of testing absolutely everything. It would be interesting to have the rough ratios.)

What's left?

- Some small stuff: matching Pollock behaviour for the last 20% of Wrapper feature, plus the promise list ("Will Pollock do X?"; "Oh, of course.", Sames has said to (too?) many people at earlier StSs) and the bugs.

- Some medium stuff: the API has evolved over the last two years of coding and so some old widgets must be brought up to the final state. Internationalisation support must be plugged into Pollock (straightforward and not too long).

- Some big stuff, including the following

- No wrappers!!!

- A future. Wrapper has no future. At production release 3, wrapper will not even be in the image by default. It will load and run fine but it will be all yours. We will look at bugs if you pay us and not otherwise.

Pollock will have 3 production releases, plus quality intermediate releases will be dropped to the Cincom Open Repository and announced.

Pollock will have very configurable Grid widgets (per row, column, cell, can put Grid inside of Grid, TreeView column, selection/edit-changed views). Text will be much more powerful. All widgets will have powerful API: no need for aspect adaptors. TabControl has optional tabs; someday you will be thrilled by that. And much more (see the list on Sames' slide).

Q. Drag-drop integration with native platforms? That is scheduled for the Chagall project, TBD during next two years.

The production 2 release will provide conversion tools to convert wrapper specs to (XML or Literal array as you wish) Pollock specs, and wrapper menus to Pollock menus or toolbars as you wish. It will convert ApplicationModel code to Pollock UserInterface code. The expectation is that 80% of code will just auto-convert and run. Another 16% will need the developer to choose a classifying pattern from a small list, after which it will auto-convert and run. The remaining 4% will require manual intervention. Production 2 will also have business graphics kit.

Chagall is the GUI project after Pollock. It plans to do native widgets, including various expanded sets: GTK Gnome, QT (KDE), ... see slides.

After Chagall, the Peaches project will move the gra.2(r)10phice (ing-4.2(c)na c(r)10ph)--

Q. Why the name 'Pollock'? ParcPlace had project called 'Van Gogh' for GUI (the idea presumably being that it would let you build GUIs as powerful as his paintings). For the new GUI project, Sames wanted to build a more modern abstraction, and there are few artists more modern and abstract than Jackson Pollock (the fewer the better, say I :-).

The withStyle team have been working with Pollock and giving feedback.

Tools, Testing and Process

Opentalk, Load Balancing and Multi-Image SUnits, Leonard Lutomski, Cincom

Load-balancing is a deceptively simple problem. Leonard has seen people make horrible mistakes. He hopes he now know knows how to do it. He has a multi-image SUnit facility. He also has a load balancing facility in OpenTalk that he thinks shows a good pattern (but welcomes comment).

There are only 6 rational ways he knows of doing load balancing. (Anyone know a seventh?) Configuration is part of the load balancing problem; nothing gets done without it.

His demo showed multiple images running on one host, whereas usually one looks at multiple images running on multiple hosts. However his experience and theory is that OS time-slicing does not change the relative efficiency of compared load-balancing strategies. His demo omits examples of the balancer itself failing / fail-over (not enough time in 45 minutes; he had more slides than minutes as it was because he decided to make a slidepack that was a good introduction to load balancing).

His demo has two kinds of request streams

- a stream with 60 17ms-to-service requests, then 60 173ms-to-service requests, then 60 11709ms-to-service requests (all primes, he likes primes)

- a stream guaranteed to frustrate round-robin balancers (or any sequential distribution scheme)

Start-up and shutdown is complicated and takes time as much configuration is going on.

He then ran the first scheme against a simple sequential strategy and the graph showed a load balance we would like; servers fairly evenly loaded. he measured the total time clients waited for requests to be serviced (455secs), the total time to service requests (152.844secs), and the total server-side wait. The last two measures indicate how efficient the strategy is (the ratio is the real issue but for a given test set, the absolute numbers are valid and magnify differences, making them easier to spot).

He then asked us to guess whether rival strategies random and least-loaded would be better or worse than sequential, then ran them. Random was worse, as you would expect, since it will sometimes put three requests on the same server, creating back-up.

Least-loaded sounds like it should be better but is not, partly because the data it gets about server loads is always out-of-date but mainly because the data is refreshed at a periodic rate (with some variance) e.g. 20ms. If you get 30 requests in that 20ms, they all go to the least loaded server. In short, it is vulnerable to burstiness (request rate exceeds server-state refresh rate).

If you have a simple request stream you want a simple strategy. Cincom had a customer who requested least loaded, which took much work, then found they had to revert to round robin for just this reason.

Then he ran his 'tough on sequential balancers' stream with the sequential balancer and it did not look good. Random looked much better, least-loaded better still. He then ran a time-sequential strategy (ratio between average request service length and time interval was 9:1) which was even better. (Not much but the least-loaded run was particularly good. All this was one run; in real life, he would run a dozen and discard two outliers.) The diabolical request for time-sequential balancers is a very long request.

Random-least-loaded-half (randomly select server from less loaded half of servers) usually does better on the tough stream than least-loaded but not by much (and did worse in this run). Random-under-mean-line is similar but randomly picks server from those loaded less than the mean.

Load-balancing is for synchronous connection-oriented protocols. Asynchronous or multi-cast protocols, e.g. that used in the ISIS shipping system, use a root game. Some routers implement load-balancing in hardware; it is faster but less flexible than in software. Load balancing minimises the time a client spends waiting in a sever's request queue; it cannot minimise anything else. It has a cost of its own: it takes time to redirect requests, the servers and balancer must use the network to communicate, the server image must take time to do administration tasks for the balancer, and the balancer image must also run on some CPU.

The OpenTalk load-balancer is in VW7.3 in four parcels and is commented (unlike much code Cincom puts out :-). It has all you need to implement any of 6 basic load-balancing architectures. It has an API for configuration. It has some multi-balancer support (you do not want your balancer to be a single point of failure) which may be further developed. Time was short so Leonard skipped many slides describing the OpenTalk L-B's components.

The basic load-balancer architectures assume that all servers can handle a given request in the same time. Do *not* use an architecture with sessions or transactions in *any* case where you expect *any* maintenance of shared state. He gave an example of an object reference routed by the load-balancer to a lightly-loaded ORB but it is not the ORB that exported that object so the lightly-loaded ORB will not be fast handling it.

He started to talk of architectures. Client code must not see whether load balancing is present or absent, so it always references a wrapper object.

In the 'balancer always, no loads' strategy, the wrapper always goes to the balancer. The balancer has a static distribution policy (so no server reporting of state). It is simple, fine-grained, cheap in messaging overhead but costly in redirection overhead. It is not good for sessions or transactions, but is good for a high average base request arrival time.

Time ran out. His slides list each of the 6 strategies, with its costs and benefits, and its scenarios of use.

Pragmatic Enterprise Software Delivery, Angus MacArthur and Sean Morrison

This talk is about problems they've had delivering software in a timely manner, and processes that have helped them with that.

OTTP is the pension fund for all public teachers in Ontario (private teachers must contribute to it by law). The development environment is Smalltalk and Java. The Java is for some web developments. Their client-server desktop app is written in VisualWorks. When the web came along, the ease of refocusing Smalltalk meant the same Smalltalk code powered a web app that lets plan members inspect and change their data. However other web apps in the company are written in Java and WebSphere. Their system has 173 packages, 2360 classes, 57,000 methods, 285,000 LoC.

Of 500 people in the company, 50 people are pension managers and they all use the desktop application. Their users usually sit within a few hundred feet of them. Their process is to gather requirements, then do SUnit test-driven development. This SUnit-associated process is very quick. Next they release to test which means moving between departments. If a tester or customer reveals a problem, that takes longer to fix than if an SUnit test reveals a problem.

The developers and QA use different tools. The QA people are business people with some computer knowledge, not developers. They, like end users, do not like seeing walk-backs. By contrast, a developer is happiest when working with his computer, not with QA or customers. Bugs interfere with ongoing work. A bug is usually a high-priority issue he must address. So the less time it takes him to fix, to deploy patches to the floor, etc., the more ongoing work he can do.

Whenever a problem is generated in testing or from the floor, there are logs and forms and process to be done and that takes time. So they are augmenting the tools that come with VisualWorks to help them do these process things by building tools for testing, building and deployment.

Testing Tools: One testing tools is SUnit. They are not an XP shop as such (yet) but they find it valuable and growing more so. They also have BRITE, an in-house testing tool (a poor-man's TestRunner). BRITE runs test at a much higher level of granularity and, like SUnit, converts walk-backs to failure reports. The testing group have 100,000 expected-result tuples which their BRITE tests confirm (or not). BRITE is open source, in the Cincom public repository and you are welcome to use it. It files into 5i4 but

will not work out of the box. It needs a database to store results and is tied to their schema. It is also somewhat windows-specific. The licence is GPL but is being made BSD. (It now runs in 7.2 in their environment so will be released as such, and more schema-agnostic, soonish.)

Building and Deployment tools: originally they had a single deliverable of a desktop client application in Envy. A build was a monolithic image file. Change management was well-handled in Envy. Deployment was simple but error-prone, since the process was to have a developer with instructions on a scrap of paper load all the code and run scripts. When they deployed a broken build to the testing team, that would halt their progress for a day.

Currently, things are more complex. They have a batch processing framework that distributes tasks to machines and uses a server farm, a rack of machines that hold Smalltalk images supporting websphere. The result is that a build has at least one Smalltalk image plus web pages, .ini config files, dlls and Java code. These files change as rapidly as the code itself.

They built two tools to help.

- Autobuilder: this is an extension to the runtime packager that scripts the build assembly and then tests it to ensure you are not distributing a broken build. A command line switch tells VW to use an XML file to load specific bundles, packages, parcels, or by blessing level. Images are saved according to a predefined script or by external .st file. Load errors and warnings are logged and a build manifest created (they are working on a hall of shame for undeclared variables on load, etc.). An ANT script invokes the process. Typically it is run once or twice a day.

- Configuration server: this puts all the supporting files under version management. The configuration server holds the files and serves them to the build. Previously they broke many builds by deploying the wrong set of .ini files (which are lists of key-value pairs). Now they use CVS-controlled XML files (fewer namespace-collision problems plus allows more complex data if needed). As websphere is now the department's standard for web development, the front end is websphere.

These tools were developed to meet their needs but may indicate more common requirements. In their environment today, Smalltalk is one of several players and so they must cross-reference across platforms.

Q. How do ANT and VW interact? Only by invoking a command line with appropriate switch data. They used ANT because they had ANT scripts, but they cannot call back to ANT. They write to a file which ANT then reads to see if the build is OK. It works but they are looking for a better solution.

Q. Timing? The QA dept is very good at keeping them in line. As they approach a release date, thinking they are feature-complete, they do twice-daily builds and the QA department gets these builds, runs tests, checks whether requirements are met and tells them what is missing or wrong.

Q. Where is the build info kept? In an XML file which specifies pundles by version or by blessing level (e.g. latest released).

Q. Who writes your tests? QA group says, "Need to test this". Developers converts their objects into strings and numbers. They do many of their 'what it should be' calculations by hand. Occasionally, developers can show that QA are wrong. They do not have an automated 'smoketest', partly for time and partly because some tests do not back out cleanly so automated running is not quite there yet though wanted.

Q.(Niall) A CM tool in Smalltalk, with external call ability, is better than an external tool precisely because it allows such add-ons. Some pure Smalltalk shops have created similar tools, except for the configuration manager, which sounds like a good idea that should be used more widely.

Inside the VisualWorks Tools, Vassili Bykov, Cincom
My write up of Vassili's talk 'The Secret Life of Tools' at last year's ESUG in Bled is relevant; it starts on page 19 of my ESUG 2003 report at: http://wiki.eranova.si/esug/DOWNLOAD/nfrESUG2003reportPublic.pdf.

I missed this talk. One addition since Bled is that, while most of the talk was done live in an image, as then, Vassili has added slides on how to use Pragmas, which are on the website and worth looking at if you need help.

Building a 'Test-Friendly' Application, James Foster
I missed this but James kindly re-presented it for me later.

"WinRunner works first time and is reliable", says Mercury Interactive; oh, yeah!!!. WinRunner sits between windows and your app, capturing and translating events. Playback must find the right window, widget, etc., to work. Their GUI map tracks widgets 'characteristics'. It is hard to identify windows and widgets uniquely, especially when they are custom widgets.

The testers can only do so much towards making a tool like WinRunner work; you must change things in your app to help the testers. Example custom widget containing widgets. James added a button menu; testers (and users if they wish) can use the menu to select the button and be sure that WinRunner will find it. Likewise a tab menu can drive tabs. (Users could want these if using VNC or whatever but his customer did not let him release them as they'd have to redo the documentation.)

Now, the testers record a script and it won't play back; they ask why? The script usually looks weird in widget selection when examined, suggesting they try using the menu to select. After a few such discoveries, testers started to recognise patterns. Example; in a many row list, clicking the 40th pixel down may well not get the same row as last time, so use 'select 40th row down. It's not exactly the same thing but the actual press of a button is almost always not what breaks.

Window labels can be non-unique and inconsistent. Mostly, we have the label; how to make it consistent and unique? James adds a class name to each label when in testing mode. Put a dot (.) in front of the non-latest referenced windows of same class to hide them. Before this, WinRunner's

pattern always found the first, i.,e. wrong, window. James' pattern likewise makes an assumption that could at some time prove wrong but it is usually true and the testers using it know how it works.

Widget positions are fragile and their names are non-unique. However Windows assigns a unique child identifier to all widgets so James just took that in Smalltalk and make sure WinRunner uses it. Before that, e.g. every widget had a help button and WinRunner was tending to select the wrong widget's button. James demoed a WinRunner script, made by point and click, some of it using his menus.

To sum up, given that WinRunner has a fixed and inaccurate view of what applications do, Smalltalk lets you tweak your application, either to fit WinRunner's expectations or to provide WinRunner with idiot-proof data.

Q. Why not TestMentor? TestMentor was less known than WinRunner and (not quite as unreasonably) management had a desire for testing to be done wholly outside of development, to avoid (probably unworthy) suspicions of `self assert: true`. Lastly, VSE does not have a TestMentor port (but SilverMark would have ported it for the WinRunner licence fee).

Squeak Tools, Colin Putney
Colin talked about the new generation of Squeak tools. The turning point was at OOPSLA 2002 (just down the road) when Squeak Central turned over Squeak maintenance to a new group. SqueakMap had just appeared. Colin showed the SqueakMap packageLoader. PackageInfo is Squeak's packaging system. It was a quick and dirty way of starting packaging. Then Ned, Avi and Colin started talking about how to manage development in packages. The outcome was Monticello, which is unlike most packaging systems. Monticello works on snapshots, so it does not watch you as you work. You tell it you are ready to package (c.f. Store as against Envy).

Monticello is version-centric. It does not keep careful track of the history of source code within a version. Thus it cannot show all prior versions of a method (see below). However it can merge very well. If two version streams have a common ancestor, you merge them by the union of one merge parent with the difference between the other merge parent and their common ancestor. The new version is the common ancestor of both parents (the rest is reminiscent of Store but Store can only hold a single parent).

By the properties of Set addition and differencing, this operation is commutative and transitive for merges without class-shape or method-source conflicts. Obviously, if there are conflicts, you must resolve them and that could break the commutativity and transitivity properties. It only handles code but could handle any flat structure (Smalltalk is set of class and method definitions, each of which either conflicts or not between versions). There are a lot of tests for the merge tool. It seems to handle cross-stream merges, repeated merges and multiple merges well.

Monticello can use various repositories, XMTP or HTTP server, directory, ftp site, GOODS database, or a SqueakMap cache or release (latter puts it on Squeak Foundation). Any individual package can have versions spread over lots of different repositories. That's why you don't get all prior versions of a method; you get prior versions that are in packages you know (and they could filter down to just the unique versions). You don't need a big central repository. Colin and Avi worked well for a time with only read access to each others' separate repositories.

(Usual demo hiccough; Colin had closed his image and reopened it but that does not reinitialise the packages in the browser so he got DNUs when he selected packages; he reinitialised.) Versions have UUID to distinguish them from all others, and metadata that tracks prior versions, etc. You send this data to end users so if they do any hacking in your package, you can easily merge what they send you.

Monticello is an optimistic, distributed, quite unusual but quite pleasant-to-use CM system. It is important to an open source community that people can work well when disconnected (on planes, etc.), as suits them and merge easily. Because repositories are so simple, you can start a project easily, save versions on your hard drive, then later move versions to a more general repository. You don't have to create infrastructure up-front for what may or may not later turn out to be a major project.

Then Colin described the new browser. When he started to rewrite the browser, he spent two months beating his head against morphic and then got on implementing functionality. Rewriting the browser will let many things progress much faster. The RB in Squeak is bad because it has to inherit from the browser. Namespace's main issue was not breaking the browser. Packages should be made visible in the browser. Colin showed his OmniBrowser acting as a package browser.

Q. I can put this into the Star Browser? Yes. Colin would like to steal classifications and other stuff from the Star Browser.

Then he showed a package-limited senders browser that let him see senders of senders of senders in a given context (here the context was a package, a particular kind of generic code context) to help him track down the eventual sender of a message and so on.

Q. Duplicates Refactoring Browser Environments (RB always displays an environment, which may be the whole image or a more limited code view)? Yes, may need to refactor to reuse RB.

Then he showed the versions browser. The file browser is work in progress; it is just the OmniBrowser retargeted to a different domain. It is currently a viewer, will be developed to drag-drop files, etc.

Colin has tried to abstract all the commonality that browsers use into a generic browser of a graph of nodes, which here is a tree of Smalltalk source but need not be either a tree or Smalltalk source. Node classes wrap

the actuals (which may be dumb, e.g. a category is just a string) and know how to handle them. A meta-graph describes how the browser should walk the graph. Colin talked through the meta-graph for the file system, just the ability to send `directories` and `files` to a directory node to return other nodes. You can attach actions to nodes (e.g. add class category). Thus you can extend the browser into a new domain with no UI work.

(There was discussion but I had to leave promptly so could not record it.)

Is Software Development in Developed Countries still affordable?, Georg Heeg

I only caught the end of this. Georg's answer was that yes it is and that the current enthusiasm for outsourcing underestimates its associated costs and risks. The middle section of Georg's talk at last year's ESUG in Bled is also relevant; my write-up of his talk starts on page 8 of my ESUG 2003 report: http://wiki.eranova.si/esug/DOWNLOAD/nfrESUG2003reportPublic.pdf.

Q. Statistics? Georg Heeg once competed with an Indian company for a contract. Heeg sort of won (neither got contract, Heeg got two consultants, India got none). He's heard of single offshore projects that failed but not of a proper study. Offshore staff cost one-quarter of onshore prime-site staff.

How small can Smalltalk be?

Smalltalk Mobilizes, Georg Heeg

Georg covered much of this in the last part of his talk at ESUG in Bled in the second half of 2003. See my write-up of his talk on page 8 of my report: http://wiki.eranova.si/esug/DOWNLOAD/nfrESUG2003reportPublic.pdf.

They will release this month. The mobile community is very keen on this. Even Gartner had a hard time being negative. One of their consultants wrote, "Smalltalk is the only technology that is the same on mobiles as on standard machines", which is a very positive point. Another Gartner consultant (one we know will always slate Smalltalk) said, "It doesn't matter what Cincom does, the market will not go for it.", which you could take as meaning "I can't actually find anything specific to gripe about."

Squat: A Minimal yet Extensible Smalltalk System, Craig Latta

Craig wanted to free himself from the intermingledness of all the subsystems you use to build Smalltalk systems, instead letting you choose the components you need. Snapshotting is essential but needs to work with much more fine-grained composability. He wants to simplify remote collaboration and behaviour distribution, not always sending one huge thing. Lastly he wants to enable portable embedded use in e.g. PDAs.

Craig started this in May 2002; Duane Maxwell, Tim Rowledge and Eric Arseneau had these common goals. The first task was an installer (small C program) for delivery of music and graphics. It was a tiny http program that got a VM for you that would do the work, called Relief (how Craig feels when Smalltalk starts up after which he can ignore all else on the platform).

He wanted people to visit a web page and have everything start up with no need for user work. Issue: is this a tracing (declarative) or simulation (imperative) problem? Do you already know what should be your minimal snapshot? Or do you e.g. run a very minimal simulator that complains whenever something is missing and you then put the bytes there and carry on (crazy initial idea they had)? They ended with two Smalltalks, one driving the target minimal VM.

Alejandro Rewimando's group traced several small snapshots (e.g. start, do 3+4, exit) but extension is difficult. (And their documentation is in Portuguese, which alas none of Craig's group knew.)

Craig felt they should start with Dan Nichol's 500k squeak image and strip. He invented a framework Other (as in 'self' and 'other') for remote message sending. Instances of Other forward messages sent to them, with help from the VM (not using DNU) which also protects Other from inheriting methods. This was easy to do; Craig showed the Slang (the mappable-to-C subset of Smalltalk that the Squeak VM is written in). They just added to the special objects table and did an extra comparison; the overhead is very manageable. They also serialized to another live Smalltalk system in real-time, which Craig noted as far easier than via BOSS or similar files, because being able to say things in the right order is simpler.

Craig stripped much of the browsers by putting them on the remote system, driven by remote message sends. It was easier to do this by using the MVC browsers instead of Squeak 3.2 Morphic browsers.

Craig demoed an MVC snapshot talking to a headless snapshot, another MVC snapshot and an OpenAugment snapshot. using a Sorcerer server to establish connections. The headless snapshot has an informant object which the headful tells to do things. Craig launched a browser on the headless system's classes. The round-trip time was 1/4 second which can doubtless be greatly reduced were it ever needed (it is good enough for Craig's purposes).

Q. Storing code? At first he did, following Dan's approach, but he blew away almost all the trailers at one point so you will see a lot of t1, t2, ...

Thus he can refactor without having any UI on the headless to get in his way. He had to change 11 methods to make it work (but e.g. the browser's 'Find Class' searches the local cache instead of asking the stripped image to find, etc.; as this is stripping, not changing, that's OK). The first thing he did when reaching that point was to strip out the controller and so on.

Q.(Eliot) Where's the compiler? Swapped out.

Next he did an activation marker. He found that much behaviour may be used soon after startup but not at startup, e.g. the compiler. Each successful method lookup sets a 'run' bit he has allocated. He regularly clears the marks, runs a scenario, then swaps out what was not run, replacing them

with loader methods. Running a loader method swaps back in the original method and runs it, all in-line. Thus he replaced thousands of methods with 12 loader methods; this reduced the stripped image from 300k to 200k. He demoed compiling something and we watched how the compiler was swapped back in. The Squat website has a few stats on this, but not much.

Q.(Eliot) Motivation? To reduce the initial startUp. System loads and then distracts you (by supporting user interaction) while stuff it will need 60 seconds from now is quietly loaded.

Craig is now developing a module transfer system. To transfer behaviour from one snapshot to another, the receiver creates a new empty module, establishes a connection, and tells the source-holding snapshot the id of the empty module and of the module it wants, so the source snapshot creates a proxy for the empty module. The two modules then talk to each other directly, so the proxy tells the empty how to get what it lacks (the receiver may already have some of what the module contains though it does not have the module, thus it may not want some things, etc.). During module transfer, identifiers and names are distinct. Source transfer is optional.

Craig demoed this. The port and URL are embedded in the .exe filename. The program started, distracted us with a progress indicator and then opened a trivial app (showed the time).

This is currently being used in OpenAugment, Jon Hylands MicroSeeker project, Jecel Assumpco's NeoSmalltalk machine and Maurice Rabb's Microlingua. The idea of all these is to shrink to just the classes they need. Craig demoed OpenAugment loading and leaving behind all the behaviour it did not need.

The future: Squat is an internal name, so ugly they'll have to change it. Craig is thinking of Spoon. Visit http://netjam.org/squat for info (including an answer to "Why Spoon?" :-). They will have to think about security implications; how do you trust a module? Craig hopes that Squeak 4 (or maybe Squeak 5, because Squeak 4 is changing the VM to 64bit) will help.

Craig thanked Tim Rowledge for conversations, and Dave Thomas and Jeff Eastman for giving him real code to investigate.

Q. Thought about long-running systems? Craig wants to set up an IRC-like system for serving modules, in which at all times the objects involved in those modules would be live.

Keynote: Making Embedded Systems Serviceable, Lars Bak
Lars talk at ESUG2003 in Bled covered much of this and I wrote it up in detail there. To avoid duplication, I concentrate mostly on new information and additional insight below. I recommend that you read my summary of Lars talk at Bled to get the whole of Lars' presentation on this interesting Smalltalk development; it starts on page 33 of my ESUG 2003 report at: http://wiki.eranova.si/esug/DOWNLOAD/nfrESUG2003reportPublic.pdf.

Serviceable means being able to fix the system if it fails in the field. One of their venture capitalists had a $100,000 car whose software failed (lost speedo, etc.) every fourteen days; he had to pull in and reboot the system. He visited his dealer and said he had a problem, but that it was fixed when he rebooted the system. "So where's the problem?", said the dealer.

Lars started in the Beta project, thence Self, thence StrongTalk, thence HotSpot. He wanted to sell StrongTalk to Wall Street but that was just after SUN told Wall Street not to waste any money on Smalltalk when Java was where it was at, so one weekend they made the VM take Java bytecodes and sold it to SUN as HotSpot, which has had 150 million downloads so far. Meanwhile, Lars returned to Smalltalk in 2002 to produce Resilient.

Software in embedded systems today is done in a horrible way. Images are dropped into the device plus debug versions with printf statements plus gdb. Problem: the debug version may not fit on the system. The deployed product has hardware device under RTOS kernel under C libraries under your application(s). Once shipped, it is a block of stone that is hard to change. It's a slow development, low productivity environment with an unsafe programming language (can't be trusted to keep memory safe, etc.)

Gartner says that embedded software on devices will double every 2 years, that the software costs of new car development is 35% of the total for some manufacturers, and that in four years devices will be almost all wireless. You can't believe everything Gartner says but this Lars does believe.

The industry want to reduce the number of hardware components. They *really* want to avoid product recalls, so really want dynamic software downloads. Real-time is essential.

Lars worked in Java for a long time; it is not the solution. Java, like C, does not support incremental execution. Serializing and restarting the processor is not the way when the system must run all the time. The Java VM spec is very complex; it is hard to be sure you have a correct implementation. Java bytecodes are not designed for speed and compactness. Java configurations are already too big and growing every version. Smallest Java embedded config (CLDC) on Cellphone needs 1Mb ROM and 0.5 RAM to run Java.

So Lars prefers a dynamic programming language. Bang and Olufsen want to use firewire for speaker cables but this needs computation within the speakers. Lars put Resilient on top of the chip and it lets customers connect to B&O and get online diagnostic and fix in a pilot project, which is better than telling the customer, "Take speaker back to store and we'll look at it."

Cellphones want to push out minimal size updates while the software is running (obviously, as you must use the phone connection to get update). Another use is an embedded program in e.g. a wind farm windmill, being debugged by the remote system e.g. by snapshot taken back to lab and carry on executing to study. Small method updates can be done by SMS. One seller of home entertainment systems (using C today) needs an open secure platform to make their kit work with others' kit; for this, Resilient is ideal.

Resilient is an embedded platform and an IDE. When running hosted, it interfaces to host services. When embedded, it can e.g. provide its own TCP/IP implementation to a network card. They use a single heap because memory-constrained systems cannot afford boundaries with space on one side, crowding on the other. The IDE is based on Eclipse.

Unlike Craig's work, the embedded image has all that it needs to run unconnected (they compute the closure). Their technology preview GC is a standard mark-and-sweep collector. They are rolling out a new one.

You download VM (needs 32k) then classes as needed. It is a dynamic language so you can update device drivers while program is running. For one cellphone, they used so little space that they threw in a webserver.

The VM on the device has a simple reflective interface to which the IDE talks. Provided you don't need on-the-fly code-generation in the embedded system, this is a good model. The IDE was at first implemented in ST/X with a webserver front end but the webserver was hard to progress (let's add this - some Java script, let's add that, - some XML - suddenly you had to run in NetScape version ...) so they moved to Eclipse with their plugin.

They reduced the Smalltalk bytecode set to 20 in a more uniform format. This made correctness-proving much simpler, and left them 236 codes to play with for optimisations. They create a byte-code-pair histogram, and use the other 235 bytecodes to do common pairs of bytecode; this saves speed and space. Currently, they use 35 more bytecodes for these pairs. They have an optimised interpreter and inline caching. 10% space is saved by having methods that share implementation share bytecodes.

They also unified resource management; Lars was fed up with how hard it was to handle running out of memory in Java. The VM also has control primitives to transfer control between the scheduler (written in Smalltalk, customisable) and other threads. Stacks are 512 bytes and grow as needed.

The language is Smalltalk with a few tweaks. They have full syntax for classes (inspired by self) because the customers (all C coders) are so used to files and full syntax lets them go on using vi and CVS if they want to (see slides for examples). Lars asked for comments; using = to separate method names and bodies was queried, as that could look odd when the = method was itself defined (maybe use something Smalltalk does not).

They added atomic test and store; crucial for synchronisation semantics, written as `?:=` as in `owner ? nil := Scheduler current` (if owner is nil get the current scheduler and assign it to owner). See my write-up of Lars ESUG2003 talk for a discussion of why this is essential, and of why they do right-to-left evaluation (user sees normal behaviour). They have no pool variables (I approve) or class instance variables (I regret this).

They added namespaces (any class can be a namespace) and typed LIFO blocks. Block activations can only exist on the stack. Blocks are a challenge to allocation, which you cannot solve by standard Smalltalk ways in 32k. It gives them a factor 5 speed up for some benchmarks.

(Avi) Ruby does something similar with blocks but can convert to heap block; could Resilient? We could do that but prefer to keep the VM small.

As there is no reflective behaviour in the embedded environment, just the reflective interface to the IDE, only the programming environment can create classes and there is no `perform:`. Reflecting class shape changes to their existing instances will be fully supported in the shipped product.

They have a minimal basic class hierarchy. Writing device drivers on a 32 bit computer requires 32 bit arithmetic so their LargeInteger is only 32 bits. (However true LargeIntegers, e.g. for encryption, can be implemented.).

They have a device driver API, an abstraction in the VM. You get a pointer to the beginning and range of the device' memory range; system protects you from writing out of range.

Visit www.oovm.com. The final 1.0 version is scheduled for late summer; it will be free to evaluate and will cost to use. OOVM is looking for commercial projects. Contact Lars at info@oovm.com with any ideas.

Lars then demoed. IDE had class hierarchy pane, source code pane, and outline pane that showed what was being sent via the reflective interface to the embedded device connection, i.e. what it was doing. Lars connected to his device. Some classes turn blue (already on device) others grey (not on device; would be downloaded whenever needed). Lars connected up a display, executed some code and made the display show stuff, updating code on the device to do so. He then made a slew of associated font changes and then uploaded them; the display font changed. (Important to be able to manage batching up your uploads to the device so as to upload consistent sets of code.) He showed the debugger (which is being further developed).

Q. When uploading, how do you handle when old methods are still on stack? They keep old methods around for old activations (AI conversion of old method to new is unsolved problem :-). When activations for an old method are done, the method is GCed like any other unreferenced object.

Q. Exceptions? But of course. He scrolled back in the stack to a throw. You can throw to any object; currently their pattern is to throw to symbols.

Q. Port to PalmOS? We're focused on small devices, so not sure if PalmOS is a market for us.

Q. Security? First version will not have security and will aim at markets where you have good grounds to think your connection is secure. They will use an existing secure connection system (he can do this in 100 bytes).

The VM itself will not be open-sourced (we keep the crown jewels) but you can have the hardware abstraction layer and can implement that on your device, after which Resilient will run on that device with nothing more needed doing. A wiki exists; by all means read and comment.

Q. You traded expressiveness for performance? Yes to a degree, but Lars actually likes LIFO blocks.

Q. Why switch to Eclipse? We liked Smalltalk/X but our target audience of C programmers know Visual C as their favourite environment. Eclipse looked like what they knew and so helped early uptake. (Lars remarked that he had found Eclipse' other language support was not too clear.)

Q.(Travis) We can easily write a VW, Dolphin or whatever UI. Travis looked during the talk and all you need is there.

Q. Emulation support? No plans. Most of the code is platform independent, which would help.

Finally, Lars dropped the device on the floor while trying to untangle his wires but it still worked; "It's resilient." :-)

Microlingua, Maurice Rabb
Rabb does work for Katylystic, LLC Chicago, and Stono on display products for watches, etc. He wanted to do this work in a language that would be easy to use, have expressive concurrency and polymorphic argument dispatch, plus all the obvious embedded real-time needs, and be small and fast. He's moved Microlingua back into Squeak.

He demoed animated watch numbers. DynaGlyph and FoldingGlyphs offer a fun new medium for watch designers to play with.

When he started, he was frustrated by number immediacy and immutability issues. He found it absurd to have to use value holders to hold small integers when the latter just needed to be mutable. Other goals were:

•   things said frequently should be easy to say

•   important things should be easy to say correctly

He wanted fast fixed-size integers but he did not want to violate how one thinks of numbers in Smalltalk generally. He therefore created new notation (~ same precision, ! immutable, ' mutable, # unique) which he used to define additional number-handling operators

•   ~+, ~&, ~<< and so on all return an object of the same type and precision as the receiver

•   '+, '&, and so on copy if necessary (i.e. original was immutable) otherwise overwrite original (numbers are immutable by default)

leading to code such as

```
a := `123.
b := a.
```

```
a '+ 12.
```

after which `a = '135` and `b = '135`. This gives equality behaviour which matches Smalltalk's.

He found he sometimes needed to see unreduced fractions e.g. to see 5/15 in source code even though 1/3 in compiled code. His ExactDecimals are not FixedPrecision as they are fractions, not floats; this is all about display. Thus

```
(3/15) + (2/15) == !5/15
```

The exact rationals ensure that WYSIWYG; you never accidentally get a false equality between non-equally-precise numbers.

Floats are inexact numbers; they cost performance and memory. He displays inexact numbers with half of their precision plus ~ to ensure he is always aware when a number is inexact. His ~ comparison operator compares inexacts as equal if they are equal to half their total precision.

Rabb feels that over-focus on `self` prevents you from thinking of methods as generic functions to which all arguments are equal. He thinks this inhibits better read/write and concurrency models. Thus he has eliminated self and provides implicit accessors for all instvars.

Alan Kay: "The selector space is Smalltalk's most precious resource". He feels that the methods `at:` and `at:put:` have poor names (should be `valueAtKey:`, `atKey:putValue:`) and improperly low precedence. To his #, !, ', ~ notation he has added @ for key, % for value. Except for bounding symbols [()]{}, any characters typed together are evaluated as one token, so that 2/3 + 4/5 works as expected. He adds a :: to handle the issue that, as there is no assignment operator (message used instead), you will end up needing more parentheses. The idea is that as keywords are lower precedence, adding a colon reduces the precedence of any operator; thus

```
  newPoint:: Point x: 2 y: 5
```

instead of

```
  newPoint: (Point x: 2 y: 5)
```

' can also be used to reduce precedence.

There are hundreds of millions of cell phones in use. People are searching for killer applications. They plan a contest with cash prizes for the best cell-phone based DynaGlyph.

Q. (me, Avi, Lars and others) KISS: why have two ways to do the same thing? Rabb: I dislike brackets. Us: yes, but Smalltalk's simplicity is a great value; concentrate on what there is business case for. Rabb: the new notation is the thing I will argue for least.

Q. (Andrew) Using APL right-to-left would let you get rid of assignment without needing new notation just to reduce the number of brackets.

Q. What size applications? Their target is 128k. That should be doable as Lars has one that works in 120k ("and I use standard syntax": Lars :-).

The syntax stuff got a rough ride; otherwise interesting. Expect to see some cool dancing numbers on watches.

Talks I Missed

To make it easy to see whether I have reported on a talk or not (and to prove my point that I could not attend all the conference :-), here is a list of all presentations that are not reported on above. In rough chronological order, Talks:

- **ANI's Digital Archive, Dan Antion**

- **Dialect Portability: Smalltalk to produce cross-language consumable components, Giorgio Ferraris**

- **Cryptography & Smalltalk, Martin Kobetic, Cincom**: Smalltalk 'everything is an object' consistency between small and large integers makes for effective and performant cryptography. Martin published his Smalltalk security VW presentation app into the Cincom Open Repository so you can walk through it.

- **BottomFeeder - a Smalltalk Development case study, James Robertson, Cincom**: my write up of James' talk at last year's StS gives background. He plans to replace TwoFlower by withStyle.

- **FastCGI for Smalltalk: Integrating Smalltalk Into An Apache based Web Site, Peter Lount**: Peter was unable to attend; the talk was cancelled.

- **OpenAugment: Preserving Engelbart's Augment Heritage, Jeff Eastman**

- **SmalltalkDoc, Mark Roberts, Cincom**

- **Cincom Smalltalk Protocol News, Leonard Lutomski et al**

- **Clean Slate Smalltalk and its Progress, Brian Rice**

- **Building structured drawing editors using connectors and Squeak, Ned Konz**

- **Advanced VisualAge Programming, Eric Clayberg**

Tutorials:

- **Seaside, Julian Fitzell and Andrew Catton**

- **Modular Smalltalk -- Refactoring, David Simmons**

- **Introduction to Web Technologies, James Foster**

- **Using GLORP, Alan Knight**

- **Smalltalk Garbage Collectors, John McIntosh**

- **Introduction To GemStone, Bill Erickson**

Panel: **Evolving Smalltalk**

STIC and Vendor BoFs

STIC meeting

(I only caught the end of this). Lunch on the final day was added to the schedule as STIC was under budget for the conference, a good sign. The 2005 Smalltalk solutions is now planned and for the first time in a while there will be a cash surplus from this conference to help it along.

Everyone registered last thing (Just-in-Time registration). STIC asks you to please register earlier next time, as it is less frightening for them.

Store BOF, Diane Severide, James Robertson, Cincom

Publishing

Should we publish the base? Yes. I asked that it be made a one-operation choice: with each VW major release, there should be a menupick (or better a script that you swipe and run; no point having a menu item you only execute once) to publish a set of pundles that Cincom thinks should be the 'base' publish for that release? Jim and Diane thought that made sense. I argued that the fact that publishing the base takes quite a few point and clicks, with several choices along the way, discourages many people.

Someone thought he was observing slower publishing in 7.2 than in 5i4 on Oracle. Others had not noticed any such effect.

Browsing Published

Search tools would be useful in large repositories. Let people search on package properties. People asked for a versionless pundle comment, visible when browsing published pundles in the public repository. (c.f. squeak map functionality). I agree; at present, I find myself writing this in my blessing comments, and relying on my general knowledge of the Smalltalk world to guess what someone else' pundle probably does. (The obvious thing is to make the pundle's comment itself visible.)

Someone also suggested adding a license property (which public domain licence is this under) so people could see if there were any issues with reusing and republishing this. Diane pointed out that pundles have properties, so adding one was straightforward. I felt that you would probably already have downloaded something in which this was a real concern to see if you wanted it, and that such info would usually be in the pundle comment.

Merge process

The process is described on Travis' Griggs Blog. For a two-version merge, the process is:

- Load the head package you intend to merge into the image (or otherwise reconcile the image version with the Store repository, if it is not already).

- Select the divergent package from the version list, or graph, tool and invoke 'Merge into Image'.

- The Merge tool finds all differences and resolves all it can, showing the results and options, each marked with a square icon:

    — open (not yet resolved, merge tool does not know how)

    — cross (merge tool thinks it can resolve)

    — black (resolved)

    You select from alternate resolutions, or select the nearest choice, edit it and 'Accept'. You can 'Apply' each resolution one by one or just 'Choose' resolutions and 'Apply All Resolved' in a single act at end.

- If the parent version of the package you loaded on starting is not the one you want to be the merged version's parent, re-reconcile the merged version that is now in your image with the repository. This re-points you at the head of the chain, gives you correct version numbers, etc.

- Publish merged version

The re-reconcile prompts you for a new parent and you choose the one that you want to regard as the prior version of this merge. You may want to do this because, unlike Squeak's Monticello, Store pundles always have just one parent; after merging, invoking reconcile again lets you choose which prior version you wish to regard as this parent (other is also recorded but only as a blessing level comment).

The three-way merge process is similar but you need to be aware that the tool's three-way merge behaviour interacts with Blessing Levels. The default behaviour looks at the 'Integration-ready' blessing level. You subclass PackagePolicy class and MergePolicy class, overriding a couple of methods, to adapt the tool's behaviour to your blessing-level usage.

Someone thought merging slow. It takes a while to complete a merge. Then you have to review everything and then publish. Could it be done in fewer point and clicks? However they noted that it does much more than most merge tools, so taking longer to do it was understandable and acceptable.

Overrides
Someone had found the override system confusing. He got frustrated by seeing overrides vanish from his image. An AR is out to preserve overrides when reloading. However, as several people stressed, do *not* use overrides if you can possibly avoid it. They are there to stop you being blocked by conflicts with code you cannot (or should not) manage. They are *not* there to let you avoid sorting out conflicts in your own code. Only use them to meet this necessity. Never use them from choice.

Line-ups
I (and most people) use bundles for line-ups. Travis (and some others) use pre-reqs. To a degree, bundles were what there were so we used them. In Envy, you could overlap configuration maps, like bundles.

An open edition in Envy was a mini-stream of development. In Store, you use the blessing levels to store Work-in-Progress. In Envy, a 'release' action makes the change visible to others immediately, whereas in Store

you must look to see if there's been a change. I thought that was OK as regards how people actually work. Diane thought you only needed to know when you first started work in a given pundle; if later it is updated by someone else, you have a merge issue anyway. Alan pointed out that if you wanted Store in your image to poll given repository(ies) for updates to loaded pundles, that would be very easy to do and would recreate exactly what Envy did (polled when you re-entered an Envy window); is it wanted?

I find that Envy, practically-speaking, rarely warns me of rival releases unless I think to look (if I'm working on an unreleased edition, then it goes on looking unreleased in my application manager when what is released changes), which is much like Store. I don't want to be warned on release; as Diane says, it's too late then. I want to be warned when I start work if there is later work in the same place. Envy could give earlier warning since it knows when I and other users edition the same things. The Coast tools did this. Store's more frequent motivation of versioning offers earlyish visibility. Alan's polling idea might find users in large separated teams.

Bundles and Parcels
When a third party provided parcels they could not provide a bundle structure. They can now; a utility is in open repository.

Post-BOF discussions: Tools and Replication
Do/will RB code tools exist that show version lists, version graphs, etc.? (I might attempt such tools myself if not.)

Can the replication UI more clearly indicate when 'Export changes ...' as against 'Export' (speaking in Envy terms) is being done, and/or (maybe just another way of asking for the same thing) how far export of specific versions is integrated with reconciliation with the export's target?

Incremental saving / loading of binary?

IBM VAST
VAST BoF, Greg Bonadies[1] and John O'Keefe, IBM
Greg began by saying that IBM's general public strategic direction is WebSphere, Java and J2EE. VAST has hundreds of applications and many customers. Their customers have told them these are strategic to their businesses and need to run for another five, ten or twenty-five years. These applications have to evolve or perish so VA must support this, and must provide resources from the IBM community.

Greg wants customers to contact IBM and make known their wishes for VA Smalltalk. The long-term viability of VA Smalltalk does not come out of thin air. There are specific revenues to handle.

Some people may move *to* VAST. Some may be compelled to move *from* VAST. Greg mentioned the Synchrony BoF that would occur next day. Their wish for VAST is to provide viable options for their customers.

---

1.   I mis-identified Greg as Greg Curfman in an earlier version of this report; apologies.

Q. How do we talk to IBM? Greg emphasised that he and John will gladly hold conference calls, and are here till Wednesday. "Talk to us." They can arrange NDAs with customers to share confidential details on future plans.

Q. VAST has a revenue stream and must support itself from it? Yes. IBM has 2000 products to support and must put its strategic resources behind its strategic initiatives.

Q. Any effort to increase the marketing base for VAST? No that's not happening. IBM has finite resources for marketing and has made its strategic choices. VAST is still selling new licences but the revenue stream will not support marketing.

Someone asked if anyone in the (fairly full) room wanted to leave Smalltalk for Java and J2EE, as IBM's overall strategic thrust suggested. The answer was a very clear no.

Someone pointed out that the upgrade licences are a lot for what you get, and pointed out that many VAST users are not upgrading for that reason. There was discussion whether a changed licence funding model would in fact increase the net revenue of VAST. Cincom has benefited from a changed funding model (which is not to say that VAST should necessarily imitate their licence).

Greg stressed he had to allocate finite funds to things essential for users, so please tell him what those are. Points made included:

- Keeping VAST working on new platforms is much more important than exploiting features of new platforms. If MS bring out a new Windows version, get VAST working on it. Exploiting some new feature it offers matters much less.

- Do not let the WebSphere + Java thrust prevent the direct VAST-to WebSphere connection from continuing to be available.

Off-line discussions pursued the question of how revenues are allocated for Passport customers; how does the VA Smalltalk group get attribution for their contribution? Rumour hints this is a sore point with the VAST team. Allocations are determined from on high, based on license counts and other factors (presumably chicken entrails and/or tea leaves are also involved), but it is conceded that they truly have no idea of how many active licenses are out there, so revenue allocations are not particularly accurate.

My opinions on all this are

- fortunately, the productiveness of Smalltalk means that Greg and John will be able to do more for VAST with the finite revenue they get than otherwise.

- IBM top management's wish that their customers would consolidate on their core strategic offerings is nothing personal to Smalltalk (e.g. IBM says there are still 200 million lines of Cobol in use worldwide, and they recently publicised Cobol-EJB interworking tools).

Why are IBM top management still telling people that Java and J2EE will take over the world. Do they still believe it? Or do they, on the contrary, fear they have finite time left to grow critical mass for their own suite in this domain before it becomes yesterday's hype? Will not most VAST customers who can be so influenced already have been? Will not most who practically-speaking can port already have done so? (For example, GEICO management, influenced by IBM strategic marketing presentations, are now talking of reworking their VAST system in Java in a few years, but, like many who talked so before the millennium, it may never happen. The GEICO developers are ex-COBOLers trained in Smalltalk, so they have no C background to make Java attractive.)

Other sources, unconnected with this conference or Smalltalk, tell me that many IBM consultants are unhappy with the 'push our strategic products' drive from on high; unhappy to the point of resignation.

Synchrony Presentation
This was entirely a Synchrony sales pitch. We started with Timm Vonu going over the objectives of 'modernization'. Although not overtly stated, it became clear that 'modernization' == 'conversion to Java' (not how I would use the word :-). He then talked about options in a transformation roadmap, the critical success factors, and, finally, the Synchrony vision. He emphasized that Synchrony was pro-Smalltalk and was anxious to provide a transition platform built in Smalltalk.

We then had a presentation by Slavik. He spoke of Synchrony's long expertise in migration. He alluded to the virtues of strong typing in programming languages, and mentioned that they had extended the refactory browser kernel for use in their language transformations. They import the code into a code analyser and then they use an interactive type inferencing methodology to derive a typed object hierarchy. The code is stored in a kind of lexical repository, and once all the typing is in place, transformation rules are applied depending on the source and target platforms to create the converted code.

There were a number of questions, all from Bruce Badger, interspersed through Slavik's presentation:

Q. What kind of code can be imported into the code analyser? Any sort of Smalltalk in, any sort of Smalltalk or Java out.

Q. Can Java be imported? Not today. The code analyser only handles Smalltalk presently, and, for Java output, translated code is sent to Eclipse.

Q. This seems like a useful tool for analysing current code. Yes. Eric Clayberg commented that it might be considered a 'modernise in place'.

Q. What would it take to get a Java to Smalltalk translation? A client that wants it.

Q. Have you done any migrations from VW 2.5 to 7.x? No, not yet

We were treated to a very quick demo of their tool, showing how classes are brought into the repository, type information is initially inferred, then refined by a user. Once typing is in place, transformation of code is looked at. This involves looking at type mapping and message mapping, with both of these relying on looking at things such as how method returns are used.

My own take on this is that many have tried Smalltalk-to-Java generation from the late nineties on. When the project was complex, the results have been dismal. Verison tried generation in one of their three failed attempts to port a large project from Smalltalk to Java. I know of a large German trading system that attempted to reimplement in Java and simply ran out of money. Many others tried and failed. Smalltalk is too granular for Java interfaces, resulting in namespace pollution and performance degradation. A more serious problem is the interaction of generation with the 80/20 rule. The generation rules work in most cases, not all, producing a system that must be refactored, but the generated code is not very human readable and not very refactorable. So the money and time you 'saved' by generating you then start to burn as you push the generated system towards usability.

It is fair to note that Synchrony have background in Smalltalk inter-dialect porting and are building their tool in Smalltalk, so may do better than others, but I'm dubious that 'better' would actually mean practical success for large complex projects. As for Return-On-Investment!!! Synchrony offer (for a noticeable price tag) to do an ROI analysis as the first step. Perhaps it is a good idea to show managers a real cost as computed by people who want the work and who made you pay for the information (conveys plausibility to some types :-). I could do an accurate ROI for less, but the manager who proposed the port might not like its phrasing. :-)

Other Discussions

General

Most of world's shipping is managed in Smalltalk. I talked to Cherie from Nynex (shipping firm). Nynex are moving from their VA application to a VW one through having merged with another shipping firm that use VW.

There was a strong Australian contingent at this StS. It's the first time someone has given a seasonal date (Sames, "summer / autumn for release of Pollock") and been asked, "Which hemisphere's summer will that be?"

JWARS is now on 35 sites with another four deployments upcoming. They are running some major scenarios this year, including some anti-submarine warfare scenarios.

The Smalltalk market is picking up again in the U.S., no doubt in part because of a more general pickup. There are twice as many Smalltalk-mentioning adds on the lists as there were this time last year, and some people who, this time last year, were interested in UK work, now have contracts, or expect to have, closer to home.

Eliot has sent several Watson-dumps to the blue screen of death team in MS and gets frustrated when the support people can't see some details of these, and he can't cut and paste them; if only they used Smalltalk.

GemStone

SUnitGS and GemKit, Joe Bacanskas, Washington Mutual

SUnitGS has no UI, you run it from a workspace test by test. The standard pattern is that setUp starts transaction, tearDown aborts (get example tests from Joe). Running suites: be aware GS is capitalisation aware: call your category 'Testing' not 'testing' to get buildSuite working OK.

GemKit for Envy was rationalised by Paul Baumann, download it from sourceforge. GemKit for Store is being worked on by Joe for UBS (via Magnus of Georg Heeg). In the Store version, the GemStone namespace is a peer of the Smalltalk namespace, not child. Movement is always a push, from VW image to GemStone or vice versa. The comparison UI uses VW DiffList, etc., facilities to compare code in GemStone and in VW.

Other GemStone

Bob Nemec does canonicalised dates and canonicalised zeros. He has powerful UI-driven tools for instance migration: turn this collection into class with seven instVars, split this class into three subclasses, etc.

Monty Kamath mentioned that GemStone databases with 2 Billion objects exist with OTL, Cosmom, etc.

Follow-up Actions

- Ask Joe Bacanskas for GemStone SUnit test examples.
- Get Store parcel bundle-structure utility from the Open Repository.

Conclusions

Seaside and withStyle had a high profile at this conference. "Seaside is to other Smalltalk web toolkits as Smalltalk is to most other OO languages; it's as simple as that." Cees de Groot knows whereof he speaks; he runs a hosting and web company. My Smalltalk webwork has only been hobby projects so I speak with less authority when I say that Seaside justifies Cees' opinion, and when you add withStyle you get a very strong intranet and internet-developer-environment story. I will watch with interest what others with serious commercial web-app experience make of it, especially as it offers the possibility of starting small and growing.

Lars and Georg carried on the 'Small is beautiful' trend with impressive developments linked to well-designed business cases with plenty of room for growth. Most Smalltalk survived the lean years in the complex-app, fat-client world of systems so challenging that repeated attempts to rewrite them in Java failed. Maybe we're now breaking into new territory.

Written by Niall Ross of eXtremeMetaProgrammers. REPORT ENDS.