Smalltalk Solutions 2006, Toronto, 24 - 26 May 2006

I spent the days before and after the conference with a friend at Oakville, 20 miles west along the lake from Toronto. Unfortunately, I forgot to move the bottle of fine merlot I was bringing her from my hold baggage to my hand-baggage before check-in so my clothes drank it instead of my host. I advise cold-soaking before washing for removing red wine stains. (As my friend remarked while cleaning a wool jacket, when the wool is on the sheep, it can be hot or it can be wet but in the UK it is rarely both at once.)

This year's Smalltalk Solutions was combined with Linux World and Network World, and so was on a larger scale than usual, though not so large a scale as the Toronto Metro Convention Centre, whose space syntax was quite something to encounter first thing Monday morning if, like me, you arrived at the south building (where numerous coaches disgorged their contents outside the entrance), eventually worked out that you needed to cross the skywalk (over what seemed like most of downtown Toronto) to the north building, and there found numerous trucks parked *inside* the building, disgorging their contents into the remainder of a space so vast there did not at first seem anywhere else for the convention to be.

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 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 at bigwig dot net. I thank all the speakers and participants whose work gave me something to report. Thanks also to the conference sponsors and organizers, and to the Smalltalkers, especially Suzanne Fortman, who worked hard to make us part of it.

Summary of Presentations

I have sorted the talks I attended into various categories:

- Keynotes and Overviews
- Frameworks, Applications and Experience Reports
- Coding and Testing Techniques
- BoFs and the Coding Contest

after which I describe some Talks I Missed (from the Smalltalk track; that I missed lots from other tracks goes without saying) and Other Discussions, note some Follow-up Actions and give my overall Conclusions.

As there were usually two Smalltalk (and three to five other) parallel programme tracks, I could not attend, still less report on, all I wished to see. (Some choices forced by the schedule were painful; listen to Bob Nemec on Northwater apps or Michael Lucas-Smith on WithStyle?) Talk slides are on the conference' website http://www.smalltalksolutions.com. James Robertson's blog posts cover some talks I missed, as do those of other bloggers (e.g. Michael Lucas-Smith, Blaine Buxton and Avi Bryant).

Keynotes and Overviews

Invited Talk: Avi Bryant and Andrew Catton, Thinking Small

At Smalltalk Solutions two years ago, Avi's talk "Winning the application server arms race" argued that there was a niche where Smalltalk gave web apps real competitive advantage and that startups should start up and take advantage of this. As a believer in eating your own dog food, Avi then did this. He and Andrew have started a company and developed Dabble DB. They are one week away from launching this thing. They are thus not (yet) saying, look at our success. They can say, look at the interest we're getting: 10,000 sign-ups and a great deal of blog interest (especially for a Smalltalk product). Thus they are pretty happy about how it is going so far.

In their official and unofficial consulting to web companies in the past, they saw vast business process problems to do with spreadsheets where the best solution was a DB and web app. They would say, "Email and spreadsheets: bad!" and the customer would reply, "Well, what should I do?" Dabble DB was built because there was no ideal answer.

Ten years ago, or even five years ago, taking money to start up their company would have been an obvious choice. Now, applying Smalltalk ideas to business, they advocate late-bound business plans. If you take venture capital money, that forecloses many options. Making a million dollars a year in revenue is 'failure' not success. Taking money to build web software can be very premature. For most of the time, their company had two developers (living in different countries and time zones) and no office. Deployment costs were absurdly cheap (under a dollar per month), so, "What do you need the money for?" Well, you do still need to eat (and their wives might think of other things).

Solution: get a job. Take the best option, especially the best for opening up other options later on. Andrew took a job with the agile projects group in the University of British Columbia. It was then using Java but Andrew soon found ways to add Smalltalk. The manager came and said "I need a survey application and I need it by this afternoon". A few months later a project needing to model a complex changing domain appeared and they convinced them that Smalltalk was the best choice; that gave a year of Smalltalk development and the managers became hooked on Smalltalk due to their habit of answering 'yes' to the 'can you' questions they were asked.

Andrew brought Avi in as a contractor and then, having got the group addicted to Smalltalk, they quit and founded new company Smallthought. It was the reverse of, "How will we find Smalltalk developers?"; the group was addicted to Smalltalk so their company started with a nice contract.

Avi has described Smalltalk's technical advantages in other talks. Now he spoke about Smalltalk's psychological advantages. "If it hadn't been written in Smalltalk, it wouldn't exist", or it would be very different. They demoed to show how a flavour of Smalltalk was in the app. You start with adding entries or importing data; their target customers start with some data they want to enter on a spreadsheet, usually before they know just where it should go or what they will do with it. They wanted an equivalent of on-demand / program-in-the-debugger style editing. Over several iterations, their product evolved from a style that felt like designing a schema to the on-demand style, with halo-like manipulation.

Avi imported some data (about another conference) from a spreadsheet, confirming the field names. He added columns for the room and presenter of the session. He then clicked on the column header and started filtering the room view. If you want to change something, you click on it to discover what you can do with it. He grouped by session type; the view rearranged.

Databases usually demand that you define types upfront, with strong constraints on what can go where, etc., whereas users of Excel are not accustomed to that style. Thus everything in Dabble starts as text and gets annotated when type annotation is of use and benefit to the user (e.g. annotate that this is a date field, so get calendar views offered). However nothing forces you and you can enter invalid data without causing an error (you get warned but it will be stored).

Databases often do not offer refactoring. Dabble supports 'extract subtable out of another table' and similar. They pulled the presenter out of the session table to let them start adding data about presenters. The text field became a link. They pulled company from the conference sheet to the person sheet. (One company per presenter; some refactorings may not be semantics preserving. You could refactor company from a text field to an object and then refactor to give it a one-many relation from person.)

All these refactorings are undoable. The UI offers one-level undo since otherwise concurrent users can create subtleties. Andrew feels they should offer multi-level undo for most scenarios. Other refactorings exist and yet others will be added; they find the idea of data refactoring powerful. For example, a field for children may start with comma-separated first names and later evolve into detailed data on, and relations for, each child.

Another flavour is that Smalltalk has lots of small methods. Conventional programmers can be annoyed at this ('I must step through method after method to see what is happening') but business users like small increments.

Avi added session durations, then mapped to summaries to show total talk time per presenter, to show people who did more than 5 hours of talking, etc. All this was done by simple point, click and menu operations.

Getting the right UI is absolutely critical, probably in general and certainly for their target audience. They therefore got an external UI designer involved from the very beginning. As a developer, it feels good to be using something that looks good. It also deprives you of the ability to rationalise your mistakes. “We’ll redo that to work better when we upgrade the UI”, becomes, “The UI is great except for this ugly thing I just did”. They paid the external guy for umpteen versions of the UI they never used but it was still the best investment they ever made.

The business equivalent of real estate’s ‘Location location location’ motto is’Networking networking networking.’ Avi did a demo to some people at a software meeting in Vancouver. Those people started a company while he was in the Netherlands for a year. When he came back they met and he was introduced to someone else; she invited him down to conference. A speaker dropped out (when Google bought their company) and left a slot which Avi was invited to fill. The bottom line is that just going to some meetings and saying “Hi, you don’t know me but let me show you a demo...” can start a chain of effects that ends somewhere you would not have reached by planning. They get 2-3000 hits a day purely through word of mouth through blogs. Also (as James remarked) Google loves blogs and makes Avi’s blog very visible. You know you’ve made it when someone puts a google add-word using your blog.

Q. Paul Graham (Lisp) said bug-fixing was his great advantage; you too? Yes. Customers love getting an email 15 minutes later saying “Sorry about that, it’s fixed now” (It feels so good, he had to overcome a temptation to leave the bugs in. :-)) Likewise they can go to a customer meeting, be given a spreadsheet they’ve never seen before and do impressive things with it during the meeting.

Q. The web designer did what; write CSS, advise, ...? All interaction is through CSS classes. Sometimes he provided a static mock up. At other times they gave him the raw HTML and he instrumented it. Instrumenting the code with his CSS felt like running tests to green; you fit the CSS ids to the code and suddenly there’s that great-looking UI in front of you.

Q. Database? Their database is 50,000 Smalltalk images on the server. If the session is idle for more than a few minutes, the image saves and the session goes down. (And there’s lots of infrastructure to keep all the code up to date.) You would never use this if your users were going to scale up to millions of records but it gives them competitive advantage because it is simple.

Q. Testing? There is cool work to test web UIs but they do not use it because they have a small visible UI. They may get into using it. The market (people who want to test data) is huge so they can easily get a representative cross-section of people to use it. They are not XP-fanatical about testing absolutely everything and at some level they are ashamed of that and at another they think they are testing the ‘appropriate’ things.

Q. Concurrent users? They are not trying to build an ACID database. They are trying to replace EXCEL. The intentionally set their sights low. For example, you cannot do an import of more than 5000 records at a time.

Q. Costs and scenarios? We host for $10 - $150 per month. Later, we may let a customer host it.

Q. Issues of scaling up, data protection and other legal issues? Data for which these issues matter will not be put on a publicly hosted service. Owners of such data will pay more to someone specialised to give them firewalls and all that stuff. Dabble DB aims at a specific market segment.

Q. Users? They have sent 5000 invites, 3000 have signed up for a trial. Of course, most will use it for a day and never come back. So their actual users are in the hundreds. At first they had a differentiator be the amount of data but now it is the number of concurrent users since customers find it easier to know, “I won’t have more than 10 users in the next two years”, than that “I won’t have more than ? data.”

Q. What are your selling points? The ability to offer lots of views of the same data. (A typical Excel spreadsheet application can have six copies of the same data oriented to different uses: finance, management, planning, etc.). To-many relationships do not work in Excel.

Q. Charts? Some work done, not yet released.

Q. Customer interaction? They have done lots of InstantMessaging, not so much phone interaction, and very little screen sharing. However they’ve had experience of screen sharing with their web designer who lives in Nottingham, U.K. InstantMessaging was what they were used to and it does leave a log of all these conversations to remember what was said.

Q. Competition? Competitors mostly use MySQL and suchlike at the back end so their refactoring manipulations puzzle their competitors. It would be very hard to replicate this via Ruby-on-Rails for example. SalesForce had much difficulty to make a much less capable rival function. Interestingly, the continuations stuff, much noted in his Seaside talk of two years ago, is used only in a couple of places as most of the application is modeless.

The Chronos package reimplements date and time and they may do that; they have had performance problems with Squeak’s data and time implementation.

Andrew expected their market to be a lot more geek-heavy. In fact, there seems to be a lot of overlap. (James confirmed; blogs live along an infinite range and are not at all geek-specific.) One customer is a symphony orchestra; maybe the clarinetist’s husband was the software guy or blog-reader who suggested them.

Keynote: I have nothing to declare but my genius, Brian Foote
Opening the session, Alan Knight described Brian Foote as ‘very hard to describe’ and ‘a well-known computer gadfly’. One of the first things Brian saw on arriving in Toronto was Suzanne Fortman talking about Smalltalk with a baseball game going on in the background. This is how life should be; Smalltalk should be naturally encountered everywhere.

His next slide showed an empty goal (for Canada, he changed it to a hockey goal). He compared a strong typing advocate to a football coach with a theory proving that no shot on goal will be possible. Unfortunately, should this theory of the coach be found erroneous in practice, there will be noone keeping goal. Late binding is about having someone there to save the day.

You must be careful with titles like his. Customs may ask you why you are visiting. If you reply, “to speak at a conference”, they tend to ask you what are you talking about. When Brian reached the point about using a hockey goal for Canada, the customs officer said that was fine and let him pass.

He described how Smalltalk has changed from his start in it during the reign of James VI (James Earl Carter) to today’s George III (George Bush).

Three ideas came from object-oriented programming that tend to save goals. The first was dynamic polymorphism. It lets you swap pieces at runtime. It lets you wrap things.

The second was callipygian compilation (callipygian *adj*: possessing a sightly back-end). This came from Deutsh and Shiffman via self and hotspot and many others to JITs and Complets. Everything Brian learned about OO he learned from the image. The browser and IDEs came from there (with a pinch of Lisp). Window GUIs came from there. Dynamic translation (JITs) came from there. His greatest delight was to find that Smalltalk was a language built out of first class objects; you could get your hands on them.

Idea three was reflection. (In a sense, objects lost but reflection won when XML came along.) Reflection literature grew out of the Lisp tradition but was what Smalltalk was already doing (CLOS had the benefit of seeing what Smalltalk had done). The one hole in Smalltalk meta-objects is dealing with variables (GemStone had to deal with that). The CLOS folk had no contexts, a serious omission. Put these together and you have it all. (Brian circa 1990 tried to define a class menagerie of a vast (74) set of things that could be first class objects: profilers, database views, etc.) The rest of the world rediscovered the idea recently with manifests (XML schemas and so on).

Brian sees the new idea of Actors (Actors, Models, what a glamorous world we live in) as how to integrate distribution with all the above; distributed computers go down a lot so you must have asynchronous messaging models. “20 years research into virtual memory has demonstrated that there is no substitute for real memory.”

- Meta considered harmful: false.
- “Meta” considered harmful: true.

Meta has been very badly explained and marketed. If something is a first-class object, don’t hide the fact behind mystical names.

The vision was that everything would be open, even the JITers. That compile time would be anytime; change something and the JITer will go back and deal with it. Linguistic imperialism was part of the vision. “It is manifestly clear that our language is better so use it everywhere.” The Lisp people and the Smalltalk people discovered that utopian monocultures were wonderful idea incubators but more pragmatic languages prevailed: tools designed for a particular job *and* a particular person doing that job. Gabriel’s paper “The rise of worse is better” argues that adequacy defeats excellence everywhere.

In 1997 the great schism occurred between Aspects and Objects, while pragmatists were left in the middle and the formalists retreated to academia. Meanwhile, the world ignored us. When Brian reads XML he is reminded of the legend of the Medusa, a creature so hideous that all who looked on her were turned to stone. When he sees XSLT, he thinks of it as ‘diet cobol’; “it is more verbose than even I am”.

A favourite example of formalists is the Mars climate orbiter tragedy. Brian watches NASA TV and he was watching the night it failed. What people say is that a perfectly good spaceship with good software was lost because of a simple unit conversion. An army of carpet baggers then turned up to say that if they had been carpet-bombed with money, they could have averted this tragedy: management consultants (“It was a people problem”), testers (“If you didn’t bother to test it”), formalists (“We can prove it”), etc.

Brian tried to get his hands on the actual code. The official report says the failure to use metric units in a file called ‘small forces’ was the cause. Reading the report, Brian figured out what happened. A satellite uses thrusters and big spinning discs (reaction wheels) to keep itself oriented. The latter are far better than the former to use for reorientation; you speed them up whenever you want to turn and restabilize. However, if you always keep destabilizing in a given direction, eventually you have to use the thrusters and reset the wheels back down to their start range.

In the orbiter, they decided to save weight by only using one solar panel instead of the customary two. Hence solar pressure was unbalanced on the craft, so it destabilized always in the same orientation. Hence they used the thrusters ten times more often than usual. Ideally, this would not matter. In reality, there was a slight imbalance between thruster and gyro effects, as they knew; the corrections needed were these ‘small forces’.

They dumpster-dived for an old Fortran program to do the small forces calculations. It used manifest constants in one line and the programmer porting it to the orbiter assumed that the required 4.45 unit-conversion factor was implicit elsewhere in the program. It wasn’t. Thus by the time it reached Mars the orbiter was off-position by ~100 miles because of using the thrusters 10 times more than usual. When entry time came, there was no feedback to check, ‘Is there a planet out there?’

Was this a typeless tragedy? There were papers at OOPSLA about adding units to Java to avoid this. However paper specifications are not executable (nor are programmers, luckily for the anonymous guy who made this mistake). Brian argued that this would never have worked. There was no end-to-end way of identifying the units in two different programs in 2 different languages (the input was from a C program). The guy knew the program had to be right but had probably never seen a spec for it.

Expecting paper to save you is voodoo science (sticking a pin in the paper and expecting something to happen).

- Brian started his check expecting that unit type info would have saved the program but once he had grasped what had happened he no longer believed this. NASA is (in)famous for its paper documentation but nothing in all that paper enforced the requirement.
- An end-to-end test would have saved them but they were running late for a launch window.
- None of the other theories the carpet baggers proposed would have helped much or (mostly) at all. A few weeks ago Brian reused a 15-year old program he though he had got completely right, and found a new bug in it. Large bodies of code are unreadable; people like to write it rather than read it (like poetry or weblogs :-).

Thus a multi-billion dollar firework display was watched by noone on Mars in late 1998.

Kent Beck recommends the olfactory method; if it stinks, change it. Code smells are a none-to-subtle indication of a possible problem. Brian thinks the end-to-end principle of web architects is what we need, not things like Java generics. (Java has *seven* type systems now; why does it need so many?) With processors as fast as they are, you can leave the goalie there. The program model is central. The things we have always done (swappable dynamic objects) are going to be good in that space.

People don’t want componentised housing. Similarly, they want programs that look customised to their needs. The new model is vegetable, not animal. Once we thought objects would ply the web. Now we think that data plies the web and programs are more static, sucking in data nutrients and (we hope) producing fruit.

Over-engineering in civil engineering is done by making components stronger. Three times as many lines of code does not make anything stronger but we have a resource we can squander; cycles. We can afford to leave the goalie there. We can afford to test. Not just because of Moore’s law but because the DB and the network are going to be a lot slower than our reflective programs.

Static languages throw away all the information we need. We can keep it. Humans are fallible (except the Pope in matters of Christian doctrine, but he’s not a programmer and Brian has found programming can be a pretty unChristian activity). Our programs are in our fallible image. We must embrace failure (he instanced the pine open-source project, who found they core-dumped less often if 7 were returned instead of zero as the default.

Science is theoretical, observational or simulational but computer scientists are in denial about using computers to study computer science.

Like many cultures (Ireland was one he mentioned), Smalltalk has influenced many more people than can recognise a square bracket. However, Woody Allen once said he did not want to achieve immortality through his works but by living forever; that is an attitude we could take.

Georg Heeg, What is so special that Smalltalk is still hot after 25 years?

This talk was aimed at non-Smalltalkers (of which, Georg noted, we had at least 3 in the room). Like his grandfather, who was a carpenter, Georg works with objects and frameworks. His company is located in Dortmund, Zurich and Koethen/Anhalt. They work very closely with Cincom.

The company has been running for 25 years, which happens to be the time since a certain Byte magazine issue with a balloon on it was published. It is even longer (30 years) since Smalltalk, in 1976, began to have its present form. In August 1978, Byte had its ‘Pascal’ edition and he showed that one too, with its ‘Land of Small Talk’ island (the same island the balloon was leaving in 1981). We were allowed to touch these holy volumes.

Georg quoted the article: “...the Kingdom of Smalltalk where great and magical things happen. But alas ... the craggy aloofness of the Kingdom of Smalltalk keeps it out of the mainstream of things.” Georg once ran a week’s course for a customer that started from this and asked whether Smalltalk was indeed the princess or one of the ugly sisters.

In 1976, objects were created in Smalltalk that still exist. These 30-year-old objects (nil, true, false, ...) have been cloned in Squeak and VW to the present day. This is not the case in other languages.

In 1983, his boss showed him the blue book and said, “This is new stuff - learn it”. There was no VM; he had to build one. Digitalk started in 1983 and his company started in 1987 and is still here. Many, many successful projects were started in 1990-1995: banks, insurance, car and chip manufacturers. Then the slowdown happened. Java’s promise was: “the magical things without the craggy aloofness”; you can get it all without learning a new syntax and all that hard stuff. At the same time, the Smalltalk projects were deployed. So Smalltalk’s customers were happy *but* they believed the Java hype so they started to migrate. Several failed and such migrations as succeeded were very expensive: migration always cost as much as the original ST project *minimum*.

Then the second spring occurred. Eventually, some large customers made Smalltalk strategic again to revitalise stagnating projects. Meanwhile, very small companies (e.g. Avi’s) started to use Smalltalk.

What is so different about Smalltalk?

Everything is an Object. There are only five concepts: object, message, class, instance and method. These are defined in terms of each other so “it is almost as though the reader must know everything before knowing anything.” [Adele Goldberg].

Everything is an Object. Collections, numbers, classes, methods, messages processes and contexts are objects. In C (or rather, in assembler) you can hack the context. In Smalltalk, the context is an object and you can safely manipulate it. When you write a debugger, you need it.

Everything is an Object. You are in control of your *entire* system. This is like Linux (with sources).

Everything is an Object. You get addicted to this. Smalltalk is “a vision of the ways different people might effectively and joyfully use computing power”. In some management organisations, joy is not favoured but management motivation classes will admit its relevance. Most people in this room are here because of it.

What are the alternatives? In the 1940s, von Neumann invented the architecture of CPU (procedure) and Memory (data structure). This schism between statements and data thus becomes the norm. Soon, it was noticed that this would not solve the world’s problems. Other models were proposed influenced by mathematics: functional programming, logic programming, set programming. These had no representation of time because 2 * 2 = 4 and has done for a long time and will do for longer. Time is therefore represented by tricks (or ‘impure features’ as the theorists call them). This is undesirable and not just to theorists; try spending two weeks debugging a Prolog program at exactly where the axioms no longer hold.

In the OO world, the main question is ‘who is responsible’. Concepts map directly to software; we model, rather than program. If you have toothache and visit your dentist, they look in your mouth and at your insurance and record it on a form that models this. 15 years ago (in Germany) traditional computer programmers arrived in dentists’ offices. They wore spectacles just like Georg’s except that one lens saw only data and the other saw only processes. The phenomena in the mouth (and in the insurance policy) were separated in these two kinds. By contrast, OO models much as the old form would: a viewpoint determines a model and its adequacy to the application.

He then showed us two abstract descriptions of pencil and pen. However his son John just wants to draw with them. Thus we have an abstract class with which you can draw, and subclasses Pen and Pencil. Class `Boy` can have a `hand` and can send the message `Mummy givePen`.

Now let’s do it in Java: we need some private declarations and so on but it is straightforward enough.

John grows older. Now he uses his hand to pull his toy cart, and he sits in cars. Now John sends `Daddy getWagon`. All is done easily in Smalltalk, In Java, look at all the points where we must change the types.

In Java and other languages, a program is software which is started. Smalltalk is a system which is modified. Thus it is much more like an operating system. In Smalltalk, new features are e.g. parcelled in. In Java, the sequence needs a stop and restart.

He showed a block that recursively defines factorial. If you understand recursion you understand it; blocks are easy.

So it’s all easy, right. Yet there is still this craggy aloofness. However he talked to a new customer (who had never seen Smalltalk or any programming language) for two hours and then the Object, Message, Class, Instance and Method made sense to him: “OK I understand these 200 classes are all just controllers, now what shall we look at next.”

Q. (Brian) Could a rigorous programme of mandatory testing help with Smalltalk addiction? Discussion (and some laughter); conclusion: XP will probably make the addiction worse.

Q. There are Smalltalks on Linux? Yes, indeed (and Abuntu have gone out of their way to make Squeak available).

Q. Smalltalk is a set of small but powerful ideas, as was recommended in yesterday’s keynote? Georg had to do some C to facilitate Smalltalk in a project; average time to find a bug was 5 hours in Smalltalk, 5 days in C.

Q.(Petr) So how do we sell Smalltalk? (Sames) As well as being viral with developers, you don’t talk about Smalltalk, you sell solutions (and I can show you the whip marks from Suzanne to teach me this). Show the companies that are succeeding with Smalltalk. (Georg) this second spring differs from the original hype in a crucial way. AMD deployed a $2.5 billion development last November pre-schedule! How many projects of that size do you know that go into production ahead of schedule. AMD have stated their next project will be in Smalltalk.

Suzanne explained that in the old days they marketed Smalltalk to the application developers, who were very keen but were not the decision makers. Now Cincom are publishing success stories. Why is Smalltalk still around after 25 years? Because of success; it had nothing else going for it but that. (I remarked that the Cincom hand-out is right on message for this.) Suzanne in Cincom is seeing the benefits already. In the past, our line was “If you don’t use Smalltalk, you’re crazy” - not always persuasive to non-users. Now we say, “Look at who is succeeding with Smalltalk.”

Frameworks, Application and Experience Reports

Real World Uses of Smalltalk, Martin McLure, GemStone

Smalltalk was started in 1972 by Alan Kay and his team. It originated many of the programming ideas we now use. The world learned of it in 1981. It takes a day to learn the Smalltalk syntax, a month to learn the style and a year to learn the class library (in the old days; the class library is bigger now).

JPMorganChase are the second largest bank in the US. Their Kapital system handles pricing and risk analysis for exotic derivatives. It has made their investment division more money than anything else. It uses Cincom VisualWorks Smalltalk and GemStone Smalltalk.

The Penn state eLion project gives 24-hour secure web access to records.

The OpenSkills project is covered in Bruce Badger's talk tomorrow.

The US Department of Defence has a Joint Warfare Simulator project, running since 1995 in VASmalltalk from Instantiations.

Progressive are an automobile insurance company. Many insurance companies can only create a couple of ratebooks a year. progressive can role out a new ratebook and new product in two to three days. They use VA and GemStone.

MicroMuse were acquired by IBM recently. Their NetCool suite of products monitor network performance. It runs in VW.

NorthWater capital has a division northwater objects that uses VA and GemStone for financial applications; see Bob Nemec's presentation.

Adventa control technologies were spun out of Texas instruments. They've been using VW to run wafer construction since the early 90s. Plants cost billions and must be replaced every three years (because they've become obsolete) so you can see what even an hour of downtime is costing them.

Florida Power and Light: summer 2004 saw three hurricanes in Florida. At loads far higher than ever projected, their VA, GemStone and VW systems worked fine. A rival Oracle system had many complaints.

Liberty Basic from Shoptalk systems is written in VW. Some major books e.g. 'Basic Programming for Dummies', reference Liberty Basic and use it for their examples.

Martin then demoed adding a feature to David Buck's solitaire game app, working in the debugger, inspecting, changing values in the inspector, executing proposed code, halting and typing. Eliot pointed out that the autocompleter he was using was not part of the base system. A user (Travis, sitting next to me) wrote it and uploaded it. Smalltalk is very malleable.

(At this point the usual demo hiccough occurred - a method he called should have been in the image but was not.)

SqueakSource is a website for projects written in Squeak. Last Martin looked, the squeak mailing list was getting 12,000 messages a year.

Fractal is a Czech company doing online flight search, using Gemstone/S.

IntercontinentalExchange do energy trading using VW and GemStone/S.

The Canada Border Services Agency is a customs intelligence agency. They use Smalltalk to diagram the chain of information of why a person is of interest, flagging that someone should be looked at, etc.

Dalektron: Andy Bower taught his 9-year old son to programme by writing dalektron collaboratively in Dolphin Smalltalk (see www.dalektron.org).

MetaCase is a meta-data-driven tool that lets clients (mobile phone companies, chemical analysts, etc., etc.) develop domain specific languages in which to define their solutions and generate code for them.

Jack-in-the-Box tracks food resources.

Key Technology use VW to control high-speed food sorting machines; see Travis talk.

Caesar systems use Smalltalk for petroleum systems analysis. (I recall someone gave a talk on this at Smalltalk Solutions 2002; it’s in my report.)

SqueakLand is a website on using squeak in education, with Etoys and much else. There is a video available; he showed a brief excerpt (a pupil, Amanda, described how, “Squeak makes math funner.”).

There are many more projects listed in the hand-out.

A few years back, the flying Karamazov brothers had a show using physics, juggling, music and etc. The IT media lab had sonar emitters on their heads to detect position and wrist transmitters to connect to the computer so they could do virtual juggling of e.g. planet images. Martin toured with them for six weeks and he showed a video of their original program and then the version he rewrote in Smalltalk. His version had a better background and better behaviour. He started bouncing it with the cursor then with the camera. He had an automatic camera aligner (in the original, camera alignment had to be set up just right).

Silt: Lessons Learned in a Smalltalk Web Deployment, James Robertson, Cincom
Because I already knew a lot about this impressive system, I went to the competing talk. For background on this system, see my StS in 2003 report (Jim's talk 'WebLogs and RSS'), my ESUG 2004 report (Jim's talk and Alan's demo) and my ESUG 2005 report (Jim's demo). This time Jim talked about the system's scalability.

On The Development of a Platform- and Domain-Neutral Enterprise Application Integration Framework, Tom Hawker, OOCL
OOCL manages shipping. Their GemStone database has 135 GB and 2 million objects. They are moving to 64-bit GS. Their database tracks where anything is and all the associated paperwork and the particular custom data. It manages the total paperwork involved in knowing that a particular cargo with a particular security seal of a particular customer is in a given container on a given ship in mid-atlantic and is due to be off-loaded with a particular priority.

IRIS2, the system that tracks all this data, is a centralised monolith with 50 downstream systems. 15 sites provide 24 x 7 distributed call centres. Their network is T1 and above to all these locations. They have status to track every time something happens.

Each player in the shipping market has their own way of doing things. OOCL foresees a future in which they must interwork increasingly others. Thus they want to move from a single monolithic Smalltalk system to a distributed system (and able to support modules in multiple languages). In a pier-to-pier system of distributed computers, one system will handle the bookings, one will track ships, etc., All these must interact with each other and with the distributed locations. Interactions will centralise in the particular database that is most relevant to the function being performed.

The global container shipping market is $100 billion/year. IRIS has 4155 classes in GemStone and 4295 classes in VW. The new architecture had to deal with bidirectional messaging over multiple h/w and s/w environments. Some messages are queued but others need immediate processing, and messages must not be lost: reliable robust messaging is a hard requirement. Messages must be sequenced and need traffic volume control to manage runaway applications.

However the very worst requirement of all by far was that "it must be standards-based". They have 1500 to 2200 clients online at any time running 4.5 transactions per second and rising. Hundreds of thousands of objects are moving across the VW/GS boundary. WSDL has non-dynamic types. Making WSDL and SOAP work with their system was no fun.

They had to avoid some easy unwise assumptions. They could not assume their private network was unhijackable. They could not assume that its topology would not change frequently. They found the XML and SOAP variants were not always compatible. They had to decide things:

- Where did they want to put the software?

- Could they treat the objects in the database as read-only passive data?
- Could they merge asynchronous and synchronous messaging?
- Sequence messages on transmission or on receipt?
- Maximise throughput how?
- How to handle faults?

They decided to leave the IRIS2 client alone and not to port WSDL or domain objects. They chose a VW7 and GemStone solution (no way to do web stuff in GemStone). They guaranteed sequencing at transmission. They communicated by channels: channels provide a non-conflicting insertion point. The dispatcher sequenced using minimal transactions. The transmitters allow multiple insertion with a single extraction point.

All this development raced past the Java code work. The Java system froze and the Smalltalk fixed it time after time.

The architecture is of client and reader Gems talking to each other and each to its own client VW image. The reader VW image passes stuff to the sender image. A feedback loop talks back to the Gem-Gem queue which models the state of the messaging.

They already had a batch processing facility. They augmented it to queue, sequence and run requests. It also wrote to files for recovery and rerunning as needed.

Synchronous exploits both client and server sides. Rather than put socket transmission into GemStone, they package WSDL in intermediaries. They generate their WSDL dynamically. Execution interacts directly with domain objects (and they have found one multiple feedback loop already). Their servers have traffic controllers to limit denial of service risks.

They load-balance through the network front-end so GemStone sees only one entry point. They have been able to run several tens of thousands of messages an hour without loading the machines especially.

They cannot queue because the sender is expecting a response. The handshake was therefore made the actual response; this is a major difference. (See the slides for the for overall architecture diagram.)

They had various hurdles to clear:

- It tool them 6 weeks to work out all the interoperability issues between ST, Java and C#. The latter two do not handle NULL the way one might think they should; their typing mechanisms had to handle Java’s “can’t send NULL to some values” and C#’s “its optional”. They had to generate the WSDL and that helped adapt client to server.
- At a larger scale, they had to handle almost every combination of client and server. Some operations require a dozen properly sequenced messages. Management were very keen that no message ever be lost.

They had various hoops to jump through:

- They had to optimise the VW GS interaction as the servers were being stressed.
- They needed channel status reporting and error reporting.
- They needed to redeploy easily. Tom can reboot any daemon in the system in 3 minutes!
- They had to keep operational debugging; it is incredibly valuable.

The new system is 15 packages and 145 classes in GS, 40 packages and 400 classes in VW. XML-based configuration uses 9 files and 2000 lines of XML, all optional. In both GS and VW, the application just sits on top of the layered architecture. It took 2 years of effort and went into production use in February. The frameworks have already proved to be reusable; he added a logger in a morning by reusing the underlying capabilities.

They did all this without affecting their existing GemStone application behaviour; management was very happy about that. It could be pulled out tomorrow and the original application would still run. Nothing in this system is hard-wired. The run-time logic is very easy to extend. He can write a new client in 2-4 days, a new server in 4-6 days, and a new config in minutes.

Q. (Eliot) What kind of sandboxing do you do when building these? The object encoder and binding structures say which objects are wanted. The application usually has a lot of sanity checks built in so should protect him from doing damage.

This is a model of distributed application processing. It was produced without a precise spec (initially it was half a page) but with a good target. Complex configuration is necessary; making it easy is also necessary. Design in flexibility because you are going to use it. Having an extensible baseline is valuable.

The Java system, despite having twice as many threads as they did, could not accept stuff as fast as they could transmit it.

A typical shipping transaction might take 15 seconds of real time, several seconds of CPU time and 0.6 seconds of their system’s dispatching time. (and that was a maximum high complex case). At maximum volume they are still incessantly starving one of the queues.

Over the last few years, they have increased revenues 50% but their market share has halved. This has in various ways influenced the belief that their system must be able to interact with others in a distributed way. They are gradually moving the less volatile capabilities out of the core system.

Q. How do you avoid over-engineering? It is not the requirements or their changing that bites you; it is the rate of change that will bite you. One day he will have to talk to a Java person and a C# person. Luckily Smalltalk is very amenable to interoperating.

They have a C# .Net client doing a very low volume vendor maintenance task (from some shippers they use in addition to their own fleet). It only needs one update a week but it was useful as a .Net test. Every Monday morning they take 2 hours to dump 120,000 routes to the booking clients.

February was the beginning of transition; not all has yet been transited.

Building an Optical Food Sorter with Linux and Smalltalk, Travis Griggs, Key Technology
Travis is a mechanical engineer who discovered he could write programs. He simultaneously learned C (‘you can insert NULLs into a string and this lets you tokenize’) and Smalltalk (‘this is a dictionary’); guess who won. :-)

Of all the OS systems out there, the Linux one has most in common with the Smalltalk image. Smalltalkers tune their image; Linuxers tune their OS.

Key Technology was started a while back by two farmers who decided to make equipment for farmers instead. Now it is 500 people, of whom 8 are software developers.

In an old-style food processing plant, many people sit watching conveyer belts doing tasks such as flipping 50% of asparagus heads the other way to get them all lined up (and noone has automated that particular task yet). In their sorters, some 4-8khz true-colour line-scan cameras scan the food and pass the data to the Image-Processing Board. The image processing done is not that special; the rate at which it is done is Key Technology’s claim to fame. The IPBoard talks to the Sorter by firewire and this similarly talks to the Ejector. The UI processor talks to all these.

One kind is an air-sorter. Pneumatic valves divert the potatoes, vegetables and other food items as they pass over a space between two conveyers. The firm started with a product to sort potatoes by colour. Then the Hamlet project (to bean or not to bean) introduced sorting by shape. Now they sort tobacco, coffee, nuts, cereals, and many many others. Travis gets to eat all these when they get left in the lab after testing (alas, he does not like green beans).

The have in-air sorters, on-belt sorters, an ultra-fast (1000 ft/sec) tobacco sorter (runs in an air tunnel). Their french fry sorter just cuts off the bad spots. They sort cherries by size. When they first sorted oranges, their airjet was so powerful it bored holes in the oranges instead of moving them. Some sorters use lasers instead of cameras as lasers can see glass and remove it. Travis showed a demo animation and a couple of movies of these sorters. The salad sorter Travis finds really scary to watch; the reject screen caught grasshoppers, cardboard and suchlike.

Q. Do you have samples? :-) No, just videos.

The machines cost $300,000 and last for years. They are not perfect but they are better than humans. This is not pharmaceuticals so they are not paranoid about perfection. The value is in percentage recovery and minimising friendly-fire losses.

They have a ‘soft realtime’ requirement. The core sort algorithm is written in C for historical reasons.They have a uni-node processor for 4 cameras; it is much easier than multiprocessing would be. They do a lot of p-thread stuff (the originally used ANX which has lots of cool threading but p-thread was adequate). They have 9 realtime priority processes, 3+ normal.

They used firewire because the push was to use of-the-shelf technology where possible. They used paid open source to get some drivers written and that worked fine. Isochronous Data Transmission is data that may have latency but you will get it with the timestamp of when it was sent. In the C program they use Boehm and Weiser conservative garbage collection (see http://www.hpl.hp.co.uk/personal/Hans_Boehm/gc/); this saved Travis a great deal of pain. They had some syslog() and stopworld() honing issues which they fixed.

The UI (in Smalltalk) has to do a great deal. It must calibrate the machine, tune it, troubleshoot it, configure it with the sorting recipe for this task. It is touchscreen-based (you quickly run out of space with windows widgets). They need the UI to keep the workers engaged as working in a food sorting plant is not the most motivating job so they make it fun and straightforward as much as they can.

He showed the UI. Everything is kept big (fingers are a finite size), some look and feel is inspired by Mac, multi-language support is important. They also have passwords for various functions. It was very visual and fun and gave good feedback of where things were happening. It was also multilingual (Adriaan commented that the Dutch translation was ‘a bit strange’ :-). Travis showed tuning the spot size for chips.

Normally they run inside a VNC manager without a window manager. They have a DLL to do zooming and pseudo-colouring. The rest is all custom widgets in VW: all vector graphics and caching. VNC also lets them restart X windows after changing its parameters without closing their application.

They are 100% VW because it is a wonderful ADE; this is what Linux needs. There are 84 packages of which 51 are theirs and the rest are SOAP and stuff not in the base image. They load parcels into the base VW image each release to get their image. They use SUnit tests.

They code in C., C++ (shells for embedded boards in cameras), scripts and Smalltalk. Smalltalk has easily the best thought-to-code ratio. They are an XP shop (SUnit tests, pair-programming and refactoring). They find VNC improves pairing. The have a planning game every fortnight. They have a fully automated build process; a tested build takes 30 minutes. The system monitors changes and does a rebuild whenever a given time elapses since the last change. Planning is tough, especially explaining to marketing that once it is built and mostly working, the lab testing to polish and make perfect still takes time and is very hard to estimate.

They use Linux everywhere, top-to-bottom, uClinux on the embedded boards, Debian derivatives for the UI and Sorter. Debian packages are used exclusively including for the VW base image and parcels. (They are not as hard to build as the manuals imply; the manuals push you to use policies.)

He once took down a plant because he lost a string in a pun. he brought it back up and asked how much did that delay cost you - about 11 million. Luckily, they are used to these plants going down at times and it was not deducted from his paycheque but its important that these things work.

Travis has uploaded a number of tools to the Cincom Open Repository: RBBetterRefactoringWarnings, ClassCloning, ExtraEmphases, SUnitToo and ExtraRBForSUnitToo, CoolImage, AutoComplete, LessProgress, ConsistentProtocols and DefaultPackageNamespace.

Their Choose Control system is how they build their ST system from parcels (Smalltalk does not fit easily in make files and such). It monitors multiple projects for changes. It has a one minute debounce to let changes settle, then writes all the packages it needs. A secondary ParcelBuilder converts the Store packages to parcels. Their 11,232 tests run in 10 minutes to authorise a package rebuild. They run it in a VNC server and look at it if they need to. They added UNIX cookies to make the build systems' emails more fun to read; then they had to sort the cookies to eliminate too long ones. :-)

He showed a recording of the VNC server running 10 images (control and 9 building, then testing). The KDE bar used to look amazing as 9 Store progress widgets fought to raise themselves every update; sadly, Randy used LessProgress to make it more sane and less fun to watch.

Building UIs in WithStyle, Michael Lucas-Smith, WithStyle
Version 4 is the pollock version of WithStyle. See my earlier reports (from 2004 on) for the background to this product. They are user interface enablers. They wanted to give Smalltalk the rich web-content interface abilities of some other apps, e.g. Gecko that Mozilla uses. They did not want to be a black box you throw an HTML doc at to get rendered.

Their intent was always to make products with this but they also make it available for others to do likewise. Along the way they've contributed stuff to the open-source community: XSLT and ICE libraries and etc.

WithStyle was first built in the VW Wrappers UI and hit the reasons why Cincom is replacing Wrappers. Therefore WithStyle 4 uses Pollock.

Desktop GUIs give the user a rich experience and custom widgets. Web GUIs give CSS-styling look and feel, and cross-platform familiarity. (VW's platform-look-and-feel approach says, "We'll make your UI look different on each platform"; web browsers say, "We'll make your UI look the same on each platform" - and succeed to varying degrees.)

Michael showed examples of using XHTML to make text flow around subwidgets and so on. Dabble DB and Piers use this to good effect but tree views cannot be handled in any consistent way. So he showed a Pollock widget for a tree view inside his XTML - and yes this was just what we saw a few slides back when we thought it was just a slide.

There are so many versions of the file dialog in windows he thought it would be a good demo to show how just changing the CSS could change the file dialog. So he did - "and I only changed the CSS."

He then showed pong (Smalltalk Coding Contest UI) as a CSS-controlled Pollock component. He set the CSS to change the background from black to green whenever he put the mouse over the game. He then demoed a pause and resume button. His slides became a (borderless - another thing you can do in WithStyle) mini-window and reappeared on click. Context-sensitive menus interact with XML events to 'go to slide' or whatever.

He then showed context-sensitive editing. The idea is that any XHTML for which you have a schema can be edited (and, in free form, any for which you lack a schema). He edited WSYWIG in WithStyle3, first basic XHTML and then his help doc whose schema is nothing like raw XHTML. When he moves the mouse over an element the border appears indicating where the element is. He showed deletion-enforcing: the menu shows them, controlled by the document rules, so delete on a title is 'delete title, step, ...' and so on to maintain the schema rules. All this was in WithStyle3 which works within the limits of wrapper. In Pollock, the limits are far off.

He listed the various standards they've tested with: the usual XHTML, XSLT, XML-Events, DocBook etc., etc., and Smalltalk UI XML, their own standard, and CSS3 (not really standardised yet but it has good ideas). See Michael's slides for the webrefs to these standards.

XML Namespaces: XML is a misunderstood language. It is so verbose that people think it can do anything. Then it gets a bad reputation when people use it for stupid things. XML is meta-information around the information - in essence it is a lisp S-tree, just commands with data. But it is multi-namespaced and you can cross namespaces within the same tree. You set up aliases, e.g.

```
<...
xmlns:ev="http://www.w3.org/2001/xml-events"
...>
```

The new VW Announcements framework replaces triggerEvents, sending an object, not just a symbol as the old events did. Announcements are things like Pressed, RightPressed, PaneExited, etc.; WithStyle has added Configure, PreLayout and PostLayout. WithStyle has a secure sandbox environment that prevents scripts from using reflection, sockets, etc. If you want a script to do that then you must provide a method to do that since you presumably think its OK. Otherwise, a dialog will oblige you to permit the violation.

XML-Events have two phases, the bubbling phase and the capture phase. The click on a widget travels up the tree from the target back towards the root encountering any observer node that wants to do something with the event, i.e. it ‘bubbles up’ the tree and gets ‘captured’ by an observer. In capturing, you can just write `smalltalk: ..smalltalk code..`, e.g.

```
ev:type=”Pressed” ev:handler=”smalltalk: ui helloWorld”
```

They have some script pseudo-variables: this. thisURL, ui, ... (They have used Smee. Quallaby has been bought by MicroMuse and MicroMuse by IBM so they are trying to find out how they can bring back Smee from its current lost-in-the-ether state.)

He showed some CSS2.1 for setting hover colours and suchlike stuff. WithStyle has added -ws-widget: property. It takes any Pollock widget as parameter (including custom ones you’ve made) and these now can be styled with CSS, made to flow to the left or whatever.

Q.(Dave Simmons) Have you looked at ZAML, thought about a translator for ZAML? Yes and yes; it may be possible but they must bed WithStyle4 down before doing anything (they have also looked at Zool).

Michael then showed us a page of CSS they might do. They pride themselves on having lots of tests and on conforming to the CSS standard. Try using the CSS ‘centered’ construct in Mozilla or any such browser; nothing will happen.

Q. How are the tests expressed? How do you validate against the spec? You read the spec again and again, and look at what existing browsers do.

There is one test in the world, the ACIDtoCSS test, that uses every bit of CSS there is to draw a smiley face. Opera passes and Mozilla does not nor any other browser (including he must concede WithStyle 3 and 4).

He walked down some code. One readable line launched a popup menu where you click (try that in Java script and see how long and how unreadable). He then showed the line that does the Pollock tree widget (note the call of `x subclasses` will cause a sandbox dialog). They map all the ids so they get true reflection to what they draw on the screen.

Q.(Dave Simmons) could you take ids into local variables and reference them directly? Michael thought this a good idea and will give it a try?

BTW, his code was simple and/or long-winded for easy demo explanation; in real use, he would factor the code better and have less of it in the CSS.) He noted that as he expanded the tree and selected items, the (Smalltalk-obtained) item name was shown (appearing, thanks to a WithStyle bug, a couple of pixels lower than it should).

Q.Use in existing browsers? Most browsers will ignore XML-Events and all browsers will ignore the -ws-widget tags.

Q. PDF rendering? Smalltalk PDF creators exist. They have not looked at it.

(Bruce Badger's SPDF for PDF generation is in the open repository. He created it for an insurance company who needed to be able to reconstruct documents. It has a test suite which will show you what it can do.)

Q. Why done this way? If you want to deploy both regular and rich application this can help. (Also it can be used by developers who need a rich set of tools and when debugging need to see how styles are being applied to things. Regular tools are now getting these abilities.)

Q.AJAX? AJAX-enabled web apps are the tip of the iceberg of what we should be doing.

Sames: Pollock does not offer rich text capability (a design decision); WithStyle is the place to go for this.

Wizard customers want custom versions of their screens. In the past, that would mean a development cycle. Now, their strategy is to plug XSLT and CSS, and use a web developer if they need to.

Exubery, Bryce Kampfjes, RMB
Bryce computer had a mechanical screen failure just before the talk. It was lucky I had asked him to give me a run-through half-an-hour before starting so he had time to realise this and borrow Avi's computer before start-time descended upon him. Meanwhile the run-though was much truncated, as my short report following indicates. (However Bryce' contribution to the proceedings was not slides but a full-scale paper on Exubery, so you can get more details there.)

Bryce has done and will do a number of small releases leading to release 1.0 in about a year hence. This will have everything working except 64-bit which will be in a 1.x release (32 simulating 64 should already work).

Exubery maps Squeak bytecode to native. (It does not map from syntax trees because Squeak has some imported utilities that produce bytecode directly not from Smalltalk parse trees; thus Exubery will work on these.) The aim is to make costly operations run fast by compiling just those methods, not necessarily the whole image. The Exubery code is held in-image, and regenerated as needed, the squeak code remaining the configure-controlled source (An Impara user wants to make the Exubery code persist so game-users see the speed from the beginning, not after 5 minutes, so this may be made available.)

Exubery uses a background thread to do slow compilation and is 15% faster than VW on the bytecode benchmark. It runs on Intel and other hardware. Mac may be a simple recompile and release or may take a week or two.

Karen Hope, Smalltalk to Java: the Good, the Bad and the Unbelievably Ugly
I only caught the end of this talk.

There are many specious reasons for translation; Karen went through them stating the actual situation. You will *not* use EJB or whatever in your translated code; it will be straight Java. You will *still* need Smalltalk-aware people to understand the code. The cost of an IDE is *trivial* compared to the cost of reduced productivity.

They saw some performance gains: Java ran some policies in two seconds instead of eight and generally the system was faster at the end. However the these comparisons are not quite apples to apples as the language-independent system rearchitecture affected them.

The executives were happy because they said, “Rewrite it in Java” and they did. The business partners kept being asked for more and more money but they too had bought into ‘Smalltalk is dead’ so they endured it.

Lessons learned: crappy Smalltalk code will smell 10 times worse in Java and there’s more of it. Good Smalltalk code may be bad in Java. Clean it before translating, and rewrite blocks as much as possible. If she did it again, she would have rewritten the module entirely, exploiting off-the-shelf components wherever possible, she would not have translated it; it would have been cheaper.

Q. What version of VA smalltalk? 5.0. They never went to 6.0, let alone 7.0. They were running under websphere.

Q. Were you translating Toplink itself? No, they found they were doing sufficiently simple stuff that they got away with just using JDBC.

Q. If you had not been told to use Java, what non-Smalltalk languages would you have used? We would have looked at alternatives but we would not have left Smalltalk anyway. Working in IT means you don’t make the decisions. Decisions are made by people who believe what they read in USA today: ‘Java is the wave of the future’. (Then she went to OOPSLA and asked an IBMer about the VAJava they were using and was told, “We’re retiring that.”) Rational thought is not part of the decision process. What a manager reads on the plane or in the bathroom is.

Q. If you had stayed in Smalltalk, how would you have overcome the performance issue? I would have had 6 engineers working on that instead of on the translation. I would have been able to go to VASmalltalk 7.0 (or e.g. Smalltalk/MT that supports multi-threading).

Q.(Heeg) Did these 75 developers still have 4 major releases a year during and after the rewrite? No, they certainly did not. (She noted there was a lot of other company churn going on which also impeded things.)

She resigned over the silliness of these decisions; she could not endure being architect of the Java system, after 3 years spent migrating during which nothing of value was delivered.

Coding and Testing Techniques

Efficient Smalltalk, Travis Griggs, Key Technology

Travis audience included VW users, one ST/X, a few Squeakers, a Dolphiner, some GemStone users and some VA users. Finance was the most common application domain with a few in each of science and engineering, IT, and academia.

Travis started Smalltalk to escape an oppressive boss, having previously done a lot of Fortran. Some years later, he attended a tutorial on this title at Smalltalk Solutions 1997. He has now picked up the torch to dispel the myth that Smalltalk is slow.

Preparing this taught Travis that many things that he thought he knew, he actually did not know so well. However John Brant told him that one often learns something new from watching how someone else solves a problem.

Smalltalk makes things easy because it has the best thought-to-code ratio of any language: you don’t need a long time between having an idea and expressing it in code. This is an amoral truth: you can build bad code quickly as well as good. In Siemens, he once encountered a single seven-page-long Fortran-style Smalltalk method. More often, poor coders make instant spaghetti; we’ve all met systems where it is very hard to work out what is going on where.

However it is easy to fix designs and in particular you can easily fix performance. When refactoring Smalltalk, Travis often obtains a factor-of-ten performance improvement easily. In C, he sweats for every 10%.

Early optimisation is the root of all evil. Nothing is slower than a system you cannot fix. (Alan Kay pointed this out to Apple when they insisted on doing their Pink operating system in “the only technology that will be fast enough”.) One reason above all other is that you cannot guess where the performance bottlenecks are.

- Travis spent a long time making logging faster in a ‘flogging the logging’ blog thread before he realised that Troy Brumley’s alternative approach had avoided being IO-bound and that was the real issue.

- Algorithms for the Sieve of Eratosthenes (not for ‘C rasters’ as I and others at first thought Travis said) can be made very fast but are usually small parts of large systems. Travis’ only college class was to write one - his was really fast - but that is not where using programs lose time.

Also, you tend to optimise in one context, then to reuse in another that alters the original performance.

It’s a truism that 90% of a program’s time is spent in 10% of its code. Travis has been in Smalltalk for 15 years but he still makes bad guesses about where a program will lose time. There is also the tent analogy: your tent bag must hold the longest pole. Break that pole in half and then your bag must hold the next longest pole; you may gain nothing by making the pieces of the original longest pole even shorter.

XP tests are of course very valuable for optimisation, not for finding where real operation is slow (test usage styles rarely resemble user usage styles) but for ensuring your optimisations do not break your system. We should plan for that final third of ‘make it run, make it right, make it fast’. Often, people just throw hardware at it and that is sometimes the right way forward and sometimes not. This leads to profiling.

`Time milisecondsToRun:` is the simplest thing that could possibly work. Travis gets more from this than any of the more elaborate profilers.

- Make your runs aligned with the garbage collector and/or leave allocation out of the loop unless allocation is what you are studying.
- Dotted reference is slow: `obj := Object.DependentsFields` is a lot slower than `object dependents`.
- `Time countIterations` and `Time milisecondsToRun:` live in different optimisation domains. Do not combine.
- Ensure your iterations don’t go into large integer range inconsistently. That really impacts apparent time.

Be aware of these when writing transcripts. Travis recommends normalising out your overhead.

```
Time millisecondsToRun: [repeats timesRepeat: [nil]].
```

can be a useful normalise to subtract from a run in which `nil` is replaced by the code you are profiling.

No profiler he has yet met is perfect.

VisualWorks has the TimeProfiler, MultiTimeProfiler, AllocationProfiler and MultiAllocationProfiler. There are useful class comments. The UIs are usable but functional - `TimeProfiler profile: ...` (Eliot: there is also the ObjectEngineProfiler. Three people on the planet know how to use it; demo at end of talk.)

Squeak is like VW.

The VA Tools PerformanceWorkbench has a fairly amazing range of outputs. In the past, Travis has had to write a method in a certain class to run it, but someone in the audience pointed out it also has workspace you can use to write the code to profile. There is also the profiler that displays objects and moves them closer together if they exchange many messages. This makes a great demo but Michael, Travis and others felt it had no real use. However Carl pointed out that its original purpose was to find distributed application objects that talk a lot to each other and show you the groups, letting you move them together. (Today, GemStone only finds that objects talk to their proxies so it needs integration with GemStone.)

The VA profiler beats all the others for detail of information offered. Some in the audience also used the KesMemoryModifier that shows differences in usage between two runs. VW has a class reporter but not a differencer.

Michael found the VA profiler good for colour-coding where the system spends time so you work to move that to the core libraries, out of your own domain. (Eliot thought that would sometimes be just shifting the blame. :-)

St/X just has message tally outputting to the transcript.

Dolphin has Ian Bartholomew's Profiler. It was good interactively but he could not see how to use it programmatically (but not doubt you could).

Brian Kampfjes noted that sometimes optimisation just moves the time from one place to another.

Eliot remarked that all of the in-Smalltalk profilers have Heisenberg effects. Specifically they all allocate to save the said data. They are another process so may especially impact multi-process profiles. The base VW profiler will not track finalization since that runs in another high-priority process. The profiler can be confused: e.g. if you are in a large integer primitive when the timer interrupt happens, the system cannot interrupt that so it reports after you return as if you were somewhere else.

Eliot also stressed the importance with the JIT (VW, VA) of eliminating the JIT potential. On VW, global GC will flush the native code cache. A scavenge can be triggered by one instance of a class. Do one of these and run the block with the repeat count set to 1, *then* run the real block and collect the time. This is the *only* way to get reliable timings. Otherwise you never know if the profiler ran out of JIT cache and thew it away. For long profiles this does not matter but you must do it for speed-hound work. And throw everyone off your machine.

Q. Workspaces have temps and that impacts? Yes, but it is OK if you avoid dotted references.

Next, Travis reviewed some patterns for performance: his examples were mostly VW-oriented and he would be delighted to be told of others. The patterns have some overlap and his estimates of benefit are rough.

Pattern 1 is a meta-pattern: recalibrate your expectations:

- dialect to dialect
- version to version
- platform to platform
- context to context

since optimisation tricks may not be reusable.

1.1 Everyone knows WriteStream is faster than concatenate? It is now five times *slower* in VW. Someone rewrote comma to be faster!!! WriteStream is now slower at growing. Presize your WriteStream and it is still two times faster, and another 23% if you just grab the collection at the end, but if you fail to do these, you are better writing commas (see his slides for code).

Travis spent 5 years whitewashing commas out of his code :-/

1.2 Smalltalk/X has an `at:` method. In smalltalk, `at:` is slow; we convert to small or large integer and compute the offset where C just dereferences a pointer (20+ cycles versus 1). He sometimes knows he does not need Smalltalk's safety blanket so he wrote an unsafe `fastAt:` because that would be faster wouldn't it? Wrong! He opened St/X and showed us the raw C code of `at:` and then commented it out to make `fastAt:` by dropping the smallint check, bounds check, etc., then demoed. It was sometimes slower, never much faster (five years ago it was always slower).

Eliot mentioned that he has lived through the change from normal to superscalar machines and they completely recalibrated expectations since to a first approximation you only pay for memory fetches and indirect fetches; to a second approximation, you also pay for conditional fetches. He wrote ten versions of the VM that all ran at much the same speed as these machines are almost JITs internally. Travis agreed: this has also impacted C ideas about optimisation (but that is another story). Brian Kampfjes noted that some optimisation patterns had inverted in their effect because in the past memory accesses were very cheap but now memory is cheap and memory accesses are slower.

1.3 Ward's old trick `#(1 2 3 4) collect: #squared` is cool but we know it's not fast. Wrong! Now it is only 16% slower.

Q(Niall) maybe have XP tests asserting order-of-magnitude effect. (This was discussed at intervals in the talk and afterwards; see Michael Lucas-Smith's remarks near the end.)

Pattern 2: use a different algorithm. Chiselling away at profiler results gets 5% here, 17% there. You can stop seeing the forest for the trees.

Eliot: performance of C compilers has improved by a factor of 2 although machine performance has improved 1000x.

2.1 Alan Knight told him of a large interactive diagram layout. The profiler only told them that they were spending time in the code they thought they should be. So they redid it recursively and it was faster.

2.2 Before the talk, Michael suggested to Travis the example of the Boyer-Moore algorithm (80 examples on website) on substring matching. It tests backwards so can always eliminate a match-string-length of characters. He tried to make his code show the effect but someone has put a primitive in VW that makes finding the index of a character fast so VW is still faster than his implementation. (Michael, please advise. :-)

Pattern 3: match your data structures to your problem's scale.

3.1 The Refactoring Browser's RBSmallDictionary uses a linear array instead of a dictionary hash. It is 50% - 0% faster for the first five entries in the dictionary. It is 2x faster for `do:`, 10x faster to `add:` new item.

Michael: we can change things in the background and should do this with dictionaries. VA did this at one time (3.0 IIRC), with abstract and concrete implementations of collection classes. Travis wondered whether it could detect it was needing to optimise for `add:` as against `remove:`, etc.

3.2 Stack: OrderedCollection is growable at both ends. Most uses only need it to grow at one end. He made a Stack that only added at one end. (Eliot: List has that implementation. Travis: yes, but List has dependents baggage (yet runs only a little slower than OrderedCollection).

3.3 Use a real object: turn your Array into an object. `person name` is 2x faster than `person at: 1` and `person name: 'Travis'` is slightly faster than `person at: 1 put: 'Travis'`. (Eliot: the two memory fetches for an array versus one for an object gives that 2x). Making a class is cheap in Smalltalk so don't be reluctant. We are often ready to use Arrays longer than we should. (I wonder if this array-to-class conversion could be offered as a custom refactoring.)

3.4 BCDObject for palindromes. David Buck needed to detect number palindromes (e.g. 39693). A first implementation created the palindromes in very quick math but was very slow to compare them. His BCDNumber was slightly slower at the math but quicker comparing made it 31x faster.

3.5 Distance-reduction matrix: a first multi-loop version took 400 secs in Smalltalk and 100 secs when recoded in C. He then represented the data differently by viewing the problem as a triangular dataset of row elements mapping to value distances, and it took 20 secs. Noone wanted to recode that solution in C because a dictionary implementation would be needed.

Pattern 4: Savvy math. Transcendental math is great. Sometimes you pay for it because the VMs still like to add normal numbers. NotLikeNumber numbers must deal with failure, coercion and retry.

4.1 Use SmallIntegers. They are highly optimised, no collector or allocator work, 3x faster than floats. Maybe floating point values scaled as integers would give enough accuracy for your problem.

4.2 Do higher generality first. 'The crap flows downhill' because VMs are written to coerce upwards. `100.0 * 10000` is 4x faster than the other way round. `100.0d * 100.0` is likewise 4x faster.

4.3 Limit large integers. Float * LI is 4x slower than Float * SI, and timesRepeat is 30-60x slower on a large integer. Use a reference value if possible. Or use doubles: one million iterations between two large integers is three times faster with doubles than with large integers. (Q. (Bruce) they are approximations so does it skip any? Not unless you go too high; he has checked.)

4.3 (Number duplicated in slides) Avoid fractions: `3/4` is seven times slower than `3 asFloat / 4`. Generally / and // are expensive in all cases. a / b < c is far slower than a < (b * c).

4.4 Zero is special. An unhandled exception is very slow. An explicit zero check is much faster than wrapping in an exception handler. (Of course, exception handling is generally best avoided in high-performance code.)

Pattern 5: make fewer objects. Travis has yet to find an allocator more impressive than the Smalltalk one. However you then initialize objects etc., Nothing is cheaper than nothing. "It has to be a good object to be better than no object at all."

Q. Bruce: are you in favour of lazy initialization? Not when it hides invariants between variables.

5.1 When moving rectangles, you can save 31% by using `moveBy:` instead of `translatedBy:` as you create two objects instead of three (and none in `fastMoveBy:`). Michael joked that class Point continually does pointless (point-full :-) creation of points so always change the x and y in the point instead of recreating whenever you can; he's seen this 5-6 times in the last few years. Travis agreed, subject to ensuring that it is safe to assign in the receiver, that noone needed that point to retain its old values.

5.2 Smalltalk collections create new collections (select:, reject: genuflect: :-) and this is great for prototyping and safer to avoid bugs. You can save time with `allClassesDo:` instead of `allClasses do:` and so on. You save an iteration and a collection. You can also combine expressions in a `do:` (this trades some loss in readability for the speed, of course).

Michael: Alan Knight's package ComputingStreams combines streams and collection to avoid the `allClassesDo:` protocol explosion. Travis agreed it was much more readable but he's not yet profiled it.

5.2 `SomeCollection asSortedCollection first` (or `last`) is a lot slower than `someCollection fold: [:a : b | a min: b]` (gain 10x for collection of 100 elements, 15x for 1000 elements).

5.3 He has seen a system with a 20-tree deep 730 MB allocation of Filename>>construct: calls through using the wrong type of intermediate streams (needed the allocation profiler to find this).

5.4 Reuse large objects: a system was using 4Mb of FixedSpace because BytesArrays are zeroed (Eliot: the non-zeroing primitive is in the VM) and it took 150ms to allocate so they used a registry to recycle them. (Travis tried hacking the VW VM to make byte arrays be initialized with nil, not zero, and the VM mostly worked but one or two things relied on those zeros so he pulled it. Travis still thinks that compatibility with ordinary Array initialization would be philosophically better.)

Pattern 6: Do expensive things less often:

6.1 RBParser scanToken has a great comment and is a large case statement. John and Don moved the most common case on top after profiling to find out what it was and this made it 2x faster.

6.2 For backwards compatibility, ExtraEmphases needed to check whether a message existed `...pt y] on: MessageNotUnderstood ...` (Michael dislikes this as MNU may be raised within the call of `y`) versus `pt respondsTo: y`. The first is 42:1 when y is not understood whereas the second is 1:1 so it depends on whether you think the exception will be thrown often or rarely.

6.3 BCDInteger needed expensive // and \\ so David Buck created a lookup table cache.

6.4 When implementing code to brighten his image, Travis found that using `replaceBytesFrom:to:with:startingAt:map:` with the last argument a lookup table was many orders of magnitude faster than computing and assigning each value within a `keysAndValuesDo:` loop.

6.5 Avoid hidden repetition. Extract a group of operations to a method called once in an iterator instead of repeatedly iterating for each operation.

6.6 Quit switching collections. Collections have a lifecycle. Sometimes OrderedCollections grow only while being created, then are sorted, then are sent `asArray`. You can do this faster via

```
ws := Array new writeStream.
... do: [:each | ws add: each].“growth”
array := ws contents.
SequenceableCollectionSorter sort: array using: ...
“fixed sorted array can now be used”
```

6.7 Cache expensive resources: when drawing a rectangle, it is faster to use a hash pattern than to draw each line but pixel resources are expensive so cache and reuse the hash pattern.

6.8 Cache repeated computations to their outermost using scope. Cache loop computations in a temporary. Cache object-local computations in an instance variable. Cache object-shared computation in a class variable.

Pattern 7: Skip using frameworks to swat flies with howitzers.

7.1 ExternalStream versus IOAccessor. ExternalReadStream is well-protected to be many things to many people. In some cases, using an IOAccessor directly is 100 times faster for reading contentsOfEntireFile.

7.2 Drawing text. ComposedText (was called Paragraph and should still be so people would realize what they were using) is a very powerful, multi-line, etc., utility. VW therefore created Label for uses that only need one line, etc. You can use GraphicsContext>>displayAt: if you really need the speed; it's more brittle and less expressive but it may be right for your case.

7.3 XMLParser: Travis was querying the creation date of multiple files. He found that `skipThrough:upToAll:` to find the one tag was a lot faster on his 200 files than creating the whole XML parse just to find that one tag. (Niall: you see a similar effect if you condense changes, parcels and/or source between 'XML' and 'Chunk' in VW7: for ordinary browsing, the speed difference is invisible but if you need to scan all the source in a huge image, it is quite visible. The speed difference in low-level access is 10:1.)

Pattern 8: Inline, Inline, Inline:. Messages are fast but not free. Inline to get to the messages that actually do things. Inlining can make it possible to cache things. Inlining can sometimes make code easier to read. However you do not usually get big gains.

8.1 Different ways to check for nil have different costs (see slide table; the largest difference was `nil ==` versus `ifNil:ifNotNil:` which had a 22.21:1 ratio). But be aware that proxy systems may need a method check and think about whether they could appear in your code. He also demoed that an argument block ran faster than one without that had to close the block for a reference, i.e. `myval ifNotNil: [:val | val yourself]` is faster than `myval notNil: [myval yourself]`.

8.2 `to:do:` or not `to:do:` You can gain by rewriting iterations from `array do: [... e to 1 to array size do: (array at: e)` (Michael mentioned `array basicAt: e`). Someone in the audience mentioned that recently VW have made all the iterators fast so `to:do:` no longer buys you anything.

Pattern 9: get primitive: generally, getting close to primitives is faster. Sometimes what a primitive actually does can surprise you; Travis likes to look at the source when he can. Expectations may need revision.

9.1 Not all primitives are equal. `diamond displayPolygon` is slower than creating a mask of the diamond and drawing that. Kent used to use `displayWedge` as the example but Travis found that if you are a developer checking this and then shipping to users who use VNC everywhere, it will not be faster for them but much slower.

9.2 Make your own primitive: this is a last resort as deployment is then harder. There is an implementation of FloatArray (NumericCollections versions 5 or 16) that takes less memory but can still cost more in call overhead versus C.

Pattern 10: Make it *look* faster: involve your user in the process and give them feedback. Keep your progress-bar implementations light and no quicker than the user can see.

10.1 Windows flying folder copy is 20% *slower* than command-line copy (he has checked) but it *feels* faster. Store progress dialogs force redraw on every IncrementNotification. Travis implemented LessProgress to make it refresh no faster than 4Hz: it saves 9%. Store can spend more time calculating what 100% of an operation will be than doing the operation.

10.2 Pathological Notice Dialog use: he wrote a loop that ran in 0.2 secs without, and in 10 secs with, increment notifications. With display of them, it took 150 secs. With LessProgress it dropped to 23 secs, still too long. Using a value holder to instrument progress dropped it to 0.27 secs but was tedious to code. (Further tweaks and times on slides.)

Michael: I and my team will put all our optimisation assumptions into unit tests immediately I get back to Australia so we can test our assumptions. I've realised during this presentation that one is probably wrong and maybe others are. (Eliot: It's my job to make sure you have to reset your expectations on every release.)

Travis thanked Ken Auer for giving the 1997 talk, Instantiations for giving him VASmalltalk to test, Claus Gittinger for suggestions (others named in talk above), and Randy and other co-workers for ideas and review.

The VisualWorks Object Engine Profiler, Eliot Miranda, Cincom
At the end of Travis talk, Eliot demoed the object engine profiler. He uses it to see what is going on in the VM. It just measures time and it shows you *exactly* where time is taken. (Ralph Propan added methods for other stuff.)

It is in goodies because the UI is an embarrassment to him. He, like Travis, likes to profile programmatically. He created an OEProfiler instance and inspected it; `self open` showed us the embarrassing UI (and yes it was rather hard to tell the difference from the inspector). He then asked it to `self profile: a block` (the start/stop/clear profiler buttons changed) and we saw a graph with very illegible names of c-functions. This is useless and noone gets further unless they know to select all in the left pane, select the + low pc (and avoid the usual demo hiccough; Eliot got a DNU, had to quit and restart the image, then carefully avoid having the inspector over the main UI). Then he zoomed in to make it readable and so investigate where the time is going. He showed FindMethodForOpenPIC being used a lot for his ‘find allImplementors’ block as every send to the metaclass was different so the PIC kept being missed every time (see ESUG 2000 report).

It can also show the integral of the examples on the same graph and that can be more useful than the peaks. Invoking ‘display costs’ from the popup menu will show you a text window with percentages and values for each call and is usually more useful than the UI. He compared profiling a ‘find references’ block to his ‘find implementors’ result.

He then opened SystemBenchmarks and profiled one of its test methods that used DLLs etc. CompactedCode is a bin to hold stuff that was run and then got thrown away to make room for other methods. OtherCode has the system DLLs and it is not helpful just to have them as a single value. They plan to extend this to make it say more.

Q. Many of your percentages are greater than 100? That is why I use a small font. :-) If you have a lot of OE stuff going on ...

Advanced Seaside Tutorial, Avi Bryant and Andrew Catton
Avi and Andrew had expected a modest number of Seaside users. However the talk attracted a lot of non-Smalltalkers, so Avi introduced Seaside.

Much web development focuses on statelessness, well-formed URLs, etc. Seaside does none of this. Avi will be defending these ‘web heresies’ later this year at the OS conference. Meanwhile, he agreed these were all good things but invited us to see what we could get if we abandoned them all.

He started with the counter app. Its `renderContentOn:` is much like any widget builder but those who have done web development will notice there are no URLs and no recreation of state, just an instance with state (the integer counter) representing a piece of the page. He then showed a multi-counter built from several of these. Because the callbacks use blocks, we never have to worry about which counter this is in the page or how many there are on the page: all that is handled for us. Avi also showed the halos (plus usual demo hiccough: their icons were not found on the demo laptop).

`call:` shows the next component and puts the caller on the stack. How do we get back? Every component also implements `answer:`. This pops its caller off the stack. Its parameter is the return value of the original call. The tutorial example was to build an AJAX-using to-do list with drag-drop, etc. Andrew took over and modified the counter demo so increasing the counter returned a starting component for the to-do list; soon the multi-counter had some parts as lists while others were still decreasable counters, etc.

Seaside has chosen to stream out HTML instead of a DOM tree. This means some blocks need some things to come at the end, e.g. the `text:` call in the example’s code has to be last in the block. That used to be a common requirement and it does help you build a large page. It might of course be viewed as premature optimisation.

After a break, we AJAXified our to-do list. In typical web apps, every time the user does something you get a full reload of the entire page. AJAX allows interaction with the server without a full page reload. An example is GMail, which loads a ton of Javascript. Issues are that the back button no longer has a history of every action, the error reporting is not as good, the user just sees silence if it fails often, and you must consider how well your web app will work with an old browser.

Andrew then showed the good side. He submitted a to-do item without page reload, adding an item without page reload and (this one got oohs and aahs from the audience) dragged items to change their order in the list. Visit scriptaculous.smalltalkhosting.st to see CS styles for this, etc.

To use AJAX you must trigger callbacks, you must render whatever small snippet of HTML the client is to receive and you must tell it where to put it, i.e. you must tell it the ID of the item you intend to replace. To set IDs you can send the `div:` selector to anything. You can have a method to return an id, usually a meaningful string, which you then use in various places. If you are iterating a loop, you will need to acquire a unique ID. Usually this is a meaningful string followed by some uniqueness garbage. You can use `html nextID` to get a unique ID; store it and use it.

Avi noted that caution and scepticism is sensible about the whole AJAX premise. They advise using it lightly for small effects. A lot of users get very excited about it.

When you e.g. check the checkbox, a Javascript handler gets all the check boxes, send them to the server so their callbacks get done, then does the overall callback, then the server sends the answer which the browser puts into just that item. First you do the model-part callback, then you do the render-part callback.

Andrew looked at reordering; he can just mouseDown on an item and drag it, other items rearranging around it as he does so. He showed two lists, dragging and dropping between them as well as within them. This is all using scriptaculous code:

```
SListView>>reorderItemsOn
...
html unorderedList
  script:
    [html ajax
      triggerSortable: self itemlistID
      callback: [:v | list items: v]
  with:
    [list items do:
      [:each |
      html listItems
        passenger: each
        with: [html div...
```

He also showed the scriptaculous AJAX Javascript library (which is in Smalltalk so it can be demoed when offline). There is a 'ridiculous' amount of Javascript (100kloc) because Javascript needs a lot of code to do anything. "This is what we are trying to save you from." An audience member pleaded to go back to the Smalltalk as "the lesser of two evils" :-).

They could make the Smalltalk components very clean, e.g. here are some unordered lists you can sort, but this has not yet been done as much as it could be since in practice people want to customise and the abstractions are still emerging (some of this code is only a few weeks old). The ugliness of Javascript shows through in places. IDs for example: the rest of Seaside hides such things. A seaside framework to hide them here as well is possible but not yet written.

Andrew then showed the very small amount of code the example needed. The server needs to replace the items in the list. The passenger notes that the reordering of the list maps back to the model reordering the Smalltalk items list. `onFailure: stringOfJavascriptCode` handles problems on the client side (no you cannot provide a block, alas). The scriptaculous code is not in SqueakMap yet but soon will be. It is in the Cincom Open Repository, thanks to Michael Bany.

More often than not, AJAX is used as an optimisation.

Q. Date-related example? Yes, see the 'short' related Seaside components.

Q. How do I use a Javascript component I find? The script block returns Javascript; you can just provide a block that quotes the script. The best Javascript libraries slot in very easily.

Andrew showed how they do autocompletion in the SUautocompleter:

```
renderContentOn:
  callback: [...
  onBlur: ...
```

The `onBlur:` means that whenever you unfocus, not only whenever you submit, the callback is triggered. The callback just captures the text input to an instance variable and the autocompleter then acts (`onBlur:` is of course the opposite of `onFocus:`). You can overuse effects but some, e.g. highlighting the last item added, are useful and easy to do.

```
html span
  script: [html effect fade duration: 2];
  with: items last description
```

Q. Persistence framework (Ruby on Rails style)? Seaside considers persistence an orthogonal concern. People use Glorp, OmniBase, GOODS, or careful use of image persistence. Often you work for 8 - 10 months just using the image and then add persistence. This keeps you agile and they find persistence is not hard to add when needed. Image persistence means that each customer's data is in memory when they are using it and on disk the rest of the time. It works for smallish well-partitioned datasets;

otherwise use a database. Pier and Magritte provide a lot of Ruby-on-Rails-equivalent scaffolding; see Lucas Renggli's talk (which I missed, but a write-up of his talk on Pier at ESUG 2005 is in my report).

Ruby on Rails is a clean dynamic approach to the old traditional 'deal with each request separately and keep your URLs clean' style. That style has some advantages but it obstructs rapid start of development by forcing you to worry about these things up-front, it makes reuse hard and it causes coupling between elements on a page. They went through a new iteration of their UI every day for weeks and they could never have done that in Ruby on Rails. If what you are building is fairly straightforward and you know ahead of time what it will be, RoR is OK. Agile developments are better with Seaside.

How do you coordinate web designers and developers? Coordination is at the level of CSS classes. They agree the basic semantics of the page with the web designer. Their designer tells them, "You need 10 CSS classes." They gradually add those classes to their evolving HTML. (Their designer lives in the UK so that helps the separation of concerns; they can talk at certain times of day only. :-)

Q. Cross-browser CSS issues? Comments add extra CSS to make IE happy (and you need a good web designer who knows the issues).

Q. Unique id issues? You can use the extra info you can add to the URL to reconstruct an expired session.

Q. (James) You could hold the Javascript, templates and CSS in files if wanted? Yes, and we do with CSS so the designer need never see the Smalltalk image if they so wish.

Q. (James) Custom Javascript? That is the designer's domain. They could ask us to add specific handlers. (This assumes your designer is happy with Javascript which is not always the case, as Jim agreed; his blog Javascript does not work on Safari for example.)

Bruce has tried to switch Javascript and CSS at the root but he finds it very hard to work back far enough? Avi has not tried that yet so cannot advise, but thought it might be a good idea.

Controlling Pain, Blaine Buxton, www.blainbuxton.com
In this talk, 'pain' means code assumptions that will come back to bite you. Pain comes from tight schedules, from broken contracts, from unclear vision, etc. SUnit can be used to enforce team decisions and contracts.

Blaine's team were not always remembering to ensure their files were closed. So he wrote a test that did an around check that they were. Another test checks that no `self halts` or similar have been left in code. He found that using the RewriteTool to parse and verify the pattern was better than raw code handling. (I find the same.)

SUnit and Lint (and the RewriteTool) can find many bad code smells. In practice, he tends to run basic SUnit all the time and Lint-invoking tests before packaging. Combining these is easy, e.g. (see slide)

```
self assert:
  (CompositeLintRule ruleFor:
    BasicLintRule ...
```

He created a convenience RTLintRunner to make this very easy.

MethodWrappers replace a compiled method in a class method dictionary with a wrapper that will call the wrapped original but also call before or after methods. These are useful for code coverage, encapsulation violations, timing constraints and legacy code. Method wrappers in the bowels of your legacy code can check arguments and state to make it safe(er) to create the seams that make legacy code testable.

He showed an example for coverage. Grab the methods you care about, install wrappers on them then run from the top, finally ensuring you uninstall the wrappers again to leave you image in a clean state. (You must be careful to ensure uninstalling your method wrappers, and it is wise to have a test that checks no installed wrappers are in any code you are about to publish.) If any wrappers were never called, fail the test.

Sometimes programmers move their code to the GUI where (because :-) it is hard to test. MethodWrappers can help to test such things.

SUnit and metrics are your canaries to take into the legacy code mines. Metrics have been ill-used in the past to manage people, to demand they make code meet target values and suchlike silly things. Instead, you should use metrics to prompt refactoring sessions. The high-scoring code gets looked at for refactoring. A refactoring session is more positive than a code review. His team use cyclomatic complexity, average message sends and coupling. You must keep the metrics simple; more complicated metrics do not correlate any better with the problems that you are actually studying.

Coupling is just the count of messages sent but not implemented by a class. This is great for finding ‘god’ objects. Average message sends are interesting when low; it reveals dead data objects. For high values, he looks at methods with values greater than 12 (some say > 7 should be looked at). The team must choose the metrics. Anyone fit to be a coder can get around metrics so *never* enforce them. Let people choose metrics to help them to understand their own code.

SUnit can also be used to check source control. You can test which areas of your code are volatile. This can detect bad code, areas where people are tending to overwrite each other, etc.

Squeak and VW have pragmas (not VA yet). It lets you add meta-data to your methods. This makes it easier to detect deprecated methods, to verify that all public methods are called. Blaine finds pragmas are better than categories.

To sell reflective testing to your team, use team consensus (especially about what metrics to use and how to use them), start simple, be positive, and have refactoring sessions. Let your team know what you are doing (five minutes after he released his 'ensure files closed' test, an angry colleague was asking Blaine to explain why there were three violations in code he was trying to release). Be sure to put good comments in these tests so that people know what to do to fix their code when the test fails.

Others have done this and written good books on this. Blaine listed one from 1983 and three by Stephane Ducasse and colleagues (see slides).

Q. Does test code ever itself need tests or become brittle? Yes. They had been excluding their test cases from their test scores as they were the top offenders. Sometimes this is legitimate because of what the test is doing, but generally you must be ready to refactor your tests as needed.

In discussion, I explained that wrappers can help test UI, and hard-to-reach areas in general, not only via before-after behaviour but also by altering behaviour within the test. If a pop-up-dialog-opening method is called deep within a test, a wrapper is a handy way of making it return the value the test wants instead of showing a dialog. Similarly a wrapper can make the very last stage of preparing a refactoring return the refactoring to the test for study instead of compiling its code. (Examples are in the custom refactoring project downloads.) Someone argued for policy objects instead of method wrappers so, e.g. the dialog class would have a settable policy allowing it to do what the test wants instead of what it normally does. I argued that the method wrapper was itself a policy (i,.e. an alternative to the default policy represented by the compiled method in the class' methodDictionary) but less intrusive to production code and/or code not under your control, and easier to tailor to the specific needs of the test.

Acceptance Testing in Smalltalk using FIT, FitNesse and FitLibrary, Randy Colman, Key Technology
Randy's talk was in the parallel Smalltalk stream at the same time as Bryce'. Just like Bryce he had a laptop meltdown on his screen just before the talk. :-)

He used the game of mastermind as his example (player chooses a four-colour code, opponent makes a guess and is told how many colours are right and how many are also in the right place). He has converted the game to FitNesse tests and put this on the FitNesse wiki.

Ward Cunningham created a Framework for Integrated Tests a few years ago. Tables drive code that convert their data into tests. Ward's idea was that customers and business people know how to make tables in word documents. Later, FitNesse made a wiki wrapper round FIT as people can also write on wikis. Rick Muggeridge then created FitLibrary to support it.

Randy supports the VW implementation, David Buck is working on a VA port and there are versions for Java, .Net, C/C++/C~, Python, Ruby, Perl, PHP, etc. The idea is that your acceptance tests do not change even if you change the implementation language.

He showed the mastermind tables. A test entered a guess and checked that the feedback keys were correctly set. He ran it and got feedback: line went from white green. Fixtures subclass the appropriate generic class (e.g. ActionFixture) and implement `signatureFor:` to provide type adapter information (he would love to drop this type info but has not found how to yet). Then you implement the methods you need. All fixtures are table types. He showed us the fixture class that starts the game, and the very simple table that uses it at the start of his test.

ColumnFixture is the simplest type. Each row is a different scenario, with columns e.g. ColumnFixture subclass: FullGameFixture might have columns guess, response, game over? and score? The class names (fully qualified as needed) are typed into the table headers and fitnesse finds them. `signatureFor:` maps the strings immediately into domain objects.

The start game action test could never fail (looked like `setUp` action) and kept its game in a class-side variable.

You can use ActionFixture to do GUI tests but the result is verbose and brittle; he does not recommend it.

RowFixture lets you check the collection of results; order is unimportant; you must add an order column if you wish to use it. RowFixture needs a target class defining the type of the collection and a query method that returns the data to be checked.

Thus the core fixture classes are simple wrapper classes calling into your domain classes.

Q. These tables are created how? Edit the wiki in the usual way to create tables. The wiki image (the fitnesse server) produces a file the development image is watching for; when it sees it, it parses it to create Fixture instances from the tables, thence run the tests.

The FitLibrary adds DoFixture, SetUpFixture, SetFixture/ArrayFixture, CalculateFixture. The first fixture on the page controls how the rest of the page is interpreted. He showed us the initial, verbose test rewritten to use DoFixture and SetFixture, making it less verbose.

Fitness has the concept of a setup page. DoFixture normally will be the first fixture, so must be in the set up page if there is one. The DoFixture first table creates the DoFixture and makes it the controller of all other tables so he was able to add an instance variable game and retire the shared variable. All subsequent tables that lack Fixture class headers are handled by the DoFixture. DoFixture uses auto-delegation: it passes any messages it does not understand to its instance variable(s).

Q. Safety? Could this call unintended and/or undesirable methods? There is some checking but not much, e.g. if you use a method name the same as a fixture name you will be asked which you meant. DoFixture>>calculate treats the rest of the table as a Calculate. Other methods are check and ensure/not/reject (I noted regrettably different terminology from SUnit).

(Notation: Rick and he italicise selector keywords in keyword-parameter expressions in the wiki text for readability.)

A SetUpFixture does what you would expect. SetFixture and ArrayFixture supply Sets (order unimportant) and Arrays (order important; no other difference from Set). The auto-delegation means when he wants to treat the result pegs as an IdentitySet he can just provide a SetFixture wrapping it with no need for a SetFixture subclass.

A CalculateFixture separates expected and result columns so you do not need the ? terminator for the latter. He showed us the CalculateGameEnd subclass, using a helper method to get a game whose completion status could be tested.

He ran a test that errored, then worked on the code. He ran it repeatedly, successively creating the instvar, the accessors and the type signature required before it would run. After he added all these, the wiki no longer showed debug lists; instead it showed a failed test.

In Java and .Net, they have to do a lot of work to make a failing test show up in the debugger. In Smalltalk you just insert a breakpoint and run the test. After more work, more of the table entries were green. He dropped an `asArray` and got a debug table again (stack dumped into table entry which resizes to show it). A type adapter problem was the last hurdle to overcome (response value was a domain class, not an array as he had thought) so he added it to the `signatureFor:` data.

Q. Although the test data is cross-implementation-language, you must write fixture subclasses in each language to run the test? Yes.

Q. If you are just using Smalltalk? Use SUnit, the tools are better. This is good for customers who are already used to it when a Smalltalk application appears, or who are happier with a wiki and tables than with reading Smalltalk code (or who have tests that fit tables especially well).

In summary, it is a way of developing a domain-specific language to communicate with your users. Since my impression was that FIT tests are more verbose and rigid than comparable SUnit tests would be, I agree with his recommendation to use SUnit when you can. If a customer knows FIT, it is good to have Fitnesse available.

A Regression Testing Tool written in VisualWorks Smalltalk, Emil Markov, Ontario Teachers Pension Plan
BRITE (Best Regression Interactive Test Environment) has been is use for some years at OTPP and has been a huge part of their success.

OTPP covers all public school teachers in Ontario. It has 60 developers, testers and managers. They use Smalltalk on the desktop, a couple of Java applications for webservices running on websphere, and AS/400. BRITE tests all the Smalltalk and some of the Java via RPC, and now they are using it to test webservices.

What is wrong with SUnit? Nothing - if you have a programming background. Most of their testers have an in-depth knowledge of the business and no programming skills. They know the function they are testing and what the result of these inputs should be and they do not want to write code. The developers write SUnit tests for infrastructure functions but BRITE's users do not use SUnit.

In the late 90's, there was nothing like BRITE on the market. BRITE was written in 2002 and open-sourced in 2003 (and presented at StS2003, but I missed that talk so it is not in my report of that conference).

BRITE has TestSuites containing TestCases containing TestScenarios which have test inputs and outputs. All these are executable entities.

Additionally, the design started from a 'business function' concept idea. A meta-data descriptor for the test case defines the test class and method, and describes the types and names of that test's parameters. He showed the UI: a business function has a name and text, and contains test cases. The latest run history is captured. Business functions are not executable entities.

He selected a TestCase and opened a browser on it. Its main content is a set of scenarios. (Outputs can be included or excluded from regression test.) The left pane shows the tree of runnable objects (TestSuites containing TestCases containing TestScenarios).

They have created a Web Services Business Function editor. You tell it the files for the WSDL and the rest is as before. He showed us the WSDL model for the persistence model. When they connect to a data source, they suck up all the XML and then create the data and application models required. BRITE needs no database; everything persisted is just XML.

He demoed, launching VW 7.4 and starting BRITE from the menu. He created a new web services business function and found the WSDL over the network. You set the package and namespace where the WSDL classes will be created and then it generates them. The operations now appear in the drop-down list. He chose one (the GoogleSearch spelling checker webservice) and created a test for it.

The data for the test parameters was then entered in a table (there were resemblances to the tables in the FIT talk). Rows can be duplicated for easy editing. He ran the test, realized he'd misspelled the access key, and reran. The failure was now indicating that the user must accept the initial returned values as valid as part of creating the test. He did so and the test ran green.

He used the shortcut navigation to the test classes to show the code generated by defining this test.

Any item can be cloned. He cloned and added rundate to the outputs. Since the date will be different tomorrow, he excluded the date from the test.

Q. Use `Date today` as comparison value? You can go to the Smalltalk code and do that. You must remember not to overwrite with regeneration (or else to move it to a linked method).

The code to generate WSDL is the Cincom parcel integrated with their system. Code is generated with straightforward `expandMacro*` calls on class and method definition templates. They use the appropriate RBChange subclasses and the RefactoryManager to map changes into the system.

The web services testing is new (just the last few months). For WSDL, they need to improve the type handling: a complex type has to be flattened into an Array to pass to the test code; this will not be hard to fix. WSDL-defined exceptions and faults need more specific handling. All test code is saved in the repository but the test data is saved in configure-controlled files; he would like to integrate these to a single location.

They also want to add things to help their users. Tests should not need to be all loaded, with the possibility of system impact, to test one function.

BRITE is open source. The Cincom Open Repository version is old but will be updated soon, as will the similarly old version on Source Forge.

The development team write parametrised tests to start the system and get outputs. The test team then load and add inputs and outputs to create tests. There was discussion re integrating with SUnit, and re offering menupicks to let testers view expected and actual values compared in the differencer, and/or to write the actual to the expected to be the basis of future test runs.

Cryptography 2, Martin Kobetic, Cincom
Martin ran out of time 2 years ago in Seattle so this is the second half of his talk. Much new has been done since then which you will see - until he runs out of time again.

Symmetric algorithms are 32 bit register arithmetic operations. Public key algorithms are more complex as they are derived from known hard computational problems: prime number factoring, discrete logarithms. The mathematics is not that hard. It needs to be conceptually simple since for security you must be able to reason about all possible modes of failure.

Symmetric cyphers use a secret key that the two parties must share. If there are more than two parties you need more keys unless you are happy that everyone can read everyone’s mail. Sharing keys conflicts with the need for security.

Asymmetric algorithms have a public key to encrypt and a private key to decrypt. Thus anyone can use your public key to encrypt something they know only you can decrypt with your private key. It also means you only need one public key to receive messages from many people.

RSA was released in 1977. Let the modulus value n be the product of two large primes p and q, public key e be a prime relative to p-1 and q-1, and private key d be 1/e mod (p-1)(q-1). Anyone can send you $C = M^e$ mod n (where M is an integer, less than n, representing the information to send; in VW, use ByteArrays). Then you can recover $M = C^d$ mod n.

You therefore need to generate keys with these properties: e is often set to 65537 (which is smallish but performant and safe provided your other values are set to correspond, i.e. d will be much larger).

He inspected examples:

```
keys := RSAKeyGenerator keySize: 512.
alice : RSA new publicKey: keys publicKey.
msg := ‘Hello World’ asByteArray encoding: #utf8.
msg := alice encrypt: msg.

bob := RSA new privateKey: keys privateKey.
msg := bob decrypt: msg.
msg asStringEncoding: #utf8.
```

If public key is so great, why do we have symmetric. Because it is 3-4 orders of magnitude slower. So people use the public key to encrypt a one-time symmetric key, e.g. `key := DSSRandom default next: 40 ..`, and then attach that to their symmetric-encrypted data.

DH cannot be used for encryption or signing but it can create a common shared secret between two parties over an unsafe channel. You need a large prime modulus p (usually > 512 bits) and smaller q (usually > 160b). and generator g (order q mod p). Private x is random between 1 and q - 1. public y is g^x mod p. Public y’ is the other party’s $y = g^{x'}$ mod p. The shared secret is then y'^x mod p which is the same value for both parties.

```
key := DHParameterGenerator m: 160 |: 512.
alice : DH p: gen p q: gen q g: gen g.
ya := alice publicValue.
ss := (alice sharedSecretUsing: yb) asByteArray.

bob := DH p: alice p q: alice q g: alice q.
yb := bob publicValue.
ss := (bob sharedSecretUsing: ya) asByteArray.
```

Diffie-Hellman can be done offline with one party publishing their public value where anyone, including the other party, can see it.

The most important use of signing digital signatures is to see if anyone has tampered with the content; you can’t prevent it so you need to spot it. It can tell you who you are talking to. It can provide evidence for non-repudiation (which you should consider the value of in a given legal or social context).

Hash functions are used for data finger-printing. They must be unable to reconstruct the original data content and yet *very* unlikely to collide. Someone in the audience claimed to have seen it once, which Martin compared to the probability of being struck by a meteorite during his talk. (He recommended that person to ignore lottery tickets; he has used his improbability quota. :-)

MD5 is now becoming breakable in this sense. Martin says you can still use it today but the researchers are working hard on new functions. SHA-1 was broken in 2005. Its variants SHA-256, 380 and 512 are harder but researchers are seeking new cryptography with basic structure differences.

You can use RSA for signing. You hash the plaintext and encode this digest. Then you encrypt the digest with your private key. To verify, the recipient decrypts the digest with their public key. The API is straightforward, e.g.

```
| keys alice bob sig |
keys := RSAKeyGenerator new keySize: 512.“or larger”
alice := RSA new privateKey: keys privateKey.
sig := alice sign: 'Hello World' asByteArray.

bob := RSA new publicKey: keys publicKey.
bob verify: sig of: 'Hello World' asByteArray.
```

Digital Signature Algorithm (DSA) resembles Diffie-Hellman with somewhat more involved mechanics.

```
keys := DSAKeyGenerator keySize: 512.
alice := DSA new privateKey: keys privateKey.
sig := alice sign: 'Hello World' asByteArray.

bob := DSA new publicKey: keys publicKey.
bob verify: sig of: 'Hello World' asByteArray
```

Martin used his slide display utility so he could select and run his code on his slides. He demoed creating a file using an X.509 certificate. This is recent from 7.3.1 - 7.4, along with ASN.1.

Q. Advantage of writing in Smalltalk? The OpenSSI protocol is involved and our API is straightforward. We are no slower except as regards symmetric side and hash functions and we trust we will solve that issue.

Q. Any additional licensing needed for commercial use of anything in VW? No. Almost all patents have already expired. One will expire in 2008 but would have to impact the US government before you needed a licence anyway. Elliptic curve variants are new stuff which will be very useful in letting you use shorter keys for the same security but have been described as a patent minefield. CertiCom (a Toronto company) owns many of the patents and intends to loosen them slightly to get industry traction.

Q. Export restrictions? Cincom’s lawyers say no; the rules have been relaxed. Some ‘let them know’ requirements are all that remain.

BoFs and Contest

Smalltalk Coding Contest

Michael recommended that the next coding contest schedule more time for phase one and have a simpler problem; he felt his had been challenging. All the more kudos to the winner, Andres Valloud, whose blog describing it is here: http://blogten.blogspot.com/2006/04/regarding-pong.html (IIRC, runner-up Blaine Buxton also has a blog entry but as I type this his blog archives are having problems; work back from blog.blainebuxton.com).

GemStone BoF

A new symbol-specific Gem improves symbol management in GemStone 64. Symbol lookup is faster. Apps that create many new symbols may be slightly slower. All use cases from customers all pass in the new design.

Extents can be safely copied while the system is running. Thus 24 x 7 can use extent copying rather than the more expensive full backup.

The shared cache management has been offloaded from the Stone to a new system Gem. This increases throughput.

Garbage collection is handled by an admin Gem and reclaim Gems (0 to 255 of them; configurable). This reduces the Gem types (6.1.x had 3 or 4 GC gems).

He showed some benchmarks. The big win was creating and dereferencing large amounts of data where the faster GC gave a 10x improvement. Most benchmarks were just 1-2x. OOCL and COSCO (both shipping companies) and ICE (energy / gas trading exchange) have moved to 64.

The 2.x goals for GemStone 64 are to replace the 2 billion object limit for with 130 billion in 2.0 and ultimately 2 trillion (2^{40}).

More than 50 new byte codes have been added to the VM based on analysis of customer databases, giving a 5-30% speed increase. SmallIntegers are now +-2^60. They’ve added SmallDouble which is also fast (it is a 64 bit double bitwise-identical to Cincom’s SmallDouble; Martin hopes to persuade VASmalltalk to use the same standard when they go to 64 bit).

GemStone 64 2.x restores indexes. They have same syntax but are completely redesigned. Dale has redesigned it no longer to need to physically modify every object in the collection. This will be ready for production probably in the 2.01 release (due in a month or so).

GemStone 64 2.0 shipped end-March for Solaris and HPUX. They are now working on AIX and Linux. They hope to ship these end-summer.

Q. What is the migration effort? It is all done by us, using your old GemStone database as a backup. Scripts spin up as many Gems as you wish on the new database to copy from the old. He converted OOCL’s database (165 Gig) in 4-6 hours (after which OOCL had to reset their indexes and so on, using 20-30 Gems in parallel, so that was not the total time taken).

GemStone 64 3x is being planned. They plan multithreaded GC and tranlog; many other things are being considered. Smalltalk source-code management is a sore point for some customers and for them; they have cvs and Store and Envy and would like to use GemStone itself. They would like a 100% thread-safe GCI layer. The class library could be upgraded, especially performance-wise. Reduced conflict indexes would be nice. They might offer new specials.

There was discussion of what should be special. Special ScaledDecimal and DateTime would be a big win for some clients. Some people suggested having a user special class. Publishing the canonicalisation patterns would help some users but users needing canonicalisation behaviour in both client Smalltalk and GemStone need more than GBS canonicalisation patterns.

GemStone find that Windows is not the most pleasant platform to work in. Many things that are very easy in Linux are less so in Windows.

Q.Shared page cache developments? Primary dies, backup script-controlled starts and replays tranlogs. Warm offsite is done (tranlogs copies at regular times to offsite which is in permanent restore mode, ready to take over if there is a problem). They have no plans to add more in 64 as yet; all already done will still work. Customers must tell them whether they want a warm standby or a hot standby.

Q. Single-sign-on, using windows API to use same sign-on password and user ids? It is common for Windows users to have applications reuse their Window's login. Auditors are the issue. Their Java product authenticates using public private key; that would be acceptable.

Q. Better access to `allInstances` for class migration? Also offer a distinct `allInstancesDo:` method. The algorithm hasn't changed but bringing up a shared page cache and using the cache warmers will help (`allInstancesDo:` makes sense; thanks for reminding us).

I asked about the fixed memory allocation. They will not be going back to no memory limits; that would mean back to the NotConnectedSet and garbage to disc which they do not want. However they are working on mitigation of various kinds: things commit more, there are APIs, they will exploit Linux and other OS' ability not to allocate all memory at once, etc.

(Bruce, also Bob) Common Filename handling would make porting some things easier. Likewise exceptions and sockets. Perhaps STIC can help get VW, VA and GemStone talking? ASCII exceptions are now done. File it in for 6.1.5. find it built-in for 6.2 (ask James Foster for info). If Gem could do the full socket thing Bob could avoid some issues. Bruce has his Sport socket wrapper which already makes VW and GS sockets the same (it is in the Cincom Open Repository).

Q. How are you doing? We're doing well. We still have to support the Java side but not as much as previously. GemFile is more lightweight than the full GemStone/J DB. So they are generally doing better than at any time in the recent past.

Packaging BoF
I only caught the end of this. There are plans for a BodyBuilder project. Please describe your build process on the BodyBuilder wiki. Travis Griggs has a build system which he will share to help things get started.

SmalltalkDoc publishes HTML, which should be kept with the packages. Give James Robertson feedback re what it should have. Reinout would like a wiki-in-his-image metaphor; something he could take notes on, with people given that image seeing those notes. Recently Reinout did some work with COM tools in VW and noticed the need for type annotations.

STIC BoF
Bob Nemec has been elected the new head of STIC. Bob is not a vendor. He enjoys all the reunions at past StSs but he wants to grow the community. It is excellent to see the Smalltalk Solutions logo given equal billing with the LinuxWorld logo; such little things can have real impact two years down the line.

The STIC and whysmalltalk websites will be cleaned-up (hosting, ownership, etc.) within the next few weeks. Over the last 10 years the community has fragmented between the various fora. It used to be all in c.l.s. Now Bob knows less than he would like of what is going on in Squeak and he suspects Squeakers know less than they would like of what goes on in GemStone. He'd like to see links to what is there; maybe weekly posts with links to a community wiki.

They would like to demonstrate Smalltalk technologies. For that to happen, they will need a little cash. Before, vendors sponsored the conference which made money for STIC. Now, they need to sell individual memberships. Individual membership is $40. Corporate is being reviewed.

Monty spoke of how nailbiting the conference could be; STIC commits to spend $40,000 then waits to see how many will attend. He thanked Suzanne for her effort to arrange coordination with the Linux people.

Q. Renewals? STIC renewals date will be chained to the conference date in future.

Q. Bundling renewal with Linux world made it much easier to get approval in my company (MS); can this contine? We intend so.

Overall attendance at this conference is 3800 of whom we hope 200 or so will sign up specifically for Smalltalk. Thus we look much larger than a pure StS would be. We will see also how many Linux people ‘cross over’ to attend Smalltalk seminars.

Attendees are only part of the impact. This Linux World, Network World and Smalltalk Solutions conference was marketed to 100,000 people. Most of them are not here but they have seen Smalltalk given a major mention and many will think about that.

Bob then invited feedback.

Q. Have we any connections to the Ruby community? If someone heard about Ruby on Rails, how would they learn what the Smalltalk equivalent was? Various suggestions were offered. I recommended STIC's websites linking strongly to the Seaside hosting site. Someone else suggested the 'Delicious' site ('social bookmarking') as one place where Smalltalk-marked bookmarks could accumulate. Bob is aware of the value of strengthening our visibility to the sort of people who google RoR.

Eliot recommended focusing on core competencies on the whysmalltalk page and having really good links (he's just improved the wikipedia Smalltalk page; that is where people chance on things).

Q(Georg): this conference has no opening and closing session. How can we present ourselves to the rest of the audience? Alan explained that there was an opportunity for a speaker at the level they would accept (C level executive) to present an excellent case study but alas the schedules did not work out. James added that this year was a proof of concept; next year we will have more ability to say can we get X and Y.

Suzanne described how she got us accepted into this conference. The Linux and Smalltalk communities work well together (imagine going to a Java one). Their upside was that many communities get too focused and they saw value (also of course more actual revenue and attendees) in having us. (Other conferences may now be more open to us.)

Q(Dave Simmons) Can we manage a single timeslot to present 'latest cool technology'; such talks are always packed? Suzanne agreed that for next year that would be good; we have apps that would have a great wow factor. You would pitch it as wow for Linux and have Smalltalk stuff all running on Linux. Suzanne mentioned that we committed late (in October) for this conference but we'll have a full year to plan for next year's conference.

Discussion: some volunteer must offer the wow talk items and other volunteers must do the political and scheduling tasks of getting good slots.

Q. What is the $40 STIC membership buying? A $300 discount on this conference. Mainly, what it buys is marketing dollars; we are so bad at marketing a self-evident advantage. It also bought the prizes for the coding contest; Michael has been an exemplary volunteer arranging it. This conference cost them $12,000 dollars (booth, Tshirts, iPods, website).

Sames wants the community to be more outward-looking. Squeak has a large spirit but how do we make them part of the larger community? Bryce described how mainly Squeakers recently organised a Smalltalk meeting in the UK. There was an earlier meeting in Brussels and another meeting is happening in July in Sardinia. All this is done by Squeakers. Suzanne mentioned that if schedules coordinated, Cincom would be glad to let their people visit if they were there; tell Cincom as early as you can and you may get a speaker. James: just set up a google calendar and mention Smalltalk and it will show up.

(Someone mentioned 'summer of code'; students could be made interested. Dave Simmons thought it would have only a small visibility effect for the effort that would be needed.)

Dave Simmons is sitting in Microsoft hearing people talk of IronPython. Noone in MS noticed Ruby until Ruby on Rails became a threat to certain interest spaces. James stressed that it takes effort to blog, report, etc. Any site you want people to visit must add content regularly and the audience will gradually show up.

All publications are always short of content. Articles can be placed if they are written.

Eliot and David instanced what they heard at the mono talk. Mono was a download and install that was not on any of the linux distros until someone wrote cool camera, desktop search and wiki-as-notetaker apps, when Linux distros suddenly started including the mono updater. James blog is an excellent app; we need these killer apps.

Bryce: we have at least one killer app: Seaside. We invited Avi to London and we had 70 people, mostly not Smalltalkers. It cost £200 in expenses and 3 organisers doing some work. Old Smalltalkers got some people in but far from all. Seaside with AJAX will wow even more.

Bob summed up and noted that we have a lot of people who care; this room is full. Suzanne stressed the need to get your next year's conference submission in before the deadline (this year's was earlier than is usual for StS, catching me amongst others :-).

Some Information about Talks I Missed

I missed Mike Hales talk on KnowledgeScape work used in the copper mines in Chile. I got a summary later from him and from Jeff Hallman who heard it.

They used image analysis when processing ore to sort good ore from bad. They used Smalltalk MT. The system recovered its total development cost in a fortnight in increased ore yield. They also used knowledge-based strategy approaches to optimise the performance of the grinding mills (huge machines in which steel balls whirl round with the rocks grinding them; think washing machine on a large scale). Last year, they increased efficiency across a range of mines by 2 -13%.

Overall, they believe that Chile's 2005 GDP was 0.5% higher due to this.

Bruce Badger’s OpenSkills Sport pundle wraps exceptions and Regex11 and sockets to let you write code that will run unchanged in GemStone and VW. It is in the Cincom OR. I missed Bruce Badger's talk. Info on it is at www.cincomsmalltalk.com/blog/blogView?showComments=true&entry=3323549680

For Lucas’ talk on Pier and Magritte, see my ESUG 2005 report or visit www.cincomsmalltalk.com/blog/blogView?showComments=true&entry=3323549556

I only caught the end of Karen Hope’s Smalltalk to Java talk. More is at www.cincomsmalltalk.com/blog/blogView?showComments=true&entry=3323459262

Other Discussions

Gemstone already has a backend framework for taking data from SQL databases into GemStone. James Foster has code to make a front-end framework to let apps talk SQL to Gemstone, not something you would ever choose to do for heavy performance apps but useful to let customers’ minor legacy tools talk to their Gemstone database with minimal port effort. He discussed with Alan Knight hooking it up to StoreGlorp to put Store data in Gemstone; it seemed very doable.

Monticello

Colin Putney demoed Monticello. Slices hold Elements. Snapshots hold ElementVersions. An ElementVersion has a variant (its state), which is a definition (class, method, whatever) or denotes removal (you must distinguish ‘removed’ from ‘was not there’ when merging). When a Snapshot is loaded, the image’s WorkingCopy adds to its dictionary the mappings from the Slice’ Elements to the Snapshot’s versions of those Elements. If the user republishes, it determines whether the slice has changed and if so makes the loaded Snapshot the parent of a new one.

The 4-pane merge tool has an interestingly different layout from the usual style. The Smalltalk elements being merged are in the top-left pane as usual and the image’s definition of a selected element is below in the bottom left pane. However the rival snapshot definition is diagonally opposite in the upper left and the definition version you will get is in the lower right. I can believe this makes it easier to avoid the occasional uncertainty I’ve seen in Envy, Store, etc., over which is older / newer / image / repository. Bold shows conflicts, grey shows ‘my history includes your history’

The tool offers two modes of loading:

- Load: snapshot code replaces image code in all cases
- Include: this is a catchup load that takes the snapshot’s code *if* its history includes that of the image

In discussion, there was also mention of the approach being worked on for Slate. Slate’s aim is to have an image *and* files of source code: methods and classes know which file their source is in, will note update need and write new versions, with namespaces being determined by asking the method, then the file/package, then the class.

Follow-up Actions

Niall To Do:

- Get VW3 profiler from Eliot for use in old systems.
- Look at Bruce Badger's approach to writing VW and Gemstone compatible code.
- Mention Hernan's ESUG talk on time modelling to Avi.

Conclusions

There were welcome signs that Georg's 'second spring' is indeed happening:

- Suzanne's strategy for publicising Smalltalk by presenting success stories looks good to me.
- Dabble DB is, as Avi would say, "eating its own dogfood" and looking good on it so far.
- Several experience reports described how people had used Smalltalk systems to make an impressive addition to their clients' profits.

A Smalltalk-oriented, or Smalltalk-including, evening event at next year's conference would be good. Some of us attended the tail end of a reception happening next door to the last Smalltalk talk on Tuesday, and there was a good but noisy Irish pub nearby where many congregated, but I missed the usual early evening reception or buffet as a forum for industry gossip.

Written by Niall Ross (nfr@bigwig.net) of eXtremeMetaProgrammers Ltd

* End of Document *