CS9 and ESUG 12, Kothen, September 6th - 12th, 2004

I found getting to Kothen straightforward. A cheap flight from Stansted to Erfurt was followed by a two hour train ride with one connection. In Glasgow central station, the staff often have no idea which platform a train will arrive at until it does. In Germany, you get a print-out of the departure and arrival times *and platforms* of your initial train *and your onward connections*. I was impressed and surprised, but my surprise grew less when I learned that German train-management systems are written in Smalltalk. On the way back, I also learned that if the station information desk was closed, the automatic ticket machine could give you the same information in the language of your choice. (This ticket machine would not accept the credit card that Jim Robertson had forgotten to remove from his jogging shirt pocket before putting the latter in the washing machine, but I think that was excusable, given the card's new and unusual shape :-)).

Although few places could compare to the spectacular landscape and architecture at Bled, Kothen was pleasant enough, with well-maintained old architecture rather than DDR-style concrete blocks. Kothen has a long-standing musical tradition (the counts of Anhalt appreciated good music and Bach spent the pleasantest years of his working life there), reflected both in the Bach that was played to us (in the same room of the castle in which the master played it formerly), and in the jazz that Georg arranged for those who wished to hear something more recent. Between recitals (of Bach; I skipped the jazz :-)), Georg's wife gave a brief talk on 'Language, Culture and Software', with some amusing examples of what could happen when software localisation was done without attention to context.

Cincom sponsored us to tea and cakes in the open-air railway museum (I especially liked the elegant old-style dining car) after which we first toured Ferropolis (formerly a hideous open-strip DDR coal mine, now flooded to produce a peninsula concert ground much liked by techno bands whose 'music' recalls the noise the mine used to make) and then went to the far more elegant 18th century town of Worlitz for dinner.

Style

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

Author's Disclaimer and Acknowledgements

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

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

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

My thanks to:

- Adriaan van Os for helping run our CS project, Petr Stepanik for working in it, Alan Knight, Martin Kobetic, James Robertson, Wim Lybaert, Bernard Pieber, et al. for demos, goodies and/or explanations that helped it, and Daniel Vainsencher for setting up what, for me at least, was the smoothest-running network of any Camp Smalltalk
- the speakers whose work gave me something to report, the conference sponsors (Cincom, Eranova, GemStone, Georg Heeg, Impara, Bedarra Research Labs, smalltalkedVisuals, Hochschule Anhalt), and the ESUG organisers (Stephane Ducasse, Noury Bourakadi, Roel Wuyts)
- above all, Georg Heeg and his team of local organisers for a conference that was impressively professional, both visually and administratively, while at the same time being cheap to attend.

Summary of Projects and Talks

I begin with some Camp Smalltalk 9 project summaries, followed by the talks, sorted into various categories:

- Smalltalks, Old and New
- Frameworks
- Selling Smalltalk: Commercial Issues and Experience
- Applications and Experience Reports: Commercial and Technical
- Tools
- Porting Projects

and then the Academic Track:

- Awards Candidates
- Research Talks

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

ESUG Programme, Stephane Ducasse and Roel Wuyts

Stephane thanked ESUG's sponsors. He mentioned smalltalkedVisual's sponsoring a student to attend as an example of the range of ways one could sponsor. ESUG has a new wiki, built on SmallWiki which offers a website that does not look like a wiki yet is as easy to manage. ESUG continues its teacher starter package and teacher excellence package activities: books, slides, exercises. If you can use these, visit www.esug.org/sponsoring.

As usual, they have experimented with conference format. This year we

have:

- a business day
- a parallel-track education day
- an academic research half-day, as at last year's conference track
- technical innovation awards prizes

Roel welcomes feedback.

New Smalltalk user groups have been set up in Italy, the Netherlands and Sweden. It is easy for a group to start with a simple mailing list, then become an organisation; in France, they started with a mailing list of 3 and now have more than 100.

Seaside: the opportunity exists now. Go to Avi's and Adrian's talks. Show Seaside to PHP programmers.

Stephane discussed open-source Smalltalks and their contribution to visibility. Smalltalk/X is free but the licence, though it has no real issues, is not a standard known one. Squeak has a known licence but there are some issues with it. A known-to-be-issue-free licence is essential. If the licence is not immediately recognised as a known issue-free one, many organisations will not even take the time to look at it.

The head of the hosting institution, Hochschule Anhalt, then gave a speech in German. I think he was essentially just saying, "Welcome to Kothen, Smalltalkers." Georg Heeg's reply was witty, to judge by the laughter.

Camp Smalltalk 9: Project Summaries

The decision to hold the Camp Smalltalk weekend after the conference, instead of before, was required by room availability; it was not a schedule experiment. We had a lively Camp Smalltalk. However I think holding the CS weekend before is better, since then you do form groups and do significant work during the initial weekend session and can use odd hours during the conference to discuss and advance specific points. You are less ready to exploit spare time during the conference if you have not had this preliminary weekend to get you started.

Camp Smalltalk 9 ran during conference breaks, afternoons and some evenings of the five conference days, and over the following weekend. There was much activity with 18 people still there on the last Sunday. I heard a lot of, "Come and look at this!", and almost as much, "Come and look at this bug!"

The Custom Refactorings and Rewrite Editor Usability Project

I spent most of my CS time on this. Adriaan van Os and I, helped by Petr on Sunday, worked through the bug list, almost all to do with the dynamic refactoring UI. One exception, a problem reported by several people in loading our current release from the Cincom Open Repository ('XML element not unique error') puzzled me greatly for a while since I was getting warning-free loads. Alan realised it was nothing to do with Store or

our release. The VW7.2 and VW7.2.1 ESUG disks have a corruption in one method of one parcel that does not appear if you install with a key from the Cincom NC release; very odd. If you see this, reinstall from a Cincom CD.

We also looked at a suggestion Blair had made to me for 'extract with holes': the ability to select text for 'extract to self' or 'extract to component' and then select within it those parts that were not to be extracted. The main issue for this was working out a UI. We had thought of doing the secondary selections in a read-only code tool pane. Discussion with Martin and Alan persuaded us that it would be better to extend the extraction dialog window with this pane; display the extracted text and let a user iteratively select sub-parts and add them as parameters to the dialog.

Wim Lybaert integrated the Zork-Analysis Browser (which does code coverage and type recovery) with the RB's navigator. After he and I worked out why Wim was not seeing the type-recovery-coloured code on his machine (compatibility with something else he had loaded), he began work on integrating the type recovery as a code tool. There was also discussion about integrating Michel Tilman's abstract grammar, which powers this, with the Refactoring Browser's parse nodes.

Michael Prasse could not make it this year, alas, but we considerately :-) saved a VA code-colouring bug for him to look at off-line, that being his area of expertise.

Other Projects

Daniel, Avi and others worked on Monticello. From the conversations I overheard, I gather they found a counter-intuitive but effective solution to a key issue. AOSTA was progressed; see the 'future plans' section of the talk on it for details. Alan Knight did some GLORP work (see his talk) and James Robertson added withStyle to BottomFeeder; he was impressed with how straightforward withStyle was to use.

Georg Heeg worked on integrating ObjectStudio and VW. Allowing the declaring of blocks inside literal array declarations was a key enabler of this, for reasons of syntax (Georg thinks it may also enable certain optimisations).

A good deal else was going on; try Jim or Avi's blogs for more info.

Smalltalks, Old and New

Keynote: Dan Ingalls

Roel introduced Dan as, "Well, Alan Kay had this vision and Dan was the guy who was able to code it all up." You can still find his code in Squeak.

Walking down the street to his hotel, Dan passed Georg's 'Smalltalk' van; Dan feels welcome here. His talk was about a couple of things he's been doing for the last year, and about why he is excited again about Squeak.

Dan showed a weather station website he has designed, displaying data collected by sensors (the demo displayed old data; people remarked it was

hotter than that today :-)) along with tides, phases of moon, wind data, etc. Will it be a good day for sailing, for golfing, for any sport that cares about weather? Dan once did this in VisualBasic 15 years ago (took quite some time) and sold it to companies. Recently he did it again in Squeak (very quickly) and now sells it to sailors and golfers, who are people with money to buy home weather stations. Now in Squeak he can run this on a Mini-ITX board off 12v (so his customers can run it in their cars, getting data from their home weather stations).

Q. What OS? Michael Reuger did a compact Linux port.

With Ian Piumarta, he has just ported Squeak to 64 bits, on request from HP who has a customer who wants to do big things in Squeak. Serious industrial use of Squeak is good for it.

Dan then looked back at Smalltalk's past. Dan worked (on speech recognition) across the hall from Alan Kay and one day he and Ted Kahler asked him about all the interesting things they heard talked about in his office. Alan had noticed that many specific projects all needed message interfaces, so why not treat everything as a message interface. Two days later, Alan brought them a hand-written description that Dan implemented as a Basic interpreter; it took one minute to compute 6 factorial. Then he rewrote it in machine code and that was Smalltalk 72. Dan showed us the bootstrap file for ST-72. It was read by a token interpreter to which a colon meant bind the next thing in the stream to the name following the colon. They also tweaked the font so @ was a smiling face (Alan liked to change the fonts). It was easy to make something happen on the screen, in those days not a common feature, so easy to interest children.

ST-72 was not very reflective: you could access and edit the token vector that was the code. To Dan a meta-system is not just one in which the language is described but must also be able to describe the environment; in an afternoon, people could change how their windows looked. Dan showed a few Smalltalk-72 things (in his Squeak presentation environment).

They learned that ST-72 was a delightfully simple language and a great scripting language, with a small kernel (15 classes, 160 'methods', nice primitives (easy to draw lines, rectangles, ...) and a hint of reflection. It was too slow, had no inheritance and inadequate modularity; integrating two large applications was very unpleasant.

Q. Funded? Alan had a project with a local school using it to teach 7-9 year old children OO programming. (Later, he hired a couple of them to the group at age 14 or so.) They also had their research budget.

ST-74 (more reflection: objects to control memory stream, debugger). Alan wanted to make things easier while Dan wanted to make things faster so he wrote a byte code engine in ST-76. ST-76 had inheritance. It preserved ST-72's English-like syntax.

Dan showed us the ST-76 definition file. He had not managed to revive ST-

76 to the point of demoing it as he did ST-72. Then Helge Horsh realised that he could get ST-76 working by hooking the primitives to Java. One day, Dan received an email with a jar file. He clicked on it and there was Smalltalk-76. Dan demoed it, opening windows (after the usual demo hiccough) and showed the base class hierarchy. “It’s fun doing archeology in Smalltalk because you have all these power tools available.” (Helge’s ST-76 URL is http://home.netsurf.de/helge.horch/esug/signum.html)

There was a brief excursion into ST-78. ST-76 had a virtual memory (called OOS or more often ‘ooze’ because it moved so slowly). ST-78 had a much simpler 16k virtual memory built for an experimental portable computer (‘the notetaker’) that Xerox did not develop.

ST-80 had metaclasses (ST-76 has class Class but not the parallel metaclass hierarchy), booleans (ST-76 was more like Lisp where everything is an object or nil), blocks (ST-76 was more like an `eval:` approach). Dan has ported Vassili’s Hobbes system to Squeak. He demoed it running in Squeak (doing Smalltalk at a 45 degree angle :-). Dan also got Apple Smalltalk (ST-80 from 2 years later; easier to use with one button and simpler public domain licence) working as a version of Hobbes.

These old Smalltalks taught them key ideas.

- You can win with messages + inheritance.
- Bytecodes are small and fast. (Alco had 512 of microcode that ran 10 times faster than other code. That prompted them to think about how to get the inner loop into microcode, and so to do bltblt and bytecodes).
- Meta is a win; classes and contexts as live objects. You can change it day by day and that is how systems get better.
- You can have a small yet complete Kernel.

ST-80 is the same engine; this has stood the test of time. Looking back, he feels that:

- The parallel meta-class system is a little confusing and over-elaborate. Prototypes would be more general
- Side effects are problematic. Lex Spoon tried to make a secure Squeak (Island’s project) and learned that shared mutable state is a problem. Local side effects are OK but global ones are not. There needs to be something in the meta-system that makes that scoping distinction.
- Some subclassing is problematic. Somehow, clean kernels get messed up. Clean packages are hard to build. Try destroying class Integer and then reaching a safe system; Dan has done it but it was very hard and should be easier.

Dan’s Smalltalk Zoo project will be a reconstruction of 72, 76 and 80. Squeak has to model every memory cell because it must translate to C but if you skip that requirement it is simpler. He aims to have a boot-anywhere CD from which these can be run, thus a Zoo of live Smalltalks, not a museum of dead ones.

Q. Put the zoo in Croquet so we can visit it. :-)

In July of this year, while finishing the 64 bit Squeak VM work, HP had a security conference. Dan almost did not go but did. In the course of the day he became a believer in security. E is a language/system/architecture for secure distributed computing (some similarities to Linda; for E info, see www.ERights.org). It can be implemented on Squeak. A version exists in Java but it is much more powerful when run on a pointer-secure language. Mark Miller: "Security is just extreme modularity." Squeak, ST-80, etc., are simple but just loose enough that as we make them bigger they get more complicated. Security architectures are complicated but what they are trying to do is simple. Dan thinks that a 64-bit VM with security done right in Smalltalk would not just be very commercially significant. It would also solve some basic issues. Fast, secure, distributed will pay the rent and can be kept simple so it is still fun. It may also address exploiting multiple-processor architectures, which require similar solutions. Implementing E in Squeak is half-done and Dan will be working on it when he gets home.

Q(Andrew Black) ST-80 had (which ST-76 did not have) the ability of user to change the protocol in a class? Yes, it gave instance-specific behaviour on the classes (instance creation methods were the major initial concern). Dan recalls using ST-76; it was not so bad without them. Georg Heeg has been told by Adele that meta-classes were a demo of how to do instance creation methods, done by Dan one night and never pulled out (mind you, could that not be said of many features of Smalltalk?); Dan confirmed this.

Q. There is a Java project (called Groopie?, Groovie?, ...) in which meta-classes are being added. Is it a good thing? Hmmm...

Dan prefers to recreate these Smalltalks in Squeak rather than e.g. on an old Alco as he wants to understand how the interpreter, base system definition, etc., works rather than how the old instruction set worked.

Q. Children have problems with the precision required by computing; we need to make systems less brittle for them. Could security by modularity help that goal as well? Interesting thought; could be.

Q(Roel) "Connectivity begets Connectivity". Objects rather than strings passed around can improve security. Thus Roel agrees with Dan that this is an opportunity.

Q. Modularity should be unit of deployment as well as security? Dan is not yet sure. The question is, with images or without? We could move to an environment where each module is like its own image, rather than moving modules into larger images. Dan must do more work to find the right solution.

Q(Claus) VM-level mechanisms borrowed from the functional programming guys? Support running multiple versions of the same class in the same image? Reference classes by 64bit UUID, not by name? Dan has found his favourite Smalltalk programs were functional. These ideas could

show which pieces to configure.

Q. Can we have an asynchronous Smalltalk? Andrew Black's answer is yes. Timber is their asynchronous language in which each object is like its own monitor and all messages run to completion. It has no inheritance because they can't work out how to do that yet. This can be done in Smalltalk by changing the evaluation defaults, doing side effects explicitly. Roel: Smalltalk already has Promise, an asynchronous construct.

SW Vision Smalltalk, Frank Lesser
His smalltalk has deep OS integration, .Net integration and 10,000 classes and 100,000 methods in its frameworks. Frank started this in 1994 as a private project, influenced by VSE. DaimlerChrysler ordered it as a replacement for VSE in 2000. In 2001 they ported part of the VM to Squeak and also began work on .Net bindings. They built a reflection browser for .Net and now have a team development environment. The current released version is LSW DotNet-Lab 2.0.

Their OS integration uses native widgets and binds to most Windows APIs. Their .Net integration uses reflection at the binary level. The base development environment has class and hierarchy browsers, inspectors, etc., and a team development environment. He demoed; the IDE reminded me of Dolphin - package browser with windows feel, expandable tree lists, etc. It addresses the common Smalltalk IDE problem of too many windows open: panes are switched depending on the selection in the globals view.

Libraries can be used in pure form or embedded in Smalltalk assemblies. A smalltalk assembly can be made into an ordinary DLL. Method browsers can be switched between Smalltalk, Smalltalk Assembler and decompiled source views. A cross-referencing tab shows various cross-references to other selected items (a little like the Classification Browser's pre-provided method-oriented intensional classifications).

They have MSDN integration in the browser, based on their HTML pane (appearing as a code tool in the browser). Refactorings are integrated into their package browser. Frank showed the inspector (tree-list expansion of instance variables), debugger, source code control tools, etc.

Their low-level Windows bindings are subclasses of ExternalHandle, ANSIWindowHandle and UnicodeWindowHandle. Their framework does not use events; they saw them as unperformant and hard to debug. Instead, they use entirely the visitor and strategy patterns, i.e. not

```
    self triggerEvent: anEvent
```

but

```
    self eventVisitor: event from: self
```

Every class must have an eventVisitor: method. Context-sensitive refactorings to add these are provided. For every Window class there is a Strategy which owns one or more visitor instances. These retrieve parameters for API functions. One drawback of their approach is that the Object class is huge with all these methods.

Q. Are you planning for future evolution of .Net, especially in GUI? Yes, that is an issue for us. Our refactoring framework for visitor pattern, etc., should help here.

Q. Commercial use? .Net reflection browser and other tools. They do this simply, since .Net is such that it makes a lot of difficulties for Smalltalk vendors (c.f. Dave Simmon's work). In Java or C#, adding a method needs recompiling; good for hardware vendors but bad for developers. Frank finds that .Net developers don't believe what they see in Smalltalk. Frank thinks the .Net kernel is rubbish, thus they prefer to use their own bindings. Use of .Net for Smalltalk: web servers need .Net bindings.

(Frank had to stop for time here; he had more to say re .Net integration.)

Smalltalk/X, Claus Gittinger
Externally it looks like other Smalltalks but the internals are different. Claus does not use native widgets, everything is unicode. The browser is basically the multi-tab RB (which you rarely see elsewhere). Claus built a small GUI to show how (use builder, set some parameters) and then used it to raise a debugger and write the model method. He invoked some very bad GUI code to demonstrate how robust the debugging is (and was hit by the usual demo hiccough; base system interrupts not set in the correct way).

He showed using the debugger on the many process he had running. The dispatch loop catches the control C event and opens the debugger on it. The same architecture has been implemented in VW in recent versions.

He browsed the frameworks (HTTP server, etc.). Everything is dynamic; no recompiling. Claus configured services for the HTTP server and brought up a browser, looked at a 'hello world' service, changed the code and we saw the change take effect.

Smalltalk/X differs from other Smalltalks because Claus started without any image, etc. He read the paper in 81, decided he wanted that, had nothing else, knew C translation, so generated C (or primitives) from Smalltalk files. Claus called

```
make hello.c "to show use, otherwise do make hello.o"
```

and showed the C. He pointed out his in-line cache, using an indirect call as you cannot actually cache in C. The send function gets a pointer to the in-line cache and thereafter points directly to the cache. Dave Ungar inspired Claus to do it this way. That's how Claus started 15 years ago writing this in vi. The runtime library (send function, cache, etc.) links into the specific Smalltalk code. Claus then showed hellomicroworld.c (using primitive, statically linked) to show how small it can be.

Thus Claus created a non-GUI smalltalk. Next he built a parser, then a view library, initially for X windows (thus the name, Smalltalk/X); nowadays it also supports Windows (and anything else people pay for). He browsed the Array class, showing its primitives written in C. He showed compiling to C and to assembler. Thus very optimised, very tiny executables can be

created, or you can build the whole Smalltalk/X framework and your app into a single executable. Claus showed writing stdin / stdout filters (which worked once the audience corrected his spelling :-).

Q. Can save image? Yes but images not portable across machines presently

Q. Licence? Open-source except for ST compiler (he will give it to you but he does not want that open-source) for (almost) any use, subject to always crediting ST/X. They tried to sell it as a tool but only sold 50 - 100 licences a year max, so they make their money from projects (for users who are sufficiently fed up with Java or C# that they just ask for fast executables). Thus they cannot find time to support novice users, e.g. 'Why does this (windows) file not start on my (Linux) machine' type users.

(I think a question on code management elicited that Markus Schwarts had done some work on an Envy-like system, but I missed the details.)

Squeak Demo, Stephane Ducasse
Squeak is a dream catalyser for language research, web development and etc., with a powerful community. Stephane showed Seaside to some PHP programmers; now they use Smalltalk.

Stephane showed a range of impressive and fast graphical displays, some driven by the code on his slides. He showed grabbing a graphic (an avatar) and moving, dilating and rotating it, etc., while the avatar still followed its program (to wave its arms and smile and so on). He created a piano icon and played it while rotating it, useful for simulating the livelier sort of pop group. He showed a gas simulation, atoms bouncing around as their bounding box was moved and changed.

Squeak 3.7 is the current version. Internationalisation is being done and will be in 3.8. Chinese, Kanji, etc. Stephane showed it.

The look of Squeak can seem a bit way-out-there so they also offer conventional windows looks, and some less conventional ones. Some native widget work is being done, as is an Objective-C and Cocoa bridge.

Faure and Dynapad are PDA Squeak projects. Genie is the handwriting recognition project; steer Squeak with a pen. Genie outperforms standard windows software. there are many music, sound analysis, etc., projects. (Alan Kay, who was a Jazz band player, can 'play' Squeak impressively. Alan has a demo of synchronising music to soundtrack of movies, just speeding or slowing tempo etc. to make notes coincide with action.) Dan Ingalls showed what the various lines meant, changing modulation and timbre of the notes. Thus you can work on one of the instruments while Squeak is playing a midi score, changing that instrument's setting and hearing the effect as the score plays on. Dan had a customer of his weather station who wanted to call home and get the weather. Dan provided that feature in an hour using the avatar-speech code.

Stephane built a plane controlled by joystick, to show how easy it was.

Then someone from the audience stepped up and did a fractal. He gave himself a field, three corner objects and a pen. Using the script editor only, he wrote a script for the pen to choose a corner at random, move half-way to it, draw point. A triangular fractal pattern was created. Then Stephane drew a curved road and a car and programmed the car to stay on the road.

Squeak is used in Spain on 80,000 PCs taught by 50 teachers. Morphs can be moved between desktops to create distributed games.

Scratch is an MIT MediaLab project built by the LegoMindStorms and StarLogo inventors. It is a much leaner Morphic. Stephane showed StarLogo, an ant-pheromone system simulator. There were also 3D demos of Alice, ships, etc., and Squeak in web browsers.

The slides list the URLs for all these projects. Download and try out if interested. Stephane learned almost all these things just by trying. “When looking at a Squeak project, it’s important you press every button - you never know what you will find.”

AOSTA, Marcus Denker, Software Composition Group, University of Berne
(See talks by Paolo at last ESUG and Eliot at last two Smalltalk Solutions). He reviewed what AOSTA was and then looked at compiler generation.

We have really good JIT compilers already (there is one for Squeak) but we do not have the higher-level optimisations that Self introduced. AOSTA uses the JIT compiler plus Smalltalk to let profiled execution identify hotspots and compile to optimised bytecodes, dynamically decompiling for debugging. The idea is that optimised code works as now but unoptimised code also does profiling: counters for each send and backward send, collecting type information, etc. Bytecodes build up Single Static Assignment form where we optimise and compile back to bytecodes.

Marcus’ talk is about how to generate bytecode back from SSA.

The design’s front-end (bytecode-to-SSA) and middle (SSA framework and optimise) is already implemented in VW. In 2004, they are doing the back-end transformation out of SSA, a simple code generator (written in Squeak, since he is a Squeak user, so he ported the whole AOSTA framework to Squeak).

In SSA, each variable wants one assignment. Hence code like

```
a == 3 ifTrue: [b := 2] ifFalse: [b := 1]
```

is transformed into two assignment paths that are then merged (the program will go one way when it runs), thus a third assignment. This is good for optimisations but you must remove the path-controlled merge-assignment (called the ‘phi’ function) before you can generate code. The canonical method is to model what the phi function means, rework to this and then generate. Alas, this combines wrongly with some optimisations. An additional parse to handle this is not pleasant.

A better idea is to notice that you can transform the SSA tree, so transform it that all variables in phi are the same

```
a1 = phi(a1, a1).
```

Now we can remove phi easily. This means that if you have no optimisations then you need no copies and if you have some, simple heuristics let you handle the copies. You always have fewer (e.g. 16000 with the canonical method reduced to 1000 with this approach) for the same degree of optimisation. The tests system is easy to build:

```
ir := IRBuilder new
  rargs: #(self);
  pushLiteral: 1;
  returnTop; ir
aCompiledMethod := ir compiledMethod.
```

Execute
```
aCompiledMethod valueWithReceiver: nil arguments: #() 1
```

Install
```
Float addSelector: #test withMethod: aCompiledMethod
```

Marcus walked us through an example, selecting his code on the slides and doing it (talk presented in Squeak).

Future experiments: do AOSTA on Squeak with jitter. Does it make sense to run this just with an interpreter? Use Exubery as a back end?

Now that we have a translator, what does this enable that we could not do before. In Squeak, instVars are accessed via offsets. If you add an instVar to a class, then you must recompile or else recompute the offsets of all the affected classes - bad! With a translator, you could late-bind instVar access and generate bytecode on the first access. When you add an instVar, just invalidate the cache. You can also improve the meta-object protocol. MetaClassTalk in Squeak runs 500 times slower than Squeak. ClassBoxes are slower too. Both can be helped by this.

Q(Niall and Avi) status of VW? Eliot hopes to restart work on it this year. Marcus work will be on SqueakSource soon and can be ported to VW.

Q. Could be used for generating other language bytecodes? Yes, perhaps.

Q(Andrew Black) Use this for Tau-optimisation? (Answered by Bryce) You can do tau-optimisation even without SSA and with blocks but then your debugger cannot tell how you got to where you are when it opens. What do you want, tau-optimisation or a working debugger. You could do tau-optimisation after you found that last bug - if you think you are ever in that state. (Avi, you can do a lot towards it by adding just five lines of code; he did work on that a while back.)

MicroLingua: an extended conceptual model for Smalltalk, Maurice Rabb
I also recorded this talk in my Smalltalk Solutions 2004 report. Here I

summarise more briefly, noting fresh points.

DynaGlyph is a dynamic font engine for fancy watch graphics, etc. Inspired by Alan Kay at StS2001, Maurice wanted to build a fast, lightweight, small real-time VM for concurrent flash apps. He demoed watch and raffle-ticket apps. Alas, 2001 was not a good year to get funding. He lost his second round of funding. He has finally resolved intellectual property issues and now can open-source it.

He wants things that are said often to be sayable quickly and things that are important to be easy to say correctly. In embedded systems, fast access to fixed size integers is important. He use a ~ prefix to +. -, *, etc., operators to mean 'return something of the same precision as the receiver'. He uses a ! prefix to mean immutable, ' to denote mutable (e.g. in the debugger, just to make it crystal clear). In MicroLingua, FixedFractions do not reduce (improves performance and programmer understanding). InexactNumbers display themselves showing the trustworthy precision.

Smalltalk has a great object story but, when you work on concurrency, he sees self as violating encapsulation. He therefore eliminates self and so eliminates direct access to instVars.

"The selector space is Smalltalk's most precious resource", Alan Kay. Smalltalk tries to model spoken language but he feels software is a written, not a spoken, language. This motivates him to suggest changing some basic selectors (see my StS2004 report; following discussion there, Maurice is ready to drop some of the more contentious syntax changes) and to propose treating symbols typed without whitespace as evaluated together, thus changing the precedence rules (2/3 + 4/5 = 22/15). He uses double-dot statement delimiters to make statements asynchronous. In the future he wants to reduce the overloading of nil and refactor the collection hierarchy.

Q. Licensing? MicroLingua is open-source BSD licensed.

Q. Maybe concentrate on the domain additions (fixed precision) rather than contentious language changes? He agreed; the former will help Squeak, the latter he cares about but will sacrifice to gain forward movement.

Frameworks

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

(I've already described this talk in my Smalltalk Solutions 2004 report but Avi made some additional points this time and the questions were different, eliciting additional explanation, so I have detailed it in full again.)

Web apps are really awkward things ("I'm more of the mind that HTML-based apps suck.", James Robertson.) so why do we bother. They are used because they are everywhere: you would not download individual clients to do all the things you do via web apps.

Applets failed not because Java was not everywhere but because they were not the web. By the time applets came around, users had a mental model about the web that included the back-button; browser-based apps that were not web apps did not fit this model. A web app should feel more like a website than an application. Applets failed but Java didn't (alas!); it had its success on the server. The message is: server-side is a second chance).

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

Carsten's talk about a Dolphin Smalltalk stressed that without its excellent, simple windows deployment, he would have been forced to go outside Smalltalk because his customers cared about that. On the web, noone cares.

Why should the Smalltalk community be interested in Web development. Because we can. On the client side, rivals have an unfair advantage: Squeak looks funny. It is difficult to deploy a desktop application using Squeak for all sorts of trivial but real reasons. On the web, this does not matter so we can leverage the fair advantage we have. If you think web apps are awkward to use, you should try writing them! Building web apps is a hard problem and, as Niall's talk illustrated, Smalltalk is good at solving hard problems.

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

- user types URL
- that sends request to webserver (e.g. GET /foo?x=3 ...)
- server responds with an HTML doc (... <a href="bar" ...)
- doc gets rendered in your browser (text, buttons, links, ...)
- user reads, then invokes some operation in browser (... /bar?y=4 ...)
- that produces another request

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

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

- get shipping address (state is shopping cart info)
- get billing address (state is shopping cart and billing info)
- get payment info (state: cart, billing, payment info)
- show confirmer ...
- get user acceptance ...

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

Global session state is a commonly-used bad half-solution to the above. A session is created to last from the start of the user's site interaction to the end. Thus you assign a unique session id and every request and response includes this session id. So you just send the new data and the session id instead of all the data (so e.g. GET /foo?sid=a32cf6d could respond with <a href=bar?sida32cf6d).

Locally this is better but global use of the session id is very bad. Suppose you get near to booking a flight, then use the back button to check something, spawn a new browser, then decide, that after all you will book the original flight. The last thing you did was look at the flight you decided not to buy but the global state now has that linked to the session id and so the wrong data. (Generally, global state combined with threading is always a problem due to race conditions, etc.) So it books the wrong flight (or it books both flights, as Georg Heeg has experienced when booking flights.)

WebObjects is written in Objective C which is Smalltalk-like. (Simplifying slightly) it arranges that each page instance within a session has a unique id. That instance, helped by the overall session, handles the response and returns the id for the instance of the responded page (e.g. input name="page" value="a21">..., responds POST ...page=a21&flight525). Thus if you backtrack, you get a fresh instance of the page with a fresh id (e.g. ...page=c5&flight620 ...); you do not get confused. You don't have to marshal your state back to the user because these instances are creating themselves on the server and can use normal accessors to communicate state between each other. This has problems but is a lot better and is a smalltalky way for approaching the problem; you are interacting with live objects, not marshalling state.

WebObjects does as much as you can do in ObjectiveC, or in Java (Apple ported it to Java when they bought Next; see also WebTapestry, the only other web app framework Avi knows that uses some of these ideas). However it does not address coupling, the next huge problem. This way of writing apps means your code needs to know that page A is followed by page B which is followed by page C ... Many web developers envision their site as a network in which this web page points to that one, which points to that other one, and likewise for the objects pointing at each other on the server. That's fine if you have a really rigid setUp but it's a problem if you need to rearrange things and a huge problem for evolving your system. One of Avi's early clients wanted a box office system where the main tasks had

many similar steps - pick show, pick seat - but in different orders - selling a seat and exchanging seats did not start from the same place or go through the steps in the same order. The solution to this problem is continuations.

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

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

But suppose they entered the billing address and then checked a box saying ‘use that for my shipping address too’; we don’t want to invoke the Shipping address. Suppose you pass a block instead of an instance; then you can add conditional behaviour, and have the state as parameter.

```
WOComponent
  subclass: #BillingAddress
  instanceVariableNames: ‘ship bill’

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

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

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

Avi then opened a browser and showed essentially this code in Squeak. He opened a web browser and demoed the code running. He reordered steps and showed how the apps behaviour changes.

```
bill := (self confirm: 'Use shipping address as your
billing address')
  ifTrue: [ship]
  ifFalse:
    [self call:
      (WAStoreAddressEditor new
        addMessage: 'Billing Address:')]
```

"Users use the back button more than they go forward" (Adrian Lingard). Avi demoed changing his mind about whether his billing and shipping addresses were the same by using the back button. That rewinds up the method and forks off a new version that continues. Having the back button work as users expect is essential; users use it all the time. It works because `call:` makes a copy of the context stack. Avi demoed cloning the browser and going forward and back on the two tracks.

Q. What is returned by the `call:` above? It returns an address object. Avi can use the same object for shipping and billing by just parametrising it.

Q. User creates new record for shipping, then discards (shipping = billing); how do copying stack and database transactions interact? To ensure only one state is right, when that is necessary, Seaside has a method `isolate`. A call of `self isolate` done e.g. after the user has confirmed their shopping cart, invalidates the other stacks. Avi demoed getting a 'this page has expired' followed by redirection, by OKing one purchase after splitting the browser, then trying to OK the rival fork, in a `self isolate` case. You would end your transaction inside the block just before the block ends.

Q You can nest `isolate` calls? Yes.

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

Q. Space for these continuations? You configure to expire continuations for all but the ten most recent cases and that keeps footprint down. Seaside uses more memory than its rivals; you are trading space for ease of development. He showed that his demo was using 10k, that a complex session may use 100k. In 95%+ of webapps you will write, this will be a good trade; maybe Yahoo would hesitate to use Seaside, but mostly you just add another server (by definition, you must be being successful to have the issue). Provided one user session is on one machine (standard anyway for webapps), you can add servers easily.

Q. Time to copy the context? Negligible; Squeak (and VW) have primitives that make copying and restoring very fast.

It is great that you can write the whole process in one method. You can describe the flow of your application in a much more natural and maintainable way. Avi first used Seaside in a box-office booking system where the steps were the same in several process but their order varied: ticket purchase, exchange or return, use the same steps in different orders.

It is a happy add-on that you get dynamic context across multiple requests and this helps exception handling. You can put an error handler around a high-level method, not around each request-response. Avi inserted an error and demoed how the developer sees a walk-back page, hit the debug link, saw a debugger, fixed it, proceeded. He is debugging *in the same context* as the error occurred and he can *still* use the back button to reinvoke the error over and over again to understand the error. This is a very Smalltalky way of doing things and one of Smalltalk's strengths.

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

Avi demoed this on a simple counter application: page with count and links to increase and decrease it. Avi increased it to 5, went back to 2, then decreased it; saw 4 because he hadn't registered the counter for backtracking. Avi registered it and did the same process. Now going back to 2 and hitting decrease responds 1. He opened several windows and played around, showing it worked as a user would expect in all cases.

Avi showed a Graydon Hoare quote to the effect that things you can rearrange within a single method are simpler to handle. Paul Graham said something similar about colour selection. He had a colour swatch - go to colour page from anywhere, make colour choice, go back. This is very easy to do with continuations, very hard without.

In Lisp, Scheme has continuations and Common Lisp does not; it is a major at-the-VM-level choice. In Smalltalk, it is ten lines of code that you add. Avi showed the code; stuff all the temps, program counter and etc. onto an array. To restore, kill the current context, restore the saved one with all its values, and swap the context's sender, so you return to where the context was captured (usually the self call: point). N.B. context termination does not handle `ensure:` perfectly; in Seaside it is desirable not to have ensures that go across multiple continuations (the interaction of continuation and ensure is an unsolved problem and is the reason Common Lisp does not have continuations, since it does have ensures).

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

This is fine between pages; what about reuse within a page. Suppose a page has two addresses (e.g. shipping and billing address, or old and new address); we'd like to reuse two instances of that object on the page. The problem in the old approach is that duplicates of two keys must not occur

so street must be street_1, street_2 and you're back to marshalling unnatural data. aRequest('name'->'Avi', 'street'->'...', ...). Avi showd code for generating an HTML page that demonstrated this duplication of the text labels you are displaying as method labels in your code.

Seaside says, when you do this, don't give the fields a name, give them a block. The HTML is generated with the unique ids of these block objects so when the request comes in you just marry it to its block and execute.

```
<form>
Name: <input name="1"><br>
Street: <input name="2"><br>
City: <input name="3"><br>
Country: <select name="4">
          <option value="5">Canada</option>
          <option value="6">US</option>
       </select>
....
```

Seaside as against other web app approaches reminds him of the difference between Smalltalk and Java source code configuration management systems: in Java - what directory is this in, what files - versus in Smalltalk - my source is in this object. In Smalltalk, we use objects, not filenames and other strings.

Q. Huge gap between what Smalltalk can do and what you can print on the page, e.g. dates? You can handle this via `renderContentOn: html` using a canvas-like `html` object, or via many specific methods e.g. `dateInputWithValue:`. You could also have a special date component to handle specific requirements and reuse it from anywhere.

Q(Andrew Black) (missed some discussion) This gives to web development what structure programming gave to those who previously used goto? Yes, Avi's blog is named 'HREF considered harmful' for precisely this reason. Seaside adds subroutines and parameters to what was a goto environment, so it is an advance.

Q. Seaside and withStyle? It needs a proprietary client; part of Seaside's value proposition is being able to deliver web apps to standard browsers. Like Seaside, withStyle uses CSS presentation and barebones HTML generation. Avi stole from morphic the ability to toggle halos tweaking your CSS so that a barebones generated HTML of links can be displayed beautifully and edited in place (e.g. by a web designer in a commercial app). Avi demoed several completely different looks for his demos.

Web heresies:

- It is claimed that you should use meaningful URLs. Seaside has session data which is useless if bookmarked and revisited a year later when the session has expired. Seaside lets you include meaningful information from which a session could be reconstructed; Avi showed an example.
- Seaside has a different philosophy from REST. REST is about using HTTP as HTTP was designed. Seaside is about saying we can build higher-level abstractions.

You the developer determine the semantics of the back button. The common semantics are roll-back the web page, and some, but not all, state.

Avi showed the other side of `call:` - what happens when the user clicks the OK button.

```
ok
  self answer: address
```

returns the user to the correct context.

Q. Configuration? Avi walked round the configuration pages that let him control his apps, provide parameters, etc.

Q. Performance? He has never encountered performance problems; indeed people have sometimes told him it was faster than other things they used. However it has often been used in intranet or smallish internet applications so massive scale use experience may be to come.

Q. Which dialects? The main development happens in Squeak. Michael Bany of Cincom maintains a VW version which tracks the latest Squeak state closely. Dolphin has an early port; it would take two weeks to port the current state.

Q. Seaside's licence? In Squeak, use it anywhere for anything for free. In VW, the position is almost the same but ask him case by case.

Q. Can more be done for the Squeak website and documentation? Avi's priorities have to be with the framework state and customers. If anyone wants to help, great.

Q. This conference has a teacher's day; how should we teach the value of Seaside? Building a UI with Seaside is at least as easy as building a desktop UI; it might be good as a teaching exercise. (Stephane: a French group has received a demo and was very impressed. He read the email he had received afterwards, in French, which I translate as - "I saw Seaside and was totally <hard to translate word - blown away? - but I gather he liked it>". Every time Stephane has shown Seaside to people they start to learn Smalltalk. Avi mentioned how he was blown away by Smalltalk three years ago when he discovered it at OOPSLA.

Stephane discussed whether we could somehow arrange a little money generation for Seaside. There is an open question about how to fund Seaside; support contracts or whatever.

Q. Who does it where? Switzerland and Canada are the focus points. There is also work going on in Slovakia, Colorado, some in France and the UK, and Avi is in Holland for the next year.

Relational Persistence in Smalltalk, Alan Knight, Cincom
Alan hopes to see us all at Smalltalk Solutions in Orlando late July 2005.

Generic Lightweight Object-Relational Protocol (GLORP) is an open-source project that will sometime be part of an overhaul of the VW Lens. He showed a motivating example of using GLORP: a Cincom internal application. It is small (19 tables) with significant amount of data and an 'interesting' hard-to-change schema, interesting enough to give the average DBA a heart attack. Performance is critical and some connection is over high-latency links. Alan was of course talking about Store. He started a GLORP session (in code so we could see what was going on)

```
session := GlorpSession new.
login := Login new
  database: OraclePlatform new;
  username: 'system'; "Store default is BERN"
  .......
Smalltalk at #MySession put: session.
```

and brought up an inspector of Package versions. He examined one, noting various 'uninstantiated ....' values; GLORP uses proxying so although he read in 25,000 package versions he did not have to wait before looking at them. He sent it `self packageYourself.` The primary key was first looked for in the cache (where it was) so did not have to run more SQL.

Requesting a versionless package runs much faster and retrieves package names. What table does versionlessPackage map to? The same as package; it just uses an SQL 'distinct' clause, an example of 'interesting' mapping.

```
MySession
  readManyOf: packages
  where: [:each | each versionName = 1.2]
```

On the screen we saw the SQL scrolling up in the Transcript whenever database access was invoked.

Q. Multiple and long-running transactions? GLORP will start a transaction if it needs one and one is not already running, so the user can start one and manage it (long transaction) or do nothing and GLORP will create and commit as needed.

You can optimise the query, e.g.

```
  "when running query, also get the definition"
methodMapping query
  alsoFetch: [:each | each definition];
  expectedRows: 1000; "allocates suitable size caches"
```

In Store, "Package" means one version of a Package. "Class" means ClassDefinition (no equivalent of Envy class version letting you browse methods held just by that class version). MethodDefinition may be in many tables. Naive table mapping does not connect to straightforward Smalltalk. Thus Alan introduced additional entities such as StoreVersionlessPackage, StoreMethodInPackage, etc., that let you talk about things in standard Smalltalk ways and write queries more easily.

Alan then showed a UI "that shows clearly why Vassili works on tools and not me". Alan finds John Brant's replication facility slow so wrote a faster

one using his tools. Example: 'In my local database, find all top-level (i.e. not contained in bundle that is also found) pundles that were published in last 30 days, whose names match <string>'. He uses this to keep pundles in his local Store in sync with the on-site Store, etc. (The UI looked quite usable despite not having Vassili-ish quality and after seeing more of it in a later demo, I plan to use it in my Camp Smalltalk project.)

Alan demoed a browser he has created that can load code from Store into VisualAge. He used the Refactoring Browser since it has a lot of cross-dialect code (but enough dialect-specific code to give Alan work), thus the tool runs in VW and VA; he demoed in VA to make clear what it was doing. This tool runs threaded in the background and is fast. He still has to complete making it do writing of code back to Store (but otherwise, it looked pretty good to me and a useful step towards making Store and Envy work nicely together; Alan talked me through how it worked off-line).

Cincom did Alan the great favour of letting him test all the above in an environment with a lot of database and network hiccoughs :-). Writing uses Oracle ArrayBinding so 1000 rows can be written as a single round trip. Alan showed some relevant weblinks: I noted

- www.glorp.org, glorp.sorceforge.net is just for bug list and mailing list
- www.agiledata.com (for Scott Ambler's Object Primer article)

He also mentioned relevant other work: Avi Bryant's ROE, etc.

People tend to come from either the OO or the Relational camp and wish to build systems that hide the other world completely. Alan thinks it better to build systems that are aware of both.

Q(Niall) Can the community use this? Replicator shipped in 7.2.1, Browsing not yet there (just finished) but published in Store except latest

Q. Other ports status? Squeak port being used, Dolphin a little old.

Smalltalk Web Framework:, a comparison of Aida/Web and Seaside, Janko Misvek, Eranova
Janko build Aida but Avi is in the audience so should keep him honest :-).

A web application server provides web pages built dynamically on-the-fly. Web pages are not like GUI applications. The client is thin, just a web browser. Seaside and Aida/Web are two Smalltalk web frameworks. Swazoo is a Camp Smalltalk project, a web server plugin structure ported to Squeak, VW and Dolphin.

You can build web pages in two ways:

- embed tags into HTML: Cincom Web Toolkit, jsp, asp, php. WebToolkit embeds <% smalltalk code that returns html %> tags in HTML pages. It is simple for web designers who don't like to program but the functional logic gets lost inside the presentation logic. The mix of two languages creates hard-to-maintain spaghetti code.

- use programs to build pages: Aida/Web and Seaside, Struts, Java. This needs no html, separates presentation from model and aids componentisation and reusability. Web designers dislike having to learn programming and having to move design into code. However graphic design can be separated from HTML content using CSS stylesheets.

```
element := WebElement new.
element table width: 500 ...
```
is Aida,

```
self contentsOn: html.
html form: ...
```
is Seaside.

An object system is a web of objects connected with object references. A web system is a web of pages connected with URL links. Janko built Aida to use this parallelism. Every object should present itself as a web page. Object references should map automatically to web pages. Aida supports get, post, put, etc. (www.xfront.com/Rest-Web-Services.html), and answers requests by presenting the object as a resource. Aida uses MVC to separate WebApp from domain object using an the observer pattern. Every domain object has its presentation counterpart. Aida also provides support services for mapping URLs, security, etc., and for presentation. In operation, the WebServer sends a printWebPage request to the object which then communicates to the WebPage and thence to the HTML Page.

In Seaside, continuations let you define control-flow very easily. There is a session-like process/thread (like Process in Smalltalk). Components have UI state and logic, and can call other components. Janko showed examples of Seaside and Aida code doing the same thing (a URL counter). He noted that the Seaside code was shorter; Aida had to separate logic and presentation, whereas in Seaside:

```
WACounter renderContentOn: html.
html form: html heading: count.
html
  submitButtonWithAction: [self increment]
  text: '++'.
html space.
html
  submitButtonWithAction: [self decrease]
  text: '--'.
```

and adding control flow, e.g.

```
html
  submitButtonWithAction:
    [(self confirm: 'Do you want to go negative?')
      ifTrue: [self decrease]]
  text: '--'.
```

is easier in Seaside.

Janko finds Seaside very good at supporting the back button, locating control flow in one place but it seems more procedural than OO in programming style, the URLs are not REST-like and cross-linking pages is difficult. Aida has persistent URL for every domain object and automatic

cross-linking of pages and is REST-oriented. However it has GOTO like programming for more complex apps and does not fully support the back button. See www.eranova.si/aida for more information.

Cryptography for Smalltalkers, Martin Kobetic, Cincom
Cryptography aims to achieve confidentiality by encryption, integrity by message authentication codes (MAC) and authentication by signatures. Martin's talk was on encryption.

Since encryption algorithms are hard to develop, they tend to be made public, with their secret part extracted to a key. Encryption approaches:

- symmetric secret key cipher: stream cyphers or block cyphers. These are used for bulk data encryption.
- asymmetric (public) key cipher
- one-time pad

Stream cyphers (Pike, A5, ...) generate a stream of random digits, used to encrypt with xor. Martin demoed by encrypting his slide with a key cypher. Never reuse the key of a key cypher, since

```
(a xor k) xor (b xor k) = a xor b
```

i.e. the key k can been eliminated by this simple operation.

```
key:= 'secret key' asByteArray
  "use bytes not strings, to avoid encoding issues"
alice := ARC4 key: key ...
msg := '...' asBtyeEncoding.... "msg is btye array"
alice encrypt: msg
```

A block cypher (Blowfish, DES, ...) is a fixed transformation applied to blocks (64bits, 128 bits) of the data.

```
key:= 'secret8key' asByteArray.
alice := DES5 key: key.
alice encrypt: msg "must be whole number of blocks long"
```

You usually pad with the remainder number (block size 8, message 15 chars long, pad with '1'). Block cypher is a fixed transformation so it does not well-encrypt higher-level structure. Martin demonstrated this by encoding his slides with a small block size; they were still readable.

Block cyphers therefore use a chaining block, each block encrypt being xored with the previous plain-text. The first block needs an initialisation vector, a non-secret but unique and random block-size value, and again you should not reuse it.

```
key:= 'secret8key' asByteArray.
alice := CypherBlockChaining on DES new iv: nonce b8 ...
OR
alice := DES newCBC_B8 "helper method"
```

Block cyphers can also use output feedback mode or its variant counter mode instead of chaining. Counter mode uses the counter as the chain value which means you do not have to generate your encrypt in sequence.

Straightforward DES with block size 8 can be broken today with a bunch of PCs so you can use triple DES, encrypt with first key, decrypt with second key, recrypt with third key.

Martin showed code fragments for doing all of these and there are helper creation methods such as

```
des3 := DES new3EDE_CBC.
```

DES is being phased out; it is not powerful enough. A competition to replace it had 15 submissions, 5 finalists and one winner, Rinjael (ref Anderson, Security Engineering, Ferguson Schneider: Practical Cryptography).

Q (Georg Heeg): in WWII, everything was enciphered and everything was read. Is this still the case today? Martin takes the economic view. No practical cypher is unbreakable but it can be made too expensive and/or too slow. Experts in the field release cyphers with duration estimates: “I think this will be good for 25 years”. How much is your secret worth? How much effort do you expect an attacker to devote to it?

Q. Most decryption arises from user error. People encrypt XML which often starts with a known tag, just as Enigma was helped by WWII messages always starting ‘Oberst...’ and so on.

Smalltalk has a chance to shine here. People complain most security problems arise from complexity and misunderstanding. Smalltalk is good at handling complexity.

TypeWise, Ernest Micklei, PhilemonWorks
Ernest showed a slide created in typewise that his wife thought would explain typewise. At the end of the talk we will hopefully understand it.

Simple things should use a simple UI. Today, the web is too complex for some tasks. User interfaces are moving from keyboard-driven to mouse-driven. His aim is to have character-based internet widgets.

TypeWise was developed in VA and is deployed with Smalltalk MT. Its architecture has a terminal client with HttpRequestor, and an HttpServlet with client proxy-talking to the ApplicationModel. The client executable renders and interacts with character-based widgets. The server handles messages from the client and keeps the application’s view state in sync with the client.

Ernest showed some attractive web screens with various widgets. He edited these, changing colour, appearance, etc. Standard internet browsers expect you to send an HTTP request returning an entire page. Ernest instead exchanges Smalltalk messages and does widget-level updating. He has MessageSend, MessageSequence and IndexedMessageSend for exchanging messages.

To develop an application, you create a TerminalApplicationModel,

TerminalScreen and TerminalApplicationModelServlet, then start the client using the server URL. You locate things on the Grid (row and column, not individual pixel locations). Use an EventTrigger to create event to action connections. Ernest uses stylesheets to simplify development. He has a helper class to build screens.

Ernest's whole talk was itself running in TypeWise. He now reached a slide listing demos and selected them to demo

- web-based class browser built with TypeWise
- pilot-training exam application
- Google query plugin for adding to web apps as a menu item

Q. Scrolling? Not using the mouse as mouseless operation is a key aim.

TypeWise can be used for monitors (e.g. Stock Exchange or Industrial Processes), for administration tools and for reservation systems. It will hindered uptake that it is not a sexy UI (of course, it is retro-sexy :-)), nor internet-Browser-based (must download and install typewise), nor a standard protocol, but Ernest has clients. TypeWise is free for non-commercial use, has a developer licence for commercial use, and is also available for connection to Java (sorry :-)). See http://www.typewise.org.

Q. Run on PDA? Interesting idea; the talk yesterday has started him thinking about that.

Squeak Etoys, Yoshiki Ohshima
Kedama is a system that lets you describe massively parallel complex objects. It was inspired by StarLogo (MIT labs) and StarSqueak (John Maloney's Squeak version of StarLogo). Simple rules let you show complex behaviour. Thus children can learn.

The system has a world (where everything is), patches (environments represented as fields of values) and objects. He created a block containing a gold dot and gave the dot behaviour. Then he added more: soon we had a hundred, then a thousand, then the thousand dots bouncing all over the field. He changed the rule controlling them to show the effect on the aggregations. He also had a large gold block showing the behaviour, in effect providing a zoom-in view of one.

Then he changed the values and colours (quick to do, drag-dropping mini-windows and editing them) and instead of a gas-like assembly we saw an elaborately hued expanding iris. He put the gold dots back and the iris gradually swallowed them along a cross-pattern of gradient.

Next he showed practical examples of using this in teaching:

- pheromone example: the dots were ants finding food from their nest.

- gas example: a PV = nRT thermodynamic formula was illustrated, computing the gas values from the actual behaviour of the confined dots. Moving the box dimensions changed the values and children could see why. Changing the number of dots indicated the statistical nature of the law.
- Compute the value of pi by giving dots behaviour that converges on it.

Next he took a picture (of a rubber duck in grass) and converted each pixel to an active dot.

In future he will unify turtle scripts and world scripts. He wants to move from atoms to more general space descriptions, suitable for describing circuits, astronomy, biological systems, the internet, etc.) Patches need to be extended or replaced.

He showed a sensor designed to measure dryness and output a sound. Computers have sound inputs so it took him ten seconds to build a graphical interface (the fact that his computer text was in Japanese characters underlined the value of the graphics).

Squeak in Japan has a lot of take-up, especially since a TV show featured it. The squeak-ja mailing list has 380 members. Japanese employees of HP are active in Squeak. There is also Learning project (the Alan-K project). A children's book on Squeak sold 5000 copies.

The audience thanked Yoshiki for his Squeak multi-nationalisation work.

Selling Smalltalk

Commercial case studies presented in the Business Day are in the applications and experience reports section; here I put talks that were about how to work commercially in Smalltalk.

Marketing Smalltalk - Business and Community Marketing, Monika Laurent, Cincom

It is something of a challenge for a non-Smalltalker to present marketing ideas to a group of Smalltalkers, not known enthusiasts of how things are marketed. She quoted, "IT staffers are easy as long as your product is great. Fluff won't work. In fact they will hate you for it, and never forget. Think religion, think crusades. These people love and hate."

Monika never wrote a line of Smalltalk (or of Java; she once wrote Cobol). Monika's talk was not an academic marketing talk, nor a historical view of Smalltalk, nor a description of how Cincom, or anyone else, should market. It was about how an outsider, but enthusiastic about Smalltalk, sees things, and about how to talk to other outsiders.

The business audience (those whose job is to manage IT towards profit) and the Smalltalk community audience do not much overlap. (Senior Smalltalk software engineers are the most obvious potential members of both groups.) These groups have different concerns. The business audience have a business or (not-so-)technical background. Personal success and

company profit are their concerns but personal and professional interests can differ significantly. The community consists of technical people who value more their reputation and research; their personal and professional goals overlap more. The community has high personal commitment and are short in budget. Business people are more 'down to earth' in several senses and rely on 'common sense'; they have budgets but the budgets are not their own budget. Monika sees the community as more creative and also more emotional.

Q(Niall) Emotional? What, us? :-) Smalltalkers care. Business people do their job during the day but it is very far from being their life.

Which marketing channels address the business audience: large trade shows (CeBIT) and Trade conferences/seminars are much patronised by business, little by the community. Direct marketing (mailing, emailing, telemarketing) work for business people, but only email works for the community. Advertising and the press work for business, only the latter for the community who are very contemptuous of advertising puff and always wish to test claims hands-on. IRC and such-like communication channels are useless for business, popular in the community.

Message contents also differ. Business people want to hear about ROI, about real-life commercial applications. They are very 'practical', un-interested in research whose connection to commercial use is not obvious. They like much more background, more explanation on benefit. She sees them as needing more neutral language, unlike the spare but 'leading edge, appeal to ego' presentations for IT people. A business audience likes a formal, serious, commercial style, whereas technical people often prefer the casual, sophisticated, creative, funny, game-oriented style.

Presentations that say how everything was better X years ago, or that are arrogant (we are the clever ones), do not succeed in business, and nor does simple neglect of marketing. We must focus on the future and on success in business. "Think not what Smalltalk can do for you but what you can do for Smalltalk". We need to exchange ideas on this.

- Companies that use Smalltalk need to talk to the community about what they need. Tell universities that educating fresh Smalltalkers is worthwhile.
- Smalltalkers must be aware of business concerns and willing to research areas of business need.

Monika hopes she will see us at the business track at ESUG 2005. There is also a Cincom VW Smalltalk user's conference (Frankfurt 7-9 Dec 2004).

Q. Do CEOs of Smalltalk companies have particular characteristics (e.g. more technical)? In small companies, perhaps. The head of Cincom has a sales background.

Q. Is the business versus technical split just the split of standard versus early adopters; Smalltalk is small so we only have early adopters whereas other technical activities have their 'only a job people'? Georg replied by

asking for a show of hands and got quite a few people who saw themselves as on the 'business' side, and more who saw themselves as on both sides.

Q. Could an actual selling point be made that Smalltalk bridges personal and professional interests: 'love your work.'

Lessons learned from starting a company with a Smalltalk product, Christian Haider, smalltalkedVisuals
Christian built a product for himself, found a client who wanted it so developed it further, then founded a company to market it. See his talk last year for a detailed description of the technical issues and some background.

He started with a 3-month contract to do chart programs for a magazine that had someone spending all day every day drawing stock market charts. They told Christian that three earlier vendors had discussed the problem but never come back with a solution. Christian did the work from time-to-time over a year. The smallness of the charts and the need for good graphics and printing were the main issues he had to solve.

His first client had done a market survey to find a product and failed before hiring him so Christian thought, perhaps there is a market, so he created a help website. For a year, his first client (Die Telebourse) used it and then a stock market crash killed them. Their parent company (Handlesblatt) wanted it so he had a half-year contract (4.5 man-months actual work) for them. His second contract (with Seuddeutche Zeitung) was a half-year but took a long time to arrive, longer than he had expected.

Meanwhile, his help website was helping his existing customers but making no new ones so he created a marketing website.

He showed some charts, mentioning that support for logarithmic scales had to be added when the stock market crashed :-). The difficulty is not to draw any one chart but to produce charts that will print professionally and be further editable. Each customer had new requirements, tables, context-specific name abbreviations, etc. Layout is controlled in Smalltalk which means he can add new rules very fast but the non-Smalltalk-using customers cannot. He got client data software (DLL and COM with no documentation) and analysed it to connect data to charts more effectively. The clients had very strong ideas about layout. (That was the market opportunity; layout that meets their exacting wishes.)

Q. Logarithmic timescale, like what Dan demoed? yes, I'd like to offer that; it would be a unique feature.

Christian then spoke about what he had learned.

It is important to have a unique selling proposition. That was easy for Christian, automating chart creation was already recognised as a value in his first client; letting the client wait as late as possible (get latest stock market data) and then quickly get chart for printing deadline. Consistency was another value (previously, each graphics worker had their own style

and you could track the changes from day to day during the week). There is much specialised knowledge in creating small charts that look good; it was hard for the client to hire the appropriate people because the job is very boring. His product replaced a cluncky loop of editors sending requests to graphics designers and then asking for corrections.

Why did he create a company, not just trade as Christian Haider? An independent legal entity gives him some separation. It is much easier to involve partners. The main reason was because the customers much prefer not to deal with natural persons.

Smalltalk has no role except that without it he would never have built the original product and that he fixes things quickly (as his customers have remarked). It also proved useful for exploring external interfaces on the fly. He was at the customer site with one afternoon to get his code working with the 'every twenty minutes' stock market interface for which he had no knowledge, no help and no prior warning (manager said, 'It would be nice if....') but by 23:00 he knew how it worked and at the demo next day it looked good.

Being a company is about profitability. It is also about interacting with people. It needs a transparent organisation; if someone calls when he is away, where is the information. You have to deal with bureaucracy: the government, maybe an accountant or tax lawyer. You need to specify people's roles.

You need to be professional. Test. Deliver. Track bugs. Track promised features. Track your time.

You need to find customers. Christian's idea of marketing was word of mouth; happy customers will tell others. That did not work. This industry has no gatherings, conferences or whatever. They just buy each others papers. He realised that the people he had to call were all listed in the 'impressum' of the paper. Then he needed to stay talking to the correct people. He would call the editor and say 'I have a nice tool for charts' and the editor would say, "I'll switch you to the graphics people", but these are the people he is planning to replace.

He found the website did not communicate, they just did not get the value, whereas a demo communicated immediately. If he managed to present SmallCharts at a customer site, the battle was usually won. The graphics people always use Macs but the data provider always uses PCs. VW will run on either but demoing on one while getting data from another can be interesting.

Pricing was hard; he had no idea what model to use let alone what amount to charge. At first he thought: charge 10% of development cost and find 20 customers. Then he realised he would get no money when he had exhausted the market. He has ended with a renting but, for all that Microsoft does the same, the customers start by wanting to buy, not rent. Next he had to decide how much to charge. He has no competitors to help him set prices and the

customers' graphics department is a general expense amortised over many things so they often have no idea either.

His mistakes were

- Not wanting to do marketing: however, he realised noone else could explain the value proposition convincingly in depth
- Underestimating development time badly
- Underestimated customer decision process time delay (they take 6 months to call back)

He also advised choosing your partners with care. Someone you have worked with but not gone to school with may be someone you know less than you think. There will be rough times and you need mutual understanding to carry you through arguments

Q. Fonts? usually from customer, not free.

Q. Enduring the waiting? Everyone told him that companies take 3 years to become profitable so he had a timescale to help him through the trough

Applications and Experience Reports: Commercial and Technical

This year ESUG had a business day, in which some more commercially-oriented experience reports were presented, along with the usual technical reports on commercial systems. I have arranged the talks in this section *very* roughly to have the more commercially-focused talks earlier in this section and the more technically-focused ones later, but as all talks tended to cover both aspects, the difference is not huge.

Hans-Dieter Brenner, Joseph Springer, Novatec

Novatec was founded in 1996 and does project reengineering in C++, Java and Smalltalk; Smalltalk is 40% of their business. Hans described a project for a customer in Stuttgart reengineering their public sector system to use ObjectStudio. There are 9 regions, 35 counties and 111 townships in Wurtemberg. The customer implements software for the regions who then distribute it to the smaller entities for client use. The customer manages various local taxes and charges (e.g. pre-school groups, dog licences, etc.). They also manage foreign residents. The previous system was an MVS-CICS, DB2 system on 3270. The new client-server system runs on PC clients. The client is built with ObjectStudio and PVCS version management. (The server still uses COBOL; legacy software and staff.)

The ObjectStudio framework builds the application from the domain model, providing database connections, forms, etc. A layered architecture also makes it easy to respond to future requests for changes to UI, to UI style or platform, to domain objects, to database layout or platform, etc. The customer's programmers had to learn the framework, a necessary discipline that they did not immediately acquire. One programmer worked the whole Easter weekend to do by hand something he could do quickly with the framework. On Tuesday morning, Hans showed him how to discard 80% of the code he had written. He used the framework after that.

Users map new support requirements to object models of transient and persistent objects, thence to a fresh support systems. To people who were accustomed to weaving COBOL and SQL, this was a lot simpler. The framework provides audit tracing (the state of each object over time) and all the other cross-cutting aspects that previously complicated their code (or were ignored). It also connects to a range of office tools (Winword, Excel, WordPerfect, AmiPro, etc.) that various offices are accustomed to use for related tasks.

Software roll-out was an important issue. InstallShield distributed the image to the server, thence all over the region. Redistribution of 10Mb images to fix a bug was not desired. Images record their basic version and patch version, and patches are downloaded to clients routinely, checked whenever they access the server.

One issue was making ObjectStudio talk to the financial administration system SAP. They had the choice of driving the SAP's API via OLE or RPC. They found problems with OLE so used RPC. This was done in three steps: generate remote function calls, use Structure Builder (a tool they built) to parse structure of generated C files, bind calls.

Another issue was communicating with the central Berlin registry. This was done via XML. ObjectStudio has an XML parser to output their data to a Tomcat Java servlet that talks to the Berlin registry. They plan to replace this with a VisualWorks servlet.

Q.(Andreas) Here we have a very conservative customer switching to Smalltalk, not Java, and making the decision not recently but in 2001 when Java was still flavour of the month; how come? We showed a prototype to the customer showing how rapidly we could build applications.

Q. SAP interface: was it straightforward? Now we have the know-how, it is easy. The customer develops a new Business API and the Structure Builder takes half a day to integrate it. (Maybe useful to port this to VW.)

Q(Herge) Frameworks make it easy to respond to requests for generic changes? Yes. However requirements are not really specified. You need to implement something, show it to the customer and get their reaction. Where a framework really scores is in letting you show the customer something quickly.

Q. Ever been asked, "who will maintain this if your customer dies?" Yes, they heard this argument several times, but the customer has experienced over the last 3-4 years hiring people, training them in Smalltalk and seeing them becoming productive within 2-3 months so they are not concerned now. They also develop with Java and the ANT scripts they have to build with Java are very complicated and you need a good programmer, so sourcing people is hard there to.

Georg Heeg: this talk, and its level, was very different from that we are used to; this is what business people want. ObjectStudio people are more

business focused. Georg spent two days with some OS people. On the first day, he explained the differences between OS and VW and they did not get it. On the second day, he showed them and they got it. “The second day was very useful.” We can learn from this.

The Value of Smalltalk: valuing and risk-management in VisualWorks and GemStone, Niall Ross, eXtremeMetaProgrammers, Colin Lewis, JPMorgan Chase

I find it hard to explain things I have learned by experience instead of by being taught them. Noone ever told me to use Smalltalk because ... . One day, in the early nineties, my quondam manager told me I’d be working on a project in some new language “... Smarttalk - er, no I mean Smalltalk, I think ...” and I said “O.K.; I don’t mind learning another language.” (This way of discovering Smalltalk is not uncommon. Avi went to OOPSLA in 2001 with a paper on Java, but rashly walked into the Camp Smalltalk room to see what was going on there; in a sense, he never left. :-)

A few years later the scene repeated itself, with my then manager telling me that now everything would be written in Java. The manager was more keen this time, but I was less so; less keen for myself, but also I didn’t think it would do the company much good. But how to explain why?

Over the years since, I’ve had several attempts at explaining to various audiences why it is not in their interests (or mine :-) to ignore Smalltalk. One attempt, aimed at students, is at the very end of my ESUG 2002 report (see www.whysmalltalk.com Events and Trip Reports page). More often the attempt was aimed at managers. Very often, in the years of the Java wave, I had but few and hard-won successes battling against fashion. This talk is the latest attempt, using my current work as example.

Kapital is a key JPMorgan Chase’ system. I’ll give just enough background on the system and its domain to make the rest of the talk comprehensible, then describe a few examples of how its reliance on VisualWorks and GemStone delivers business value. Kapital values and risk-manages a wide range of complex financial products. Some of these are commoditised vanilla trades, where the profit margins are made by handling large volumes effectively. Others are new really complex products: here the profit margin is in being the first to be able to handle them in any serious volume. In both cases, the key point is that you dare not sell or buy what you cannot price, and you dare not hold what you cannot risk-manage. Hence the capacity of your system for doing this determines your volume. The speed with which you can extend it determines how soon you can handle new products. And the interaction between the two determines how quickly you can expand the volume of popular novel products.

In the early nineties, when the idea of a system like Kapital was first mooted, much of this work was done by traders on spreadsheets. Kapital’s goal was, “The ease of a spreadsheet with the scalability of a system.” This was a *hard* problem to solve, but *very* worth solving.

Kapital has more than 500 end-users in the major financial centres, and is

supported by a multi-site team of 60+, half of whom are Smalltalk developers. It is used in three modes. Overnight, massive batch runs value all the traders' books against an immense range of conditions. This creates a map of possible futures: the books' actual values, their values in a range of possible future states, their exposures to a range of possible risks. The next day, the businesses use Kapital to manage their books interactively, valuing fresh trades, exploring possibilities and hedging against risks. The information computed the previous night means that Kapital can tell them much more that just what it can calculate for them in a trade's time-window. During the weekend, we do housekeeping. Various runs clean up the data, archive audit records, verify various conditions, and do sanity checks to ensure the framework models are holding up.

Various features of Smalltalk make it all possible. Firstly, forget rapid delivery, cost-effective delivery and all those nice-to-have's and think simply about actually delivering anything at all. Kapital solves a hard problem (if it were easier, we'd have more rivals). One hard problem aspect among many is that Kapital's domain (and also our understanding of it) changes, so it needs dynamic domain models.

Kapital's infrastructure meta-model of graphs, closures and references ensures that financial developers don't have to worry about persistence. Complex models of financial products get persisted, and their data integrity and audit trail assured. (This meta-model has behaviour in both VW and GemStone.) Above it, a financial domain meta-model of markets, trades, cash streams, etc., lets new financial products be created rapidly. In this model, all objects can "mark to market" (i.e. value) themselves, all objects can walk their graph to explain their values against variations in time-period, pricing, etc., etc. (This domain meta-model has behaviour in VW only, GemStone acting as a virtual memory extension at this level.) It took 1.5 years to get these meta-models right. It was *so* worth it. "New financial developers find the ease of the persistence framework unbelievable."

The bottom line is that meta-modelling is so easy in Smalltalk - that you can deliver something that makes money before your project runs out of money. We know of a bank who attempted to replicate the system. They had legitimate full access to Kapital's codebase and employed skilled staff some of whom knew Kapital. Their only problem: as this work was being done at the end of the nineties instead of in the early-mid nineties, they 'of course' used Java and a relational database instead of VisualWorks and GemStone. The project ran for about the same length of time as Kapital's initial start-up, burned all its money and ended; nothing was delivered.

Meta-modelling is easier in Smalltalk because Smalltalk exposes its own meta-model: everything is an object; there are few reserved words. Kapital's meta-models were developed by trial and error, mixing and matching what was done by Smalltalk classes and what by 'meta' objects. Kapital uses the standard Smalltalk meta-modelling approach: a classInstVar 'meta' holds data qualifying what Kapital's model-layer classes mean and how they relate. Much code walks graphs of these objects, guided by the meta-data. As Kapital developed, fine-grained

refactoring between what ordinary Smalltalk did and what the meta-system did was essential to getting the latter right. Specific methods could be integrated with generic graph-walking methods because there were no type-system obstacles to refactoring between code and meta-code[1]. For the same reasons, it was easy to extend VW's and GemStone's coding tools to embrace the meta-system. (As a minor example, I showed a hierarchy browser whose coding tool also handled slot definitions: meta-data qualifying the behaviour of instVars for model-layer classes.)

Once the system was up and running, a second value of Smalltalk came into play: rapid delivery. The infamous DarkPlace Dogytalk letter, GemStone's Brokat phase and all the usual managerial pressures on an unfashionable language had the same effect in JPMorgan Chase as elsewhere. The company maintains 'buy' lists and 'sell' lists for technology. After the letter, VisualWorks went on the 'sell' list and stayed there for years. GemStone went on it during the Brokat troubles and was not officially removed from it until last year. Kapital survived this because we can mould it to do what our users want in volumes they cannot handle elsewhere. In frequent prototype demos, Smalltalk lets us change things in front of users as we demo, letting them see what they want and (as important) what they don't want. A single new product could be described in a spreadsheet (often is, when it begins as a gleam in a trader's eye) but you quickly hit the volume and complexity wall using spreadsheets.

I described one example. Not so long ago, *a new financial product* appeared and was very popular in the markets. (If you work in the business, you may be able to guess what it was. If you don't, its name would mean nothing to you. I'm not allowed to say what it was.) Every client of every investment bank asked for quotes for it. For *a key period of high market activity* (I'm not authorised to say how long), only JPMorgan Chase quoted new business for them. JPMC gained the business, the premium for being the only one in that market, and (also a key commercial value) the connection to additional clients. We do not *know*, but have reason to believe, that competitors used lots of extra spreadsheets and staff to enter the market when in time they did. By contrast, the new product added minimal performance and maintenance costs to Kapital, which thus kept lower overheads and freedom of action (opportunity cost).

Incidents like the above are obvious to Kapital's users. To those who support it, another key Smalltalk value is scalability. Kapital has a distribution architecture of Client, Agent and Resource Manager images. Agents do calculations. Clients task Agents and persist their results. Resource Managers control what images are started where. Detailed configuration of how clients group tasks for agents is supported and has major performance effects. Images communicate by database and by Distributed Smalltalk. The vendor said we stressed DST more than anyone else (and fed our fixes back into the product). As well driving a large server farm, this architecture lets us steal cycles from desktop PCs all night to help the batch run complete. This was started a few years ago as a nice-to-have; as volume kept shooting skywards, it became essential within 3 months.

1. See Niall Ross' talk at ESUG99 for a detailed analysis of this issue.

Scalability is a key driver for Kapital and visibility is key for scalability: you see true bottlenecks only in the production state. Because of this, unlike typical systems of this size (10,000+ application classes and not-a-few changes in base classes), Kapital is not stripped at delivery. A release image has the same codebase across all sites and businesses; production images have development tools hidden within them. This combination of a common code base with Smalltalk's dynamic features lets us *study* bottlenecks in situ. Example: recently, we noticed what was called 'the Friday slowdown'. This was diagnosed as the week's-worth of new object oops committed to GemStone (before weekend house-cleaning reduced them) passing the SmallInteger/LargeInteger boundary, thus altering the hashing method used by a key dictionary in a tight loop, so leading to extra message sends in performance-critical code. Everyone remarked afterwards that, "I would never have guessed that." Being able to see, interrupt and experiment dynamically in Smalltalk was key to this diagnosis.

That was just one example of how Kapital has overcome one performance threshold after another as it grew. Kapital on-boarded Credit derivatives, an entire new business area, significantly faster than our competitors. It switched from shared memory processor technology to blade technology in six months (an example of the fact that Smalltalk is also easy to move).

With complex meta-models to maintain, and a rapidly evolving market, another Smalltalk value, reengineering, is vital to keeping us alive. Fine-grained access is key to reengineering; you can get at so much in VW and GemStone. I described one example in some detail. Concurrent systems need canonical (i.e. unique) objects. Kapital databases store the unique roots of certain graphs of financial domain objects in a performant symbol-keyed cache, each image loading what it needs. In this cache are the keys ('descriptors') of financial time-series objects ('curves'); Kapital started with 200, and now uses 70,000 (indicative of how its volume has grown).

Retrieving 70,000 descriptors began to slow an important UI operation. At the same time, we had noticed (as have other GemStone users) that saving many copies of the same date bloats databases, and it is quite possible to synchronise image dates and unique database dates lazily, so performantly; this had been implemented. Re-using this pattern, we reengineered to use date-style lazy synchronisation for curve descriptors. No sooner was this done than our pesky mathematicians revealed that, as it was slow and costly to gather data for certain curves, they had worked out a new theory that derived all such curves from a few base curves. Now, Kapital would create new curves on the fly, so needed to generate curve keys on the fly. Deep within a transaction, after all locks were obtained, all transaction state set, we might suddenly discover that we were doing a write, instead of (real or lazy-synchronise) read; Oh joy !! :-) However, you can get at things in Smalltalk; the person who did the original synchronised dates work guided us to a pattern for lazy synchronise, read or write.

When code evolves, data must evolve to keep pace. Kapital's schema keeps evolving to keep pace with the market. In Kapital, the persistence framework is integrated with the mutation (data migration) framework.

Data is lazily mutated as it is loaded into an image, thus migrated in the database if saved. For some changes, one must write `upgrade` methods for classes, but thanks to the framework an amazing amount happens automatically. For performance, recent and important data is migrated on each release for all production databases; other data is lazily mutated if loaded. Thus data upgrade on release takes less time. More importantly, developers can run their latest codebase against copies of production databases without having to upgrade everything first. When you want to test your code and find bugs early, that is very valuable.

(Technical aside: a corollary of this is that in GemStone, selected classes - 'backport classes' - push method changes to their previous versions to avoid having to migrate. Only structural changes to such classes compel us to migrate their instances. There is debate in Kapital whether to continue with this or to move to migrating on behavioural as well as structural changes. Recall that we put no domain behaviour in GemStone, only framework behaviour. That is both why we can get away with this approach and the motive for it - to keep as much migration behaviour as possible in a single place, VisualWorks, for simplicity.)

Having given a few examples of how Smalltalk helped us, I ended by listing a few of the things we would like to do better. The original framework design forced cloning of the entire collection class hierarchy to support 'meta'-enabled collections in domain graphs. We would really like to eliminate these. We also want to eliminate dead code. Until recently, loopholes in our tools encouraged people to neglect deleting unused classes; there is ongoing work on static and dynamic dead code detection and deletion. There is also ongoing work to remove unneeded objects in our databases ('dark matter'). GemStone's object table can reach 2 Gig, occupying the whole shared cache (by copying the data we need to new databases, not by hunting unneeded data in old databases). We also want to enforce good coding patterns: we have superb code and not-so-superb code. More info on Kapital can be found at:

- https://secure.cwheroes.org/briefingroom_2004/pdf_frame/press.asp?id=4909
- https://secure.cwheroes.org/briefingroom_2004/pdf_frame/index.asp?id=4909

RUT-K: Fast large-scale train scheduling with Smalltalk, Marion Jurish and Joachim Geidel

Marion explained that in Germany 40,000 passenger and freight trains must fit on 65,000km of track, controlled by 8,500 crossings and switches. At Frankfurt there are 90 trains every hour; 400 users in 7 offices must manage them. She saw it as like solving a large puzzle. Exact construction of a train's path requires detailed knowledge down to track level and detail running time data. Timetabling is done by a team, so the system requires full multi-user capability and must show all data. You also have to explore timetabling variants, so the database must manage multiple versions and support merging them correctly. Lastly, this is done multi-site.

As in the UK, there is legal separation of railway companies that run trains from railway infrastructure management. Suppose a railway company wants to run a train from Frankfurt to Munich every Monday January to

August 2005, stopping at various intermediate stations. You need train objectives, stops, size, speed, etc. info, plus track info to output a timetable for this train that satisfies all requirements, especially the requirement that two trains not occupy the same length of track at the same time.

RUT-K supports the train path managers by effective display of complex data, by graphical data editing and by producing reports. It calculates running and blocking times interactively with the users, shows conflicts, and computes the timetables. Performance is the hard part of this task. You need very fast calculation of times, very fast zooming of displays of problems and fast update. Each client uses 100 Mb of data. She thanked John McIntosh' for his profiling work on their system which was very helpful in letting them meet their performance goals. There are 250,000 special trains every month, so this is no 'redo timetable every 6 months' task, especially as the capacity of the railway system is very packed today.

RUT-K was deployed in production in April 2003. It joined several Smalltalk systems already used by DeutcheBahn Netz AG. She showed the architecture of how these interacted. The success of this project was due to

- involving users: define requirements and tests, detailed spec
- short development and test iterations
- good organisation of a good team
- using Smalltalk: reuse of components from other DELTA applications, easy build process, powerful class libraries, programmer acceptance.

She then handed over to Joachim who demoed (using some test data as we had no link to the servers). He showed cases involving trains arriving at stations and reversing to leave them (speed calculations differ). The system produces times to within 6 seconds and trains can be (and at times are) run to that precision. You enter data for train mass, weight, etc. and its tilt on corners (the classic omission error of the UK's Advanced Passenger Train project of unhappy memory). Routes are entered graphically on maps of the track network. Location-derived popUp menus let you invoke sensible operations for regions where you have put the mouse.

The schedule running and blocking display was a multi-coloured network. The colours help users locate problem areas needing work in the big picture and then they can zoom in fast to detailed views, including schematics of the track with its signals and junctions. They can switch to textual views of data as desired. They key idea is to reduce conflict detection to a geometric problem, formatted and colour-coded for ease of user examination. It takes them 9 seconds to process 300,000 by 300,000 rectangles that make up the main graphic; conflict detection takes 0.1 seconds. (When you have millions of graphical objects to handle, John McIntosh' advice on memory management is useful.)

Q. Problem so complex you have to use humans to solve it? Simulation is used to compute running time. Conflict detection is easyish (NlogN). What is very computationally hard is solving conflicts, finding a new solution that has no conflicts. Constraint-programming algorithms might be an

approach but there are many ways a train schedule can be manipulated - increase/reduce speed or halt times, use another route, relax a requirement, etc. - it is an NP-hard problem. You also do not have all the information at one time (e.g. special trains, etc.). Today's train schedule is the result of work and decisions over the last few months. Lastly, people know some things that are hard to express in numbers - who cares more, who less whether trains run exactly to time, etc. Thus today, humans must do it. The goal is not to find an optimal schedule but a feasible schedule. (The Netherlands do use computers to provide a first-cut solution but they have far fewer trains and less track.)

Q. A discussion of how real-time scheduling around problems revealed fascinating information about how in Germany the punishments (I think he meant 'penalties' :-)) for late trains vary between states; some have punishments for dirty trains as well. Thus many factors affect how such issues are resolved, the aim being to get back to the planned schedule.

Q. Smalltalk acceptance? There are a lot of Smalltalk projects in their customer and company, enough that it is not hard to justify more.

Supporting Teachers in Computer-Aided Education and IT Administration, Carsten Harle, Straightec

Straightec has done work in various Smalltalk projects (including for DeutcheBahn, see railway talk above). Teachers have to teach pupils, to control them, to do IT administration and sometimes to do IT installation. Modern schools may have several hundred PCs and several thousand users using intensive multimedia applications. There is usually no dedicated IT administrator so teachers must install and use many software applications. Teachers must distribute software to the PCs.

They must also find some time to teach, and ensure their pupils are not playing games on their computers. Some pupils have a lot of energy; PCs may not work after encountering such pupils, whether deliberately or accidentally, so PC settings must be tamper-proof. Screens must be sharable, so the pupils are looking at the screen the teacher intends and the teacher can see a pupil's screen and help them. Internet filters are essential.

Straightec developed a tool DX-Union Olympia for these tasks. (It won a Windows award 2003.) You plug your PC into the network and it installs the software you should have. A protector guards the hardware and BIOS against viruses and manipulation. A utility lets the teacher start an app on 30 PCs, close minesweeper on 20 of them, along with the printer drives and internet, and prevent their being restarted for the next hour. The installer works from a google toolbar. The general approach to maintenance is 'don't fix it, reinstall it'.

Carsten then demoed. The tool uninstalls to an empty PC state and then finds everything to install, partitions the hard disk, and installs. This can take 5 hours but guarantees return to clear state. The tamper-proof restart takes a minute and is good enough for most intra-lesson issues.

Once running, the control page shows a picture of where the PCs are on the tables in the room. You can drag-drop, get visuals of the PC state (is on, who is logged in, etc.), control display (including blanking screen - 'O.K. stop looking at the PCs and listen to me'), correct pupil input, and share a pupil's screen.

A multi-explorer is like clicking on 30 explorers at once. The teacher can create a folder on all pupils' PCs, copy a file to it on all PCs, etc. They can also collect 30 copies of pupil's work, appending login name to each, back to the teacher's folder. All state data is updated in real time, e.g. even while dragging a computer icon you will see if it has been switched off or on.

He showed the teacher detecting which pupils were playing minesweeper and closing them remotely, and equivalently starting applications (e.g. minesweeper during a WWII history lesson) if the pupil has not started the requested application. Transferring files (e.g. answers) from files instead of typing them can also be blocked. At the end of the lesson, the teacher can switch all 30 PCs off remotely via a single command (or log off, or start new application, ready for next class, etc.).

Next, Carsten demoed detecting and remedying problem states in the PCs.

Why Smalltalk? Carsten has been in Smalltalk for a long time, so the decision was easy for him, but it also made a great deal of sense for the application. He wanted to distribute, for 1000 euros or less, a whole shrink-wrapped suite. VisualBasic and C++ give worse diagnostic capabilities, essential when building this, and slower development and deployment.

Why Dolphin? They needed good window look and feel, and integration (this may be the first Smalltalk application ever to get Windows Logo verification). They looked at VW, which was very powerful but has deployment issues for their scenario, and the executables were sizable. VAs executables were very sizable and it was costly. Squeak and Smalltalk/X did not have the right UI look and feel. Smalltalk MT was too low level in its GUI (excellent for low-level windows work but not for their application). Dolphin Smalltalk XP had excellent windows conformance, was cheap, had a simple deployment mechanism and produced small shrink-wrapped applications. Its only limitation, that it is a window-only VM, fitted their market. It source management is an Envy clone (for just $50 !!!). Carsten was very impressed with Dolphin.

This is aimed at German schools today but will be extended to an English-language version soon. It may also be adapted to the needs of small companies (few hundred PCs total) looking for easier IT administration.

Unique selling Propositions with Smalltalk, Adrian Lienhard and Lucas Renglii, netstyle.ch (www.netstyle.ch)

Netstyle.ch was started in 2000 doing websites in Windows and .asp. Now it has 6 people doing web applications in Seaside. Companies of all sizes across all industries are migrating critical business applications to the web. Quick development and deployment is a key value, especially in letting

companies react to new opportunities. The customers expect well-factored low-maintenance applications but web apps are in fact hard to do in today's frameworks, which tend not to scale.

The talk's case study is work they did for a medium-size Swiss health insurance company. They needed to create and manage offers (creating the PDF documents for them on the fly), model the process of accepting new customers, and manage customers (i.e. manage the associated documents). It was a fast growing company with rapidly changing requirements.

Netstyle did an evaluation to find a technology that fitted web applications and this customer's needs. The answer was Smalltalk, and especially Seaside. Adrian demoed the application, entering some customer info to the web pages then saving to the database (a relational database; in the demo, they used postgres) and creating the PDF file of the contract.

They find excellent synergy between Smalltalk, eXtreme Programming and web applications. The customer was happy to receive frequent early drops to test, accepting that they would see bugs, and told them when it did not work as they wished. This early feedback often saved them from wasting time and opportunity cost through developing down wrong paths. They find they have a real competitive advantage over those who work with Java, Struts and PHP.

Q(Helge) How do you market your unique advantage? At the moment we get our customers from other customers (word of mouth) so we justify by pointing to past results.

Q. PDF framework? Bruce Badger's SPDF (VW), ported to Squeak. (It is on the VW CD.)

Q. Follow-up work? Yes. The customer is giving us further work.

Q. How many customers do you have? 50 overall, but some of these just ask for web page design and web hosting, not for web applications.

Q. More demo's please? He showed a business-card-ordering web app that let consultants design personalised cards within an overall corporate identity card layout. This saved the customer the previously high cost of iterating between consultants' finicky requirements and the printers. The model of the business card is done with Morphs. They imported all the special fonts the company wants into Squeak. Once designed, they convert it to PDF and that's what the printers get.

Q(Alan Knight) All day, people have commented on how hard it is to be allowed to use Smalltalk. Now you tell us how a young startup with guys fresh from university using weird open-source Squeak can persuade Swiss Health companies to use it; well done!! (Adrian) Our customers have heard of Java, etc., but they have no IT departments to tell them what to use.

A Smalltalk-based system for Dynamic Multi-context information processing, Adriaan van Os of Soops and Tim Verwaart of LEI
Soops is a Smalltalk company founded in 1992, based in Amsterdam and Wurzburg, specialising in real-time commodity exchange systems (e.g. power trading).

LEI is an agricultural economics research institute, providing a farm accountancy data network. Sixty employees collect huge amounts of data, twenty employees prepare statistical reports, used by business managers and politicians. Rapidly-changing policies force rapid adaption of the data systems. Different projects, focused on financial, technical, ecological or hydrological aspects, want to use the same data from different viewpoints.

All this made them want a generic system that would dynamically generate system behaviour from business models (slogan 'no more programming'). Tim explained the business model, after which Adrian explained the implementation.

Data is costly to collect. The ability to reuse it for different purposes ('multi-context use') is therefore valuable. A tractor's cost is important to one user, its engine power to another. They model entities (e.g. tractors) about which they hold context-independent facts (cost, engine size). Facts about entities may be grouped into Aspects. Context-dependent data models hold many concepts. Given aspects are relevant to given concepts.

Relevances may be governed by relevance rules (e.g. if the quantity of potatoes is more than 10 tons, that is relevant to this regulation) and integrity rules (has a requirement been met) and actuality rules (time-related, e.g. an asset's value must become known within 90 days of its acquisition). These rules say when an action must occur. How that action may be done is held in procedure and interface descriptions. A procedure is a sequence of tasks, connected to relevance by CRUD specifications (yes, that's not a typo: Create, Read, Update, Delete specifications :-)).

Data entry screens must be generated *and* structured *and* ergonomic. CRUDs tell the system what can be on the screen,. layout data tells the generator where things can be on the screen. To do this without programming, the screen editor is a grid on which you define blocks (rectangles) to which you bind text fields, tables, buttons, etc. Adrian showed an editor definition and then the actual resulting screen.

Performance is the hard part since every change meant evaluating an integrity rule which could recursively trigger other evaluations. Even harder, every change means that widgets may appear or disappear according to the relevance rules.

Persistence is provided by GemStone. They made a design decision to write their domain level code so it would run, needing no change, in both GemStone and VisualWorks. Thus they do not use GemStone specifics (indexes or whatever) but they do have a very transparent persistence layer. GemStone offers a 4Gig cache in Windows so when you are using 20Gig

data obviously you cannot keep everything in the image.

To enhance performance they do all rule evaluation on the server and cluster the data. They also redesigned to reduce the number of objects by using the flyweight pattern, by making the default date implicit, and by simplifying numbers (e.g. 1.0 becomes 1).

Smalltalk on a Windows CE: Experience Report, Mattias Schnoor and Marten Feldtmann
How many have a Windows CE? (A few hands were raised.). How many program in them (fewer hands were raised). Marten's manager had one and suggested this work to him. Marten was no VW or Squeak guru but he was a Smalltalker, having worked on Document Management Software in VA for many years, so he wanted to do the work in Smalltalk. He used VW7.2.

The project was to collect working hours at building sites. Normally it took them weeks to get working hours collected from sites, and so compute payments. The manager had the latest PDA and wanted to use it to collect the hours faster but did not want Smalltalk ("Why not use Visual Basic?") and did not want to build it inhouse ("that's unproductive"). Marten had no Windows CE development experience so went to CeBIT to look at devices.

The WindowsCE client talks to the internet, an SMPT relay or directly to a server. He used the PUM (Poor Users Modeller) source code generator (a C# generator but he wrote a VW back-end). His manager did not answer various 'use' questions (inside / outside, keyboard / stylus, etc.). He found

- dBook1: 900 - 1400 euro: which they would pay (wouldn't pay that for Smalltalk licences): partial weather protection
- skeye integral: 1900 euro, full weather protection (still not too dear).

IDEs:

- VisualStudio 2003 Professional: large larger largest :-)
- Java: bad reputation from 3rd party developers, no SWT support
- Squeak: looked like a possible candidate but no solution out-of the box (later found that there were solutions out there)
- VW: being demoed at CeBIT 2004

The project had 10,000 euros. Cincom refused VAR licence for project so made it a higher cost per year (2000 euro per year, not acceptable to management). To solve this, he changed the development process and founded a one-person company just to get the VAR licence: he concludes that the Cincom licence is too complicated for small developments.

The skeye integral socket stuff did not work at first due to a false secondary exception handling issue but new CE parcels were supplied in two days and fixed this. The dbook1 did not start at all. A new virtual machine was needed due to a special machine instruction on the dbook1 that flashed the precompiled instruction cache. They got two other devices for testing: Aldi Medion and IPAQ 2210. The general advice from the machine makers and

3rd party was not to rely on testing on just one machine; things change fast. They had very good support from Georg Heeg during this process. (Georg invited Helge Horsh and Ralph Oba, who did the support, to take a bow.)

Generally, the process works. Problems were due to the very heterogeneous CE market. Native DLL usage varies (some DLLs can be missing). You need to check that your application runs on the most selling PDAs. Support for low-end devices would be welcome (will improve in the next two years).

The raw VW GUI was too slow, Pollock (VW7.2.1) if anything slower than Wrapper. Using a PDA with special graphics accelerators solved the problem, so it looked fine on dbook1 which has them; they only realised later that there was an issue on devices without them. Normally, people doing this use non-Standard widget sets. Smalltalk needs to do the same; use a light widget set. Squeak shows it can be done; see squeak demo (Small Memory Footprint project - Faure and DynaPad - at dynapad.swiki.net/1 and russell-allen.co/squeak/faure). Smalltalk other than the GUI is fast enough. The behaviour of Squeak's magnifier lens on these applications shows there is no fundamental Smalltalk GUI problem.

He then demoed his system in VW Wrapper and VW Pollock both with and without a graphics accelerator. He also demoed the effect of Squeak's light widget set in Squeak browser and Squeak DynaPad (which he would really like to have). With the graphics accelerator, it showed prompt response. Without, it was usable but you could see multiple redraws in VW Pollock and could count to 4 between invoking and seeing the full screen repainting after the slowest operation. Georg remarked that further work on their Windows CE system begins in two weeks.

Q. Multiple Pollock redraws fixed recently? You won't see them on a standard PC but you will see them on the non-graphics-accelerated PDA.

Q. DLLs? Always go to google and ask if you are doing anything with such machines. You learn that strange DLLs have been more strangely reprogrammed to make someone's program run faster.

Using GemStone, Georg Gollman, IT Services Vienna University of Technology

GemStone can be evaluated for free. The university administration has 25,000 customers (20,000 students and 5000 staff). There is not a great deal of data but there are a great many dependencies and (too :-) frequent reorganisations. A single part-time developer must keep it all working.

There is campus software, student software (they began offering this five years ago; it has been very successful), white pages and mail router feeds (translate generic external addresses to specific), plus the usual authentication, account management and system support. Georg showed the diagram of classes for handling some of this.

GemStone is a fairly standard Smalltalk dialect with the unusual feature

that Arrays and Strings can grow. It is also a database with persistence, concurrency, indexes and access control. It also has constraints (instVar type constraints) which Georg has used in reflection, to translate data coming in from the web. Classes can have multiple versions and you can control migration of instances from one version to another. It has multiple symbol dictionaries instead of a single Smalltalk symbol dictionary (i.e. it has namespaces).

Back in the nineties, GemStone attempted a GeODE add-on graphical interface builder using the visual programming paradigm but they found, as many did, that the visual programming paradigm generally, and so GeODE specifically, was not ideal, so they moved to a WebServer implemented inside GemStone/S. The mechanism is just message passing:

```
/receiver.message?param=value
```

For security, the method must reside in the HTML method category, else the Web could invoke any method. There is one web server process for the general public, others for specific tasks. Theirs was one of the first web servers to be ported to Squeak and influenced others. The front end links to SSLeay/OpenSSL for security. It uses GemStone's remote procedure call login, not linked login, to keep things in separate address spaces.

He is now working on making a development environment accessible via the secure Web interface, class browser, etc. He demoed this to us. The web interface access to the debugger is very rudimentary but this at least forces him to design for testability.

The framework has a simple interface, asLink:, asHTML:, tabulate:, itemize:, title:. Similarly there is form editing protocol, etc. He demoed some forms, with the usual widgets.

Gemstone is a transactional system but some actions cannot be undone by abort (e.g. sending an email) so you must delay them until the commit. These actions are put in a session state queue. A general exception handler keeps the system alive, returning only a very general message to the user (who might be a hacker) and a more specific email to the system administrator (since it might be a bug).

The system is 95 classes 282 methods., runs on HP L1000 (dual 360 Mhz, 786 Mb memory) and they have a Linux box for testing. The database is not large: 1 million objects and 100 MB data. The shared page cache gives good performance. They have 370,000 hits per month, mostly over the socket layer SSL. The machine spends most of its time idle.

The system holds its own user documentation, with notes able to reference any object. They used eXtreme Programming as a guideline. He finds that small is efficient and homogeneous is efficient therefore he does everything in GemStone. He is about to get a second person on the project, and will improve the development environment to have change sets and add more SUnit tests. He finds access rights the hardest things to test.

He find this a stable, efficient, rapid application development environment. He recommend that GemStone vary the licence from the current 'number of machines' licence, e.g. by offering a cheap licence for databases with less than 1 million objects, to tap new opportunities.

Call Centre Application in GemStone, Petr Stepanek
They are building a call centre application. The customer is paying a high price for each minute of the call so the agent must be served with all relevant information; when did they or another family member call before, who served them, who recorded data about them, actions taken, etc. All this data will live in GemStone and will need to be retrieved fast.

The call centre request may arrive at any edge of the data model and need to retrieve all the linked data. The approach is to implement a linked object model supporting frequent addition of new link types with new meanings. Links need to be hooked and unhooked, and invalidated, often at some specified future time. This requirement has a very poor match to relational databases and a good one to an OODB; their choice is GemStone.

They keep the links and link traversing code in GemStone, returning the data in reduced conflict classes to the VW client. All data is timestamped to permit viewing current and past-time-window data.

Petr discussed pros and cons.

- It is complicated (not a natural approach).
- It was at first confusing (they could not inspect until they used Trippy's inspectorExtraAttributes)
- It is slower (but you can fight this with caching).

As against this,

- it is less work than other approaches
- it lets past data be handled by the same fundamental model as current data
- you can change your model with no data migration requirement
- you have a high change of a successful transaction commit.
- It is good for stochastic requests, which are common in this domain (e.g. crash on highway prompts sudden flood of calls).
- It is robust in handling unusual requests.
- Because the system starts retrieving and caching related data at once, it is faster than more constrained ones at being ready to answer when it needs to.

There were some questions and discussion about time-evolving aspects; I was preparing my afternoon's talk so did not catch this.

Blogs and RSS: Trawling the web for meaning, James Robertson, Cincom

James opened his blog, where he had just written notes on the immediately preceding talk (Petr's). A blog (web-log) is an on-line journal. To James, the point of a blog is that it does not have to go through the marketing department. Other companies (Oracle, even Sun and MS but not IBM yet) use blogs for this purpose. Trade blogs are becoming common and influential. Odd though it may seem to us, there are even more people interested in politics than in Smalltalk and there are lots of political blogs, hobby blogs, etc. James now has 3-5000 daily readers of his page (and no idea how many people check his RSS feed).

At first he had ~5 readers per day. Then he got email asking him to add an RSS feed. He replied that he'd be happy to if the requester explained what on earth an RSS feed was. He learned (that RSS has its own politics so RSS formats have some surprisingly optional tags and there are modules that duplicate these optional tags, etc., all making life harder for BottomFeeder and similar newsreaders :-)). Smalltalkers are not the only computing science group who get religious about their technology, as he had previously though. In fact, there is no technology so arcane that it does not have such a band.

RSS provides blog content as XML with more tags. When he did it, he had no idea that it mattered but in fact it does. RSS is a syndication format, needed because while some people go to a blog's web page and see if it has new stuff, others have a news aggregator that gives them a summary of which sites have changed and how much. It deals with the problem that you don't want to manage a thousand bookmarks and guess which of them point to pages that have been updated since you last looked (e.g. RSS lets him deal with the fact that a few nerds like to add spam to wiki pages. It's easy to fix once spotted. RSS feeds of the pages let him spot it quickly.)

At first, he added an RSS feed by building an XML document by adding tags, which is the stupid way to do it. Later, someone told him about SAX drivers and he used one. He wrote a small bit of code (four classes) to parse RSS off the web and rashly put it in the repository. Then Dave wrote a UI for it. Then people started loading it and asking for more features. Thus BottomFeeder was born. James had thought RSS was simple (Really Simple Syndication, or Really Stupid System :-) but he found it was not so. (www.hebig.org/blogs/archives/main/000877.php has some aggregators.)

BottomFeeder started with an RSSFeed and RSSItem, shown in a standard 3-pane screen. Handling all RSS formats from 0.9 to 2.0 is hard; they're not as alike as the version numbers suggest. You have to handle lots of internet errors, lost sites, garbage XML, network errors. Storing the XML as a file took 60 secs to load when feeds rose to hundreds so he moved to BOSS. They use TwoFlower as their HTML browser (will soon move to Michael Lucas-Smith's browser). BottomFeeder started as one package of 4 classes and is now 21 packages and 120 classes. Jim (Columbia DC) and Dave (Vancouver) coded, Rich Diemers (Minnesota) created the documentation, Holger Klien (in Germany) helped with TwoFlower. They used IRC a lot.

BottomFeeder is the first Smalltalk application that James runs every day on his desktop. It has plugins to help him access IRC as well. Get it from

- www.cincomsmalltalk.com/BottomFeeder

A year ago, Michael Lucas Smith (withStyle, well worth looking at), asked him for a blog. He delayed two months because he'd written the server as a single-user applications and wondered if he could fix that, not because he was talking to lawyers, but he found he could and now there are 16 people on the server, many of whom are non-Cincom people. (Usage rules: it must have smalltalk relevance and no politics; otherwise, write your thoughts.)

- www.cincomsmallalk.com/userblogs

Why blog? A blog is an unvarnished set of opinions rather than something smoothed by the marketing department. It is also an opt-in system so bloggers need to be interesting and trusted or they loose their audience. Blogs let you get a wider audience. If you are linked to by influential bloggers, you get noticed.

Sames is replacing the VW GUI with Pollock. He has a blog that describes example applications, with parallel posts by Vassili on how he would do it. That conversation might not otherwise have happened even though they work for the same company.

Q. Can we use this as open-source? Small-scale usage noone minds, while large scale users will contact James to ask how they get support for this critical product, so he is not too concerned. Just start using it.

Q. External users? Roger Whitney at a university and someone else who has contacted him.

Q. How long to use BottomFeeder? First thing in the morning, Jim will see 70-ish updated feeds and take 45 minutes to clear that, skimming much. During the day he revisits it a couple of times (sometimes as background in a long phone call). He showed the feed updating, and read Michael Lucas-Smith's latest blog.

Smalltalk-based Speech User Interfaces: The SpexKit Platform, Thomas Brey

This framework for rapid development of speech user interfaces is implemented on VSE, from which they are thinking of porting to VW.

Speech User Interfaces are very complex (or very user unfriendly). They need

- mixed-initiative: mix of application-driven and user-driven interaction
- natural language: must parse sentences, not just recognise phrases

Guru wisdom is that implementing real speech systems is beyond us for now ('A visual rather than a verbal future': Walker). Thomas believes that for a cooperative user subject to constraints (natural constraints of context so not obtrusive to user), valuable systems can be built. Such systems must recognise and parse speech from audio. Dialog management handles the

results of this and prompts speech synthesis and output of responses.

Thomas presented a simple mixed initiative example of asking for a long number where the user may say the whole number or say it in parts, and may correct themselves or correct the system's misrecognition. SpexKit addresses dialogues of this kind using augmented transition networks, with a distinct module managing the procedural parts of the dialogue. Classes SpkDialogObject, Dialog, Subdialog and DialogStep have actions and transitions, and targets (end, back, context and self) with definitions at lower levels overriding those at higher levels (so 'repeat' will prompt the machine to repeat the last utterance, not the whole dialog). He showed an example script written in SDML, the scripting language.

Their application aims to invoke commands deduced by the dialog as appropriate. They have C and Java interfaces so can drive ST, C and Java backends. Likewise it interfaces to speech input/output technology. (One use of their system is to provide test rig drivers, connecting simple test applications to speech devices for customer evaluation.) They interface to many market-leading speech devices. Thomas closed by offering to demo his speech-enabled washing machine to anyone requiring it. :-)

Q. SDML is your creation? Yes. It has as much expressive power as many larger languages.

Tools

Abstract Interpretation, Program Analysis without Tears, Andrew Black, Portland University

This idea first appeared in a famous paper of 1977. Abstract interpretation is about taking the information you need and ignoring the rest. An aerial photo and a map both have less information that the real thing and have disjoint information. In programming the real thing ('concrete artefact') is the program with standard semantics, and an abstract interpretation is, for example, one in which variables are replaced by their types. Andrew showed the idea by systematically rewriting the `Fraction>>reduced` method to be just a manipulation of types returning a type.

Andrew is interested in inferring method requirements. If a method sends self then it needs a concrete class. He briefly reprised the Virtual Categories work he presented in Bled (see my last year's report), when he said that to analyse self sends needed source code for all methods (showed picture of himself saying this). Then John Brant told him it was not true; you could use bytecodes (showed picture of a 'John Brant' that he got from the web, probable date sometime in the 1700's; alas, John is not here to let him update the picture :-)). Thus his time in the Camp Smalltalk room at the last ESUG in Bled was mostly spent updating his tools to use this idea.

Bytecodes are held in CompiledMethod, a subclass (in Squeak) of ByteArray, which has both instructions (bytes) and the literal table. (N.B.: big-endian and little-endian machines flip literals: to browse them, use CompiledMethod>>literalAt: and not something more primitive.) The InstructionStream holds the ContextPart. The Decompiler works by postfix

byte code -> prefix -> decompiled source. InstructionPrinter is where you write a single-method-studying decompiler, but subclass ContextPart for more complex things.

InstructionStream subclass: #SendsInfo is the core class of his abstract interpreter. It runs a collectSends loop.

```
end := self method endPC.
[pc <= end] whileTrue:
  [self interpretNextInstructionFor: self]
```

Just look at InstructionStream>>interpretNextInstructionFor: to see how to parse what you need. To detect self-send or class-send, simulate the stack by pushing the stuff you care about and just pushing a nonce #stuff for what is unimportant. It was easy to write; if you miss a method you just get a DNU and then write it. For straight-line code, this just runs tallying the self-sends, etc. Loops are where this kind of thing usually get hard but Andrew assumes that all temp sends are object sends and so in a loop a send never changes from self to object or vice versa.

Q(Roel) Optimised methods? We accept that we don't track those.

Forward jumps (e.g. in `ifTrue:ifFalse:` where one branch jumps forward to get to returnTop) means we can arrive with two stacks. We must then merge the stacks

```
interpretNextInstructionFor: ...
  self at mergePoint ifTrue: [self mergeStacks].
  .....
```

Blocks are the trickiest thing in Smalltalk.They are compiled inline and copied onto the stack when executed. The only reason Andrew needs to put anything more than #self, #class and #stuff on the stack is because he needs to stuff a block's argument number.

Initially it took 27 seconds to analyse the whole image, 600 microseconds per method. The time went in dictionaries (`atMergePoint` was usually false) and in OrderedCollection stack manipulation (which is slow), plus `interpretNextInstructionFor:` was doing linear search. He did this in the last two days at Bled, much helped by John Brant and Vassili Bykov.

It was faster and more accurate than his prior work, and the code

```
(owner ifNil: [self]) cellPositioning
```

was detected as a self send in this implementation but not in his older one. Abstract interpretation is fast since bytecodes are designed for fast interpretation.

Q. In Java, there is an issue of types having to be the same at every merge point? In Squeak, types are always the same (Object). Stacks must always be the same, height is only issue. (He had an assert and they always were, so this confirms the Squeak compiler works OK if anyone was concerned.)

Garbage Collection in Smalltalk, John McIntosh, Corporate Smalltalk Consulting
(John gave a more detailed discussion of this subject in his tutorial at Smalltalk Solutions 2004. Unfortunately I missed it but I assume his slides are on the website. Back in 2001, he covered some of the same material in a talk I wrote up in my 2001 ESUG report.)

John maintains the Squeak TK4 and Macintosh VMs.

Years ago, John saw an app that ran needing 8 Mb but actually kept growing to 18 Mb and then collecting garbage. He eventually worked out this was because it was configured for a 16Mb cache machine. John showed some hilarious internet postings by professional programmers in the last year showing how ignorant people can still be about GC.

Many GC decisions were made in the ‘80s and are still governing things today. VW has improved in this respect recently, driven by the fact that there are now some visible VW apps of small size.

GC began with reference counting. Each object kept a count of its references and died if it had none. There is still a class Tracer that can check that these counts are correct. Next came mark and sweep. From the roots of the world, find all reachable objects. Objects we don’t reach are freeable. Compaction to free these objects is slow. Copying reachable objects from FromSpace to ToSpace and then flipping FromSpace and ToSpace is faster (provided you have enough memory).

Q Concurrent algorithms? Hard to do. These days memory can be swept fast and the pause in your app while it does is less risky than a concurrent algorithm.

A generational garbage collector varies this by separating objects by age. Most objects die young in NewSpace while those that survive are ‘tenured’ by being copied to OldSpace. A RememberedTable records NewSpace objects referenced by OldSpace ones to save NewSpace algorithm having to search OldSpace.

John showed an example of an application where attempting to add many records to a table found a bug of grow-memory interacting with GC.

Squeak has a very simple mark-and-sweep, young-space/old-space system that was designed 15 years ago to let music play with no stutter (and so today runs in much less than a millisecond). John gathers from Claus that Smalltalk/X has a 3-instruction allocator; Squeak’s is 1000 instructions. John also reviewed the VA and VW GCs. VW is very structured: Eden, Semi-space A and Semi-space B, LargeSpace and FixedSpace, OldSpace and PermSpace plus the Stack and the CodeCache. John showed some time-series chart for when various GC events occurred in an application with given survivor space sizes.

In pre-VW5i.x, the performance tools excluded GC time, which could

produce some odd statements about time (e.g. a free-chain effect where loop allocation would sweep down a huge linked list looking for a usable freeable object before allocating). He showed a case where a 330 second test could be reduced to 200 seconds just by retuning the GC values. In another case, a 500Mb image-size application, John reduced GC events from 4312 to 2600 in a given cycle of work by tuning, eliminating GC lock-up cases that had been afflicting his client. By contrast, a German railway system he checked was well factored (see later talk).

Prevention or Cure: approaches to Configuration Management in Team Development, Niall Ross, eXtremeMetaProgrammers
I know two approaches to configuration management in team development in Smalltalk. The one I am used to is the standard (I assume?) approach, in which large-grained code components serve for both team development and build. Developers expect to find themselves working simultaneously within the same CM units. Thus clashing versions occur and merge tools let developers resolve clashes. This might be called the 'cure' approach.

An alternative approach I have recently encountered is one in which there is a developer-oriented CM structure orthogonal to the build-oriented one (c.f. class categories being orthogonal to Envy apps or Store packages). Developers create fine-grained task-oriented CM structures. They expect these to let them avoid clashes. This approach I call: 'prevention'.

Do I believe that prevention is better than cure? Hmmm .....

In the 'cure' approach, e.g.

- Store: packages, bundles
- Envy: applications, configuration maps

code stream divergence and merging is expected. Store computes fine-grained differences post-hoc, resolves all it can automatically and interacts with the developer to solve the rest. Envy provides change browsers that can load alternatives; the three-way difference browser add-on provides the extra data and UI on which a Store-like merge tool could be written. Developers control the whole of their current code components' state in a divergent stream but have limited control over their code components boundaries in any stream; that tends to be set by the overall build structure.

An example of the 'prevention' approach is the project-specific (patented) FDP development method. An FDP is a developer-built code component orthogonal to the underlying build CM structure (which is standard Envy). The developer adds classes (those classes they have edited, normally) to an FDP they create. They can select class-by-class and/or invoke a scrape of their current image to capture all unreleased classes. Further capture of method and shape changes to the chosen classes is automatic (Envy-like).

FDPs have several characteristics of standard code components. They are loadable and exportable between repositories, can be viewed in a tool that parallels Envy's application manager, etc. In the FDP lifecycle, editioning and versioning are Envy-like but there is no (developer-controlled) release.

Instead developers submit their FDPs which their manager and the owners of classes it contains then approve or reject. The developer can then apply (i.e. release) a wholly-approved FDP to the current development base (or retract it if they discover a problem while it is going through the process).

Thus applying to the development base is the usual way of connecting the developer-oriented FDP objects to the build-oriented Envy objects. There is also a tool for controlled mapping of FDPs to build components directly, without loading either, which is useful when back-filling bug fixes.

FDPs also hold a business justification, a technical description, notes if they should (unusually) require specific upgrade actions, and data on their test state. (A smoketest system loads an FDP onto the current development base, runs many batches against three test databases, one for each of the three lines of business, and emails the results to the developer. SUnit tests were introduced more recently. A developer can add tests to an FDP, or and some 1250 general SUnit tests that are run every day.)

Having encountered this alternative to the standard approach, I've found myself thinking about its pros and cons. On the plus side, the developer gets to work with task-oriented code components that they control. To add a new feature or fix a bug, they select just the affected classes. The development stream is a daily-updated current build structure in which merge clashes are rarer than with larger code components. Production streams can likewise be updated with bug fixes and fast-track releases with less risk of merge clashes. There are also benefits in having objects that are simultaneously reified task definitions and code components describing the work done in the task.

The minus side has some implementation-specific issues and some more fundamental ones. FDPs are not fine-grained enough: they should hold class definitions and methods, not classes; work on this is scheduled. Until recently, class deletion was more tedious than it needed to be, causing system bloat that is now being removing. FDPs are not as configurable as typical build components; they cannot hold FDPs, have specific load and unload actions, etc. There are no integrated coding tool equivalents of Envy's ApplicationsBrowser or Store's package view in the Refactoring Browser. Lastly, use of underlying CM constructs could be more elegant.

The above are all fixable now Smalltalk is no longer on the company 'sell' list. A more fundamental issue is that FDPs hide the underlying build CM structure issues undesirably. One consequence of this is that problems can accumulate in this hidden structure (visible only in the many warnings on the normally unwatched fast-scrolling Transcript of the daily automatic build). Periodically, these cause problems in the build; the problems are fixable but ... . A second consequence, reinforcing the first, is that users who do not come to the project with a Smalltalk background can remain ignorant of build CM issues and techniques.

Another issue is that merge resolution tools are neglected under the impact of the philosophy that seeks to avoid them. Clashes do happen from time

to time. More importantly, merge tools are also reversion tools when what has been released later reveals problems. Thanks to the test process, this is rare, but it is unpleasant when it occurs. Development build state is gradually lost. It is hard to revert to an old development build.

This interacts with the fact that the freedom to choose your own build components is a two-edged sword; others can do the same. Classes you were not planning to change, so did not put in your FDP, get changed by others and released into the daily base. They will have tested that they did not corrupt the base before releasing, but how they interact with your code is for you to experience. This can be a plus - you are made aware that you have an issue with someone else's code earlier than if you waited till you were ready to merge - but assessing what has changed around you can be more work than with large-grained code components.

A merge tool (e.g. Store's) that computes a list of class definition and method differences post-hoc, and a tool which captures a reified list of such items as work is done (as noted, the FDP tool does not quite do this today, but it could), are doing similar things at different points in the lifecycle. When teams on several sites frequently apply work to a development base, and occasionally to test and production bases, complexity must be handled somewhere. The FDP process, devised when merge tools were less capable than today, lets the user choose an optimistic merge strategy early, while they are doing the development. I would be interested to talk to anyone who has used Store-like merge tools on a multi-site system of similar size.

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

Q. Dynamic test recovery? There is a PhD student working on that.

Porting Projects

Porting from VSE to VW with Pollock, Christian Haider, smalltalked Visuals

There are still some important commercial VSE systems out there. It was always unclear what the migration path was, GUI being the main issue, as it usually is in ports. Pollock is conceptually close to VSE's GUI approach. Christian is here to tell you that porting VSE systems to VW with Pollock is very easy and well worth doing. He will also give the checklist of things to do, using his experience in two commercial port projects.

VSE systems need to port because Window is changing (will it work on Longhorn), because already cracks tend to develop (occasional strange behaviours), and to exploit new features and be able to use new platforms.

As VSE and VW are both from Cincom on the same contract, there is no

licence cost to migrate but there is a cost to migrate and to learn the new tools. A French group migrated to Ada (!!!!), and sometimes managers take the opportunity to kill Smalltalk and migrate to Java or C#; get in first!!!

How to migrate:

- Only migrate, don't improve at the same time. Projects can fail because all their nice-to-have's get crammed in.
- The people are migrating, not just the code; they must learn new tools. Therefore always have a consultant who knows the new tools. The consultant must guide people to use the tools (keeping their own hands off the keyboard). Christian recommend a configuration of a single keyboard, table and mouse, projecting to a beamer, when doing this.

Porting steps:

- create a baseline that will load in the new platform without serious warnings
- make it runnable
- tests are very important
- build a runtime; don't say 'finished' when only the development environment is done

The baseline is a defined state. You want to freeze it but projects have to go on developing so migrate as fast as possible - Friday to Monday. The migrated source should have the same module and version structure, and its comments should also be ported to package and bundle comments.

To get all the sources into VisualWorks, decide the module structure and choose a single system namespace for all the code. Team/V maps easily to Store. If you don't use Team/V, Christian suggests an overall bundle containing a porting (base issues) bundle, an application bundle, a tools bundle and a tests bundle.

Next, decide how to handle external interfaces. The format of DLL calls `<api: name args return>` is different in VW and must be converted; this is straightforward. Sockets you should just rewrite using the VW API. The same applies to database connections. COM is a real problem. VSE is very integrated with windows but VW has only SmallCOM/X, which is what you must use.

The actual physical porting of the code should be done via a tool to file out XML VW source format. You need a VSE Porting package containing VSE namespace, VSE.Class, VSE.DynamicLinkLibrary with API parser. The parser just reads the API code so it loads without errors. You need a tools package containing the Browser list icons.

File-in by opening a GHChangeList. Ensure Undeclared is empty before starting, set the default namespace and encoding. Load the sources. Replay all to do

- missing superclass -> VSE.Class

- ExtenalInterface -> VSE.DynamicLinkLibrary

Then resolve all Undeclareds. For globals and pool variables add a global in the VSE porting package.

Now you have your baseline and the real work can start. Empty the porting package by really resolving all Undeclareds. work through the 'message sent but not implemented' list. Make the code load without any warnings. Run code critic and resolve all serious bugs (and others as you think needed). Eliminate unnecessary system extensions and unused code. Note remaining problems. Try to avoid nil return checking; cure the nil disease.

The first tool you use to help you is the rewrite tool. Be familiar with it and use it to solve problems once and repeatably, not over and over again making new mistakes as you get tired.

Will you emulate or replace the GUI? Christian showed a table of how VSE, VW Wrapper and VW Pollock differed, showing how much more directly VSE's GUI maps to Pollock. Pollock has 4 interacting classes, Pane (central class) has Frame (always) and Artist (if you want special appearance) and Agent (if you want special behaviour). He also showed the widget table, black for all that have even the same names (most) and blue for differently-named things. He showing us how similar the code is to add a widget in VSE and in Pollock.

Porting from VSE to Pollock is easy; it can be prepared in a few weeks and then achieved in a small time-window, not delaying onward development of the system. It also improves system quality as a side effect, since every method is effectively being checked in various ways.

Smalltalk Modernisation Technology, Michael Vosgen, CEO Tricept InformationSysteme AG

From a fat client to a multi-channel server. Frameworks, the subject of the last talk, are fundamental to this talk too. A smalltalk system developed in the early '90s for banking needed to be upgraded.

Tricept is an IT-consulting company, 30 employees, working with Smalltalk since 1989. They build OO applications and frameworks for the banking and insurance sector. They have delivered one of the largest ObjectStudio Front-Office applications. They are very keen in Smalltalk but expect a shaking out of some Smalltalk IDEs in the market. They work with Synchrony systems.

Existing Smalltalk applications have a high value to their users. However there can be issues adapting them to new uses, for example if they must become multi-user or servers of other systems. Thus new projects are built in current Smalltalk dialects, or Java or .Net, rather than in the old dialects (VSE, digitalk; Michael suggested that ObjectStudio may be approaching that point). The result is a standstill in development, and management discussion of dropping Smalltalk entirely.

Two or three years ago new projects were initiated on technical arguments.

Today, solution of business problems is what is discussed, not technology, IDEs, etc. This is a chance for Smalltalk because it is productive. Their Java developers (more than half the firm's income in 2004 was outside Smalltalk, in Java etc.) confirm that complex functionality is easier to implement in Smalltalk.

A large Smalltalk application may contain three-to-six thousand classes, of which a third are user interface. Assume there are 4,500 classes and that a Smalltalk developer costs 1000 euros/day (he noted this was a generous rate but it makes the calculations simple).

Firstly, we ignore tools and look at new development (in Java say): assume a Java programmer needs 2.5 days to reimplement a Smalltalk class, so it will cost you 11 1/4 million euros to replace your application. To future-proof your Smalltalk application will take half the effort, 1.25 days per class, so a little over 5 million euros.

Now think of tool-based migration. If a tool promises to let you map your application code to Java taking 0.4 days per class (1.44 million euros), managers may thing converting to Java is cost-effective.

However modernisation is more than simple migration. It means acquiring a future-oriented base for mission-critical applications. This needs to be a development, not a revolution, to preserve the existing value. Small components need to be built and integrated into the existing system. It is safer, it shows better ROI for existing applications and it lets you learn how to modernise as you go. Above all, it preserves the existing business processes. You start with a vision of the target system: the new technical environment, the new databases, platforms, etc., that must be connected. What new modes of use are needed? Define what the results of each step are to be. Each step must be put into actual production as soon as possible.

He showed an example of a typical early-mid '90s Smalltalk legacy application: fat client, layer architecture, framework. He then showed a modern, multi-channel architecture, with various access paths (internet, thin client, ...) talking to a central server that coordinates use of services from various backends.

One step could be changing the fat client from ObjectStudio to VisualAge. This is simple in the slide (one block animatedly slides out and another in), less simple in reality. Next, the top layer of the architecture could become a communications gateway to a thin client, which then acquires an HTML client and other servers as its clients. You could stop there, or go on to start migrating your server architecture.

Smalltalk to Smalltalk: two dialects are never 100% equal. Code syntax is almost identical. The base classes map closely.

Smalltalk to Java: Michael was more sanguine than I would be, but was not suggesting it was easy.

SMT (Synchrony System's tool) maps code to an abstract model with much meta-data, and also maps the UI to an abstract model, using various tools: diagnostic, name transformation, GUI mapping and rewrite rules. The final transformation is logged in detail, to help traceability.

Michael suggested that development could continue while modernisation was in progress, by defining merge points for remerging the code. I feel dubious. He concluded that stepwise modernisation of systems in old dialects is cheaper and less risky than new development.

Q. Success statistics? Synchrony have had several successes mapping Smalltalk dialects to other Smalltalk dialects. They have some current projects mapping VisualAge to Java but no success statistics are known.

Q(Andreas) Smalltalk-to-Smalltalk preserves the comprehensibility of the design. Andreas' experience is that Smalltalk to other languages loses this, so the resulting code is unmaintainable. A Smalltalk domain hierarchy mapped to Java may look very different so, especially if generated, may be a much less clear mapping to the problem domain. What is your experience? There was some discussion. Michael thought it was doable but agreed clearly that it was of course better to map to another Smalltalk.

Q. Does EJB sell to management? Not any more. This is part of the change he spoke of that happened in the last two years

Q Why did your client choose to port to VAST? They already had a general IBM licence so moving to VAST was a no-cost option.

ESUG Innovation and Academic Track

ESUG Innovation Award Demos

As we filed in to the room, the screen showed a country-and-western style singer expounding the delights of the model-view-controller paradigm ("...model-view, I love you..."). I like Smalltalk too but there are limits :-)

This session presented the applicants, after which we dropped our ballots in the box voting for first prize (1100 euros), second (500) and third (200). (Noury also showed some Smalltalk work on the web for 3-4 year-olds.)

SmallWiki, Lucas Renglii, netstyle.ch and University of Berne.
Lucas did the presentation in SmallWiki with a slide-oriented plugin. SmallWiki, like other wikis, lets you author web pages quickly. However SmallWiki is also a content management system. It has security policies (these pages publicly-editable, these only by X and Y, these are visible to all, these only to Z, etc.). It is well componentised and an interface, in SmallWiki, lets you configure those components ('next' buttons, picture gallery, calendars, etc.) SmallWiki is fully object-oriented; every page has an object model created from parsing the page. Thus you can do powerful searches, etc. It is well tested; he showed opening the page that runs all the tests and shows results. SmallWiki is available under the MIT licence.

It is currently on VW, Squeak and #Smalltalk. It is used by the ESUG home

page, the Wiresong Monticello home page, the Squeak Germany home page and the Smalltalk composition group. It is also much used in the universities of Berne and Brussels.

They are now working on SmallWiki 2.0 in which the engine will have a meta-model, and views will be Seaside, Trippy, etc.

Q. Tables? Define a table in the usual Wiki way and use CSS stylesheet to determine how it looks. He thinks a single default table adequate (I don't).

Q. Info in repository is confusing; which bundle is top level, etc.? Noted.

StarBrowser 2, Roel Wuyts.
The model part is now on Classifications version 2 and he has managed to eliminate a class (20% code reduction !!). He has added tabs and showed switching between Trippy and class browsers. It now reuses the same Refactoring Browser window if available (faster). A vertical toolbar is not trivial to build in VisualWorks but Roel has done it; it manages the various looks available. You can move the toolbars and etc., to horizontal, vertical etc., to exploit your screen shape to best effect. StarBrowser wraps the tools it displays and you collapse it to see just those tools whenever you wish.

Roel has changed the drag-drop highlighting to make it easier to see where you are moving things (good; that was a need I had noticed). Roel showed its assignment of browsing tools, working on things in the Refactoring Browser, in Advance, etc. It also tracks what you are doing, so provides recent methods, popular classes, etc. It is in the Cincom Public Repository. He checks all tests in VW and in Squeak before publishing. StarBrowser lets you build any kind of classifications, grouping classes, objects, slides. You can use it for to-do lists, to track bug hunts, to make presentations, etc.

SqueakMap, a Web Catalogue for Squeak, presented by Noury on behalf of Goran Kempe
SqueakMap lets developers download and install selected packages. It was inspired by Gnome and Debian. The master server distributes map copies to the various servers whence they can be downloaded. A web site lets users create and manage their accounts, and view data on the various maps.

Today it has 467 packages, two more than it had on Sunday when he wrote the slide. SqueakMap work continues: they want to increase mirror robustness, do incremental update, enable local map modification (so you can do it without a web browser), and move to an architecture of one logical distributed map (instead of current single-master/multiple-mirrors).

BottomFeeder, presented by Alan Knight of Cincom
Alan explained that BottomFeeder is an application built by James Robertson, who is product manager for VW in his spare time :-). Those who know Jim know he likes to express his opinions and so wished to express these on the web. He was asked to provide an RSS feed and after an initial delay ("What is an RSS feed?") did so. He has created a very polished application that is used by people who do not realise it is in

Smalltalk (or would not if Jim did not advertise that fact so vigorously).

BottomFeeder lets you subscribe to e.g. GlorpNews (Alan updates it every month or so), BBC News (if you're interested in the less important matters discussed there), etc. Alan showed it working, looking at the various views: summary, HTML, XML, newspaper view. BottomFeeder is one of the Smalltalk applications that Alan would use even if it were not in Smalltalk.

The server that hosts Jim's and other Smalltalk blogs is also in Smalltalk and Jim can fix it while it is running, and talks about this in his blog. BottomFeeder can update while it is live (for better or worse; Jim has not become what some call 'test infected' and actually it is this that is his spare time job :-)). Load without restart is a checkbox, off by default but if you want to live dangerously you can.

Topologos, Patrick Chenais, University of Berne
Built by a team of 4 who work on it one day a week. Topologos is written in Squeak and models processes and objects together. Merging these two solves problems of inheritance and attributes.

In today's typical large company, no one knows any more exactly what the processes are and what data is available. Tools for modelling these tend to be for workflow modelling or data modelling but not both. Patrick sees a subclass as a group of objects entering a process. (What is a student? A subclass of Person defined by entering a learning process.)

Patrick then demoed, instantiating objects from a palette and relating them. He then loaded a more complex model and discussed it.

LCSTalk: a Classified Learning Platform for Multi-Agent Systems, Serge Stinckwitch, University of Berne
Michael Piel is the main author of this system.

Reinforcement learning is learning reinforced by reward. Latent learning occurs without this. (John Holland devised this learning classifier system.) Agents are given a reward (a scalar value) by the environment when an action succeeds. Agents are propagated by genetic algorithms.

LCS is a software platform for designing and experimenting with single and multi-agent learning systems. He demoed experiments using a maze, a prey-predator scenario and reconfigurable robots (see his earlier talk). The reconfigurable robot that was trying to learn to walk took a long time to master the skill, but I think that was a comment on the robot's intelligence that was being simulated. not on the simulation system. :-)

ESUG Academic Talks

Design, Implementation and Evaluation of the Resilient Smalltalk Embedded Platform, J.R. Andersen, L. Bak, S. Garup, K. V. Lund, T. Eskildsen, K. M. Hansen, and M. Torgersen
Mads Torgersen (Aarhus university) has been working with Lars et al. on VMs for small embedded systems. (See my reports of talks by Lars at

ESUG 2003 and Smalltalk Solutions 2004.) Embedded systems have lots of problems. We would rather use a high-level safe platform-independent dynamic language, i.e. Smalltalk.

In the past embedded meant resource-constrained, real-time no-GUI, i.e. not Smalltalk. Resilient is about how discovering how small can Smalltalk be by going back to the early days when Smalltalk was in charge (no OS). They have an external reflective-interface IDE, a small interpreter-based VM and real-time GC (about which Mads can say nothing because of IPR and because he knows nothing about it). They "eat their own dog-food", i.e. this is implemented in Smalltalk, not in C or whatever.

The programming environment is an Eclipse plug-in made to look like a Smalltalk IDE. The whole thing is designed to be a Smalltalk experience and can be connected to the VM at any stage, so you can repair the customer's problem while running over any IP connection. It is Smalltalk with a few tweaks.

- It is source-code based and includes a class syntax for ease of use with existing systems of likely users (and Eclipse likes it).
- Typed LIFO blocks: blocks cannot be assigned or returned. This is an essential limitation for performance: no heap-allocated environments, no stress on GC. However a block is not really an object in Resilient as it cannot be stored or returned.
- Atomic 'test and set' construct, i.e. test a value and set it if the result is a given value, e.g. `owner ? nil := Scheduler current`.
- Safe memory access, e.g.

```
io := Memory at: 16r90040000 size: 16r20
```

Its 128k fits comfortable between C / assembler and Java's megabyte.

Open issues: deployment (can change code on one device so need to configure same change on all devices) and performance critical code.

Q. What is being done at present? Several research and some commercial projects. One Mads finds exiting is measuring road conditions in your car: every 10 metres (36 times a second) measure road conditions, how icy it is, etc., via many complex calculations.

Q. Security? Would be handled at application-level as suitable protocol?

Q. Availability? Until a month ago, you could download most of this. Since the company was bought, all these links have been disabled.

Q. Platforms? StrongARM and quite a few more. (Mads did not have the full list in his mind but provided the data later.)

Parcels: A Fast and Feature-Rich Binary Deployment Technology, E. Miranda, D. Leibs, and R. Wuyts
Writing software is easy; just use Smalltalk. But then you have to deploy

it. Roel has found that when mapping a Smalltalk-oriented class-extension-using design to a language that lacks class extensions, he has had to make changes simply to make it deployable. This is an example of deployment driving design.

Parcels are a binary code deployment technology that support class extensions, can be loaded and unloaded, are very fast, and have meta-information.

- Loading and unloading: parcels remember what was not loaded (extension of class that was absent), whether a class changed shape and what methods were overwritten. Unloading restores prior state, restoring methods that were overwritten instead of deleting them. Thus you can have A and C in one package, B and D in another, with D subclassing C subclassing B subclassing A. (Niall: useful for e.g. complex platform-specific and generic hierarchies.)
- Fast: pickling is Eliot's name for storing the object graph so it can be loaded (unpickled) later on, using a recursive descent parser. In the simple obvious design, writing is easy but reading (have I got superclass for this yet?) is slower. Parcels separate and order the node descriptions from the arc descriptions. Thus reading back in is very fast, writing slightly slower (Niall: the right choice; fast reading is usually more important than fast writing for usability and commercial value). Roel showed a format indicating the structure of a pickled file

Roel showed some benchmarks of time to read and to write Store code versus the same code in parcels, and also (almost, modulo trivial porting aspects) the same code in Dolphin and Monticello (Squeak). Reading parcels was always far faster. Writing was the same for most applications except, for some reason, Swazoo, where parcels were noticeably slower. He also benchmarked against local Envy in VW3, using some fairly similar (not identical) code. Again, parcels were faster.

Q. BOSS, parcels: which system learned from which? BOSS was there first but Roel is unsure how much influence it had. He tried to benchmark parcels against BOSS but got no meaningful results.

Q. Ordering issues on unload? Yes, parcels remember the order in which they were loaded.

A Dynamic Graph Implementation in Smalltalk for Self-Reconfigurable Robots Simulation, S. Saidani and M. Piel, Laboratoire GREYC, Universite de Caen

(Michael Piel presented as Samir had exams to run.) They want to model, so control, a self-reconfiguring process. Their simulated polymorphic robots make themselves from simple cheap parts. The MAAM project is named from pseudo-BNF (Molecule = Atom | Atom + Molecule) that defines this. He started with a simple von Neuman model of cells in an array switching between two states depending on the neighbourhood, then extended this to one in which the next state depends on the neighbours and the current state. From such a model one may discover an emergent calculus. An ant colony simulation is an example of such an emergent

calculus.

Samir has built the DynaGraph system in Squeak to support exploring various algorithms and their emergent calculi. It has good UI, debugger, etc. A simple example was a star-to-chain reconfiguration. See http://www.laas.fr/robea/maam.html for more information.

An Aspect-Based Multi-Agent System, R. Robbes, N. Bouraqadi, and S. Stinckwich
Multi-agent systems have to solve problems of distribution, parallelism, and complex communication, which leads to complex applications. Their MAS model is called Aalaadin. Agents join in Groups wearing Roles: abstract representations of functionality. He presented a marketing example with a client group containing client and broker roles, etc.

Aspect-oriented programming separates cross-cutting concerns; in this case, concerns of persistence, synchronisation, distribution and debugging. An aspect defines one such concern and a join point is a message send or variable access subject to a concern. The basic logic can be implemented much more simply without the complexity of handling the aspect concerns. Weaving then applies the aspects at the join points to create the system.

MAS development is hard as there are many concerns to consider simultaneously and this hinders reusability. AOP is intended precisely to solve such problems. They have decided to unify Groups and Aspects and use a single concept (retaining the name ‘group,’ not ‘aspect’) as they are similar: aspects are transverse to objects, groups are transverse to agents. They extended Aalaadin’s groups to have real modularity (Aalaadin’s were only conceptual, not implemented) and join points, thus making them more like aspects. Groups are now sets of roles worn by agents, reified, modular and extractable. They have also added meta-roles to handle join points, entering and leaving groups, etc. They find these changes make multi-agent systems easier to design and visualise.

Q. Meta-roles? Meta-roles are needed to change the behaviour of roles.

Uniform and Safe Metaclass Composition, Stephane Ducasse, Nathaniel Schärli and Roel Wuyts
Classes are objects, instances of MetaClasses, thus need instance creation behaviour like other objects.

- Implicit metaclass: programmer does not specify class; safe but limited reuse. Smalltalk 80 wisely introduced this subtle concept in safe mode.
- Explicit creation: metaclass composition is usually unsafe. You can use ad-hoc manipulation of strings.

Stephane presented a safe, upward-and-downward-compatible mechanism. Let A be an instance of MetaA, and let B be an instance of MetaB and a subclass of A.

```
(MetaA) “c-bar implemented in A”
  A class>>i-foo
  ^self new c-bar
```

Smalltalk 80 chains the MetaClass inheritance link to the class inheritance link. CLOS lets you sink or swim. NeoClassTalk allows dynamic class changes. MetaClassTalk provides mixin composition of MetaClasses. MetaClassTalk provides Traits (see earlier talks). In MetaClassTalk, a Class = Superclass + state + traits + glue methods (class methods take precedence over trait methods). Behavior has superclass PureBehavior whose other subclass is TraitBehavior. Subclasses TraitDescription and Trait parallel ClassDescription and Class. Traits can be composed safely.

Stephane described some simple examples: TAbstract trait on Boolean class, TSingleton trait on True and False classes.

Traits is available in Squeak 3.7.

Q. Performance penalty? Traits have no state so you must use accessor methods, not in-line access, in any trait method. If you have the habit of using accessors, you will not notice any effect.

Q. The same traits can be used at instance, class and meta-class level? Nothing prevents this if it makes sense. For example, a trait with `name:`, `name`, `printWithName:`, `StoreWithName:` might be used on all levels.

Q. Traits are multiple-inheritance? We all want multiple-inheritance but there is no good language for it. Traits allow reuse of the kind that makes us want multiple inheritance, giving us its advantages without its problems. Example: refactoring of Stream hierarchy presented at OOPSLA last year.

Language Support for Adaptive Object-Models using Metaclasses, R. Razavi, N. Bouraqadi, J. W. Yoder, J. F. Perrot, and R. Johnson
Complex systems from industry cannot be simply withdrawn and redeployed once in place. These systems, once deployed, are used at two levels, expert and operator. Run-time intervention by experts changes the behaviour experienced by operators.

Such systems are best documented as a compositions of design patterns. Designing objects whose behaviour may be changed at run-time is a hard problem. The solution is to split your objects into a fixed part and a meta-part, the latter being changeable. In Smalltalk terms, the normal object is described by an ordinary class, and the changeable part is meta-data.

Videostore example: each cassette contains a movie. When movie data is altered (Terminator3: assign X rating, Fahrenheit 9/11: assign (dis)honesty rating :-)), you want to modify rental conditions for all cassettes containing that movie, so you need two objects Cassette and Movie. But experience teaches us that treating meta-objects as terminal objects creates unneeded complexity; the meta-object should not be an ordinary object.

The approach is to use the class itself as the meta-object. In the example, have class Video, subclasses Terminator, Fahrenheit. Metaclass Movie creates and initialises these metaclasses. Limitations: two alterable instances of two different classes cannot share the same meta-object, e.g.

cassette and DVD won't have the same Movie meta-object. So we need to subclass Movie to MovieTape and MovieDVD.

This is an example of the general issue of the 'item' relationship: a cassette is an item of movie, a copy is an item of book, the flight you got to this conference is an item of flight number 512. MetaClasses can adequately capture this relationship, which inheritance cannot. Razavi's thesis looked at three approaches: multi-agent simulation, Smalltalk with metaclasses and MetaclassTalk with metaclasses, and demonstrated that metaclasses solve this issue.

See http://st-www.cs.uiuc.edu/users/johnson/papers/udp/UDP.html.

Classboxes: Controlling Visibility of Class Extensions, Stephane Ducasse, Nathaniel Schärli and Roel Wuyts
They wish to add grammar to control the operation of changing the Squeak system. In Smalltalk-80, class extensions are global; any application can modify any class and everyone can see the modification. Thus two apps can conflict in modifying a class, an application may redefine a critical method and implicit dependencies may be created. (In Squeak, if Morph>>bounds is redefined to return self, you get an unrecoverable crash.) Also, how do we combine class extension with modules and with security?

A classbox is a unit of scoping (i.e. a namespace). In a classbox, classes can be defined and imported from other classboxes, and methods can be defined *on visible classes*. Within a classbox, you have a flattened view of the world, local changes appear as if global, but extending classes will not impact their clients outside the box. Alexandre showed how a

```
passwordCheck: ...
  ^true
```

hack extension could not impact a bank application subclassing the extended class in a different classbox.

Last year, their implementation had to modify the VM. They now have an implementation that does not require VM mods. Instead they use a cache mechanism that adds 5 extra byte codes for redefined methods (no overhead for invoking added methods, worst case for redefined methods is 2.5 times slower). If he redefines method #foo on Class A, a dispatcher inserted into its method dictionary controls.

See http://www.iam.unibe.ch/~scg for more details.

Q. Classboxes contain objects? Classes are in classboxes, not instances.

Q. (Georg Heeg) Have you looked at work by Andreas in '89 on security-based process models using a Smalltalk-80 VM (he has a version)? They would be interested.

Q. (Andrew Black and others) Method lookup OK; what about state? Enhanced class differs from superclass and/or has state values that are then

accessed in another thread that has visibility of extensions? Discussion.

ProtoTalk: an Environment for Teaching, Understanding and Prototyping Object-Oriented Languages, Alexandre Bergel, Stephane Ducasse, Christophe Dany
Alexandre and Stephane ported ProtoTalk from ObjectWorks and use it in University of Paris (VI), Berne, and Montpellier. The goal of ProtoTalk is to make teaching the principles of OO easier. For example, prototype OO languages use the classless paradigm; new objects are created by cloning. Composition is done by delegation and by reference; if I don't understand this message I call someone else. Examples: Self, ObjectLisp, NewtonScript. What advantages and problems do such choices give?

ProtoTalk aims to have the same syntax for all its OO languages, a very small kernel and good extensibility. A language is implemented as a subclass of AbstractProto. Send `evaluate:` to a program to this subclass to execute it. This builds an abstract syntax tree (subclasses of Smalltalk AST nodes) which is then evaluated.

Example: NewtonScript has a double-inheritance mechanism: _proto is searched before _parent.

Conceptual Code Mining - Mining for Source-Code Regularities with Formal Concept Analysis, K. Mens and T. Tourwé
They wanted a lightweight source-code mining tool, to detect recurrent patterns in the source code. Their approach, Formal Concept Analysis, is a mathematical technique. Kim explained FCA by example, showing a taxonomy of programming languages against the concepts of OO, functional, logic, static and dynamic typing. Having boolean-rated Java, C++, Smalltalk, etc. against these properties, he then studied the maximal groups sharing these properties, thus creating a graph connecting the groups from the top concept (all elements have these) to the bottom concept (elements with all concepts).

They studied Smalltalk looking for methods with class names in selectors or parameters (e.g. class Foo, method #asFoo: anObject, #import: aFoo). Having generated the concept analysis and computed the graph, they must filter greatly to remove meaningless noise. The output is captured in the StarBrowser.

- Generate formal context: they seek elements that share substrings
- The concept analysis: Kim listed methods to show how the concept of unification in SOUL would be caught by this approach
- Filtering is simple: they filter out 'do', 'with', 'by' and some more weeding, but there are still too many concepts remaining (currently filtering tends to produce a factor 5 reduction)

The tool takes seconds to analyse SOUL, minutes to analyse the Refactoring Browser. The patterns they find are code duplication, design patterns (Visitor, Abstract Factory, Builder, Observer). They also found coding patterns, accessing, polymorphism, etc. They were surprised how

many of the concepts found were actually relevant.

In future they want to use FCA to study aspects and cross-cutting. and to detect refactoring opportunities.

Q. Any study of good code from experience programmers versus newbie code? He would expect more noise in the latter case. They will study this.

Power Laws in Smalltalk, M. Marchesi, S. Pinna, N. Serra, and S. Tuveri

The web, the US power grid, biological systems and social networks have all been represented as nodes, as have software systems. A random graph is created by growing connections between each pair of nodes with probability p. Random graphs show a poisson distribution whereas real networks show a power-law distribution; as he showed in a graph, these are very different. Studies have shown that static OO systems (nodes = classes, connections = relationships between them) are governed by scale-free power-law distributions.

Smalltalk is a dynamic language so if we encounter multiple implementors, static analysis lets us do no more than assign a 1/N weight to relationships to each of the N implementors. They analysed four Smalltalk systems to see if they satisfied the scale-free power law distribution. They believe that they do. (Much discussion with audience to explain what log-log graphs actually meant. This used up the time for questions.)

In future they hope to correlate graph properties with software quality and model the evolution and maintainability of the software from a growth theory of software graphs.

I felt that dynamic recovery of types would give a better analysis than 1/N.

ESUG Activities

ESUG Business Meeting

A change in ESUG statutes was proposed and accepted. ESUG used to have a board and administrators. The proposal was to reduce the number needed by eliminating the administrators and just having the board, with a minimum size of President, Vice-president and treasurer. ESUG is current administered under French law but will consider whether next year it should be administered as a ‘European association’ (new legal structure).

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

ESUG Conference Administration

Stephane thanked Georg for the excellent local organisation and the student volunteers (‘this year, they had to work’ - I think Michael van der Gulik would say he had to work last year in Bled too :-)). For example, student volunteers made the coffee this year, so ESUG had only to pay the bare costs.

ESUG welcomes ideas for promoting Smalltalk. ESUG was more

profitable this year (last year they just broke even, helped by attendees donations). ESUG also welcomes help, help organising and doing things. The minimum help you can give is to attend the conference so if you do nothing else, do that. Next year, Squeak will be in Belgium.

Feedback on how the conference runs is also welcome. Camp Smalltalk will be before the conference again as usual next year (site issues meant it had to be after this year). This year they (almost) had no parallel tracks. In the past parallel tracks had been criticised but perhaps longer talks in two parallel tracks would be worth doing.

Georg remarked that organising it was a lot of fun to do. The visuals are available for reuse next year (and on the web-site as PDF.)

Other Discussions

There was a parallel track teaching day with interesting stuff on children using Squeak (www.languagegame.org:8080/ggame/15), use of Squeak in Spanish Schools (unfortunately Diago Gomez Deck, who was to say more on this, could not make it) and a talk by Stephane, Adriaan and Lucas (in German, so I barely followed it from its screen presentations and did not attempt to record it) on using Seaside.

Claus Gittinger has won several valuable (millions of euros) contracts recently to replace ‘90’s Java systems with Smalltalk/X systems. In all cases, the Java systems had proved unmaintainable and unrefactorable. Interestingly this was not immediately because they were hard to refactor in small increments (my bugbear about statically-typed systems) but because the code for these sizable (largest was ~5000 classes) systems was simply not sufficiently comprehensible. The systems could not be understood enough to be safely changed.

Claus described one of these contracts, being done for a telecoms company. They have concentrators that used to provide print-out of all reconfiguration (map this number to that, authorise this service, etc.). In the 90’s, a Java application was written to parse all this print-out for a wide and evolving range of equipment and formats, and send it to a central location.

John McIntosh described his GC tuning work in Germany. His client had an acceptance group, separate from users, who ran many tests a thousand times all weekend on every delivery. If the time distributions were slower than the last delivery, they would not accept it. Users found that entering text to say why a change was made after saving it was very slow. Each typed character was interrupting the GC which was trying to run in the idle loop after the save, thus running clean-up before displaying the character; John tweaked the idle loop GC behaviour to be less eager and more in phase with requirements. That affected some other operations’ times, trivially but such that the acceptance test team noticed, so he then had to reimplement it so that everything showed not even the slightest impact.

Bryce told my about Continuous, a big, expensive and horrible CM system sold by giving managers good lunches and nice bags. It has a development

system similar to FDP, but file-based with no 3-way merges or anything. Bryce suggested it was a system for letting managers have many developers doing work but without the unpleasantness of code ever actually being approved and released :-). He worked at a company which had it; he was responsible for removing it.

He has just started working in London for RMB, an investment bank who use VisualWorks and GemStone, and are also porting some C# utilities they have to Smalltalk.

Conclusions

My sixth ESUG

- There are signs that Smalltalk's commercial acceptability is rising:
  - — interesting range of projects and clients
  - — serious industrial use of Squeak !!
- Fascinating snippets of history in Dan's talk and elsewhere: how Smalltalk came to exist and by what routes people acquired it.

I'm looking forward to Frankfurt in December and Orlando next year.

---

* End of Document *
