Smalltalk Solutions 2005, Orlando, 27-29 June 2005

Jim, John, Michael, Blaine, Georgio and I went to the Magic Kingdom on Sunday. Alone, I would probably have spent the day trying to understand how to get on a single ride. Instead, I was astounded by Jim and John's incredibly detailed knowledge of layout and procedure. Led by the two of them, we moved rapidly from ride to ride, avoiding long queues. The rides themselves were enlivened by John and Jim's ability to provide a complete version history of how the layout had changed over the past decade or how a particular prop had functioned three years ago when it had been part of the 'alien encounter' ride. When I learned that Jim has an annual Disney pass and has been every year for decades, while John McIntosh has a time-share pass and has also been often, I ceased to be astounded (that they knew so much about Disney :-).

Meanwhile, Blaine nobly refrained from taking the opportunity to nobble a competitor when Michael occupied the car directly in front of his laser cannon in the Buzz Lightyear ride. :-)

In mid-afternoon, Giorgio and I had to return to the hotel. The others should have come with us; after we left, the rides they queued for were closed, whereas Georgio and I took a wrong turn driving back, leading to an exciting ride round Orlando.

Style

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

Author's Disclaimer and Acknowledgements

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

Summary of Presentations

I have sorted the talks I attended into various categories:

- Web and GUI Frameworks
- Applications and Experience Reports
- Processes, Tools and Configuration Management
- Vendors, VMs and IDEs
- Testing

- Remote Programming
- Miscellaneous

after which I list any Talks I Missed, describe Other Discussions, note Follow-up Actions and give my overall Conclusions from the conference. I also describe the pre-conference Coding Competition and the play-off.

As there were often two and sometimes three parallel programme tracks, plus ad-hoc discussions, I could not attend half of what happened, but managed to report on slightly more than that due to speakers' kindness in repeating talks for me at times. The talks' slides should be reachable from the conference website (http://www.smalltalksolutions.com/). For info on talks I missed, and others' take on some I caught, see the Smalltalk blogs.

Opening, Allen Davis of KSC and STIC and Alan Knight of Cincom
Allen described the various facilities and sponsored events. (This included, at the meal on Tuesday night, a caricaturist who drew cartoons of anyone rash enough to ask for one, which yours truly duly did. :-)

Exhibitors
The Precision systems stand was well-stocked with goodies and they were raffling an iPOD, as were the Cincom stand. With the coding competition, that made three chances to win an iPOD (but I still don't have one :-/).

Web and GUI Frameworks

Transcend the UI Divide WithStyle, Michael Lucas-Smith
Michael lives in Canberra and works for WithStyle (his own company) and for Wizard Information Services. He has been a Smalltalker for 6 years, his WithStyle colleagues for 10 years and for 3 years respectively.

MacroMedia just fakes a UI inside the browser and similarly for many others. WithStyle gives the full power of a real user interface on the web. They also have an XML editor for the end-user, not for XML experts. They began with Nyx (can it be done?), Kyx (first production), WithStyle V3 (in use plus developer programme) and WithStyle V4 (today's demos). Last StS he demoed V3 showing scripting support (he made a presentation engine in the demo) and rich CSS2 support. Now they have rich support for CSS2 and 3, they have several sites in use, and they have fewer methods.

As well as CSS3, V4 has a better box model, StyledXML, far better speed (as fast as Mozilla from a Smalltalk image and they get more behaviour for the same speed). It is much more flexible than V3 because it is based on Pollock, so any UI widget will be a web widget. It has sensible drag-drop: drag this XML to anywhere it can go, only to here, etc. They have before and after (CSS3), zooming (lets them preview in tiles, etc.) and paging (web-style, MS style, continuous stuff).

Pollock is the future, is much easier to use (even in its current changing form), gives better widgets (HTML forms, any Pollock widget, e.g. tree view). In the future, it will help them pick up XForms or whatever comes along. You can edit widgets to Smalltalk objects, or widgets to the DOM

tree, and the DOM tree to Smalltalk (your DOM tree object has an attribute that lets you connect any aspect, thus giving you a natural mapping and backmapping between the two).

He showed the CSS working in zengarden, etc. He likens WithStyle to Gecko; something you build a web browser such as Firefox on, not a web browser itself. He then went to WithStyle to show authentication and to take people round the developer programme offers. They show the test results against each version so you decide which one you want to take.

The first demo was last StS's example slide viewer. (His slides were running in it; he had been demoing from the start of the talk.) He showed the pause button, completely defined in HTML, hidden at top left until you hover over it. It minimises the slide and shows another button to pop it up again. It is very easy to make simple windows (about window, help window, etc.) that get their content from Smalltalk.

The next demo was V4 using Pollock: XML with embedded HTML with embedded Pollock widget (tree view). He browsed the Smalltalk code that assigned a WithStyle document to a window, sized it and opened it. He talked through the text (CSS flow means things move from left to right (or otherwise as needed). He displayed it and showed changing the widget, having the DOM tree update in an inspector, etc. (Sames: you can see the Pollock widget by the lines around it; this is not faked! Michael showed a WithStyle widget in a WithStyle widget - no lines to see). The tree view expand/contract menu appeared (and he'll add some content soon :-).

Backmapping is what WithStyle has that noone else does. It makes XSLT-transformed XML know where it came from in the Smalltalk. He walked through the code, showing how they could make text editable by the user in the middle of some HTML (Mozilla had this concept but does not implement it). He showed the window and an inspector. He typed and showed that the inspected Smalltalk object saw the change.

Q(Eliot) Validation? XSLT does not do validation is such. It only lets you do the things its schema says you may, so there is no validation need in the UI. All real validation will be done in Smalltalk.

The next demo was the VW Welcome Window. Mark Roberts wanted a welcome window that says, "Look how modern this environment is!". Michael showed embedded workspaces (doIt), embedded transcripts, styled XML being constructed and displayed (better than transcript). Lastly, he showed running code to popup a window. (As soon as the code browser is rewritten in Pollock, you'll be able to embed it in any WithStyle window, or pop it up.)

EzyXML is Wizard's product. They use it to edit their client's websites (8 Australian government websites). He brought up the road transport site (brief flicker in demo due to a bug Michael introduced 2 days ago). These have been in production for a long time and not one production bug!!! They just keep expanding their use of it. Michael used context-sensitive menus

and popups to add another resource for users to download in the download location. He then showed domain-specific menus: contacts info, service info (what procedure to follow to get this service from the department), etc.

The next demo was XML WithStyle. It can handle any XML and speeds you up by offering a lot of common vocabularies (HTML transitional and strict, docbook, scrum backlog, etc.). He opened some XHTML fragment in a raw plain format, then switched to the WithStyle website format and suddenly it looked cool. He used the menus to edit it and lay it out. You have a complete unlimited undo/redo stack for any XML in WithStyle. If you call Smalltalk code that inverts your entire DOM tree, you can undo it. He opened an (amusing) XML-for-CVs page and showed the hierarchic menu outlining the larger and larger affected areas a menu item applies to as he edited it. They have demos showing changing language on pages.

He then showed displaying the SCRUM backlog XML and editing it. The XML is very simple, the XSLT transforms it and the stylesheets make it look good.

At this point, time forced him to drop the SmalltalkDoc and BottomFeeder demos. He ended with the Seaside integration demo in V4. He popped up a window with a multi-line text window that was Pollock, not HTML, with the Seaside halos and so on.

Onward to Pollock, Vassili Bykov, Cincom Systems
Vassili's talk was about his experiments with Pollock and his initial work on building Splash, the Pollock GUI builder.

37signals.com once challenged people to explain what API meant in 10 words or less. As well as the jokes ("Data pimp", "1/2 a piece of computer velcro."), Vassili noted the following answers: "Another programmer's idea to remind you that you are not a programmer." and "APIs are to programs as GUIs are to humans." Last year in Seattle, Vassili noticed the hotel alarm clock (Sony dream machine): buttons for snooze, nap, sleep, etc. But the alarm mode setter took the biscuit for counter-intuitive UI, blatantly and unguessably showing the binary setting mode it populates.

Last year, Vassili decided to spend an hour pair-programming a simple font selector. Two days later it only sort-of worked. It was harder than he expected partly because of odd code in `pixelSize` that fails the least-surprise principle. We have mental models and so we misattribute blame; unusable means not corresponding to our mental model.

"What is central to Smalltalk? Modelling rather than programming" (Georg Heeg). This principle distinguishes good Smalltalk programmers from bad. Framework design is like piling blocks on each other. Soon the pile is ready to fall over. We can make the blocks very big (simplistic solution) or make them more even and better arranged (simple solution).

"The widgets in Pollock will not be stupid" (Sames). "The widgets in Pollock will be intelligently stupid" (Vassili). To explore Pollock, Vassili

does things that he knows are hard in Wrapper, sometimes so hard he failed to do them in Wrapper.

PollockPlayground is a window with menu and buttons round a workspace plus a few widgets. 'Inspect' button shows a PollockPlayground instance: eventHandlers, widgetInventory, all the usual Pollock stuff. These are all in scope in the workspace so you can doIt expressions using them, thus putting stuff in the widgets, making widgets do things or whatever. Using this, Vassili did a tutorial on fractional frames. Width of frame divided by fraction and then offset from boundary sounds easy in theory but users find it hard to understand in practice. You need only grasp three cases:

- fixed left/top - all fractions zero: widget fixed in position top left of parent (e.g. window)
- fixed right/bottom - all fractions one (usually with negative offset): widget in bottom right of window
- stretchy - top/bottom one fraction, left/right other fraction: centred, stretchable
- long floating point fractions, all offsets zero: easy to do in Wrapper, never looks good, don't do it.

You set these directly via `leftFraction:leftOffset:...` or by convenience methods, `above:`, `rightOf:top:bottom:` (good for labels), `fullyAttached`, `leftOff:`, ...

Vassili showed these in PollockPlayground inside `createInterface`, `createTextEdit`. He has its widgets set to leftClick -> inspect, rightClick -> browseClass (not good in general but useful for demos). He uses `hookupInterface` to get the Playground text from a shared variable (so he can close it, open a new one and continue the demo).

Vassili then talked about tooltips (see his Frankfurt talk, but he has changed the implementation). Tooltips need a state machine to track time since last user action and respond sensibly. A tool with a tip starts in state 'cool'. If entered, it becomes 'armed'. If the user promptly does something, it returns to 'cool'. If not after 0.5 seconds, it goes to state 'reaper': wait five seconds and destroy tooltip if user remains in widget, go to 'warm' state if user moves to another tool. In the 'warm' state, the tooltip appears immediately, not just after 0.5 seconds. After 0.3 seconds, it returns to 'cool'. The state machine is in FlyByHelpTracker and is called to by `enterEvent:` and by exit events, thence `coldEntryInto: aController`, etc.

Vassili had to protect via `self cautiously:` to allow for the debugger being able to open if the code is wrong (as debugger has widgets with entry events too). Vassili installed the tooltip system on a Playground widget and showed it working. TooltipAssistant subclasses Assistant which provides plugins to attach to widgets (`installOn:`, `uninstallFrom:`); this lets the widgets themselves know nothing of it. `buildStateMachine` builds the machine inside `setPane:`, making it easy to reuse code while building states as all are built in same scope. There is one state machine per system.

For toolbars, you want one tooltip for the whole bar but it must track button entry/exit events, so the structure (in ToolbarTooltipAssistant) is slightly different. Vassili set a tip on the PollockPlayground toolbar and demoed it.

His next demo was the EntryAssistant. This behaviour is also in Wrapper (as CompletionDriver; it would be called CompletionAssistant if he wrote it today). The aim is to insert the possible completion text, selected so user can type over it. In Wrapper, `privateShowCompletion` is spawned in a block. Vassili could not do it more nicely in Wrapper (it took him a day to see that he could not) since it looks like there are lots of events you can hook into but they are all too early: the controller is still sending events and wipes over whatever you insert. That's why he has to spawn that call.

In Pollock, it is nicer. He demoed in the Playground and then showed the implementation. EntryAssistant has four methods. In `suggest`, it just tells the agent to `insertAndSelect:`. He then demoed an entry assistant with popup (EntryAssistantWithPopup>>suggest). This has a subtlety: we want arrow keys to be received by the popup but if the user continues typing we want the entry field to get the keystrokes. The popup filters them and sends to the main widget if appropriate. This would be very hard in Wrapper.

He then put a class list into a playground widget where some class names were too long for the widget. Vassili set it to display longer names when the user mouse-overed (ItemOverlayAssistant). He then changed that assistant for a tooltip assistant that showed name and package in a popup. None of this is being done by the widget. The widget is stupid. These assistants handle all. This is what has changed since Frankfurt.

The AutoIndentAssistant just indents text as the previous line was indented. A real CodingAssistant needs more complex behaviour, auto-completing from the names of all instvars in scope or whatever. This in turn needs an intelligent partial parser.

Marker is a utility that will be in Pollock letting you place markers at points in text. A marker is like a bookmark that you can query (what is next character, prior character, charactersToEndOfLine, ...). You can move the marker, reason about two markers, etc. This lets Vassili avoid writing code with index handling, so produces more meaningful code.

Q. `nextCharacter` should be `next`? Perhaps (it is an experiment at the moment). Vassili was thinking of it in a list containing `nextWord`, etc.

Splash is Pollock's GUI builder. Whenever Sames doesn't know how to do something he says, "Splash will do that." They have created an extra component called Scallops (anyone who fixes a bug in Pollock will be told the secret reason for this name as a reward). Scallops will handle the UI specifications, their XML representation, etc. It already has 3000 tests. There is the UI, the specifications and the fluff (all the ways you could store the specifications). You can distil windows into specifications and build windows from specs. You can write specs as fluff and read fluff into specs. Lastly you can create windows directly from fluff when the fluff is

generated code, meaning your application need not load Scallops.

Vassili sent `asSpecification` to the Playground instance. He then mapped this via SpecificationCodeWriter to various fluff (to text browsed in a workspace, to XML). He showed some specification test cases that wrote, restored and compared a spec for a wide range of specs (i.e. buttons specs, menu specs, ...) via `generateRestoreAndAssertSameness`. He has extended the specs so that they can compare themselves.

Q. (Heeg) Can we do auto-adjust in Pollock? (Sames) `preferredExtent` should give this.

Q. Who will document this? (Sames) We know this must be documented. We also have to train our support people to support it. Etc. That's why my roadmap includes 7.5, not just 7.4.

Pollock, Sames Shuster, Cincom Systems
I missed most of this talk. Sames' slides have much code (plus see my reports of his talk at last year's Smalltalk Solutions and Vassili's talk above). Pollock has lots of nice features. Sames' slides show Roadmaps for 7.4 and 7.5 and for post-production: refactoring scripts from Wrapper, frameworks, native widgets.

Tweak, John McIntosh
(I missed almost all of this talk as I was catching David Schaeffer's repeat of his Seaside testing talk.) Tweak is Squeak's post-Morphic UI. John's electronic book project is using Tweak, not morphic. They have funding so they can influence Tweak. They provide testing framework extensions to let you test the widgets in SUnit.

John mentioned an example of German versus English idiom; a German Tweak developer used a 'beWare' annotation (meaning 'be aware of', so intended to tell a programmer that a state change will occur as part of invoking certain methods), which his English-speaking Tweak colleagues read as 'beware' meaning 'don't use'.

Applications and Experience Reports

Commercial Machine Control using Smalltalk: An experience report from the Semiconductor Industry, Thor Raabe, Unaxis Wafer Processing

Unaxis is a Swiss semiconductor manufacturer (also data storage and other products), headquartered in Zurich. They were founded in 1906 and now have 6000+ employees. Thor has worked for them in Florida for 12 years. They etch circuits onto thin films (10^{-6}m thickness) adding or removing material. It takes hundreds of steps to build up these wafers. These steps are done on radial tool clusters in an incredibly clean vacuum environment (deep space pressures). People in 'bunny suits' (sometimes two layers of them) add the wafers to the cluster. A wafer may then go round the circle many times before all steps are done. A cluster can process multiple wafers at once. He showed diagrams of the shapes (much smaller than a human hair) they etch to make connections, etc.

Q. (Donald McQueen) how do you create layers? Each cycle creates one layer. The wafer is doped; you add electrons to remove material. You shape one layer, then you coat the wafer with an insulating layer, then you lay a polymer-based material that can be exposed (like film). Then you etch away material that was not exposed, then that which was exposed, etc. Thus you gradually build up the structure.

Q. (Eliot) How do you register position? That is a science in itself. There are marks on each wafer that are optically aligned.

So where does Smalltalk come in? They needed a control system that could run these tools 24 x 7 x 365 (it takes £billions to start a semiconductor factory; they do *not* stand idle). They use the ControlWORKS framework in VW 2.5 and completed their first port to VW 7.1 two weeks ago. They shipped their first system in 1996 and have shipped 400+ systems. The software team has 10 developers, 6 support engineers and two managers in Liechtenstein and Florida.

Adventa was spun out of Texas instruments in May 1998 (headquartered in Piano, Texas). They provide ControlWORKS, the world's first 100% single-wafer processing system. It provides generic capabilities. What they really like is that Adventa give customers all the code. (They've worked previously with libraries delivered in C, C++, etc.; it became a nightmare just trying to keep the system running.)

ControlWORKS simulator is a valuable tool. The machines cost $5million so you don't have spare ones around to test on, still less to break. The simulator is also used for training and demos. And all the non-simulated part is exactly the code the real machine will be running. He showed the complex framework architecture slide of some 30 boxes, showing VW underneath framework support services underneath ... .

All the machines are doing things at once so it's a distributed system. Productivity is measured in wafers / hour and anything that increases that helps return the machines' (large) cost. N UI images talk to a single supervisory control image that talks to N machine control images. Each wafer costs more than the operator carrying it around will earn in a lifetime so it is very easy to justify automation; you will never recover financially from one mistake. The monitoring UIs sit on the Factory LAN and the machine CPUs sit on the Machine LAN with the control UI CPUs. The Supervisory CPU sits on both. Each Machine CPU connects to devices either directly or via the module's LAN/Bus.

They are continually trying to improve heat-handling, clock-speeds, anything that will speed production. They also can send wafers to other buildings or even other countries for further processing and return so there is now talk of Business-to-Business data passing via XML or whatever.

In the machine you must control wafer location, ionised gas, etc. Open one wrong valve and the machine will literally explode. Every mechanism has a finite state machine logic. They started with typical Smalltalk code doing

synchronous control and thread management but this caused much effort on thread management (how do I clean this up and move on) so for the last few years they have moved to FSM and alarm handling that saves state. The priorities are: don't hurt people, don't hurt wafers, don't hurt the machine.

A 300-mm silicon wafer costs only a few hundred dollars, drawn as a crystal out of a molten bath of pure silicon. But the total processing cost investment in a wafer is huge. (That drives bigger and bigger wafers; the machine costs are much the same whatever size wafer they handle; recently they moved from 200mm to 300mm wafers.)

Their FSM framework nests state machines and allows for interruptible behaviour. Time in each state's action method must be short as that is uninterruptible. Whenever an action method gets long, they rewrite it as an FSM. As each FSM can be in its own thread, they can no longer just wrap a high-level call with an exception handler. Instead, alarms are passed to the FSM that started the FSM that raised the error; it knows what to do.

Adventa provided a policy framework as they used FSMs more. This abstracts much shared behaviour from the many, slightly differing FSMs. Policies control e.g. whether to allow (not in production) or block (in production) maintenance activities.

There is a machine namespace so they can call to any object in the system performantly. He showed some UI screens; typically, these showed status-displaying graphics of the tools and tool clusters, plus tables and fields.

Experience: small teams work better than either individuals or large teams. They've been using Envy for years. They like some things about Store but wish others were more like Envy. For example, Envy unique ids were very useful when they changed ControlWORKS code and gave it back to Adventa: reconciling unchanged code to its equivalents in Adventa was a non-issue and the ancestors of changed code could be tracked provided the complete set of parents were provided. They need Store to maintain this "this came from that" synchrony between their repository and Adventa's.

The learning curve for their system is a challenge. Smalltalk is quick to learn but the domain is hard and the class libraries are large. An experienced Smalltalker can take six months to be let lose on real machines. Names in code are very important; they must be accurate and readable. Sometimes two application are identical, or nearly so, in code, just differently configured.

Would they consider anything other than Smalltalk? They find Smalltalk fast (10ms resolution) so the perception that Smalltalk is slow is now quite wrong. Smalltalk is a small community so it can be hard to find developers with all the right skills (ST + OOAD + RT control experience). Marketing-wise, Smalltalk is perceived as not mainstream. Connecting to external s/w in VW2.5 has been cumbersome (many active X controls they've had to connect to). The non-native 2.5 widgets now look old ("Why can't you

make it look like this.") Hopefully, VW7 will help in both these areas.

Why use Smalltalk? Of their many reasons, debugging is the most important. It can take him two hours to configuring machines, resources and state to reproduce a problem to the point of seeing the bad line of code. If he then had to restart after changing it instead of carrying on, he would waste days tracking down problems. He simply cannot do without it. Hence Unaxis is committed to going on with Cincom's VW and with Adventa's ControlWORKS for the foreseeable future. Unaxis is at the bottom of the foodchain (e.g. slow Christmas sales of machines or cellphones can feed back to an absolute stop on any purchases from their point of view) and must be productive when orders arrive.

Georg Heeg asked him about the ADM company and was told that, "Their Smalltalk is talking to my Smalltalk." (see 'Other Discussions' section).

Number Crunching Smalltalk, Daniel Poon, Romax Technology Ltd.
Dan has worked at Romax for 10 years. Smalltalk is not always seen as useful in numerically intensive tasks but it has proved to be extremely productive for them. Romax started in 1989. Their first release of RomaxDesigner was in 1994, then written in Smalltalk, C++ and Fortran. It lets engineers model gearboxes and transition systems. Dan showed a slide *covered* with logos of automotive firms, wind generator designers, agricultural machine designers, aerospace firms, etc. (on several continents) who use Romax products.

Dan showed several gear shaft arrangements that Romax can model. Romax uses static analysis. Force = stiffness (of all components, in six degrees of freedom) times the deflection of the gearbox under load. High distortion leads to excessive wear, failure, vibration, noise, all undesirable things. Romax is part of the virtual product development market, worth $2billion and growing at 10% annually. Designing takes 3-4 years (and much money) via physical prototypes. Romax cuts that to 1-1.5 years.

Customers are always asking for more complex simulations so they can reduce further the physical development. Romax spends 20% of its time solving linear equations and the rest on non-linear where traditional approaches don't scale for their domain.

They like Smalltalk because its simple syntax compiles lightening fast. Working for Romax, Dan missed the Java thing but suspects that their 400,000 lines of Smalltalk would compile slower (Donald McQueen: "It wouldn't be done yet."). With late binding you can run tests even when the system is broken, allowing stepwise development. They also like class extensions; using these, they extend the language for domain experts, bounce GUI and test behaviour off the model, etc.

Their FORTRAN code base does not need multiple callbacks; they have a single exception-throwing callback that the Smalltalk handles. In C# or Java, they would need many callbacks.

The technical computing market is huge. Products like Matlab claim a million users but MUnit, Matlab's test framework, has had just 48 downloads, which tells you something about their culture. Matlab-like products are designed for analysts' own use and do not address the product development cycle. Romax like arbitrary length integers, blocks, etc.

Smalltalk's quirks are noticed by their numerical analysts. They are OK with keyword methods, less happy about the unconventional arithmetic operator precedence, and would like FORTRAN-like array syntax for assigning into an array. There is a lack of native solvers and libraries.

Floating point performance is poor by default but there are work-arounds. Fortran is 70 times faster for simple floating point operations. If you profile a layered architecture (calculate in FORTRAN, do the rest in ST), you find you are spending 60% of the time transferring data between layers (plus you can't move the layer boundary easily to refactor, and can't profile across barrier). There is another way. Compared to fast-for-floating-point languages, Smalltalk is slow but compared to slow languages, Smalltalk is fast enough. Build it in Smalltalk, profile, optimise, use FORTRAN for irreducible bottlenecks. So they recommend the single language approach.

To sell this to their customers they use an analogy from the domain: the chassis/body metaphor. A car chassis provides torsion rigidity while a car body provides space for passengers to sit in. Thus these two requirements have been split, although it is not necessary to do so. This is a metaphor for a layered architecture, since chassis and body are made from different materials. But cars also use the monocoque car design in which there is no separate chassis: one structure gives torsional rigidity and the passenger space. It's stronger and lighter than its rival. It's also harder to design and construct but market forces have made it the norm. And it uses sheet steel, the same material as the passenger space in its rival; it's not the material that makes it strong, it's how you use it. Compare using just Smalltalk!!

Romax began with a traditional team structure that naturally produced a typical layered architecture: the programming team gravitated to Smalltalk and the engineering analysts to FORTRAN. To produce a global optimum, they created a single cross-function team using pair-programming, pairing a programmer with an analyst. This let programmers with no domain knowledge and analysts with no developer skills get started fast. It greatly reduced errors since coding and domain skills were always to hand. (Dan noted that engineering analysts may not be typical of your usual domain expert, since they are all numerate, literate people.)

The monocoque idea has proved valuable over time. As customers demand complexity, the tradition layer system starts duplicating function between layers. It also defers integration and testing, since you do changes to each layer first and only later integrate and find what fell between the cracks.

They used Smalltalk originally because they started before Java was known (C++ was not much of a choice) but it has proved a lucky choice. Smalltalk is malleable and malleability equals customer responsiveness.

Later, when Smalltalk was looking unfashionable, they revitalised things with pair-programming.

They would like the Smalltalk community to do more in technical computing; steal features from Matlab.

Q.(Eliot) FScript: array programming language for Smalltalk? There isn't that much array programming. Most time is spent in non-linear iteration.

Q. Your dialect is VSE. VW is not as bad as others in floating point. Might you improve your performance in that dialect? Noted.

Q (John) 3-D extrusion printers? They don't interface to CAD. CAD is a drawing market, whereas they model behaviour.

Q. Customers? They current have some 100 licences world-wide. In any simulation, boundary conditions are key, not just the accuracy of your calculations. Their product model extends so far out that you don't have to guess your boundary conditions. This is one of their selling points.

Q. How fast? A gearbox static analysis might take 30 seconds.

Q. Pair-programming; how do the domain experts like it? It started 4 years ago, deliberately starting small and growing gradually. You need to go slow, be tactful, make the organisation change. Matlab code can take 5-10 years to reach the market whereas their analysts see their code in the market in a month or two; that motivates. Some of the best coders are engineers.

Q. (Niall) Marketing? It has been word of mouth and early adopters in the past. Now they are in the second-phase and have to market themselves, but the market is growing quickly and they're lucky to be riding that wave.

Q. Migrate to VW? They're pretty happy with VSE although they may find themselves encouraged to think about migration anon.

Pushing Smalltalk on a 4GL Shop: RAD the object way, Giorgio Ferraris, ElevenSoft

(I missed this talk but caught details later from James Foster, who saw it, and from Georgio.) Georgio lives near Turin but works only one day a week there, others in Milan and elsewhere. He drives 70,000 km per year.

Smalltalk can easily beat rivals even in short-duration demos when those rivals are using statically-typed 3GL languages. If the rival is a 4GL, then it may well create a simple input form, database schema and have written one record to a simple table before a Smalltalker, developing the usual way, has done so. A target audience used to 4GLs may miss the value of Smalltalk in such a demo altogether, or may get the impression that later extensibility and refactorability is being paid for by a slow start.

Georgio has created a framework that creates an initial Smalltalk system at 4GL speeds which can then be evolved in the more usual way. Wizards let

you build a simple model, input form and database schema, and connect them. He demoed for the example of a simple personnel data application. The aim is to persuade a 4GL audience that Smalltalk is all gain, with no initial loss of speed, and it looked to me like it would do so.

You start from a text file defining the domain and mapping, and another that can define how any non-default elements of the app should be displayed. He showed an example: phone table, name, email, address, etc., with simple type info for the DB mapping (basic type or class). You load that file and his system generates the application from the spec. He opened a PostGreSQL DB and showed that the table had been generated into it from the VW (he supports DB2 and lots of other DBs.)

You can select from the tool to build/not build table, view, etc. He showed the generated Smalltalk: classes for PhoneContactView, etc. The Views appears in the VW GUI builder as layouts so their defaults work OK and you can edit them easily.

An .ini file says how to connect to the DB. Basic behaviour is to open a connection. Standard-named methods (`openMaster`, etc.) let you open the view and enter data, search for data, etc. You can add methods to let the domain or the view know what variables they want in the view. Search shows a typical view of matched instances in a paging system with the usual buttons. Search has good matching features and meta-patterns. The whole can be configured from the UI or with methods.

He can browse a tree list of the domain and select items from it to create views, search on it, etc. He can now save his choice with a name (and with description, etc.) and reuse it. Results can be exported to Excel. He can manage really large databases with this.

Giorgio then opened a real application he has delivered to a customer. It stores work agreements between companies. Blue fields denote subobjects with their own views within the main view. You can drill down into these to see views or to search (from a field you can query the subobject on all its fields). He has strategies to load just what he needs from the DB, storing and wrapping as needed.

This is all Smalltalk with the power of Smalltalk just waiting to be used. If you want checking on data entry, just edit the setter method, and so onward and upward to a full Smalltalk system if needed.

He has customers who use C# on the web and VB internally and he has interested them in Smalltalk through showing this.

Anaesthesia Information Systems at the University of Florida, Wilhelm Schwab and Gordon Gibby, University of Florida, Department of Anaesthesiology

This system would not exist if it were not in Smalltalk. Smalltalk is Wilhelm's edge. His talk focused on people and physics more than on his software, showing why Smalltalk was needed. It was very pictorial.

The medical environment is very hostile to machines. They get knocked over, moved around, etc. Various people are involved:

Clerical staff ("Please don't tell them I called them clerks."): some are techno-freaks, some are people-people and some are typists and hostile to UIs, since typing is what they're used to. (One of his pictures of these people, showing their environment, just happened to have another Dolphin product of his visible on one of the screens.)

RN data entry: the next group are those nurses (often male) who serve the physicians. Sometimes they are incredibly busy; at other times, they are sitting around playing solitaire. They tend to be good at time-keeping, finding things, spotting exceptions. They may do surgical procedures. They like forms where all they have to do is note any exceptions to the default situation. As they are paid $15/minute, it is important to track when they arrive and leave, so basic data capture is a key part of the system.

ARPN data entry: a second kind of nurse (almost always female) gets patient histories, sometimes from very ill people, and builds up much knowledge about them. These people are less likely to be in life and death time-pressure situations but they may have a full waiting room. They have much more a view of the whole patient, as against doing this procedure repeatedly on many patients. They start to see connections because they see the history. Systems must let them just describe things.

Anaesthesiologists: they are like airline pilots, who are very busy at take off and landing and may just be watching the autopilot for much of the time between. Anaesthesiology is boredom interrupted by moments of terror. They may have to flat-line someone at 17C (for the briefest time possible) to save the brain from harming itself. They must watch hawk-like at all times. They have no reserve attention at such moments and need machines that are responsive and forgiving. They are not people that a sales-talk can fool. They do not want to be distracted from watching the patient by the need to handle the monitoring machines.

Haemodynamics (i.e. what is the blood doing): an anaesthesiology record will capture every five minutes the heart rate, pressure, oxygen state. Every fifteen minutes, it records what oxygen is being breathed in, what C02 is coming out, what anaesthetic agent is coming out with it, what the brain's saturation with it is, their heart's output, BIS/EEG asymmetry (are the brains hemispheres behaving differently). All these relate to situations that can occur: for example, a patient's metabolism may suddenly make them too hot, expiring too much CO2, etc. If there's a history, anaesthesiology may be planned for it. If there is not, a sudden response may be needed.

He showed pictures of people recording this data on tablets. People used to clipboards adapt well to tablets. He then showed a picture of an operating theatre at its calmest and least cluttered (i.e. not very!!! :-). So where do you put the machine? And what does it need to do? You need a central server, tablets to record, etc.

He calls his system MEDirect (all the better names were already taken). It knows the patient index ("James Bond, patient number 007"). It has the user authentication, the schedules (N.B. it is not a scheduling system) and it can retrieve the patient's encounter documents ("I think we've seen Mr Bond before"). It has pluggable resources: publish encounters to MEDirect, find records, etc.

His clients talk to a web server. (Apache, but he has written his own web server which he will soon release. Apache's complication scares him; he thinks his server will be safer and more secure.) His server gets resources and provides them as requested.

His terminal and hub fits on the back of the $80,000 anaesthesiology machine. The hospital is more worried about the hub than anything else ("Keep our network pure!"). At first it was on a separate cart but this was asking for trouble (trip over it, moving machine unplugs it, ...). He found a better location in the back of another unit and wired the electrics and stuff on the back panel (which, he assured us, was simpler than it looked in the picture). He did not want to have air pressure pushing dust out of the cabinet into the theatre so arranged the fans to cause slight underpressure within it. He could rig up the power strip, computer and wiring. He had to be careful taping tablet pen holders in place and suchlike to avoid some obvious "Oops, moving that caught the pen top and broke it" possibilities.

His GUI remembers connection details, i.e. which monitor interface to use for each configured serial port. It can nag the monitors and has device-specific expectations (which are often pessimistic).

Lastly, he fired up the system. It opens on a list of ORs (templates: these are controlled or everyone would have their own way of doing everything). The chart is where you record. It has a grid to help users avoid, when the time is just before or just after midnight, recording events as late yesterday instead of late today or early tomorrow instead of early today.

Don't put lots of stuff in front of physicians where something contra-indicated might be buried. Physicians like to be shown things item by item so they *know* they've missed nothing. This is for the good reason of being thorough and the bad but these days essential reason that one mistake in the record makes a lawyer very happy. :-/) Physicians are good people and want to avoid mistakes.

Smalltalk lets him deal with physical challenges by giving him the time to do so (it's productive). It also lets him support domain experts who cannot give him specs because they do not understand software jargon so he has to show them the wrong thing to discover the right thing. Lastly, Smalltalk is robust: these tools run 24 x 7 and he reboots every 7 weeks to reset the Dolphin timers in synch with the system clock (which wraps around at 49 days plus change) but otherwise they would run for months on end.

Smalltalk in Medical Instruments, Andy Hodges, Medtronic
Andy is an electrical engineer who develops in Smalltalk. Medtronic's

mission statement was written in the 50s/60s and has never needed changing. People in the company like it. It focuses on tasks worth doing and preserves the feel of this 50s-started small company.

They build deep brain stimulators for Parkinson's disease (which block bad signals in the hypothalamus that cause trembling), cardiac management devices and spinal devices. The cardiac rhythm management system is an implanted cardiac device (pacemaker or defibrillator) and a programmer device that controls it. This is what Andy works on. They are not (yet, although Andy has thought about it) using Smalltalk in the implanted device; they use it in the programmer companion.

The programmer is a PC motherboard with custom telemetry, ECG and a chart recorder. They still make use of OS/2 because development began in the early nineties and you do not go to a patient whose pacemaker battery will last another 10 years and say, "Sorry, the OS is becoming old, we must operate and remove your device and replace it with one whose companion runs on a modern OS." They have been changing over for 5 years and may have retired OS/2 in two years. The programs must be the same on both OSs; people who run the programmer must not have to do different things depending on the age of the patient's implant.

Andy works in the research arm. They download new software into devices to see what happens, sometimes on the bench, sometimes on animals, sometimes in full-blown clinical trials.

Their old system required custom telemetry hardware, which did not work with new telemetry protocols. Before Smalltalk they would have spent 1 - 1.5 years designing new telemetry hardware. Their customer also did not like having to use one system to interact with device normally and another to download new behaviour.

Someone in the human factors group started using Smalltalk to show UI prototypes but she also wanted a high-fidelity prototype. She particularly wanted the prototype to use a stylus as the end-users did, not a mouse on a desktop where some UI issues would not emerge. It was also easier to use the real device to generate sample scenarios than to construct them. In 1999, she started a six-month project, helped by OTI, sub-contracted to CompuWare. Much work was done remotely, merging daily over ftp. It implemented a representative feature set for a market-released implanted device. This worked well for human factors but also met the performance needs of real devices. Sadly, the C++ / C# main development group simply ignored this. But the research group used it to replace a two-year hardware project with a three-month Smalltalk project, much to the satisfaction of their customers running the clinical trials.

They use small development teams (2-3 people), short project times (3-5 months) and projects with a customer of one researcher with the new clinical idea. They have 20-25 projects per year for 15+ implanted device platforms. They have no shortage of work to do. These are safety critical systems in regulated environments.

It takes 2 years from a device being built to FDA approval and use, so by the time they start learning about a new platform it frequently happens that all its developers whom they'd like to talk to have forgotten about it.

They use VAST 5.5 + VAAssist. The do as much as they can on Windows, the rest on OS/2, then install on the target platform and test. It's a life-saver that they have the full debugger on the platform. (Whereas the C++ developers don't even have an IDE - everyone uses their own favourite editor - and compile flags!!! The research team replace the C++ layer with the Smalltalk layer over the firmware layer for each new device.)

EnVision is the framework for all their development. It began with the code from the human factors group. HF had used VA plus WindowBuilder Pro. Research replaced the GUI with simple VA Composition Editor UI as they were less focused on UI. He showed examples: simple fields, buttons, notebooks, tabs. Typically they download the software into the device and then have to provide parameters for some customisation that must be done. The drop-down box in OS/2 is ugly (he showed how only the arrow is active, not the text box to its left) but their researchers live with it.

The use the same DLL as their market-release software. Luckily, Andy is good friends with the production guy that owns this DLL so it has been easy to get him to provide a simple C wrapper round the C++ DLL that the production team uses. Most calls are synchronous but not all. If the heart contracts by itself, that is a 'self marker' that gets sent back. In VA, these callbacks have to occur on the main Smalltalk thread or the VM crashes (bad) so they implemented a queue that holds these till the main thread can attend to them.

Why Smalltalk? Reusing the HF work to avoid building custom telemetry made sense. The C++ production stuff is *not* designed to be flexible so they would have had to build a system from scratch in any language if they had not reused HF. He can count on one hand the platform differences they have to handle (window maximisation, beeps and bells, some low-level DLL calls). Above all, Smalltalk is fun; work is not a four-letter word.

The ability to customise the IDE makes them faster. They built menus to let them configure between simulated devices. They tweaked code printing to list Envy versions for their regulated process. They've enhanced change management analysis and export to feed their change management system.

Lastly, Andy presented a few slides on the fact that agile development works fine in medical device programming. A company's high-level practices (Andy has worked for several; it's always a pyramid) are so general anything fits. The low-level is where the challenge is: make these items be part of your work. Regulations say what but not how, a fact that many people on both sides do not realise but you can work with them over this. Not much in agile actually contradicts fashionable process and QA.

So, how agile are they? Some groups use Scrum. The research group is not agile yet but seeks to become so. The three UI people are keen but the five

firmware people are less so. They do unit tests. They do some test-driven implementation (e.g. the UI must interrogate a memory range in the device long before the device firmware is ready, so the UI writes tests mimicking). They do refactor. He was glad to hear that the latest SUnit supports logging (had written some himself). They use TestResources to simulate or really connect to the DLL. They also support interactive tests on the target platform to check telemetry error handling, window formatting (text not clipped, coloured correctly, etc.; need to check this for the various platforms). They do a little pair programming They like the values of agile.

They intend to upgrade to VASmalltalk 7.0, enhance the UI and refactor the model-view separation.

Making Money with Smalltalk, Jeff Hallman
Jeff started by explaining that reserves are deposited by commercial banks in Federal Reserve Banks. It is a legal requirement that any bank worth over $50 million maintain a 10% deposit. The Fed controls absolutely the supply of reserves. When the Fed buys something it pays for it by a Federal Reserve Bank check. When your (commercial) bank cashes this cheque, it gets a deposit at the Fed. The demand is created by the 10% requirement on commercial deposits. The Federal Funds Rate (the overnight rate that banks charge each other for federal reserves) controls this demand.

In the early 1980s the Fed was explicitly targeting the various monetary values (M1, M3, etc.). Today, the Fed controls the FFR, thence forecasts the demand for reserves, so decides what reserves to supply.

The Money And Reserves System (MARS) loads several dozen reports, and adjusts for known reporting errors. Banks are good at minimising their reserve requirements, e.g. “You can have more interest on your current account if you let us put your money every night into a money deposit account (without reserve requirement) and keep your transaction rate low.”

MARS loads all this into a huge matrix called the money files (weekly money file and monthly money file). MARS then recalculates. The basic calculations are done in C code. They are mostly linear but there are a few non-linear calculations. You can then make a textual or graphical forecast which automagically adjust its components to satisfy various constraints.

MARS is support by two components, R and Smallpack (Smallpack is in the Cincom Open Repository). MARS was first built with C and a product called PowerBuilder, whose supplier went bust two years ago. Recent OSes break PowerBuilder so they had to rewrite. After a year of politics and a visit by Jim Robertson, Jeff persuaded them to let him use Smalltalk.

Jeff fired up the tool, showing a data screen. He has development tools (inspectors, etc.) available so if anything goes wrong he can fix it on the user’s desk (conveniently, the users are all in the same section so they are to hand, and he to them). Often, the data coming in is wrong but they cannot get the reports changed in time to fix it, so they correct the data, these corrections being tracked and removed when the report is fixed.

Next he showed a graphical screen, moved some data points relating to on-demand deposits and showed the smoothing effect on the Fed's deposits. Sweep accounts cause a marked monthly effect. Sweeps are banks trying to manage their deposit requirement by moving money to non-requirement accounts at the start of every month and then dribbling it back to cover cheques the users write as the month progresses). Sweeps, being new, are not yet removed by seasonal adjustment because it relies on past data.

The earlier system took three people two years to write. It had 83,000 lines of code and 27 mostly-generated make files adding another 10,000 lines of very unreadable code that broke from time to time when the IT people changed the system. His Smalltalk system is one-fifth the size, much more functional and took just him six months of work. It has 1900 methods.

Q. Niall: user interaction? Yes, he can be called round for a walk-back or just because the user dislikes the screen appearance and he can fix and resume or change the look in ten minutes. He has done in ten minutes things that took the old C team weeks. He was a user of the old system so had the great advantage that he knew the requirements.

Keynote: The Value of Smalltalk, Niall Ross

This talk was a straightforward union of the material I presented at ESUG 2004 in Kothen last September and the material I presented at CSUG 2004 in Frankfurt last December. Both conference reports are on the Events and Trip Reports page of the whysmalltalk site (immediately under where you found this one :-) so, to learn what I said, just read my very long write-ups of my own talks in those two reports (if you haven't already :-); even my vanity shies away from duplicating them into a huge block of text here. :-)

I had a generous time slot, which meant that afterwards Adriaan van Os told me, "For the first time, Niall, I realise that you *can* speak slowly."

Process, Techniques and Tools

Keynote: Domain-Driven Design: Tackling Complexity in the Heart of Software, Eric Evans

Eric started out as a Smalltalker (wishes he could say more than started out as). Smalltalk was his first love in programming. The Smalltalk community is also the root of domain-driven design (and many other things). In the 80's Eric was sure that he would not be programming in Smalltalk in 2005 but he thought it would be a superior descendant of Smalltalk, not Java :-/)

Domain-driven design: first define your terms (Niall: standard Oxford Philosophy start :-). The first key idea is that the main source of complexity is (Niall: should be) understanding the business domain. There are exceptions - a simple transaction that you must do thousands of times a second with five-nines accuracy is mainly a technical challenge - but it's true of most. The second idea is that when we meet complexity, we tackle it by modelling. When Eric says model he does *not* mean a UML diagram.

He showed a 1700's map of China. China was in the centre, with the rest of the world briefly summarised round the edge. He then showed a more

modern map (which just happened to have the U.S. in the centre - when he gave this talk in Europe, they saw analogies :-). This map shows Greenland and Africa the same size so it's not accurate. But it is conformal: the angle a navigator measures from the map is the one they can use to travel along.

The model you choose is driven by the use you intend. It is not 'as realistic as possible'. Our intent is to allow software to enter the user's domain. Eric therefore feels that a model not reflected in code is irrelevant.

His example was of a shipping application: containers are routed from suppliers to customers by truck, rail, sea. He showed a possible design of a user's current system: cargo object -> routing service -> table of bookings (each row handling one leg of travel). This is a very common style of design (becoming even more common thanks to fashionable 'service-oriented architecture' ideas).

Eric dislikes this design. He dislikes the side effect of the routing service generating a complex solution that then changes the database state. Testing it is tedious: run test, check new db state, reinitialize db. This confounds the task of mapping OO to relational with that of finding the correct route.

A more fuzzy objection is, "How do I talk about this design?" When you say, "Each stop in the route is a table row", either users ask, "what's a table row?", or (worse) every future requirement from them comes as table rows. Eric cares about the language people use in talking about the problem, the concepts they need. The concept of a stop, a cargo being unloaded from one mode of transport and loaded onto another, is needed, not a 'row'. Or maybe a cargo is a series of legs, loaded at the start and unloaded at the end. Or maybe legs are just from and to, leave loading for stops. The cargo, the route specification (start, end) and the itinerary can all be distinguished. People are to apt to grasp the first thing that occurs and stop, but they should not; brainstorming is cheap. In his revised design, the routing service takes in a cargo and a route specification and returns an itinerary that satisfies it. No side effect of table population is specified. Our code is now likely to resemble our domain-oriented conversation.

Is an itinerary made up of stops or of legs? Eric likes to talk around many ideas (e.g. ten) but eventually you want to decide which is best. Obviously, we want the one that is most useful to us. A concrete reference scenario helps find this. Scenario 1: send toys from Hong Kong to Dallas via ship to Long Beach, rail to Dallas, truck to customer site; either model describes this. Scenario 2: reroute mid-transit; customer wants the toys in Seattle, not Dallas. (Software engineers are always trying to make the world simpler and tidier than it is, so neglect such scenarios.) Drop last part of route, replace with rail from Long Beach to Seattle. Now the two scenarios are distinguishable. The reroute requires us to look inside the Long Beach stop object, but does not require us to look inside any leg object.

Scenarios in which the cargo is mis-routed, cancelled (returned to sender), etc., will all be easier in some models and not in others. Eric's rule-of-thumb is that the model that makes the *hardest* problem easiest is the one

to use: if a model made routing a new shipment harder but re-routing mid-transit easier, he would use it, not its rival.

Summary: a ubiquitous language, structured around the domain model and used by the programmers, gives your software value.

Eric then talked about 'modelling non-sequiturs'; true statements that often get followed by false statements.

- We need tools that let us express models without getting bogged down in technical detail SO we need tools that let less skilled people develop using parts developed by more skilled people. But reconfiguring legs may be much harder than modelling legs, so you get unskilled people doing hard programming using the skilfully-developed 'leg' part.
- We need tools that help bring to bear deep domain knowledge SO we need tools that let non-technical domain experts to write programs. But domain experts are not experts in writing software; you need to find ways of working with them, not of getting them to do your work.
- We need to raise the level of abstraction SO we need visual programming such as UML. (In mid-90's, Smalltalk vendors got distracted building visual programming tools, which hurt Smalltalk. I also experienced this, e.g. in GemStone's GeODE visual programming product which must have cost them resource better used elsewhere). Eric's example shows that the challenge was not to draw boxes and lines of legs and itineraries and stops but on the contrary to see which were easier to take apart.
- We should drive the design from a model SO we need an upfront analysis phase. Upfront analysis means basing your model on the greatest level of ignorance your team will ever have. It comes from the idea that things are hard to change so you must get the model right first time. While well-crafted Java code can be changed, Smalltalk shines here; it's easy to change.

As for tools, Eric thinks the whiteboard (plus digital camera) beats trying to make Rational Rose produce the diagram you want. He also thinks the Smalltalk browser is a great modelling tool. Unit tests are a good modelling tool. Lastly, he thinks that mouths and ears beat most modelling tools; strange that when people talk about software they sometimes forget this.

Eric has seen many teams fail despite doing all the above. You also need strategic design: creating the circumstances in which the above approach can work right. Strategic design needs context-mapping. Eric was in a rowing eight at University. There are no rowing sixteens because even with eight people, someone gets out of synch every now and then and the results are dramatic (and alarming if you are in the boat). Software has the same issues. eXtreme Programming is fast and works well with small teams. Somewhere between 8 and 12, we reach the limits of people's ability to coordinate with each other.

But there are projects that must have more than 12 people to complete. On a large project, there are always multiple models - always. There are always

people trying to have one model - Eric was one of them a long time ago. Fortunately, you can make multiple models work. Imagine the team building the routing service that takes route specifications and cargoes and returns itineraries. Their task gets much easier if they model their world in terms of arcs and nodes, reusing existing knowledge on that problem.

Usually, teams just have different models because they did not coordinate, but it's worth remembering that shoehorning two teams into a model that fits one or none means the ill-fitting team(s) must solve their problem in surrounding service code; the solution will be ill-structured and invisible.

So you have multiple models. Once, Eric had a project with an Invoice object in the customer billing team and an Invoice object used by the team who paid subcontractors. In the interest of code reuse, the second team reused the first team's object. Suddenly, the billing team saw strange errors since two teams were trying to use the same word (and code) to mean two different things. It was the same thing at a very fundamental level, much more fundamental than either team's model could handle. Each team's Invoice could only be understood in context.

Eric believes that making the context map explicit is very helpful. Know what your contexts are. Know what the relationships are (if any) between terms in different contexts. Duplication within a context is bad. Apparent duplication across contexts is not bad. Eric discussed some possible context relationships. He recommends mapping contexts as they are, not as you wish them to be.

VA and GemStone Utilities, Angela Martin and Kimberly Anderson, Northwater Objects

I missed this but Angela and Kimberly kindly re-presented it for me later.

Northwater objects is the in-house development team of Northwater capital, a financial company of some 90 people in Toronto (mostly) and NY. They use VA and GemStone. Kimberley started working for them this year, Angela started three years ago. (Bob Nemec has been there forever.) They have lots of tools to support their development process.

They have a custom menu on their development image for login to GemStone, icon manager, UI spec manager, direct access to Envy and debugging. Kimberley did this menu; previously, they executed code in workspaces. The menu centralises all the tools. (It was a good task for a new starter and it helps others.)

She added menus for saving a method and compiling it to GemStone immediately. They develop in manual transaction mode all the time. Every two minutes a process aborts their transaction so if you are developing without having disabled that process for you, this one-menu-click save-and-compile-in-GS is safe. The menu item is disabled (in fact not there for non-GS-stuff (UI views, etc.). They also have an RB hook-in menu to find lint more easily (easier to find, so more often used).

They have a weekly cycle: development ends Thursday and is applied on Friday + weekend. Every Friday afternoon they have a test demo plus users come and talk about their issues. One day per week for testing is enough because they have a weekly cycle. It is a positive way to end the week, focusing on what you have achieved that week, not what is still to do.

They have a group meeting on Monday: each developer selects the issues to work on that week. A major bug in the system would prompt patches but only that code would be released. They should cut the config map for every release; for a small patch, they might decide not to.

Their issue management UI holds feature requests, bug requests entered by users and bug lifecycle: new, prioritised, opened (dev working on), test (so marked on a Thursday, user knows to test it on Friday), closed (bug icon turns into butterfly icon). Users get emails whenever issue state changes (so knows to test this Friday).

They have a code-compare tool between image and GemStone. This is used to update the test database with code being worked on and also to compare all classes every Thursday. The tool scans for method code changes and (2nd pass) class structural changes, deletions/additions, etc. Recently, they refactored this to speed it up, gaining 20%.

They also have a migration tool in which they can specify which instVars to map to where, whether to map only specific subclass' instances, etc. A few weeks ago they migrated 30 million objects (10% of the database) which was done staged over two weeks. Happily, the tool tolerates mixes of migrated and unmigrated instances (by just ignoring the migrated).

Envy change reports: they compare the opened and versioned config maps versioning only those apps that have changes, releasing the parent version of those editioned apps that have no changes. (They always open all apps in the Dev map as it simplifies permissions issues and also tracking issues.)

Class comments have been made issue reports (this practice needed discipline) so their code and issues are coherent.

Their GemStone is on Linux. Cron starts statmon every morning so they have stats but they have no RT monitoring services. Instead, they have tools to view the session stats, times since last transaction (asynch abort keeps their commit counts low and transactions short; there is no transaction until the user hits 'save'). Colours change as the commit backlog increases and (since last week) the system will email warnings.

They use icons a lot. Their slides are colourful, as is the system. They store all icons in the image so they have no file dependencies; a goodie manages this. Email notification is used for lots of things: warnings, walkbacks, commit conflicts, etc. The emails include stack traces and as much information about what the user was doing as possible (sometimes they call a user before said user knows they have bug).

They use WindowBuilder Pro and have an editor for attribute specification editing. A generic attribute view lets them view any new objects that have an edit policy spec. Later they can develop a fancier UI but still use the generic for many inspectors and analysis and debugging things.

Because they use the same code in VA and GemStone, they can have a mini-model in memory so they can run SUnit tests against it.

Their group is very talkative and encourages debate to make things better.

Monticello 2, Colin Putney

Monticello started as a thought experiment between Avi and Colin. Squeakers work by mailing each other code so typical CM stuff is not quite their need. What they need is good merging. Thus Monticello 1 did little except merge.

Merging is simple in theory. Ancestors have descendent branches which then remerge. Typically, find common ancestor, apply changes of lesser branch to greater to get new version. Almost every CM system (that does merging) does this. Problem is when first branch changes #alpha and #beta, second changes #beta and #gamma. A human must merge #beta's code.

Then the secondary branch goes on and now you have another set of changes to remerge. Using single-parent reasoning, the #beta conflict reappears, but it should not if we've merged it and the secondary branch has not changed it again. Monticello 1 had a package-level solution to this (see last year's talk). Monticello 2 keeps all ancestors of all methods, a much finer-grained solution. Thus #beta knows both its ancestors and the second merge uses this to see that there is no further merge problem for it.

The next issue is cherry-picking; users merge by only fixing 'interesting' conflicts. Suppose #beta in the main branch, being un-cherry-picked, is not merged. Now the secondary merge either sees a false conflict or, worse, thinks #beta was merged when in fact it was simply ignored. Monticello 2, by using method-granularity histories, solves this. You have the method's ancestry so can see whether it was merged or ignored last time.

Q. How do you record ancestry info? Monticello 2 records ancestry with the thing so described, so a method's ancestry is stored along with its source code. When you load the method, you get its ancestry.

Colin finds this a simpler and more robust solution. Now that every method has ancestry it does not matter which package that method is in. A package can overlap another or be a subset of another. Monticello works with 'slices' of the image, where a slice can be a package or a set of methods matched by a rewrite tool expression or a change set or whatever. The ancestry of a method can have deletion in it. A slice can include a deleted item. Overall, it is a lot simpler than Monticello 1 but weirder.

There is a final Smalltalk-specific issue; how do you update a running system. Smalltalk tools are designed for this but can show problems if you

load a Monticello package that overrides a method used in loading. Colin is trying to make Monticello 2 do as close to an atomic load as possible. He aims to load while running in a Sandbox, then atomically install and migrate all instances. This is more interesting to do in Monticello 2. They have instance variables separate from classes so they are forced to do atomic loading to keep performance reasonable (otherwise it would be: load class, migrate instances, add instvar, migrate instances, add another instvar, migrate instances, etc.).

Q. (Niall) Edit Monticello version histories e.g. make two method versions with same source become same version (c.f. Store reconciles)? There are no tools to do it. It could be done. Colin knows no scenarios where he has wanted that. After discussion, it appeared that the method-level granularity eliminated the scenarios where this would be wanted; methods are either identical (compare tool then identifies them) or not, in which case a human probably has to compare them.

Q. Still need higher-level ancestry? So far, they seem to find they do not. Colin found that shift in thinking hard but is now used to it.

Monticello has Element, Version (Element, state, history), Slice (not versioned but a collection of versioned Elements), Snapshot (slice saved in repository). MetaVersion is for non-code things: comment for snapshot, version no. of snapshot, etc.; all these non-code things are also versioned. So you can search for package comments descended from the comment of the package you are viewing.

Colin then demoed the merge tool. The UI is basic, resembles that of the Store tool, shows conflicts.

Team Programming Tools, panel, Eric Clayberg, Colin Putney, Bob Westergaard, Niall Ross

I gave a brief overview of my own feelings about Envy, mostly as compared to Store, since I've used both a great deal and often moved code between them.

Starting Envy: because it has its own database and data format, Envy is easy to set up and has a small footprint. Because this DB and format are proprietary, and the code is hidden (most un-Smalltalky; some people have the code) going behind the scenes at the lowest level is hard. Component lifecycles and roles can puzzle starters. Novices can find some of the {Edition, Version, Release} x {Config Map, (Sub)App, Class, Method} details unintuitive. However Envy's choice of how versions and line-ups interact is defensible, if rigid. Configuration expressions are powerful but their UI is wretched.

The component ownership model presumably looked like a good idea at the time; I can't say anything nicer about it. It is unintuitive to starters, supports what programmers don't do, and frequently trips you up doing sensible things. (Eric's explanation of why he has added a superuser to VASmalltalk - see his talk - is a good example.) Experienced users

routinely switch off what they can and code work arounds for the rest; everyone tells everyone else where the hidden code is that lets you fool password checking because everyone knows you need it. A model that you find you *should* routinely workaround is a bad model.

Using Envy: Envy generates globally unique version ids (from machine timestamp + db-identifier). This means you never need to reconcile which is almost always good. Send a version round the world and your Envy repository will still recognise it instantly when it is sent back to you, unlike Store which needs to reconcile and can be fussy (**Thor Raabe's** mention above of his experience Store and Envy sending GemStone mods back to GemStone is an example). Occasionally, the fact that you cannot reconcile is annoying. Store knocks the spots off Envy when it comes to refactoring and this is one example: doing things in the image and/or in the database and then forcing the two sets of changes to reconcile again can be an excellent way of pulling in your too-branching project, of forcing comparison of semantically-the-same but historically-different items, etc., and here Envy gets in the way.

Another example of this 'using versus refactoring' split is the fact that Envy is always on. This is great for capturing all your work and presenting a consistent interface when comparing or reverting (Diana Merry-Shapiro's utility offers something of the same for Store). However you cannot say to Envy, "Look away for a moment; I promise everything will be packaged correctly when you look back." This makes refactoring components hard: each step must be a valid stand-alone change. Renaming of components requires destruction and recreation (with loss of history) as it clashes with how Envy identifies components.

Related to this is that loading problems can be hard to solve. Knowledge of 'The Browser from Hell' (Application Editions Browser), which is more tolerant than others of opening and changing things while unloaded, can be essential. Debugging loading problems often lands you deep within huge hidden-source methods. The good side is that it does not happen often as Envy is a mature product.

Although Envy is always on, its warnings of clash possibilities ('someone else has a later version of the method you just changed, from the same parent') are poor. Coast improved them by showing 'weather' indicators; sunshine means no clashes, cloudy or rain for states of increasing possible seriousness.

Another always-on drawback is its very chatty protocol. It is intolerably slow over a 128k pipe and pretty bad over any narrow connection.

Merging, along with component refactoring, is *the* area where Store wins over Envy. Envy's comparison tools show their age, and they are comparison tools, not merging tools. Every 'load alternatives' decision is manual and if you're not using the Three Way Difference Browser add-on then you have four clicks to check instance-side-public, class-side-public, instance-side-private and class-side-private whenever you cannot be sure

that private has not been used. (Envy's private / public distinction was presumably another idea that looked good at the time but is a thorough-going pain to users.)

The proprietary data format and unique version identifiers make exporting and importing easy. The only potential gotcha is 'Export Changes...': a very useful optimisation most of the time but very confusing if you import someone else's exported changes without knowing it and without your repository containing the base version they used, as warnings are minimal.

Extending: I've done a certain amount of this to make Envy cooperate with the Refactoring Browser, with other CM systems, and to improve the tools. Envy's UI-end is well-factored; Envy has many front-end utilities, some significant. The code that accesses the database is robust but it is less well-factored (it is also hidden but my impression is that it is innately less well-factored over and above this). Envy in VW is intrusive, sometimes in ways that trip up utilities, obstruct refactorings and generally force small but irritating code differences between Envy-VW and non-Envy VW. (As Envy is an integral part of VA, it is hard to say if the same is true there.)

Envy's Future: Envy is VA's past, present and (I assumed: Eric confirmed) future. In VW, I have done work that has made me very aware that there is no need for all the code in an image to be managed by the same system. Alan Knight's Store-in-VA, etc., work proves the point. And the fact that VW can organise code in categories, parcels and/or pundles is also an example. With a better factored DB end, it would be straightforward to have only some code in the system managed by Envy (with applications presenting a pundle interface to the VW system). Even in its present state, this is entirely feasible. (Implementation requires certain singleton state objects in the RB and elsewhere to become multiple, c.f. my demo in Bled of it in the RB.) When users can move code in an image between management systems on an application-by-application basis, taking their time to complete a port, then porting between CMs will become more XP-like and have less of the big bang style it tends to need today.

Regrettably, as regards porting from Envy to Store, the political/admin issues of making Envy available in VW7 (see Q below) mean that VW users with systems still in Envy may not receive mixed-CM anytime soon.

Lastly, I mentioned that Dolphin has an Envy-clone CM system. I believe most Dolphin users find the default cvs-like CM system adequate but the Envy-like one is used (confirmed in audience discussion; see James Foster's presentation of it in the 'Other Discussions' section below).

Monticello: Colin learned early that what you're trying to do is capture how you got to where you are. Because of this, your CM tool will reflect your process. Monticello 1 is designed for the Squeak process in which 100+ people are working and none of them can be forced to do anything. Thus Monticello is good at merging and that's about it. Most people in the Squeak world use it and Squeak 3.9 will be developed in it. Tweak uses it.

Monticello 2 will replace Monticello 1. It keeps the philosophy and breaks the stranglehold that packages had, making it easier to rearrange things. Like Envy, Monticello uses global ids, an important idea.

Store: Bob Westergaard is a member of the VW engineering time. Store was called Stash, which was called Bernstein, which was called Eagle and originally came from Andersen Consulting. It is similar to Team/V: bundles, packages. It is the VW CM system. Store is slower to set up than Envy. Store is optional in your image, which can help deployment and size. Store blessing levels let you adapt it to your process.

If methods are invoked as they are loaded, Store can have problems (e.g. adding methods and icons to the visual launcher). Bob felt the comparison tools were poor (Niall: he should try Envy) and the shadow-browser tools were old and in need of updating (true). Publishing the base is something every one does their own way today; they will provide a single solution.

Store unlike Envy does not have atomic load, but will in 7.4 (put code into shadow namespace). At the moment, you can have a load fail and leave your image in a very poor state.

Q. Envy in VW7? JPMorgan Chase has bought certain rights to Envy-VW and hopes to implement a VW7 version of it. However they do not have the rights to offer it outside. For that, they would need further permissions from IBM (and maybe also from Cincom; Eliot has indicated Cincom has no fundamental objection, which is not to say there might not be some legal/admin hurdles to clear). JPMC technical people are well aware that it would be in their interest to amortise the cost of keeping it up to date by having some external group handle it and offer it to others. (These others may use it as an easier way of porting to Store, step by step. JPMC themselves, because of the proprietary CM system they have implemented on top of Envy, might be slow to port to Store.) However there seem to be significant political/legal obstacles diffused between IBM and, it might be, JPMC and/or Cincom. It is just possible that as VAST approaches end-of-service, IBM will become less hostile to the idea of Envy-for-VW being handled by another company (ideally Cincom or a Cincom partner), or they might simply become less interested in what happens to it.

As regards Instantiations, quite apart from the work they have to do on VASmalltalk, they do not have the Envy-for-VW code or the rights to it and so cannot help advance this.

Q. Overrides in Store: when you unload the current one of several overrides, which one is then in your image? It used to be that they were restored in order (parcels) but they have lost that and need to fix it for 7.4.

Q. Why can we not see class versions? Store has a different history from Envy and though that knowledge is indeed there it has not been provided. It was a design decision.

Q. (Alan) What are common to Smalltalk versioning systems? Colin: the

syntax elements of Smalltalk are the elements used by the CM. IBM has thought of syntax-aware CM in Java but it is hard and is going nowhere fast (that he can see). It is so hard to discuss this with non-smalltalkers; compare the ease of implementing Monticello merges with the amount of code other language's CM systems must write to do a rather simpler thing. Monticello specifically stayed away from integrating with the IDE.

Niall: Smalltalk's strength is that everything is visible. You can interact and refactor CM (system or items) in small pieces. Envy is a good example of this except at the low-level data format level where, violating this Smalltalk paradigm, they keep key code hidden and undocumented.

Q. Heeg: Eric's keynote talk stressed the need for an 'appropriate' model. What rating (1 - 100) would the panel give as the appropriateness of their CM tools' models?

- Niall: Envy is a curate's egg. At first, I guestimated its model as 75 or 80 overall but after thought revised this down to 60 - 70. It would be lower still if I were thinking just of the model, not of model plus mature implementation combined. Some things in Envy are good but many are too specific and some (the component ownership model) are plain bad.
- Colin: the Refactoring Browser is an example of a tool that is a better tool because it has a better model. He would like a refactoring version of Monticello. He went for a similar value.
- Bob felt there were a lot of problems with how things are modelled in Store. Although it beats Envy in merge tool support and the like, it's schema must be changed. He initially suggested 50 for its model but in later discussion we agreed that this was for some aspects of Store's model implementation, not for its model which is mostly OK. It would also score higher relative to Envy if merging tools were considered.

Vendors, VMs and IDEs

VAST Smalltalk Transition Strategy, Greg Boniatides, IBM

(Also read Eric Clayberg's talk 'VA Smalltalk Going Forward' below.)

IBM has announced end-of-service for VA Smalltalk Enterprise in April 2006. They will release version 6.0.3 (bugfix) and that will be all ('stabilised' is the corporate term). They've been in this technology for 12 years and gathered an impressive list of customers but IBM has chosen to focus investment on its strategic offerings. (Also talk of 'where the market is investing its money' but see my earlier reports on the disconnect between income earned by VAST and the revenue stream it was allowed to receive.)

Support extensions can be purchased from IBM, or you can port over the next 6 to 18 months to Java or similar (including .Net which Greg says they will support interoperability with, clarified by a Bruce Badger question, i.e. they will tolerate it though they are hardly pushing it; Java J2EE and Eclipse are rather what they would have you use). Greg knows some VAST systems will be running long after everyone in this room has retired. IBM offer the accelerated transformation path, in which you can port in haste (and repent at leisure: Niall :-) or the staged transformation path in which

you port more slowly. Or you can stay in Smalltalk with Instantiations.

IBM IntegratedTechnologyServices front-end the extended support (i.e. do the paperwork). Support will be done by the SWG Lab (John O'Keefe). Synchrony systems does the accelerated transformation. CSC Solution Services have a Smalltalk background and focus on managing large transformations. Last but very far from least, Instantiations (Eric Clayberg, Mike Taylor) is the key player for VAST follow-on (i.e. continuance).

Synchrony claim 40+ successful Smalltalk migrations. (I found myself wondering how many of these were Smalltalk-to-Smalltalk or non-Smalltalk-to-Smalltalk. How many successful ports *from* Smalltalk *to* a statically-typed language have they done and how large were they? Jim told me that only two such are mentioned on their web site, the latest in '95)

Q. What constraints on Instantiations? There are indeed constraints; they cannot sell some proprietary technology. They cannot sell VW Envy because they do not have it, only VA Envy. They cannot sell their VAST business entirely to MS or to any third company. There are no time constraints. What they can sell they can sell and maintain forever. (Mike spoke of their easily having two decades of Smalltalk clearly visible.)

Greg closed by saying that it had been hard for him to come back to the team to do this job but it was what he had been brought in to do; he was recruited for this job, he did not seek it. He has had some colourful emails. Bruce sympathised with what he saw as a ghastly job, as did the audience.

John O'Keefe is very happy that Instantiations has taken over the code base and will be building follow-on products. As far as Greg and John are concerned, if Instantiations do manage to make more of VAST than IBM's strategic assessment suggests, well, good for them.

Greg also mentioned the number of companies IBM is acquiring: Rational, Lotus and so on, and that Smalltalk just does not fit with this portfolio. I would agree, though I might say it with a slightly different emphasis. :-)

Q. Internal IBM use of Smalltalk? There are still pockets of Smalltalk use but generally IBM is moving away from it. They wish to consolidate their portfolio; no more pockets and one-offs, the trend is convergence. Greg believes that Smalltalk is a very productive environment. The Synchrony folks have worked in both environments and they are Smalltalk bigots deep down, but Greg must accept where IBM is going.

Eric Clayberg closed by thanking John and Greg for what they have done. It would have been easier for their careers just to let things sunset. Both spent some of their IBM capital to create the ways forward that now exist.

VA Smalltalk Going Forward, Eric Clayberg of Instantiations and John O'Keefe of IBM

John has been the IBM VAST team lead for years and has been on the project from before v1.0. He will talk quickly about what happened and

will happen in IBM so that Eric can get onto the interesting stuff. John's initial talk recapitulated Greg's: end-of-service for the enterprise product (N.B. not for the server product). Smalltalk's problem is that is doesn't fit with IBM's Rational - Eclipse - Java strategy. (Niall: IBM strategy problem, rather. :-) Many other ill-fitting products have been sunsetted already and now it is Smalltalk's turn.

If you want to leave Smalltalk, you can go the staged transformation or accelerated transformation route. John heard Bruce say last night that "accelerated transformation" sounds like what happens to crash test dummies when they collide with a wall. Staged transformation is offered by Synchrony; John mentioned that some of the Synchrony tools are also quite useful if you want to stay in Smalltalk and reengineer your system.

If you stay in Smalltalk (John's recommendation, though not IBM's) then you can buy extended release-6 support from IBM or you can go to VASmalltalk from Instantiations, who will be providing future releases.

Eric is a Senior Vice-President and general manager of Instantiations Smalltalk Products Division. He played a Nasa tape that started with, "The missile knows where it is because it knows where it isn't", and developed by inexorable logic to a discussion of, "The missile subtracts where it should be from where it wasn't." as an example of what 'rocket science' could sound like. :-)

Eric first saw Smalltalk in 1986 and started full-time work in it in 1991, in the original ObjectShare company, developing WindowBuilder Pro. In the Parc-Place/Digitalk days he was development manager for both VSE and VW at one point. Now, with VASmalltalk, he will have been development manager for three major dialects. When Parc-Place began its decline, he and others spun themselves out into Instantiations.

They have been an IBM Advanced Business partner for a long time and supply many VA products: VAAssist, widget kits, and WindowBuilder. They also have some Eclipse-targeted products, Java GUI builder, etc. VAAssist won IBM's' best add-on tool' award two years ago. IBM have put Instantiations in their staged transformation strategy, which they are of course well qualified to do for people who want that, but that is not what they are seeking. They are very happy if their customers stay in Smalltalk forever. They will help, but not encourage, anyone who wants to port. Their goal is to greatly lengthen the life of VAST applications. The new product is called VASmalltalk. The first release will be 100% VAST plus their utilities (and will include VisualAge for Java if I heard aright; I know that Sandstone and others find VAJ excellent for 99% Smalltalk products that want a thin Java skin at some points).

IBM thinks modernisation means moving to Java. They think that modernisation means moving the VA IDE (a very 90's IDE) forward. Their team is as large as the IBM team was and many have 15 years of Smalltalk experience; Instantiations may be smaller than IBM overall but they are well able to drive VASmalltalk forward.

The purpose of their first release, 7.0, is to re-establish the brand and put an end to the 'dying' impression. Release is planned for July. It will not make major changes in the product, but it will include many integrated add-on utilities previously offered separately. VAAssist adds some 500 enhancements to the tool-bar. WidgetKitControls gives enhanced versions of some 20 widgets: graphical buttons, etc. GF/ST is a 2d and 3d graphics framework, very portable across ST dialects. Etc.

In version 7.5 they will

- finish ANSI support
- enhance the IDE (IBM team has ideas they will now feed in)
- include an improved Envy/QA
- more integrated Refactoring Browser and its Envy extensions
- support newer native Windows widgets
- start enhancing VA web services; they were powerful but have had little work in the last 3-4 years so they will look at newer things such as security
- include any Synchrony-contributed migration tools

In Version 8.0, they are thinking of supporting Longhorn, enhancing web services, perhaps namespace support and 64-bit VM support.

They will be including 'migration-readiness' demos from Synchrony so you can see what they can do. As John remarked, they have done Smalltalk-to-Smalltalk conversions in the past and their tools are powerful. These tools can help you restructure your applications so don't ignore them as only relevant if you were going to port.

Eric listed costs for new and upgrade licences (see his slide and/or website). He mentioned the competitive upgrade; for $1995 you can upgrade from any commercial Smalltalk dialect. (With my UK Smalltalk User Group hat on, I will add that if you are in the UK, David Pennington of Totally Objects is a good reseller to talk to for anyone seeking to upgrade with local support, etc.)

Eric demoed the VASmalltalk tools that will be in 7.0. You have customisable menus, history helpers, dynamic colour syntax highlighting (Bruce, from back, "Can you also control text size? If so, please increase it." Eric could and did), context-sensitive menus (e.g. click on a selector to get implementors / senders menu for it, corrected spelling choices for unknown selector, etc.), icons in navigator, shortcuts (e.g. select line and comment it out), and many other accelerators.

MultiBrowser synchronisation is now there; two browsers on same method will no longer risk overwriting each other. A three-pane implementors / senders browser navigates chains of calls and supports multi-select. Colour highlighting in the navigator lets you see e.g. which classes and methods are scratched. Drag-drop in the navigator is there and does all that you would expect and maybe more (drag method onto text window dumps its

text into that window). The debug-through-blocks add-on is now standard in the product. Click to bring up inspectors on visual elements; inspectors are enhanced with colour and icons to classify instVars. Text windows remember past entries and do autocomplete. Etc.

Envy tools have been enhanced with icons, colouring and filters. Versions can be renamed according to criteria (e.g. rename all after date X). You can customise your versioning template. User management used to be painful and is now easier: you can add a user to all apps in selected config maps, replace user as owner of classes, apps, etc., by new user. A super-user function finally lets you stop overriding hidden password methods and then changing who you are twenty times to do straightforward sensible things.

Prerequisite path tracing commands find all paths not just one and help you prune your trees. Menus for allInstances of a class, or of all classes in an app, speed unloading things. It supports unimplemented refs search, string search, and searching what config maps use app, what prereq config maps have shared apps, etc. (At this point Eric's microphone stopped working, so he relied on speaking loudly. :-) A good change-list-like tool lets you see the list of methods you've visited and changed, and browse any of them. You can find all classes a user controls in a list of apps.

Eric then showed some of the UI building stuff, morphing widgets in the UI builder to see what UI design would be best, easier window-resize-behaviour setting for widgets, new kinds of widgets, and much else. He also showed GF/ST 2D and 3D demos. (At this point his microphone started working again. :-)

Q. (Michael) do you want goodies? Yes. They're delighted to talk to people about goodies and also willing to talk about things that need to be in the base, not for money but maybe for cheaper/free licences.

Q. Envy for VisualWorks? Our IBM deal gives us full source for all Envy VA related stuff. We do not have the VW Envy base. (If anyone ever has a licence to use that, it might begin to be possible.)

Q. (Niall) Envy derivation tracking? There is a GF/ST graphical tool and also some stuff in TrailBlazer.

Q. (David Pennington) Macintosh? We don't think it's that easy. If anyone does, we're happy to help them and we would be ready to do it if the market were 200 licences or more but at the moment our customers do not indicate anything like that. (Question led to discussion of many non-corporate users who would use it and general question of corporate / non-corporate focus. Instantiations are readier than IBM to look at that space and to hear ideas.)

Donald McQueen thanked Eric and Mike and John for making this happen.

Preserving the Reflective Illusion in an Embedded Smalltalk System, Christian Hansen, Esmertec

Esmertec provides software for mobile phones and embedded devices

(often on Java VMs). In Denmark, a small part of the company is using OSVM. Standard embedded device development is stone-age (insert printf statements and similar). Many embedded systems are not supposed to shut down. They want to catch errors and fix them while the system is running; Smalltalk suits such implementations.

In traditional Smalltalk systems, development and runtime environments are merged. They want to fit their runtime in 100k. Hence they split these: development runs on an IDE machine talking over a serial line or network to the OSVM Runtime on the device. The IDE drives all interactions. They reduce footprint by leaving out everything that running does not need (e.g. the names of variables), by compressing the memory and image contents, and by putting some stuff in ROM (same space but cheaper than RAM).

They also omit classes that are not used and methods that are never called. They therefore have to work out what is not used, and how to give programmers the illusion that it is still there. They find what classes they need by tracing from a set of root classes. They drop methods that are not called (but do not consider the receiver; they may use advanced type inference in future). The OSVM runtime does not allow dynamic loading of classes. Instead, they have on-demand code uploading from the IDE, including evaluation support and debugging. This changes perform: to use obj perform: ##someSelector where these selector values can be restricted.

(Bill Shwaab mention Dolphin's restriction approach which he has just been studying. Christian has not looked at it, but will do now do so.)

He unpacked (literally) his demo device and connected it to his laptop running the IDE. He then evaluated some Transcript printString code (already on device). In this version, a debugger is running on the device all the time. The next version will make this startable. The code that interacts with it is uploaded on demand (and GCed again when unused). He next made the transcript code repeat in a thread, which needed code upload.

The image is on the device. The image descriptor has var names, selectors and class names for debug (so e.g. instance of 71 field [2 3] is translated to more meaningful text), and source code.

They use RAM to allocate objects, running the image directly from ROM, with debugging and code updating still working OK. Changing a class (e.g. adding a method) in RAM would be easy. With the class in ROM, they have a table of class indexes in RAM (which instances use anyway to find their classes). A shallow copy of the class in RAM can hold pointers to all the ROM methods and have a new method added to it.

Q. (Colin) Do you keep a reference to the ROM version of class (i.e. not just to the ROM methods) so if you unload your code changes you can revert to referring to the old class? We have still a reference so could and should revert but do not do so today.

Q. (Donald) Put changes into ROM? You can capture changes to apply

them all at start-up but the ROM once flashed never changes.

Q. (Donald) Use? They have projects with small devices. Where people have the resources they stick with Java. OSVM gets used where Java is too large. Example: a device with an antennae to network GSM connection. It is a simple thing but you need enough logic to handle the telephone statics.

Q. (Colin) Any changes since OSVM acquired by Esmertec? We're focusing on where esmertec has not already got solutions (e.g. they already have Java VMs on phones so are not looking to redo them).

See http://www.esmertec.com/solutions/M2M/OSVM/ for more info.

Smalltalk JVM, Allen Davis, KSC
The first version of Smalltalk JVM was done in 2000, when 50% of corporate expenditures were on IT. That paradise is lost and unlikely to return. They are now releasing it to the community.

In Java, libraries and VMs are split. The idea is that you can use a VM from from one vendor and libraries from various others. Smalltalk/JVM produces 100% Java byte code. This makes it possible to have different libraries from different vendors coexisting in the appropriate namespaces.

By 2001, most of what he presented was already there.

- Compile Smalltalk classes to Java Byte code
- Ability to subclasses Smalltalk class from Java class and vice versa
- call Smalltalk code from Java code
- full support for blocks
- servlets, applets, etc.

There are many problems in representing Smalltalk in Java. Java classes are not first-class objects. The same code will not run as a static method and as an instance methods: polymorphism is completely broken in Java on the class side. So they went back to the ST-80 book and studied how the Object subclass: #Class pattern works there, then replicated it in Java. A Smalltalk class in Smalltalk/JVM is an instance of Class, not a Java class. Thus they have created the entire Smalltalk class hierarchy as instances.

He looked at how printString was implemented in Object. It calls printOn: which subclasses override. When an instance of A is sent printString, it works the same in Java and ST. Suppose we had a printString class method and sent printString to the class object A. In Java, Object class>>printOn: would be found, not A>>printOn:, since the static method references are hardcoded. In Smalltalk/JVM, Java's `new MyClass` becomes Smalltalk's `MyClass basicNew` and `Smalltalk.MyClass newInstance()` becomes `MyClass new`. Here, the message is sent to the instance MyClass, not to the Java class MyClass. And they have a global lookup `Smalltalkat("MyClass")` to implement `Smalltalk at: #MyClass`

(The audience asked questions on various issues) Blocks are a problem in

Java. One ugly solution is inner classes, and they do use it. Assigning to local variables in an inner class is a known problem. They create a context object that holds all the variables and pass that it; the context object can't be changed but its values can.

A less ugly approach is to view a block as an unnamed method call ('the method with no name' is what I call a block in my notes on 'Smalltalk for Java programmers') so they define methods for the class and pass it the block arguments and the context object (as described above) holding the local variables. Because the method is defined in the same class as the block, it has the extended scope required.

Their final approach is to optimise out blocks. This is what they do most often; do:, select:, reject:, ... get inlined (which is not so unlike what many Smalltalks do as part of their optimisation process). By default, every block is optimised that can be, but you can suppress this.

Type constraints are another problem. They have a mechanism for giving typeless behaviour, not having to do method lookups, exploiting the lookup optimisation mechanism described in the first edition of the Java book (details removed in the second edition :-). Object puts method #abc in slot24 in the method table so every subclass puts #abc in slot24 so no need for lookup - just call method 24. (Allen remarked that Java is very mechanical in how it makes decisions. Engineering decisions bleed through to be visible in the spec.) Suppose A and B have a method #xyz, that is not on Object (or on any common superclass); now you have the lookup problem. They therefore just filter the methods up by generating a common ancestor for every single method, with implementation (you guessed it) DNU. Now every method has its assigned slot and the type lookup problem has vanished (and Object and other high-level classes have vast numbers of synthetic methods, hidden in your browser unless you check the option to see them).

(In theory, not every JVM need provide this mechanism, e.g. a cellphone JVM might do a slower but smaller footprint method lookup. They know of no exceptions.)

You could define a common interface for e.g. #xyz. This resurfaces the lookup and so interfaces are slower in Java (though it is never admitted). You can double your performance by changing types from interfaces to classes. (They have not found a JVM where you cannot re-evaluate the method table to get back to the common implementors.)

They have looked at systems with 60,000 methods. The overhead is small compared to what else such systems are doing; they see no problem.

Q. (Niall) Dynamic loading to change behaviour of existing instances (e.g. add default colour method to domain class with existing instances, then upload code that assumes all instances know their colour, then upload colour-choosing code, thus adding a new feature without ever stopping the application)? Dynamic loading in general is not something they support..

He thought maybe WebSphere had a means to do what I describe but was unsure and if it does it is very WebSphere-specific, not general to Java.

Q. Persistent state across runs? Java does not have this. This is irritating when testing. They use serialisation to get the state back but it can remain a problem; crash in a test and you cannot see the state that caused the crash.

Q. Perform:? Many Smalltalks will optimise a perform: #selector to just calling that selector and they can do that. For true perform: they use reflection to get the class' hierarchy's selectors and they cache the selectors.

Q. (Eliot) Exception semantics? They run on top of Java exception handling, which is unlike VW but not so unlike VA. They generate the appropriate Java (try-catch) code to let exception `on:do:` percolate back but you now do not have the stack, you can only pass data back, so they implement it with block behaviour and non-local returns. A block with a return is a return to the method where it was defined not where it was invoked. Hence you can't do Seaside in Smalltalk/JVM, just as you can't do it in VA.

Testing

Cooking with SUnit or How to Burn Water in One Easy Lesson, Sames Shuster, Cincom

Sames only found out a month ago that he was giving this talk. :-)

It all started with Kent Beck's article in the Smalltalk Report of October 1984 about three classes: TestCase, TestResult, TestSuite. Kent did not have the philosophy all at once (Sames was lucky enough to work on the second ever XP project - Ford, VCAP project - and could see this) but it was very Kent Beck-ish to create a three class framework. A test is your bodyguard, the last rampart between you and your customer.

The `should: aBlock` methods were because it was thought the error handling might be on what you were executing, not on the framework, so a block guaranteed late binding. Sames changed that. Now we have `should:raise:`, to support that idea, and `assert:/deny:` for the rest.

Because the framework was presented as an academic manufacturing idea, people did not get what a unit test was. By the first Camp Smalltalk in 2000, there were many styles and several dialect-specific implementations plus a simple UI, TestRunner. Sames had then used it for two years. His bosses had a hard time understanding that there was no point capturing his unit test result ratios as he never delivered code until it passes 100%; that was a strange idea in those days.

Sames and Ron Jeffries paired to produce a release (2.6) for VA, VW, VSE, Squeak, ObjectStudio, Smalltalk/X, Smalltalk MT and Dolphin, including a TestRunner that looked exactly the same on *all* platforms. It still used dialect-specific idioms. 2.7 created SUnitPreload to eliminate these by encapsulating an abstract API. This meant that all your tests, unless calling

dialect-specific code, were themselves dialect-neutral. The committee for SUnit was Sames, Jeff O'Dell and Joseph Pelrine. 2.8 added GemStone (no TestRunner), GNU Smalltalk and handled VW namespace and Store issues.

3.0 added TestResource. 3.1 added `*:description:`, etc., for better UI and documentation, `*:resumable:` to capture failures but keep executing the test (uses ResumableTestFailure). It also added failure logging (typically one logs to the Transcript). 3.1 is the current latest release.

Sames stressed that users should not touch the basic SUnit implementation. Subclass these classes, extend them but don't touch them. Changes to the SUnit base should be made by, or at least in full liaison with, the SUnit team.

Sames philosophy (with which, he remarked, anyone can disagree when it's their turn to give this talk :-) is that SUnit does *unit* tests. It is not a framework for functional tests, user acceptance tests or a UI for running tests. SUnit is public domain, copyright Kent Beck; Camp Smalltalk (currently Joseph) is the shepherd. (BTW Joseph has invited me, Niall, to become an SUnit shepherd; I'm happy to receive SUnit proposals.)

Kent said a unit test was a predictable reaction of a target object responding to a specific stimulus. By representing each predictable situation as an object, no two tests will ever interfere. Tests should not depend on each other so should not be repeatable. Sames therefore advocates TestRunner holding the tests it runs in Sets to avoid order-dependent-effects.

This prompted discussion. Colin dislikes Sets because they are at once not predictable enough and too predictable. Tests should not interfere (we all agree on that) but if they do, you want to rerun in the same order to find and fix the dependence, not see an error you then cannot repeat and don't know the order of. On the other hand, Sets are *too* predictable. They do not in fact offer much guarantee that order is being permuted from run to run.

I agreed; I think the UI should run in the same order and have another user-invoked action that explicitly runs all tests in all possible orders (or some optimised subset). TestSuites (quite correctly) use OrderedCollections so many ways of running tests do in fact result in fixed orders, and the original (pre-TestRunner) framework always would. There was much further discussion: generally, John McIntosh, Colin and I all felt that the aim was correct but a Set implementation was the wrong way to get it.

Sames then discussed test classification. To Sames,

- a Functional Test can combine multiple units of work, be defined as not repeatable, may have preconditions for success, can interact with changing resources and may have explicit test orders.
- an Acceptance Test is a written script of behaviours leading to a non-exact definition of success.

Here historical testers reappear.

How to use SUnit. (It's easier to say how not to use it):

- you don't test accessors, mutators, that vars are initialized to nil, etc.
- unit tests are white box; you can look at internals, you can use reflection (e.g. you might explicitly test that a selector is absent, e.g. 'this model does *not* respond to value)

Sames has tests that test the existence of classes and subclasses. Pollock is a framework and so the continued existence of the interface classes that programmers are using is part of the contract offered by Pollock. An existence test on a class is the best way to assert that this class name is part of the public framework, that users' code will reference it, so it cannot be refactored without imposing documentation and conversion tool costs.

In Pollock, you cannot get your hands on the scroll bar of a window directly, only its decoration. Sames does not give users accessors but he has tests that walk to it. The tests show that it works, i.e. that users can use the interface provided without ever needing to get at the scroll bars directly.

Q. (Niall) put whitebox accessors in test-only class extensions (I use these with method and/or protocol name making their test-only nature obvious)? Sames is not keen. His tests are always loaded; he'd rather use reflection.

Some people argue to add test methods on the tested class, which Sames totally rejects. Modifying your class to test itself could distort your class.

Example: rating an insurance policy. The rating is executed on a legacy server. The data is collected with client Smalltalk. Executing a rating changes the state of the legacy server. To re-execute the same rating, you must reset the server, a human-actioned formal procedure. In this case, simple tests study domain objects. Medium tests look at the UI, entering data, widget state. Hard tests do the rating. You could not actually unit-test through to the legacy server because that's not automatic or repeatable. Hence you test if the data was correct and was (would be) correctly sent down the wire and, if fed with captured data back from a pseudo-server, has the correct effect.

When handling this case in a system he worked on, Sames wrote a tool that captured the data sent to the server and the data returned. He ran a functional test from time to time to capture this data. His tool grabbed it and stored it for use in the unit tests.

There was discussion of TestResources. There was general agreement that the key issue is not basic idea - people get that - but the magic that makes resources work. (Niall: see the SUnit BoF write up on pages 25 - 28 of http://wiki.eranova.si/esug/nfrESUG2003reportPublic. Joseph has written a draft TestResource manual which will be reviewed at ESUG2005.)

Sames recommends always nilling out your tests instvars in tearDown because you should not make assumptions about how the test runner will run your tests. He (almost) always does so. We discussed two possible

exceptions:

- I once had to code tests for a meta-data system where TestCases needed some key instvars to identify the meta-data configuration that each test instance was running. I therefore recommend writing such nilling code so that TestCase subclasses can exempt selected instVars, and have an example pattern. From experience, I'm sure this is needed when testing meta-data systems.
- I've also encountered a pattern that nilled only if the test completed correctly; its coder's idea was that users want to inspect the values of a failed test. However this makes a rerun of a failing test not an exact repeat of the initial run, conflicting with the repeatability criterion. If it did affect the test's behaviour, that could be *very* confusing to users.

Do you want to be proud of your work? Do you want to have pride in your code? Write unit tests!! You do not want someone to come to you and say, 'Your code is broken'. Your unit tests will protect your code from you and from others. The other great thing tests do is let you refactor.

Seaside testing, David Shaeffer
(I missed David's talk but he gave me a quick version during a lunch break. He will be at ESUG, where I hope to catch the full version. Meanwhile be aware this talk was noted in great haste and may be poorly written up.)

David's tests run on the server side, so have access to live instances of components being rendered, to the debugger, etc. His framework is a web test runner. It is available in Squeak and in VW. He had to tag some Seaside code better in order to test it, since tests need tags to find the things they are checking. This is why his framework overrides some Seaside methods.

David provides a web browser simulator which can e.g. simulate a click on anchor, tracking it by CSS. If there is no CSS key in the component, he has to dig in via e.g. ... `elementNamed: hr` ... . SCComponentTestCase is the root class. Subclass it and write tests such as

```
testMyComponent
  self newApplicationWithRootClass: MyComponentClass.
  self establishSession.
  self followAnchor: (self lastResponse anchor...
  self assert: (self componentAnswered: ... ).
  ...
```

`lastResponse` returns a SeasideResponse wrapper of the XML DOM tree. The wrapper has convenience methods for pulling things out of the tree, e.g. `anchors`, `anchorWithId:` (least brittle), `anchorWithLabel:`. These return wrapped XML elements typed in buttons or text areas.

Seaside's use of continuations, implemented by interactions in the call stack, created an issue he had to solve. The normal way of writing tests is to create a component instance, put it in the state you want, then invoke framework calls on it (e.g. create a calendar instance, select a date that suits your test, then have Seaside display it). The problem is that Seaside components *must* have the appropriate Seaside framework calls in their call

stack but your test case, as constructed above, has not got there yet. The solution is to set up your component in a block which you provide to the appropriate call in his framework,

```
testMySeasideComponent
  ...
  self
    newApplicationWithRootClass: aComponentClass
    initializeWith: [:myComponent | “do set up”].
  ...
```

He provides a fourth halo, appropriately named ‘test case,’ to invoke tests. Note that this runs a test on its own component instance that it sets up, not on the particular instance of the component class that you were viewing when you clicked its fourth halo to invoke a test of that class of component.

His test runner has some extra features over the Squeak one. It keeps a history of the last page downloaded with which you can interact to study what went wrong. What you cannot see is what Seaside submitted.

His talk uses a FormDemo example, in which he tags all the input elements, then tests. The test sets data, simulates button presses, etc., in much the same style as one uses when writing Smalltalk UI tests.

What are you testing:

- what’s being painted, what component thinks going on (state), and the component API (does it answer when you call?)
- components have hooks (blocks to evaluate) to test callbacks
- internal state: ask for last component and look at it (example of WACounter test)
- Do `back` and see backtracking works (Browsers can do different things on back. Some always get it from the cache, some check page meta-data to see what to do. He intends to implement subclasses of test browser to replicate these possibilities.)
- check answers (What was the answer? Would it crash if the answer was fail?)

You just call `componentAnswered` if you only care whether it did. You can also mark tests for visual inspection. This spits out flat files that can be opened in your browser later and eyeballed.

David also has slides on the other free frameworks. SmallHttpUnit runs outside the server but is better than his at pulling apart the web page. Selenium is a Java testing framework you could use for checking the behaviour of any Java script you use. If you are always on server then. as Seaside builds the DOM before rendering it, you could just check that instead of waiting till it is HTMLed, when you need tags to find elements. (There are a few exception cases but you could let SmallHttpUnit do them.)

SUnit BoF, Niall Ross and Michael Lucas-Smith
I discussed two of my SUnit patterns:

- capturing defects raised in subthreads spawned by a test
- grouping tests whose resources cannot be enabled simultaneously into test suites

Implementations of both patterns are in SUnitUtilities in the Cincom Open Repository. I will port the first to VASmalltalk; it can be ported to any dialect that allows method wrappers. The second should run with minimal code changes in any dialect.

SUnitXProcPatterns: either subclassing from CrossProcessTestCase or cloning its single method to your test causes the test to wrap the method `forBlock:priority:` while running the test so that blocks spawned (directly or indirectly) by the test process (but not those spawned by other processes) run within same test handler that the main SUnit thread is using. Thus during a run (but not during a debug) any errors raised will be trapped, just as if they were raised by the main thread. The most serious outcome (error beats fail beats pass) is added to the main thread result.

My implementation names test-spawned processes to track which ones need their errors trapped. Michael suggested holding the processes in a collection; it's less intrusive (and WithStyle also names processes).

My `waitTillSubThreadsComplete` implementation could doubtless be improved. It would be easier if Semaphore supported a (thread-safe) `procrastinate` method, inverse to `signal`, that decreased the signal count and so let one semaphore wait on the completion of several processes by allowing negative values of the signal count. Michael said he thought VAST semaphores already did this. I will extend the pattern to let specific test cases override the generic wait behaviour (e.g. with a fixed delay). [Post-BoF, Michael found that the code does not handle deferred blocks (i.e. DeferrableActions). I will work on this before and at ESUG/CS10.]

SUnitResourcePatterns: it is not uncommon that the test resources for a system cannot all be available at the same time. For example, developers working on a system that can only be logged in to one database at a time may naturally write tests that run against different databases. If each test uses a resource to login to the appropriate database, these tests cannot be combined into an overall XP suite without rework.

The Competing Resource pattern provides a minimal-cost solution, useful when using XP in a tight-deadline environment (is there another kind :-). If the competing resources can be re-parented under an abstract subclass of CompetingResource, suites using them will run correctly.

Currently, the pattern makes no attempt to optimise the switching between resources; users should construct the suite in a sensible order so that tests using the various competing resources occur in groups. This is usually very natural to do.

The pattern makes a minimal change, which I see as an improvement, to the basic SUnit code. Tests now check, not that their resources are available (that is still done by suites) but that their resource's singleton instances are not nil. This protects against the possibility that other tests in a suite, while manipulating resources (as can sometimes be needed), have not accidentally reset them.

Both patterns are in the Cincom Open Repository as packages within the SUnitUtilities bundle. Comments are welcome. They will be reviewed and developed further at Camp Smalltalk 10 in Brussels. If they win approval, I will offer other dialects' versions to the SUnit SourceForge site. (Anyone who wants them earlier, by all means port and send me your code.)

Other discussions: as I was chairing the BoF, my notes on what else was said were brief.

Three of us knew systems where the Smalllint checks have been called from tests and so made part of the system's overall XP suite. I remarked that some people use the rewrite framework to encode their project's design patterns as tests (there is an example in the custom refactoring project download: customrefactor.sourceforge.net). There was discussion of writing tests to prevent calling obsolete code or patterns,

Someone mentioned that the debugger can bring up lint rules and you can choose which rules are run. Having tests send emails when errors occur is common and someone has arranged for emails to be sent when errors are raised while running tests even if in a spawned process, as another way of addressing this issue.

There was discussion about how to address windows and other widgets in tests. I use method wrappers, following John Brant (there are examples in the custom refactoring tests). Others have various approaches. It would be useful if patterns that people found useful were added to the Open Repository or sent to the SUnit SourceForge site.

Remote Computing

OpenTalk at Large, Martin Kobetic, Cincom

Martin works on networking and security in Cincom. This talk will focus on the more exotic features of OpenTalk, things you won't find elsewhere. Martin will talk about multi-casting (IP multicasting and OpenTalk-groups). Then he will discuss grid computing.

Martin started by inviting everyone in the room with wireless to connect, browse a URL and download an installer (windows) or image and VM.

IP multicasting sends packets to a range of IP addresses. The multicast range is 224.0.0.0-239.255.255.255. The API is the usual UDP datagram sockets, differing only in that receiving sockets must join a specific multicast group (e.g. 224.3.67.119). Multicast can work with loopback on or off. Obviously, you rarely multicast outside your own network (it is an invitation to Denial of Service attacks). However it can be done using the

IGMP routing protocol: ‘if you see a packet for here send it there as well’.

Sending is standard: he showed the code at the protocol level.

```
SocketAccessor family: SocketAccessor AF_INET type: ...
group := IPSocketAddress hostAddress: #[...
sender sendTo: group buffer: ‘Hello’ asWriteArray.
```

Receiving is where multicast differs

```
(SocketAccessor family: ... type: ...)
  soReuseAddress: true;
  bindTo: ...;
  join: (IPMulticastAddress ...)
  ...
```

The rest is as usual.

OpenTalk makes this easier by offering object groups. Brokers running on the same port can join the same multicast address. Receivers are exported under the same OID. You get a group proxy #(mcast-address, port, OID), i.e. a standard OpenTalk proxy using the obvious values. A remote group a request is an STSTOnewayRequest. You could use a call back and you could send a block, noting that there are limits; you can only send clean blocks or passed by reference (so you could still send `value` to it).

```
... mcastAddress: #([224 5 6 7])
```

so if the brokers export under same OID then if Martin does

```
group := broker groupById: #group.
group show: ‘Hello World!’
```

‘Hello World!’ should now be seen on every image in the room. If the sender does not wish to participate in the group, you use a different API

```
... groupAt: id: ...
```

Martin then started the demo image, invoking OpenChat, a unicast utility. He told it where the server was running, started a client and sent a message. That was trivial. He then started a multicast example - same chat but two clients, no server - and replicated a common Cincom conversation (“When will the nav issue be resolved?” “I guess we’ll have to ask Jim.”). The unicast chat is effectively doing multicast things. Using multicast makes it simpler: people who want to talk to each other just join multicast groups.

Q. (Michael) filtering? When a host joins a multicast group, it gets all packets sent to that multicast group. The next layer of discrimination is the port number.

Thus multicast is very like unicast: oneway messages differing only in their targets. Multicast groups are tied to the object table. You can only have one export per object and there is the issue that exported objects could be garbage collected if their local fails to keep them.

Martin then talked about grid computing. When you have a complex task that can be broken down into subtasks, and you have distributed resources, then you have a grid computing opportunity. Martin’s demo was brute

force code breaking: attacking the RC4_40_MD5 crypto. His strategy was to select ranges randomly, search each range sequentially, looking for the string 'GET /' (likely to be present in target text). We all started our images and the grid drone (usual demo hiccough - locked up for half-a-minute). We inspected the drone and connected. (Needed to do `GenericProtocol setOS` or `GenericProtocol setHost`.)

When four people had connected successfully, Martin started the attack. A black square gradually filled with green dots tracking the ranges searched scattered within the overall space; 2^{12} keys to search in each range. A completion widget showed the total percentage searched. Martin started another drone on his machine, showing that it connected to the list and started working.

The Grid framework is quite simple. The controller is task specific. It maintains the grid and configures the drones. Drones are generic. They look for a controller when started. (While Martin was explaining this, the key was found by Georg Heeg's computer.)

OpentalkMatrix, Charles Monteiro
Charles runs the NYC Smalltalk User Group. He had worked for a p2p Java EAI company. Now he works for a finance company that uses Smalltalk, which he is much happier with. He planned to do two demos, one structured, the second a demo that everyone could join ('and we'll see what happens'). He configured three peers: himself, Martin and Alan (Knight) acting as the inbetween peer. Martin was to search for stuff Charles has, being brokered by Alan. (He showed this three years ago at OOPSLA.)

His demo encountered serious projector problems. Charles' machine did not like the auditorium's projector. Remembering my own experience at Frankfurt last year, I felt for Charles. He and Martin worked hard for ten minutes to sort it, finally being rewarded with slides on the screen. They used TypeVNC, an optimised variant of VNC (which he recommends) to get on-screen via Martin's machine.

Charles connected to AKnight. A configuration hiccough, maybe warming up firewalls, maybe the monitor issue, maybe wireless issues, slowed things. He rediscovered AKnight but it remained bizarrely slow to respond from Alan back to him (but worked at normal speed from Charles to Alan).

He searched for a component and found it. He switched to talking directly to Martin but still had problems. It became apparent that TypeVNC was having some impact that obstructed menu-handling and etc. Having got the component, he would have downloaded and run it. It worked when they checked it earlier. In the end, the demo had to be abandoned.

OpenTalkMatrix uses p2p to find things, including Smalltalk code. His demo would have shown finding a parcel in a peer-to-peer network (using keywords and comments), downloading it and reconfiguring the environment that the parcel needed to run (using config files and suchlike supporting resources). The idea is to make it easy to set up private peer-to-

peer networks, not be connected to some central server.

Q. (Michael) Similar to BitTorrent? Somewhat, noting he did not know much about BitTorrent.

You add components by 'adding' or putting them into your OTM folder. Searchers guard against cyclic and back-propagating queries. He is working on transmission security, with an obfuscate ST-ST protocol, wrapping connections in Stunnel, burying passwords within Smalltalk. It is on the web with a windows installer so we can download and try it.

You can compare this to Java WebStart, an HTML thing that lets you find a .jar file, using Java's ubiquitous VM. (Every Java shop he has worked at has always had to package the VM, so he's sceptical about its ubiquity. :-)

The demo problems meant he ran out of time. His slides note some features of peers in general and in his system specifically.

Miscellaneous

Serial Number Generation, Dan Antion, American Nuclear Insurers

They've used Smalltalk (VA) since the mid-90s. Recently, he had to provide serial number generation. He found an approach in the literature and implemented it. Their insurance company develops code for their own users and were not much concerned about who copied it. They reinsure world-wide and sometimes have visitors who ask for copies of the s/w. One user is a graphic artist developing rich media content around their software.

However they realised a need to track copies: brokers could use their software to replicate analysis. Hence they issued serial numbers to users and then generated a registration code (not used as an activation code at present; it could be) which the user receives.

First task is to decide who you're dealing with:

- casual sharing is fairly easy to stop
- hacking shareware and copying of limited time trials are harder to stop
- people who aim to crack and distribute the keys are not at all easy to stop

The response must be related to the level of threat. A $15 item not widely copied is little worth protecting but you may want to be able to step up protection if a product proves hot. They were only really looking to stop casual sharing, so wanted some hard-to-guess elements, plus transaction information (i.e. the sale, loan or whatever) and version information.

Dan uses a sequence of base 36 large primes, product and version codes. Base 36 has almost three-and-a-half million 5-character prime numbers. They cache them in a table and offset into it via the serial number using methods `fromTable:atOffset:`, `randomFromTable`, etc. The second prime number (4-char base 36) is generated on the fly from the serial number as there are only a few thousand of them (causes a brief delay

which is good for guess-frustration).

The product code also gives the acquisition method and release info. There is also a minor version number and a masked user and serial alphanumeric.

Lastly, you obfuscate the whole string by rearranging characters via e.g.

```
String obfuscate: 4 max: 13.
String deObfuscate: 4 max: 13.
```

The string was build modularly so pieces can be validated, modified. People who crack things tend to publish as soon as they think they've cracked so if various validations are spread through your program functions they may get caught. You can also let operations (e.g. 'don't offer this feature') be driven by what validates and what doesn't.

More could be done. 3+ million primes mapped into 1000+ customers lets you use many offsetting approaches. You can store fields in unusual places, e.g. in the structure elements of a field (hard for hacker to know what can be changed safely) or in graphic files. Delaying between each validation piece frustrates brute force attacks. Delaying before revealing that validation failed also helps; in fact, crackers are most humiliated by publishing that they cracked something and then being proved wrong.

There are very sophisticated crackers out there (Dan has an employee who connects him to this murky world). If they want it bad enough that they crack it, your product is good; it's a badge of honour (well, sort of :-). Many MS, Adobe, etc., techniques that annoy normal honest users are cracked by these people easily. This is worth being aware of: you don't want to annoy honest users. Dan had a racing game that asked you what the colour was of some random item on a page of the user manual. It was great protection (people copy games and maybe even photocopy manuals but how many will make a colour photocopy), but Dan is partially colour-blind; this is the only program he ever deliberately cracked.

Q. You use prime numbers. Would a table of random numbers be as useful? Large prime numbers are sparse in their range and Dan had code for them. If your random numbers were similarly sparse, perhaps they'd be as good.

Q. Hardware locks? Users dislike them and they add to the product cost.

OpenSkills, Bruce Badger
OpenSkills is trying to become a frictionless market people for skilled people. The SkillsBase is a CV database. It's the opposite of Monster in that you, the members control what of your CV is visible. It is free to search.

Being a small non-profit organisation means that it has approximately no money so it can't afford to build systems in Java. Java is an attractive choice for consulting companies because they get to charge for twice as long, as Bruce has seen when he worked for such companies.

The Wiki uses WikiWorks: it works very well, is reliable and is fast. All systems run in Linux machines in Docklands on UK, £15 / month with 64Mb of RAM. They make each machine single purpose, so any misconfigure loses only one service. They run GemStone on these machines and find it works fine.

The SkillsBase has a complex and changing data model (trying to reflect better and better what skills people have and want) and they hope that it will need to scale greatly. Development is distributed and the scarcest developer resource is time.

People connect to it through Squid, which unbundles https to http, a nice simple connection for their system to handle. They also cache all hits for ten seconds so if Mr Popular joins and gets scanned thousands of times then Squid gets hammered (which is fine, that's what its designed for) but their Smalltalk application only sees a load it can handle.

Bruce showed the SkillsBase code. They develop and test in VW and then move the app to GemStone, with no code changes. Bruce looked at the DB scripts, starting and stopping the systems. He then destroyed the database, and started again from a completely empty GemStone database. The VW generates a single huge string of GemStone code for portability (thus it's versioned in Store).

GemStone exceptions are pre-ANSI and unusual. Thus they have a buffer layer to make exceptions portable:

```
OSKExceptionContext>>for: aBlock on: ... do: ...
```

This is implemented simply in VW, less simply in GemStone. For the rest you just use `GBSM publish...` to get the VW classes and methods created in GemStone.

The membership system uses the same architecture but stays in VW because it needs to get to lots of places and has lots of external interfaces, which Bruce has to monitor as he's getting them working.

The reaction to Smalltalk of the free open-source community is very interesting. Bruce is starting to see more and more people going along Avi's route, getting into Smalltalk having already done Ruby, Python, etc., work. Bruce talked about the quiz of seeing something good in the Open Repository, then trying to work out which versions of dependent packages you should also get. He recommended more use of master bundles.

He expects to see compound growth of Smalltalk Free Open Source libraries: Seaside, Swazoo, Glorp, etc. He expects to see more use of GNU and Squeak too. Java and .Net got exited about Refactoring, ?Unit tests, etc., after they had been done in Smalltalk and demonstrated. Croquet and similar will be causing the same effect in a while.

Bruce wants more ANSI standard work, more standard library interfaces: Files, Sockets, Dates and Times. He also wants easier deployment.

Q. (Niall) Do you use Keyboard-Video-Mouse sharing? Yes, they do KVM for our development. Their skilled customers include doctors, etc., but they could reuse their experience for the computing customers.

Q. Contracts and tasks, not just skills? Some work done on that (someone did some Ruby work and has now realised he should be using Smalltalk).

Q. HR-XML? HR-XML is an XML schema defined by the HR-XML consortium covering resumes but also lots of other things that an HR department might do. They have an XML validator. They pull data that interests them from the HR-XML into objects where they can work on it and they also populate from other CV formats.

Q. How to manage the glossary of skills a person may have? Members enter skills and that is how the skills tree is populated. You then search for skills that match your string and then find members with those skills. Later, members choose skills from those known and can add new nodes in the tree for skills they have that are missing.

For caching to work nicely, you need every resource to have a distinct URI. Thus every member's skillset, customised as they wish, ends up as a URI which you can then email to a recruiter and they will see just what I want.

Talks I Missed

To make it easy to see whether I have reported on a talk or not, here is a list of all presentations that are not reported on above. Where I have reported relevant material elsewhere, it is noted (other reports mentioned are reachable from the whysmalltalk Events and Trip Reports page).

Talks:

- **Object-Relational Mapping, Alan Knight**: see my report of Alan's talk at the Cincom Users Group conference of December 2004.
- **The VisualWorks VM Plugin Framework, Sudhakar Krishnamachari**: see my report of Sudhakar's talk at the Cincom Users Group conference of December 2004.
- **Aggregating the Commentary, James Robertson**: I hope to report on this after I see James at ESUG later this year.
- **Complex Reports in VisualAge Smalltalk, Dan Antion and David Pennington**: Bob Nemec gave a talk at last year's Smalltalk Solutions on another reporting system that also uses David Pennington's utilities.
- **An Overview of the Cincom Smalltalk Business, Suzanne Fortman**: I hope to report on this after I see Suzanne at ESUG later this year.
- **Programming with Software Components: From Globals to URIs, Classes to Interfaces, Richard Staehli**
- **GemStone/S 64-bit strategy, Dan Ware and Norm Green**: I have brief remarks in the 'Discussions' section below.
- **Introduction to wxSqueak, Rob Gayvert**

- **Integration of Smalltalk Systems – E.g. ObjectStudio inside VisualWorks, Georg Heeg**: I describe a little of Georg's work in my ESUG 2004 report. I hope to learn more at ESUG 2005 later this year.
- **A Smalltalk-based system for dynamic multi-context information processing, Adriaan van Os and Eugene Westerhof**: my ESUG 2004 report describes the talk Adriaan gave in Kothen on this system.

Tutorials:

- **Smalltalk Garbage Collectors, John McIntosh**: see my reports of John's talks at last year's ESUG and Smalltalk Solutions conferences.
- **Web Services, Allen Davis**
- **Domain-Driven Design, Eric Evans and Ralph Johnson**

Other Events and Discussions

Smalltalk Coding Competition

The Smalltalk coding competition had two parts.

Competition: First Part

The first task, to be done over two days, was to write an application to generate questionnaires from text file specs, to present them on the web for users to complete, and to present stats on the results. For descriptions of how Michael and Blaine approached the problem, see their blog entries:

- http://www.cincomsmalltalk.com/userblogs/mls/blogView?showComments=true&entry=3293985879
- http://www.blainebuxton.com/weblog/2005/05/smalltalk-coding-contest-submission.html

I treated the competition as a chance to learn more about Seaside in VW.

I started by typing up some irreverent-stereotype specs for questionnaires to check if programmers thinking of moving to countries like Australia or Switzerland have the right skills and the right attitudes to fit in to their new societies. I created domain classes, then put my questionnaire spec files under `VersionedDirectory` management (from FileRepository utility) and started trying to parse them with SmaCC. I wasted time wondering how to match 'start of line' in SmaCC before realising that pushing a `cr` on the start of the file stream when handing it to SmaCC makes the issue go away. (BTW, how do you match just start-of-line in SmaCC if you don't know whether there's a preceding cr?) The parse was then simple, needing just a few iterations in SmaCC's UI to catch obvious mistake after obvious mistake. I wrote tests post-hoc to verify the parse; not quite test-*first*-design but SmaCC's UI is so interactive that using it is like TDD.

Now I had (pretty simple) questionnaire domain objects; over to Seaside. David Schaeffer's Seaside test framework was a godsend. He's solved problems that might have kept me puzzled till the bell rang. The main gotcha is that you can't just 'new' some component in your test setUp, assign it values and then carry on. The stuff Seaside does inside David's `establishSession` call has to be in the context stack above where you

assign those values. The solution is easy once you understand the problem: pass the assign code to the block in `initializeWith: aBlock` and it will be called within the right methods in the context stack. I tried to depart from his example style at one point, hit the problem and had an 'Aha' moment; 'That's why he does it that way.'

I knew that, with programmers who build web apps as their day job competing, I could not spend too much time on CSS and HTML choices. Watching me craft CSS is like watching paint dry; I'm just not that familiar with the various idioms. :-/) So I worked out the obvious analogy between choosing items to add to a shopping cart and choosing answers to add to a completed questionnaire, then starting hacking the Seaside shopping demo, which had styles already fitted to its components.

I then wasted an absurd amount of time looking for the Seaside component that would display a batched list of components, instead of a batched adaption of a list of components. It's not there (or I'm missing the obvious; let's not ignore that alternative explanation). It was easy to write when I accepted that I was never going to find it, but there was no getting back the time I spent looking for it.

Later than I should have, I took time to finalise my analogy into a map between all terms in the shopping and the questionnaire applications. I then used the rewrite framework to applied the map in a systematic rename of instVars, methods and classes. My code now read much more sensibly.

I built an abstract question view component, with concrete subclasses for single-answer and multiple-answer questions. Refactoring put almost all behaviour in the superclass. Questionnaire displayed but totalling code fell over, unsurprisingly as it still thought I was buying things. I changed the cash-totalling and item-number-counting code so that results were totals of questions answered, and the code knew answers can appear only once.

The shopping demo takes you from the list to an individual item view, thence to buying. By contrast, my users choose answers directly from the batched question component list. The predictable result of imitating the style of the shopping demo code was that, while the answer widget updated in the component when selected, the domain (and the 'answered questions' totals I display) only updated when the user moved to the next page of the questionnaire. This looked wrong (and might cause unintuitive results in some cases). Fixing it was not hard - I just had to make the rendered HTML nest correctly - but it was the kind of thing that someone who did web-apps for their daily living would have seen in a second.

What next? Oh, yes, the users may provide textual answers (irritating people, why can't they confine themselves to the choices provided! :-). Should I mutate the domain object or wrap it with a text holder? Should I wrap the 'Other' answer or the overall question? Users could select the 'Other' choice, then provide text, or just provide text, thus implying an 'Other' selection. Should the text field be visible in the main question view or should I call a new view with a text field when the user chooses 'other'?

Maybe I could show a small text field with the user able to see a large one by clicking an option. Maybe my question component could call a new one with a larger text field when the entered text length exceeds some value. Maybe I could finish this sometime next month. Spending too much time pursuing the perfect answer has always been something I'm good at. :-)

A side-effect from hacking the shopping demo was that, by reusing the old shopping item component, I could also very easily display, and select from, the set of all questionnaires. I again cloned shopping components, this time renaming to the new domain immediately, and made the demo two-level, so now I could create a logical group of questionnaires from a list, then complete my chosen group in a batch process, able to submit or abort the whole batch as a unit if I liked. Will this impress the judges as an extra feature or cost time I can't afford? As noted, I'm slow to hone CSS/HTML.

Reworking an existing app seemed like a good idea on the first evening and for model and abstract UI behaviour it worked well. However the second evening found me looking at the actual upfront web-layout that end-users would see: fine for a shopping app but counter-intuitive for a questionnaire. As I tried to make the end-user-look more in keeping with the app, the CSS seemed to be saying, "Well, if I were you I wouldn't start from here" (when he heard this, Michael said it was 'a novel way of building a web app' :-). After much time editing CSS, trying this and that style, it did not look that much better to my jaded eyes; nor did the clock. No real problem: it simply needed a larger refactor than my time-remaining/CSS-skill ratio allowed.

On to displaying result stats. I refactored the domain model so these could be found more easily. I gave the question component a subclass showing result totals instead of selection widgets against the answer texts. Then I 'create sibling' refactored them to abstract superclass and two subclasses (one concrete, the other the abstract superclass for question answering).

I looked at it. I thought about how Michael's could be expected to look, displaying in WithStyle, and about how any competing Seaside gurus' entries would look. I decided to place my bet on one of them. :-)

Competition: Second Part

The winners of the first part were Michael Lucas-Smith, Blaine Buxton and Andrei Sobchuck. Kevin Bradinger came fourth. Two of the winning entries were in VW, one in Squeak and Kevin's was in VisualAge.

Sadly, Andrei couldn't attend the conference so Michael, Blaine and Kevin squared off for the second round. In four hours (on Sunday evening before the conference) they had to write an app that played a game, posting moves via servlets that Alan provided. Players start with an equal set of coins (so many quarters, dimes, cents, ...). A player's move is to discard one coin from their hand into a central pool, then pick up as many coins as they like (from those available in the pool) to a total value that must be less than that of the coin they discarded. The winner is the last player with any coins left.

On Wednesday lunchtime, Alan switched on a projector displaying the

programs running against each other. As only Michael's program ran all the way through out of the box, Alan changed the rules to say that if a program raised a walkback during play, its creator was allowed one minute to fix and resume before forfeiting that game. (Michael, consulted beforehand, very decently agreed so as to make it a better contest.) As Alan remarked, it also made it a more Smalltalky contest if people could fix and resume while running, so this rule will probably be the norm in future years (but I recommend an addition so that people like Michael, whose programs don't need this, get a minute to tweak their code between games or some similar compensating advantage).

So with bated breath we wait and ... they're off!!! The smart money is on Michael. His program alone never needs to be resumed and it even displays his name in the right box! :-). Michael races into an early lead, beating Blaine and Kevin in the first round, but in the round two, Kevin's program stages a comeback, out-manoeuvring Michael's over when to play that key quarter coin. The punters begin to look worried. With Blaine out of the running, it's neck-and-neck between Michael's program and had-to-resume-but-still-hanging-in-there Kevin's. Alan brings forth his own program for a decider play-off. Kevin beats it both times. Michael beats it once but then - it can't be! - the chants of 'Mich - ael, Mich - ael' fade away in disbelief as Alan's program wins. And the bookies look happy as this surprise result leaves the crowd stunned. Outsider Kevin has come from behind to scrape a triumphant finish ...

Since Michael has to travel from Australia to attend a Camp Smalltalk, which he may not be able to do soon on top of conference attendance, he was actually happier with the second prize (an iPOD). Kevin, by contrast, lives closer to probable CS sites and was very happy to win the voucher.

As you may gather, a fun time was had by all at the first Smalltalk coding contest. See Jim Robertson's blog for the pictures

- http://www.cincomsmalltalk.com/blog/blogView?showComments=true&entry=3297504143

and for refs to Michael's and Blaine's blogs to read their take on things.

Discussions

Dolphin's Envy-like CM System, James Foster

The people who sell OmniBase also offer an Envy-like CM system for Dolphin. It needs only one system method override (of Compiler>>save).

There is a Project Editions Browser (like the config map browser). It has edition, version lifecycle, and tree display of history. Project editions contain package editions in load order (c.f. applications). There is no has no equivalent of SubApp or (more surprisingly) config-map pre-requisites (might be because Dolphin already has pre-req approaches, but one would like to see what the versioning impact on them is). There is a good shadow applications browser equivalent; it is easy to browse the contents of editions. You can browse method editions easily; all saves are captured.

Overall, it gives a fair amount of the same functionality but is by no means a clone of Envy, either in function or in nomenclature.

Dolphin out of the box has a directory tree structure and application (i.e,. package) browser. It does not define but calculates (in detail) prerequisites. Dolphin file-out is largely an import string but also has pre-reqs; these are not version-specific.

General

Michael: MS installer can open a port on your machine even if run with no admin permissions. XMLWithStyle update/restart works by creating a new image. The new image talks to the HTTP server image. If the server knows it is out of date, the new image recreates it and the old image dies. Variants of this general start pattern use the HTTP server to see if an image already exists, so the new image just hands over to it, or not, so continue start up.

Blaine works in Omaha in a group of 4 Smalltalkers running rings around a 25-strong Java group. (I thought Omaha was a state when I was young since I only knew the name from Omaha beach of WWII D-Day history where it is paired with Utah beach, and Utah is a state.)

Georg Heeg has ported ADM's ControlWORKS to the latest version of VW. Helped by this, they have just finished a huge project *ahead of schedule* (how often does that happen!) and so the Dresden site has won another major project. Meanwhile, whenever any Java and C++ code utilities they have cannot talk to each other due to minor protocol difficulties, they put Smalltalk between them as it can be quickly made to talk to both. Thus Smalltalk is becoming universal in the company.

DeutcheBahn has RUT-K (see my ESUG report) as the first page story in their latest report (before even the picture of their general manager :-). John McIntosh told me of their test facility: developers print lists and then sit in an office overlooking the Frankfurt station with binoculars, checking the trains that arrive and leave against the times and serial numbers on the list.

Energy trading systems seem to be common in Smalltalk. Adriaan van Os is doing further work on the system that Reinout demoed at StS2002. It now looks set to be used for cross-border trades in France, Belgium and Holland. Jay works in Atlanta for IntercontinentalExchange who also have such a system.

JWARS' budget has been cut (it is seen as lower priority than paying for the real war :-). It remains the only cross-service solution in its domain so Donald McQueen remains hopeful for the future.

James Foster is now working for GemStone in Testing. We will coordinate re SUnit in GemStone. 64-bit GemStone will have canonical dates, done as specials not reusing the pre-allocate oops approach currently used. I talked to him about the pattern that exploits GemStone's symbol canonicalisation to provide general canonicalisation of symbol-keyed objects.

Follow-up Actions

Tell Dan Poon about Christian Haider's good experience of porting from VSE to VW Pollock (written up in my ESUG 2004 report).

Conclusions

Since my keynote speech was on 'The Value of Smalltalk', it was really pleasant to hear several excellent commercial experience reports, each making clear points on exactly why Smalltalk had been valuable to them.

I particularly enjoyed the unusually literal significance of Jeff Hallman's talk title ('Making Money with Smalltalk'). Federal Reserve deposits ('high-powered money') control how commercial banks' increase the money supply. His Smalltalk systems literally manage the making of money, even more than if they controlled the presses that print the stuff.

The coding contest was fun. I don't know if we will have one every year but we should certain have more from time to time. The tasks were well chosen; this will be the hard part of any future contest.

Written by Niall Ross of eXtremeMetaProgrammers.

---

* End of Document *
