CS5 and ESUG, Douai, August 23th - August 30th, 2002

The accommodation was easy to find and not too long a walk from Douai railway station. Camp Smalltalk was in a convenient and commodious room on the ground floor of the accommodation, with a lecture room adjacent for impromptu talks, and had plenty of connections, hubs and plugs. Every accommodation room had a 2Mb connection. Thus it was the best-connected ESUG accommodation ever, even if in other respects it was value-for-money rather than luxurious. During Camp Smalltalk, enough fruit, snacks and coffee was provided that those keen enough (or on a diet) never had to leave save to sleep :-). For the less fanatical, a supermarket and restaurants were yards away. During ESUG, buses connected the ESUG venue to the accommodation and lunch venue. Not having lunch at the lecture site was a slight disadvantage, but the buses were well organised (and I only missed them once; fortunately, my pair programmer, Adriaan, inevitably missed them too, so he gave me a lift; chalk up one more advantage of pair programming :-)).

Breakfast on ESUG's first day introduced me to a new custom; drinking coffee out of bowls instead of cups. The seeming absence of milk was also disconcerting. On subsequent days, I found the milk dispenser and mastered the art of bowl-drinking.

Wednesday afternoon's social event, a tour of Lille, was doubtless great fun but Michel Tilman and I stayed behind in the CS room to work on the Argo meta-data framework; Lille's ancient buildings would have had to be fascinating indeed to compete. ESUG then treated us to a dinner, whereat I had fascinating discussions with Sames, Roel, Adriaan and Reinout. What I can remember is in the 'other discussions' section or added to relevant talks and projects.

Style

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

Author's Disclaimer and Acknowledgements

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

As there were two programme tracks and numerous additional discussions, I could not attend, still less report on, everything. I apologise to those whose talks are covered by a brief summary or 'missed it; see their slides'. Likewise, there were several Camp Smalltalk projects, and I spent my time there working, so my CS5 notes treat the project I worked on in much more detail than the others. Lastly, the talk descriptions were typed while I was trying to keep up with and understand what the speakers were saying, so

may well contain errors of fact or clarity. I apologise for any inaccuracies, and to those participants who are covered by the abbreviation ‘et al.’ because I failed to note down their names. (If anyone spots an omission or error, email me and corrections may be made.)

My thanks to those told me of things I missed, to Michael Prasse, Daniel Vainchensar, Aaron Hoffer, Adriaan van Os, Fransisco Garau, Michael Bateman et al. for working on my CS project, to Daniel Pfander and Fransisco Garau (again), and Dirk Roeleveld, who demoed and described other Smalltalk applications, and above all to the speakers whose work gave me something to report. Thanks also to the conference sponsors, Cincom, Georg Heeg, eXept Software AG and Roots, to Mines de Douai for hosting us, and to ESUG.

Summary of Projects and Talks

Camp Smalltalk 5 ran for three days, followed by ESUG for the next four. The consistent rise in numbers attending ESUG during the last three years was not maintained (attendance was a few less than last year) but I conjecture that this has no Smalltalk-specific implications and merely reflects partly the general current downturn and partly the fact that last year Essen was particularly well placed for the large German Smalltalk group.

Camp Smalltalk (CS5)

I spent the whole three days immersed in my CS project. The others are described from summaries obtained from their lead programmers.

Standardizing Sockets

Reinout Heeck worked on SUnit at CS5 but described the sockets project, which he hopes to resume soon, to me over dinner. The cross-product of high-level protocol and OS-level protocol gives rise to some 40 combinations of mappings to do. I advised him to leave ATM protocol till last since it has little relevance to the kind of application that will use this project. (N.B. this is based on telecoms knowledge that is two years old; if anyone knows better, email Reinout.)

The Refactoring Browser Port Project

When not helping me on the rewrite editor usability project, Daniel Vainchensar was working with Alexandre Bergel on the RB in Squeak. Bergel corrected some bit-rot that had occurred in the year since Daniel did the port while Daniel made Smalllint into a web service to encourage awareness of it among squeakers.

I gather that Squeak has the find but not the rewrite part of the Rewrite Editor. The basic rewrite functions must be there else the main RB would not work, so I would have thought the rewrite UI could be added easily.

Star Browser Project

This was further developed in VW and ported to Squeak. See my report of Roel’s talk for what it does and what was added to it in CS5.

The Custom Refactorings and Rewrite Editor Usability Project
I worked on this with Michael Prasse, Daniel Vainchensar, Aaron Hoffer, Adriaan van Os, Fransisco Garau, Michael Bateman and others. (Thanks also to Joseph Pelrine, who gave us the RB tests code, and John Brant, who emailed us the associated data files.) We addressed various tasks. (See

wiki.cs.uiuc.edu/CampSmalltalk/Custom+Refactorings+and+Rewrite+Editor+Usability

for the latest information on the project. The following summarises what we did at CS5.) Besides porting the project from VA to VW (other dialects still to do), we addressed various extension tasks.

1. The current 'Code->Meta button only maps code typed into the search pane into a single matching meta-pattern. We wish to replace or supplement this with a right-click menu in the RewriteEditor search pane. After selecting a single node, the user could invoke this menu and see a list of possible matching meta-patterns, identified by useful names. On choosing one, the selected code would be replaced in-situ by a metacode string, the remaining code being unaffected (and un-reformatted).

   By adding RBProgramNodeVisitor subclass: #RBMetaMatcher, Daniel and I got this working at the model layer and outputting correctly to the RewriteTools for some node cases. The current state is in the in-progress download but is not yet usable. I plan to refactor our solution to be more parse-tree-based, add the menu, do some more nodes and release a solution for the community to evaluate. (We will need help finding user-friendly names for the various replace options.)

2. A common source of new user frustration is making a trivial typo between the search pane and the rewrite pane. The rewrite does nothing and the user can't think why. Adriaan van Os has extended the code verification check to ensure that all meta-nodes in the rewrite pane also exist in the search pane. If not, a message (e.g. 'ERROR: 'myVart not in search pane') is added to the end of the rewrite pane.

   We also considered offering warning messages. Adriaan has implemented this, outputting the warnings on the Transcript. We're unsure about this and I plan to leave it in the inProgress download for now.

3. Next to 'Extract method...' and 'Extract to component...' we have added 'Search / Rewrite...' so that the user can select an expression in the RB and launch the rewrite tool with the selected code in the search pane. The main value of this trivial utility is to increase the visibility of the rewrite tool and hint to people about what it is for.

4. We've changed the dialog launched by the 'Find in...' (was called 'Methods') RewriteEditor button to allow selection of protocols and methods as well as classes and applications.

5. The help links listed at the foot of the RewriteEditor should be selectable so user can cut and paste them into their web browser instead of having to type them. This task is not yet done; any takers?

6. Aaron Hoffer and Michael Bateman (and latterly, myself) worked on

adding a ‘Rename InstVar and accessors’ refactoring to the RB as an option from the ‘Rename InstVar’ submenu. We refactored and subclassed John Brant’s RenameInstVar test to get a test to drive this and got it working. However, it still failed the test because, though working, it violated the basic pattern John uses for refactorings. I am currently half-way through refactoring it to conform to that pattern. When done it will, we hope be a useful utility and, perhaps as valuable, an addition to the custom refactorings examples, showing people how they can write these things.

As a side effect of our work, we had to get the RB tests and associated data files from their various locations and grasp how they work, helped by an email from John Brant during the Camp. I will document this on the wiki so future users can find things quickly. Adriaan and I sorted out some bit rot in the VA and VW3 Envy map, and wrote a

```
proceedThroughPreconditionError: anErrorString
in: aBlock
```

method (distinct VA and VW submethod implementations) paralleling John’s #proceedThroughWarning, to handle Envy-application-scratching obstacles to test completion. We include this in the release download as it may be useful for other tests.

While doing this, we were briefly confused by the bug (I think, a known issue) that VW3 reads a FileStream to the end, fails to read further, exceptions and default resumes. In a test (e.g. an RB test trying to read in a file of test-data code that isn’t on your system yet), this causes the bizarre result that TestRunner shows the test as failing but applying the debug button to it has no effect, because the debugger knows this isn’t a real error and ignores it. (We wrote code to make SUnit handle this. John Brant’s SUnitApp already handles it but we want to integrate this with CS SUnit.) In VA, a converse error occurs: a missing file error insists on throwing up the debugger even inside an SUnitBrowser test run; is this a known issue? (The SUnit team were busy when we saw these effects and we forgot to ask them about it later.)

Lastly, I will documented various patterns we learned in more detail on the wiki.

- Beware #removeNotImplementedItemsFrom: when adding menu items to the RB. If your XP-style of coding is ‘add menu item pointing at selectorName, then try to invoke to see where to add method’, the above will remove the menu item because you’ve not yet implemented the selector. (I must ask John or Don why this is there; at first glance it seems to be undesirably hiding errors.)
- The RB tests rely on a sequence of calling methods on refactorings

  ```
  aRefactoring execute
    ... aRefactoring primitiveExecute ...
      ... aRefactoring transform ...
  ```

  Do not try to compose them by putting #execute calls to components within #transform calls to composites. Better still, use

`#performComponentRefactoring...`
provided by the RB framework for that purpose. (In RB3.5.2, John introduces a ComponentRefactoring pattern, discussed on our pages.)

- Be aware of the pattern <action>* (low-level) and refactor* (high level) of refactoring methods. Our refactoring didn't need new <action>* methods, only new refactor* ones

For more detail, see our pages on the Camp Smalltalk wiki.

SUnit Port
SUnit 3.1 addresses:

- #debugAsFailure
- ResumableTestFailure: for when you fail inside a do: [self assert:...] and want to collect data and continue, to see what else fails
- #assert:description: aString, #deny:description: aString let you see something more detailed than 'An assertion has failed', especially useful in a packaged image.
- no more 'zork' tests in the SUnit suite: now pass = green for all tests

Joseph Pelrine and Reinout Heeck (and maybe others) ported SUnit 3.1 to VA, Squeak, VW, Dolphin, ObjectStudio, GNU and SmallScript. Joe Bocancas has promised to do the GemStone port off-site and someone has promised to do VSE. The project would welcome porters to Smalltalk/X, MT and StrongTalk.

Some other points were mentioned:

- The pattern for using resources has been clarified. SUnitBrowser's use of resources is now compatible with that of SUnit.
- Some people argue that, as tests are supposed to be order-independent, TestSuite's 'tests' instVar should hold them in a Set, not in an OrderedCollection, as at present. Joseph, Reinout and I all think that, a collection of tests will in fact be run in some order, the only way to verify that they are truly order-independent is to run in several orders and see what occurs, and this is much more straightforward when the order is explicit. Joseph is thinking of adding a 'Shuffle Test Order' button. First shuffle reverses order (two inverse orders being the best two-point partition of the order space), with subsequent shuffles partitioning the space further. To display the results, I suggest using my specialisation of the SUnitBrowser framework that views multiple test results against the same set of tests.

Other Projects
I may well have missed other CS projects going on in the room. I was almost wholly immersed in my own when there and mostly learned what the others had done by asking afterwards.

ESUG Programme

ESUG's sponsors (Cincom, Georg Heeg, eXept Software AG, Roots) paid for copies of an ESUG CD containing several non-commercial Smalltalks

(including the long-awaited VW7.0NC) and an impressive collection of tutorials, books and other material. Noury welcomed us to Douai on behalf of Mines de Douai. They have been training engineers since 1878 and currently support research activities, company creation and technology transfer to industry. Stephane thanked Noury; this was the fifth ESUG he's co-organised and it was the best organised from the local point of view.

Joseph Pelrine started us off on a light-hearted note by distributing a buzzword bingo sheet (who says we take Smalltalk too seriously :-)). Roel got a row before the end of the first lecture while I was still waiting for someone to say 'web services'. I never found out what he won.

I have sorted the talks into various categories:

- Writing Smalltalk: Process, Patterns and Tools
- Using Smalltalk: Framework Applications and Web Applications
- Using Smalltalk: Experiences and Opportunities
- Extending Smalltalk: Languages, Metaclasses and Aspects
- Open Source Projects
- Miscellaneous and Impromptu

followed by Other Discussions, ESUG affairs and my Conclusions (and an appendix with my contribution to the ESUG Education Symposium's request for a brief 'Why Smalltalk is Better' text).

Writing Smalltalk: Process, Patterns and Tools

From C(VS) to shining C(D): Optimising the Delivery Process, Joseph Pelrine, MetaProg GmbH

Inside every large system there is a small system trying to get out. Package early, package often is the motto. Smalltalk has an image problem :-). Joseph aims to push packaging operations out of the image. (Joseph made an analogy with the clean climbing ethos: don't leave the mountain covered with scaffolding.) Stop doing everything inside the image you are working on. Also, he hates Java as much as you do but don't refuse to use a decent packaging program just because it is not written in Smalltalk. These are examples of Joseph's four rules, which gave his talk its refrain:

> Package early, package often.
> Automate all that you can.
> Take advantage of the tools.
> Remember how it all began.

Various issues: declarative v. imperative, strip v. building; you don't have to remove something you don't have. Stripping goes well with imperative syntax, building goes well with declarative syntax.

Joseph demoed packaging the COAST UML editor in VW. Press one button and you have nothing more to do. The work you have to do to get to this one button state is a good thing. It reveals the problems you have in packaging your image.

Undeclared is harmful. You're in the development image, you see the message: fix the problem immediately. (Q(Niall) test that checks Undeclared? Joseph will have one in his next VW project. He has a VA equivalent; see the book.)

Module ~= Namespace. Squeak has modules wrong, VW has namespaces wrong. Doing Envy for VW5i.2 gave Joseph many illustrations of this. Namespaces have become a management instrument (quoting Reinout) because of the way they are implemented in VW. They should be about visibility. You should not be using full paths to reference class names.

Joseph tests packaged images by running SUnit headless and logging results to emailer. He demoed this in Dolphin: a TestLoggerResource captures failed runs. It grabs a FileStream on setUp and closes it on tearDown. (He needed this for a project where a Smalltalk package ran against an external interface whose maintainers sometimes changed it and forgot to tell people.) Demo (with Mission Impossible theme :-)) showed a one button sequence: 'change detected' to 'recompiling' to 'all tests run'.

He showed a statistical summary of results for successful, challenged and impaired projects (published by the Standish group). The clear lesson is that larger is less likely to succeed and over $10m are never successful, though 66% are challenged rather than impaired. Agile technologies address this by (among many other things) forcing you to treat your large project like a succession of small projects.

Traditionally, you develop on the development platform and then, late in the project, you move to the pre-production platform. It is a part of the agile approach to say 'package early, package often'. Joseph demands that his team gives the customer a ready-to-install CD at every milestone. We have time to do this: we work in Smalltalk, not Java ('a stiffly-typed language'). (Dave Simmons: I use firewires like tape drives, eliminating need for CDs.)

Packaging is boring so automate it and eliminate boredom-induced mistakes. 'Hail the power of the command line'; IDE macros, shell and batch scripts, cron jobs. You can use frameworks (Alexandria, Gump, Cruise Control) or (may well be better in Smalltalk) roll-your-own (and maybe earn money by selling it; there's a market). Automation uncovers issues by forcing you to write down all the things you need. It may also show you how to refactor your environment to support packaging better. Rule: automate the third time you do something. Do not automate at the start of the project; you don't know then what you need. Build it when you need it then you'll build what you need.

Test dependencies: one test may affect data of other; make data a TestResource and take it from a file. Use MockObjects for unit tests. Ivan wrote a tool called Jester (on sourceforge) in VA to massage your test cases to verify that the test cases make sense by changing assert: to deny: and seeing if the case still runs. Change code and 20 tests break; where do you start. Solve this by refactoring your tests to exercise disjoint code.

The VMWare tool (VMUnit due out) compares HTML on 30 browser-OS combinations. It's not too costly, can be a pain to set up but meets a need.

RUP is the sound a project makes as it crashes. One can do XP, yet let managers think you're doing lightweight RUP, with careful presentation.

Don't package on developer's machine. Have a gold-standard packaging machine and do it there. (Q. if this machine breaks tomorrow have you lost part of the process? Have an automated, or at least recorded, set up and configuration process for that machine.)

A project manager *can* be useful. There are managers who are inapplicable to agile processes but the right manager can stop going to meetings and bossing the team and start organising and supporting. QA people come to first meeting and say, 'Let's write the testing strategy', (expecting this to take six months). 'Fine', you reply, 'we start testing tomorrow. The test machine is there. You're most welcome to turn up.' Put them on the spot to do the job their being paid for. It is one way to help introduce agile.

Joseph recommends using ANT to automate builds. It has several 'must have' features and some 'nice to have' features. It sets up in 10 minutes. It takes a long time to run and is missing some things. Do not store your generated results in the repository because they should be reproducible (possible exception, if sending CD to customer and need historical record of state at that point.) Standardise on ANT targets (distribution, clean, test, zip and doc). He demoed using it to build some smallscript utilities he has, including the results of running the tests. He then showed the not-to-long XML file that controls it; behind the ugly XML syntax there was a straightforward list of automated tasks to build, test, zip and put everything in the right place. (Q.(Roel) what's the advantage over shell scripting? It's become common / standard in many areas, it took him 10 minutes to understand the syntax and use it, it runs on all platforms VW runs on.)

GUMP is only available on a few platforms, requires you to write a lot of shell scripts and has a polling approach to seeing what it needs to do. CruiseControl runs from an ANT script. Both are imperfect solutions.

Summary: its an onion. You can package, automate packaging, make it an ANT task, have tasks to call these tasks, etc. You do as much or as little as you need. The image is an artefact, a static cache for objects. Build it new every day and use new support.

InstallShield (much as he hates it) beats WISE because you can call the former as a task. There are many other solutions. FinalBuilder costs as much as InstallShield and does less but, being Java, sells as 'we run on all platforms'. Why doesn't VW capture this market by building a scriptable, extendable, ANT-callable rival? (Smalltalk can be the scripting language.) Sell it for $300 instead of InstallShield's $1200 and make money. (Dave Simmons: the market is split into high end - focus on single complex customer - and low end; aim for the low/middle end.)

Joseph concluded his talk by describing the Common Base Smalltalk project. Dave Simmons, Joseph Pelrine and others will coordinate an effort to complete what ANSI left undone. ANSI Smalltalk is autistic: it has no UI or I/O, no Modules or Namespaces, no sockets and minimal (read-only) reflection. How often have you reimplemented #notNil because you were accustomed to it but the dialect you have just moved to lacked it?

This task cannot be done by the vendors, though they are supportive. The Common Base Smalltalk project invites all interested developers (except dialect bigots and egomaniacs) to join them in working on things that should be common across dialects but are not in ANSI today. The core group will resolve clashes, apply consistent naming and coding standards, and issue releases of the common base from time to time. They have a process and tools defined that should make it easy to take new team members on board.

Joseph led a detailed audience discussion of this during which much interest was shown. I could not record it as I had positioned my laptop to record the upcoming Seaside talk. See the slides at the end of Joseph's presentation for more information.

RsSets: a pattern for fine grained access control in object systems, Alfred Wullschleger, Swiss National Bank
(I wrote detailed notes while Alfred was speaking. Then, while he was demoing, my machine crashed, losing them all :-/. The following was written from memory during the latter part of his demo while trying to follow said demo.)

Alfred has worked in software for 20 years, in Smalltalk for 10 (currently in VW3.0/Envy and Gemstone) and adds his voice to those who say you can build anything much faster in Smalltalk.

Four objects, ProtectionSet, UserCollectionObject, RestrictionSet and SomeObject make up the pattern. SomeObject has optional parent (inherit restrictions) and ProtectionSet, which has own and inherited (from parent) RestrictionSets. RestrictionSet must have rsModify (grants access to RsSets themselves) and insert (grants right to change children) instVars to hold access rights to manage itself. You then add whatever other rights you want. They use read, modify, delete (in that order: delete can modify, modify can also read). Other possibilities include execute or move (two checks: can delete here, can add elsewhere). The pattern is not concerned with business logic. For example, a rule 'can modify if three months old' would have to be implemented as some process that would update the access rights at the appropriate time.

Each right is a UserObjectCollection. An empty collection means noone has that right. A nil value in the RestrictionSet instVar means that everyone has that right (i.e. nil = infinite set). (I started what proved to be an ongoing discussion of whether a nilObject pattern, implemented by a singleton InfiniteUserCollection class, would be better than using nil. Martin Feldtmann mentioned that when your system writes to and from a

relational database, nil may be easier than the nilObject pattern.)

They have evolved various standard patterns for using this framework. He gave some examples:

- collection is parent of its contents, so its restrictions are their default
- composite is parent for its contents likewise
- legacy objects can be put in collections to group their access rights

He demoed: UIs let you assign rights, show their enforcement, etc.

The main value is that a user of the framework finds that right assignment parallels the relations between the assigned objects within the system, so it is much more usable than other approaches. The framework has been in use for 1.5 years and is liked.

Preparing For Pollock - The Future Of The VisualWorks GUI, Sames Shuster, Cincom

I missed this talk at StS2002 so was very glad of this second opportunity to hear what's coming. Sames described what the new VW GUI will do and how to write code that will be compatible with it. Previous GUI projects were Van Gogh and Jigsaw. Sames researched a large number of GUI frameworks in Smalltalk, C, etc. He liked Van Gogh best. (Why is his project called Pollock? See 'Other Conversations' below.)

He needs to obsolete:

- wrappers: everything wrapped in something else which wraps... Hard to find where you just enable this widget. SpecWrappers hold onto too much. ScrollBars are two buttons and a scrollbar, all three wrapped to show their bounds. Bounded, bounding, borders, preferred bounds (e.g. text that may be larger than window) v. bounds (actual physical bounds) and you have to know which wrapper to ask to get the answer you want.
- look-specific widgets: change look to MacOS doesn't change dynamically because the chosen look is set at build time. They have complex build rules (widgets don't know how to build themselves) forcing duplicated build code and mingling of common and look-specific code.
- ball of mud controllers: ifTrue: [ifTrue: [ifTrue: [self doThing]]]. They've lost their MVC meaning; because of wrappers, they are involved in upcasting and downcasting events. The controller was supposed to be the locus of where keyboard/mouse input was handled before passing it to the view. Today, they are doing lots of view work. The also have to use lots of navigation messages to find the wrapper to do a simple change.
- static edit keys: hardcoded or in UIFeel (because someone in VW3 said 'we have the look but we don't have the feel' :-)). CTL-C = copy hardcoded so if-check to see if you're on Mac and must use apple-C.

- fragile and noisy change/updates in guts: VW had first Observable pattern but this means all dependants get the changes and do lots of if-aspect-is-this-do-this (case statement). One of Pollock's goals is to be quiet, not to use your cycles.
- doTooMuch builders: huge builder builds window and then stays there, taking up memory (but having lost important build-time information). self builder componentAt: !!! Hard to get nested subcomponents.

VW is an emulated GUI set. It was done 10 years ago and needs changing.

- configurable hotkeys: Pollock will use Roel's MagicKeys.
- dynamic look changing
- simpler (stupid) controllers
- proper use of trigger event system (i.e. publish/subscribe, only tell who's interested)
- separate builder and widget inventory
- easy developer widget creation and extension: today, try to find where to change the entire background colour of a drop-down list :-)

Sames showed the VisualLaunchers wrapper hierarchy: I counted 12 layers and three sub-hierarchies in the picture but only 10 in the class diagram he also showed; either way it is too much.

VW will retain emulation widgets (for at least two years). However Pollock, like Van Gogh, will be capable of doing native widgets. Pollock widgets will do more, better, faster and, where needed, more platform-faithfully. What they don't do will be easier for you to do.

Pollock will have tools to migrate old-UI to new. It will be a visible project; community can see where its going and comment. It will be XP:

- test-driven development: he has lots of test and never releases a single line until all tests run
- continuous integration: do it and use it
- make it work, make it right, make it fast: e.g. currently some widgets flicker as they appear but they work, he'll fix the flicker later
- you won't need it: (it's a visible project: if you do need it, tell him and have a good reason)
- simplest thing that works
- let code teach you

He showed the wrapper hierarchy picture for the launcher in Pollock: 2 layers, 2 sub-hierarchies.

The Beta 1 is on the ESUG VW7NC CD. it has:

- Win9x, MacOSX and Motif looks

- ApplicationWindow, Button, Label and Image
- Build read and write from XML and Array
- full trigger event usage.

so you can build a pane with a button. The current state is well beyond this. Beta 2 will be in VW7.1: it will have many more widgets (Vassili, who's using it to build tools, drives widget priority), plus maybe fractional layouts. DragDrop and a UIPainter will be in Beta3. The first production version is currently scheduled for VW7.2 (but might be 3): it will have all widgets with at least basic functionality and UIPainter, MenuEditor and ToolbarEditor. That release will also have the last changes to the current framework. WinXP look and feel will come later. A later production version will also have the old GUI hard-obsolete, i.e. present as loadable parcel, not in base image. (Q. avoid Microsoft and Mac copyright problems? We avoid some specific things e.g. we won't use the apple-tab).

His goal is that the translation tool will translate 80% to Pollock with immediate perfect run. Of the remaining 20%, 80% will have the problem method marked with 'choose this or this to fix'. The remaining 20% * 20% = 4% will be marked with 'problem here'. (John Brant and Don Roberts who have done VA/VW UI translations, think it will do even better)

The Pollock metaphor is about panes and frames, agents and artists.

- A pane is anything you can see (and may hold other things you can see), c.f. VisualSmalltalk: all subclasses of AbstractPane. For example, a pollock button has no label: it can contains one label, ten, a list, another button, whatever.
- A frame is a layout with visible and displayable (i.e. can display within) bounds, and an origin; subtypes have alignment, fractional positions, relation positions and viewports. A pane holds a frame (but no further containment therefore frames are not like wrappers)
- An agent handles some behaviour that is currently on the controller. The agent tells the artist when to draw. It maintains most state (visibility, selectedness, pushedness, cursor position). It provides behaviour for mouse and keyboard and focus. E.g the agent holds the start and stop position of a selection
- An artist does all drawing. It interacts with agent and pane to learn what to draw and when. So far, the artist is passive and will stay that way unless the code teaches him otherwise. (Q(Jan Schuemmer) when to draw was the main issue in the COAST framework. The agent can ask the artist what is drawn and then say if it has to be redrawn.)

He will announce (in c.l.s) when the Cincom open repository has the latest work.

Sames showed the class diagram. the AbstractDraw class is superclass for AbstractAgent and AbstractArtist. (Q(Andy Berry) AbstractDrawer is better name: noun v. verb. Some people hate 'er' at end of names. His choice was either offend them or offend the grammarians.) There is some pairing (e.g. ButtonAgent and ButtonArtist, RadioAgent and RadioArtist).

Sames showed RadioAgent subclassed to MacOSRadioAgent, MotifRadioAgent and Win95RadioAgent. Usually, these subclasses override a single method. This caused much discussion about its blocking extensibility. Many people (Reinout, Michel, Niall, etc.) remarked on this; a MyRadioAgent must be done in three places, not one. He looked at a Georg Heeg employee's framework that used pluggable looks but that reintroduced wrappers. he made a trade-off. Alison remarked that introducing a new look (e.g. Windows XP) means making change in several places in the hierarchy. Sames replied:

- they will always support a new look within one release (or earlier if a customer pays them). Sames did the WindowsXP look for the current GUI in 1.5 months; he will be able to do it in 2 weeks in Pollock
- One day, Pollock will offer native widgets and this issue will go away.

However he conceded it was a more important issue than perhaps he had considered and asked me to collect the suggested solution off-line and record them. I am aware of two:

- Michael Prasse, Jan Schuemmer and others: use a delegation / bridge pattern. RadioAgent>>genericLookMethod calls an OSLook (or OSRadioLook) delegate that all agents (or all where this matters) hold.
- Niall Ross: use the visitor pattern. Push all the osSpecificLookMethods up into the superclass, eliminate the subclasses and have a visitor singleton (or a few if different widget classes cannot all use the same visitor) for each OS look. Have #genericLookMethodForRadioAgent, call the global that holds the current one, with self as parameter, which in turn calls RadioAgent>>osSpecificLookMethod.

Sames used the break to run 3179 Pollock tests (all passed) in the RB with SUnit (and it has XP scrollbars; 'when the UI is right, noone notices; when it's wrong, everyone notices'). He demoed some dynamic updates and programmatic control of a list box: send command to widget (never to wrappers) and it scrolls, changes single/multi selection, etc. while preserving selection, etc.; all much simpler than in the past. He showed the code for creating, configuring and opening a window (currently Pollock.ScheduledWindow differs from classic ScheduledWindow; after classic is retired, it will be the default).

```
"Demo code, user would use higher creation method on
pane only"
window := Pollock.ScheduledWindow new.
window frame: ((WindowFrame new)
  maximumSize: 400 @ 400;
  ...
  yourself.
window border: #etched.
window open.

"Do it, see the window"

radio := Radio new.
radio frame: (OriginExtentFrame origin: .. extent:..).
window addComponent: radio.
```

He opened a window and changed its look dynamically, then added a radio button to it, changed the button look between motif, XP, etc. Labels are not part of panes, they can be subpanes. He has labels and displayLabels to separate ampersand from actual display text. (Label controllers currently don't have mouse behaviour but it would be trivial to add.)

You have to have a valid frame before you put things in other things, e.g. add label to radio before adding radio to window. Otherwise, order is unimportant. Any Subclass can have any number and kind of subpanes. Common subpanes (e.g. button with label) are provided. Pollock supports 3 ways of building windows: Arrays (as current), from XML and from code, all of them practicably doable (try building from code without a UI spec in the current framework).

Controllers are still there but much simpler (typically 3 methods). They forward all events to the Agent. Eventually they will be removed. The Agent holds the keyboard processor.

Decorations are inside a pane (scrollbars, 3D look of button, etc.). Borders surround a pane, clipping its bounds. Not all panes have borders because someone told him radios and checkboxes should not have them but he now thinks he may well change to everything having borders. Some panes have decorations. Borders and decorations have artists but not agents, though their parts (e.g. scrollbar in scrollbar decoration) may have. (Scrollbars discussion; can they be just the same as others? No, fixed thumbsize obstructs this.)

(Example of pair-programming: he talked with a colleague, Rita, who wasn't even a Smalltalker, who suggested action-display (could have called it action-action) in which the button and check box, spin button, drop down, etc., are ActionDisplay subclasses.)

Agents, Artists and Frames are run-time pluggable. WidgetPolicy remains, supplemented by BorderPolicy and LookPolicy; FeelPolicy replaced by KeyboardPolicy, based on Roel's MagicKeys (currently a goodie: load by hand as current pointer is to old version). It will be application and platform-assignable. Thus you can have multiple buttons with different looks and you can have just one widget with your own special look without needing to subclass that widget's class. DialogWindow separate class from ApplicationWindow. (Discussion (Claus and Niall) need to be able to change modality of e.g. 'Find' dialog dynamically.)

DropDown: he thought drop down and menu differed only in their model but alas Motif and Mac do treat them as visually different. He still hopes to combine them into a single widget, possibly with a subclass. TabControl: tell him if it should still have minor and stacked tabs. Some HCI gurus say you upset the users if you stack or scroll tabs but he hesitates to deny developers the ability to upset their users. NoteBook is not scheduled to exist in Pollock: tell him if you think it should?

The trigger event framework is fully documented in VW7: who raises, how

to configure, etc. In VisualSmalltalk, events were strict: runtime halted if you raised an event you had not constructed for that location. However developers sometimes find that an obstacle. So you have the choice of turning on and off strict checking. Protocol:

```
EventTable
...
canTriggerEvent: anEventNameSymbol
hasActionForEvent: anEventNameSymbol
actionListForEvent: anEventNameSymbol
...
when: anEventNameSymbol do: aBlock
when: anEventNameSymbol evaluate:
when: anEventNameSymbol send: to:
...
whenAny: eventNameSymbols do:...
...
anApplicationModel
  widget: aWidgetDSymbol
  when: anEventNameSymbol
  do: ... “or send:, send: withArguments:
```

The handling is ordered so you can (should you?) rely on ordered handling of events.

Q. make events first class objects? In a distributed world, events want to travel across the world so stay with symbols rather than marshal events objects.

ApplicationModel>>widgetAt: replaces ... componentAt:; use it now to make your code pollock-compatible. Likewise different changed:... lets you become pollock-ready now. Use mainWindow, windowMenuBar, controllerAt:, widgetAt:, wrapperAt:. Use trigger events, not update/change. Don’t use the ‘default Smalltalk’ look. Don’t affect the GUI guts: if you must, subclass your change and how you get to it.

The model hierarchy is replaced by the EventModel hierarchy. He showed the new DragDrop system state-table: read it, tell him if you don’t like it. SelectWhenDown is junked (never worked). DragDrop button will be configurable.

CoolImage (from Travis Griggs) will replace ImageEditor in 7.2.

He showed what the enhanced UIPainter will look like: similar to 5i.4 but never more than three tabs and no apply and cancel (it’s dynamic). Instead it will have event configuring and coding in the painter, unlimited undo/redo (already there) plus list, widget morphing, XML save/load, etc.

Sames ended by indicating his XP process for creating a pane. It was very different from mine. He writes a test to verify that the Window subclass he wants exists, then that the Pane subclass exists, etc., whereas I would write a single test that did window instance creation and keep rerunning the same test till it passes, letting the debugger tell me where it was failing. Over dinner we had an interesting discussion (see ‘Other Conversations’ below)

VW dev list tells you where, every week, they put that week's image for download, play, etc. They've closed the list because they have as many members as they can support.

In 7.1, they will have multiple processes for the window manager so you avoid the problem of events you generate (e.g. test event to fake user input) waiting while another process (e.g. your test process) refuses to yield and even queries the window and complains 'why haven't you responded to my event'. This gets rid of the need to fork UI and is 10% faster (and may become faster still because they can get rid of the unhandled events that windows sends you e.g. paint the toolbar). Thus there is no heartbeat in VW: no events => VW will use zero CPU (current has 10hz heartbeat always and 100hz heartbeat with forked UI). Terry Raymond donated 90% of this. This is a prereq of Pollock so be aware just now it will warn that it is not loaded; don't try to load it.

StarBrowser: Classification in Action, Roel Wuyts
The whole talk was a demo, showing basic browser usage, operations on classifications and finally adding a custom service and item.

The star browser is implemented in VW. It knows about Items (classes, methods, etc.) and Classifications: nested groups of items. It reuses existing editors such as RB, Trippy, CodeCrawler and SOUL. Select an Item/Classification and the default editor for that item is shown.

Classifications are extensional (user chooses their contents) or intensional (user defines rule which selects items). Non-first class items like protocols can also be treated as items (silent generation of classification containing that protocols methods). One useful extensional classification for a singleton class is the one that has the singleton instance along with the class and its methods. A useful intensional one contains the most recently visited 20 methods (one line Smalltalk expression finds these).

It is easy to add editors and open them on classifications. Roel changed one to ADvance and instantly got a UML diagram of his current scope. Multiple views stay in sync (Roel had to write a dependent tree list to make this work). Camp Smalltalk 5 work lets you swap between current editor on viewed item being the whole screen and being inside the browser (Eliot asked for it at StS2002).

Roel has done much work (more to do) on operating on Classifications. He can convert extensional to intensional, and will soon add the ability to have mixed extensional and intensional classifications (e.g. the 'most popular classes' intensional definition keeps showing 'image' but I don't care about that one so omit it extensionally). You can take intersections, differences, etc., of classifications to e.g. browse senders of #add that also send #initialize. (Demo demon struck here; he called wrong menu item and it froze; well, development is ongoing.) A classification showing all methods in current work containing 'self halt' is useful: check it is empty before packaging.

You can add services to the browser. Yesterday he added items to let him show slides and code in the same browser. His seven slides were simply seven items in the 'Slides' classification under his current 'Root' classification in the star browser. SlideShower is their default editor. He will add a text file viewer service, etc.

Roel uses the visitor pattern with a #doUndefinedObject: method (to avoid ifNil: checks) as well as #doClass:, #doImage: and all the other visitor pattern calls for specific types. These calls have access to the subcanvas and browser so can use this information in displaying (e.g. choose display type based on what this classification contains). He experimented with overriding a classification's getting of its children, e.g. to give a Dolphin like look of 'click on class classification name, see its methods in its expanded classification', but removed it again for performance reasons since a classification's accessing its contents is very basic to the Star Browser's operation. Camp Smalltalk has just added an export facility.

The Star Browser model has six classes. He ported to Squeak (chose Squeak first because models ported to there will usually then go into VA, Dolphin, etc., easily) and it passes all its tests. Now we need to do the GUI.

As clicking on an item and viewing it may put you in any editor, you may need to use MagicKeys to ensure that you always get e.g. CTL-S = save.

Q. port to SmallScript and view whole computer in Star Browser? Yes, that should work.

Q.(Andy Berry) This is terrific but 'Classification' is an off-putting name. Call it the Groupy Browser (or the Groupie Browser?).

Using Cassowary for Scheduling Verification of Components, Roel Wuyts

Since the admin session delayed the start of Dave Simmon's talk, and Dave could in any case have talked for two days without exhausting either his topic or many in his audience, Roel kindly cancelled his talk to give Dave Simmons more time to explain SmallScript. The slides are on the wiki.

Using Smalltalk: Framework Applications and Web Applications

German Language in 7000000 Shared Objects, Jan Schuemmer, Intelligent Views

(It was actually more than 8,000,000 :-)). Jan reported his experience using their K-Infinity and COAST frameworks (see my past ESUG reports for description of these) to develop an application for a German dictionary company.

K-Infinity is written in VW Smalltalk. It has a knowledge builder, text mark-up tool and usage tools (some in Java) to work with web browsers, etc.

The Duden dictionaries are very widely used in Germany. Duden have many products: standard dictionary, ten volume dictionary, children's

dictionary, etymological dictionary. Currently the word ‘apple’ has definitions in each of these dictionaries, all held in different places. They needed a single repository.

Jan showed us the structured definition for apple (‘apfel’): keyword, grammar, all meanings, use in sayings and quotations, cross-references. He then opened K-Infinity and showed a tree browser of the grammar knowledge, a schema browser showing e.g. that adjective can become adverb, and the dictionary instances. He then scanned for compound adjectives and found ‘forget-me-not-blue’ (in German, one omits the hyphens of course).

Thanks to COAST, multiple users can remotely pair on editing knowledge in the tool. As well as the general K-Infinity interface, they built specialised interfaces for dictionary knowledge elements.

They have to deliver at the end of September. In true XP fashion, it all runs and is now right but one or two things still need performance tweaks (demo speed seemed fine). Their tests get entries from K-Infinity and compare textually with existing entries (they started COAST and K-Infinity 3 years ago without test cases so have them now but not complete coverage; Joseph is helping them). The customer provided a DTD for entries but it was rather ambiguous and was often violated in actual data (which was one of the reason why the customer needed them).

30,000 entities was the most K-Infinity had previously handled. They knew at starting that this would have 240,000 definitions each with many sub-objects, so they estimated 7,000,000 overall. Currently they have 8,000,000+ which threatened to give a 480Mb client image. They measured and tuned and sorted out the performance by:

- only loading into memory what they needed: in K-Infinity, model objects are organised into clusters. A cluster is the smallest unit that can be requested from the central server. In COAST 1.0, a cluster was loaded on demand in its entirety. For the first commercial installation, they added a lazy unmarshalling strategy of the frame content, so cluster has frame dictionary which has frames, each of which has an just an unmarshalled ByteArrays until/unless needed (if not, can just be written back as is). That worked for up to 100,000 objects.
- They now stream the clusters to a disk cache and only keep pointers to it. This gave them a huge memory saving; 22Mb of ByteArrays now live on disk, not in image.
- They wrote some index structures and short-cuts to avoid loading intermediate objects.
- They reclustered to have objects needed together in the same cluster.
- They avoided large Smalltalk dictionaries and sets (hashing poor when they grow big) in favour of Arrays.
- they had to teach WeakDictionary to shrink: VW grows value collection when entries added but does not shrink automatically when dictionary shrinks; they now check and copy to smaller when needed.

- They changed some Dictionaries to IdentityDictionaries.
- They eliminated instance variables when they didn't need them. (They now sometimes reuse instVars for several things; a questionable style but saving 5 instVars for each of 8,000,000 objects is worth doing).
- They looked at memory allocation for temporaries, refactoring code when it gave a big saving.
- Lastly, they looked at the VW VM. Bulk load problem was solved by reducing object allocation and footprint through lazy unmarshalling (it was not early tenuring, which was the first suggested cause). Thanks to John McIntosh' talk at ESUG2001, Jan knew enough about MemoryPolicy to specialize it. They had a problem while importing: 'out of memory winscavange.c'. It was not an endless loop but a GC problem. Eliot told them this and gave them a solution (set FreeMemoryUpperBound to max). The cause was Smalltalk handing memory back to the operating system too aggressively.

So much for the model layer. Some UIs just don't scale up. One that draws each object won't scale when you ask it to draw huge numbers of objects. A previous/next pattern for pages and drop-down lists solves this. (Rob Vens: he solved the same problem by overriding scroll-bar behaviour to add to the list as the user scrolls. They've done that too but the customer wanted a previous/next in some places).

The K-Infinity architecture has various tools of which they only need the administration tool, KB builder editor, COAST server and (specific to dictionary project) a query server, all in Smalltalk. COAST shares objects to support distributed computing. He stepped us through how the UI client and job executor client would cooperate over COAST. COAST provides them with transactions and concurrency control so they don't have to rely on a database for these. COAST seems to be evolving towards becoming an OODB; they are getting requests for 24 x 7 operation with backup. They are tuning it (12,800 transactions/second on 500Mhz PC).

Visit www.i-views.de for their current state and www.openCoast.org for background on where COAST came from.

Q.(Niall) other application domains? They are selling it also to Krupp to represent their enterprise structure, what companies they own, etc., and are thinking about product management, market evaluation, etc. It has many application domains.

Exposing Business Value with VisualWorks Web Services, Kirk Blackburn, Qwest

Kirk's last ESUG was Brechia in 1998 when he presented 'Interoperating between Smalltalk and Java'. This year, web services is the 'technologie du jour'. Co-developers of the work this talk presents are Stan Benda and Robert Mashad in a programme called Integrator within Qwest. Qwest is a baby Bell company in the western U.S. Their key product has 4000 classes in VW3.1 integrated with Oracle. It's an operation support system to handle service requests, inventory, provisioning DSL circuits, customer

record management, trouble ticketting, etc. (He joked that Verizon, in the same business, is also using Smalltalk that they can't get rid of, as I knew; see my StS2002 report :-)). Kirk described how they introduced web services to the trouble-ticketting portion.

Web services claim to let businesses expose, in a language-neutral way, the value contained in their legacy applications. 'Web services' is an XML-based standard for information exchange systems that use the internet, or an intranet, for direct business to business application interactions. It has all the buzz-words (open standards, platform-neutral, app-to-app connectivity, etc.) but so had CORBA; what's different? Answer, CORBA was based on an IIOP protocol you could never get through a corporate firewall.

Web services use:

- Simple Object Access Protocol
- Universal Description Discovery and Integration: like CORBA's naming service; a repository of information about services and where to where to find them, especially their WSDL
- Web Service Description Language: a grammar for defining object types and protocols very abstractly
- xSchema: the typing language; lets you say that a given parameter requires an integer, etc.

The hype is that this will make EAI easy and you don't care what the app is written in. The reality is that security, transactions and workflow have still to be incorporated in the standard or otherwise handled.

The model is that a user (with UDDI) sends a SOAP request via XML to a UDDI registry node. The request goes through the HTTP server, SOAP processor and UDDI registry service, which responds, the reply going back the same way to the user. The SOAP inquiry API is

```
find_...
get_...Detail
```

where ... can be business, service, binding or tModel. The Publisher's API can save... and delete... the same things and get security info.

In VW7,

```
UDDISearchTool open
```

lets you find and drill down for services currently offered (N.B. if you're behind a firewall, you may need to play with settings to see all that is out there on the web; this is just like using a proxy for internet access behind a firewall. By convention, web services run elsewhere than on port 80, e.g. theirs run on 4343. Perhaps port 80 would be insecure or subject to denial of service attacks.)

WSDL is even more wordy than XML (‘Simple’ is a misnomer if you are reading it as raw text :-)). It contains

- <portType>: operations performed by service; think of it as an object that encapsulates some operations
- <message>: messages used by service
- <types>
- <binding>

Kirk showed a short Smalltalk example

```
createTicket: aTicket
  ...
  ^Array with: aTicket with: errorParameter
```

and the longer IDL and the even longer WSDL that corresponded to this call and return info. The most common choice today is J2EE and BEA to do web services; .Net and C# is another.

Qwest IT Objectives are that everything will be a web service, *without rewriting the existing applications*. The goal was to be (and appear) a leader by being the first to do this (ultimately to save money by reducing the number of active languages and technologies within the company). They chose their trouble-ticket app (already CORBA-ized) as their first case, with services:

- findTroubleTicketByID
- createTroubleTicket

The existing app has a server for queuing and scheduling. A thin Java client talks IIOP to a 3.1 Promia ORB that talks to 6 TT worker ORBs that talk to the VW3.1 application. Their options for adding SOAP clients were:

- Build on CampSmalltalk XML support: the main issue was this needed a SOAP server which would take too long to write.
- Use .Net and VisualWorks COM Connect; this would run really fast but when he wrote a white paper his colleagues were not keen. They were worried about their lack of COM Connect expertise, the hardware (they’d never deployed on Wintel hardware) and the seeming complication (many pieces).
- Use the existing SOAP to CORBA bridge: this was a shareware product in C++
- Ask to be beta tester for the work Cincom was doing on it for VW7: SOAP Server, WSDL client, UDDI, XSD, XMLToObjectFramework and (as of 20th August) a WSDL generation tool

James Robertson gave them the OK to put the beta into production (great contrast with the old ParcPlace attitude!) so they went for it. They were well-supported by Cincom.

He showed an architecture of how the old CORBA and new web service fit together. Initially, they had a slight problem: they did not have the WSDL

and VW7 did not yet have a tool to generate it from Smalltalk. Java has a tool, .Net has a better tool, and since this week VW7 has a tool. Previously, Cape Connect from the Cape Clear company does an OK job of generating WSDL from IDL; it saved them!! Their steps were:

- create WSDL
- create Class for the web service
- create Classes for XSD types
- write XML to Smalltalk bindings (WSDLClient created default bindings. Stan changed the binding document, which is in XML, so that they bound XSD types to objects instead of to structs; bindings are in registry in BindingBuilder class).

(Now they have the VW tool, they would create the Classes first and generate WSDL from them.) It was much easier than any other solution because, being all Smalltalk, they could put self halts in code and see wherever there was a problem. It was cool that their VW7 SOAP server also had a DST ORB so the SOAP message comes into the server, gets marshalled into a Smalltalk object and can then be remarshalled into CORBA to get the proxy reference and call the method on the old implementation. Thus the SOAP server wraps the ORB which wraps the old app and so a CORBAed app can be SOAPed very easily.

The system is now in production. CORBA calls and SOAP calls have order of magnitude performance differences e.g. 800ms for SOAP, 30-100ms for IDL. They noted some issues:

- object references don't exist in web services; they were lucky they had CORBA
- WSDL is hard to learn
- SOAP works and can interoperate with ORBs
- .Net interoperability issues
- VW7 provides solid support (still needs some coding; not as automatic as .Net but .Net has interoperability issues) and has potential to leverage entire domain
- selling web services still a problem internally (because the groups that might use them are not doing .Net yet)

Overall it has been a big win for Smalltalk in their environment.

VisualAge Web Services, Alison Dixon, Hidden Ridge Consulting
Alison spoke while Aaron Hoffer (who works for her client), opened files and highlighted things.

Alison has spent the last 6 months studying what VA web services might do for a large financial (savings and load) application. It has a VA fat client (OS/2 moving to WindowsXP). The servers are OS/2. They need a high uptime, high performance and multi-peripheral support. They would like to move to thin client but can't see how a thin client could do all they need.

So when web services appeared, they looked at them to see whether they could use them to make the client thinner.

The did some web service prototypes:

- simple string concatenation: send two words from the client, get them back
- an insurance example: VA provide one, they played with it
- client search: they had an app that did this, so made it a web service using an external service that validates whether a U.S. address is a valid one or not. It had (relatively) complex input and output

Other groups in the bank want to use Java or .Net so the complex business logic in the Smalltalk server wants to work with various thin clients as well as their VA one (or at least not-so-fat clients, which they call rich clients; the company is expanding into small outlets like grocery stores who want rich clients, not fat ones). Web services must be described, published, found, bound to, invoked and composed to get an end result. The service requester goes to a service broker to find a service published by a service provider, to which they can bind. Alison found SOAP straightforward to use.

VA 6.0 has good support for web services, much improved on 5.5. It supports XML and SOAP parsing, SOAP over HTTP (beta) and has Server Smalltalk (transport and dispatching). It maintains Smalltalk semantics, important in terms of raising faults, etc. The online documentation seemed sparse to her.

Alison and Aaron talked to developers at DST (a massive US financial company), a lead IBM customer who began to use web services in VA before 6.0 came out and are very happy with it. DST use Server Smalltalk everywhere.

It was a little work to figure out how all the files fitted together:

- The .wsdl file separates implementation from interface (in microsoft, it is .as?wsdl or something like that). .Net has colour coding which helps read the file. One file does the imports and says where another file that defines the service is. VA generates these files and, for simple cases, you can copy and paste from one to create another. The types are entered by hand since Smalltalk has no types. She found it fairly easy to do and once you have one you can copy and paste to create others quickly. Most of the work comes in deciding what classes and what attributes of a class to exclude, not in typing (in either sense) what's left (I remarked that one could easily use e.g. SmallTyper to get the types automatically if needed, but from what Alison was saying, it was not enough work to justify that approach.)

  (C# differs from VA: C# uses ids to reference objects whereas VA uses tree references. At first VA would not parse C# output but now it does. There are still conflicts between how Java and how C# regard types: a problem for the standards people.)

She's only used immediate binding styles (VA also supports delayed response) and a few of the other options, e.g. encoded. She walked us through a .wsdl file example in detail. At first she returned results in OrderedCollections everywhere but Java and C# really don't like them so she now returns Arrays. The VA insurance example has an error which broke Array returns but this is now corrected. VA lets you map Arrays into OrderedCollections so you can keep all your existing code.

- XML files are fairly unique to VA though a few of the external services do use complex objects. The XML defines the mapping specs and .wsdl. There are many VA-specific goodies, e.g. handlers to deal with specific types of input, ability to specify client and server side independently (one XML file for each), etc. VA provides some standard services (#locateClientUsing: and so on).
- Mapping files convert between e.g. Arrays and OrderedCollections. She showed a server side and client side mapping file, the latter doing an addAll: from Array to OrderedCollection.
- The server has a container where you publish services. The client likewise has a container of services to bind to.

The serialization manager is the place to look if you can't figure out why things aren't mapping. It knows what files it thinks its using. (VA has many pluggable managers: Transaction, Serialization, etc.) They've managed to package what they've done.

They had to overcome various obstacles:

- From behind the company firewall, calls to

```
(SstTransport configurationRegistry ...
  proxyCredentials: ...
```

  need Base64 encoding. She spent a day searching the web for tools to do it, then found them in VA :-).
- Arrays in C# and Java issue
- Multi-references in C# also needed fixing
- It wasn't easy to see just how to use the various files from the docs, which is why she's described it above.
- She showed some code examples she'd had to find to do things: locate to search web services, etc. (see her slides for details).
- While developing, always disable Tools > SST > trap exceptions and Tools > SST > forward exceptions so you get a debugger when there's a problem. Always fork web service launch code lest you hang because it is not found.

She used handlers to get performance data, capturing the results by adding them into the SOAP headers (this didn't break the non-performance-tested case because when there is no handler it just ignores the handlee). The slides show the code for this, all relying on the invoke: method. The only annoying thing was that each piece (server side, client side) was slightly different to implement. They found that their network speed varies. They also found they were making persistent objects that did not need to persist.

Their asp .Net pages for client search were very fast. Overall, she was impressed with VA web services.

The immediate business need was because a bureau with an isolated system had loan information that customers could see on the web but not if they walked into a branch. They're now ready to receive this information but the bureau has not yet made it into a web service. Had it been their task, they could have done it quickly (modulo any security considerations, not looked at in their work so far).

Q. what benefit over distributed Smalltalk? It makes Smalltalk valuable to groups who don't use it. Not so long ago, another group in the company, whose apps were not written in Smalltalk, wrote yet another non-Smalltalk app to grab information from databases and process it. Now, their Smalltalk app could offer that processed info as a web service to the other group, saving the company the cost of the other group's reimplementation and reusing the business value of theirs.

Q. how long to make a simple VA app into a web service? Now she knows what to do, she could do it in a week.

Q. no wizards? Only for the .wsdl but she feels wizards don't add that much. Most of the work is in making decisions. Copy and pasting the templates is not much slower than what wizards do.

Seaside (Squeak Enterprise Aubergine Server), Lukas Renggli, Adrian Lienhard

Seaside is a web application toolkit for Squeak developed by Avi Bryant and Julian Fitzell. To demo seaside they have written a very simple app SWebMail that gets email from POP server, send to SMTP server. It took a day, reusing the existing Squeak Celeste app model layer.

Seaside is server-client, with the client using plain HTML; no need of client stuff. IAComponent subclasses to your app, in their case, SWebMail, SWebMailMain, SWebMailInbox (nested in that order on the screen). Instances are rendered and sent to the client. Squeak reuses by composition. Widgets such as tab sheet, sortable tables, dialogs, etc., are composed to make your pages. The idea is that programmers should create logic, while designers should create HTML (templates).

To get started, you subclass IA component to get a ViewControllers for your model objects (in their case, the models were the Celeste model layer). They all have method #html. Screen refresh happens when you click browser buttons or from refresh commands put in the HTML.

```
SWMail>>html
  ^'<html>
      <head>
        ...
        <swebmailmain sea:id=”main><...

SWebMailView
<td>Subject:</td>
```

```
<td>[message.subject]</td> ...
```

```
SWebmailCompose ...
```

(See their slides for all the code.) Seaside looks for a thing to bind to in an expression in template locals, then in accessor methods, then in instVars. Bindings are unidirectional or bidirectional. With action methods you can execute any method, get a parameter, and redisplay the current component, display new component, or throw an exception.

The common web app way is that the user types some data, clicks a 'return' button and moves between pages. It is error prone, needs cookies, or URL manipulation, etc. Seaside puts traditional Smalltalk send/return, loop constructs and control flow onto the web. It supports backtracking (user hits back button, app goes back to context) and transactions.

```
IAComponent>>callPage: aComponent
IAComponent>>return: value
```

For example, to login to the email application, the flow is InBox page -> (callPage:) ->(if login not already set) Login page -> return: -> InBox page

They then demoed adding the ability to delete email to their app. They added a 'remove' button that expects one parameter. No method yet written therefore button throws exception in web browser (i.e. does not raise Smalltalk debugger). They then wrote a simple two-line remove: method, removing from model object and from page cache and it instantly removed (no need to hit refresh or anything). They then refactored the method to do

```
self confirm: 'Really remove?'.
```

and reran it (the speaker being careful to tell the demoer which of his emails to delete; an overhasty demo might lose him an important message; maybe they should demo something other than delete next time :-))

Squeak has some goodies: e.g. a WebInspector: browse your instances over the web. Eventually you will be able to do all your implementation over the web if you want to.

Seaside 2.0 (Borges) will be a full reimplementation supporting layered design and more styles. Comparing with Zope, Seaside scores heavily on OOP-ness, reusability and separation of concerns (and you can write in Smalltalk, not Python). Debugging is much easier. Seashell (an OODB app) is being written to be the equivalent of ZODB. Both integrate with Apache. Obviously, Seaside is much less tested against large systems to date than Zope.

Squeak can be used with the Squeak-L or Berkeley BSD licence. See www.beta4.com/seaside for current state, www.beta4.com/squeak... for 2.0 work.

Q. does it support handling timeouts? Yes

Q. access request objects? Currently you need to go into the guts to do this but could be made much easier.

Q(Martin Feldtmann): in seaside, begin RDB transaction, do stuff, start commit, go to other page, start transaction, sometimes see strange errors. Speakers could not comment.

Q(Niall) status of ports? They have used the VW port once, but think it's not yet stable. The Dolphin port seems to be complete.

Using Smalltalk: Experiences and Opportunities

Dave Thomas - Invited Speaker: You Can' Do That With Smalltalk! – Can You? Lessons From The Past – Challenges For The Future

Dave is the founder of OTI (Envy and much else). He now works with Bedarra corp (Canada), and Carleton and Queensland University.

Brief History of Commercial Smalltalk (pre-1984): Smalltalk started on specialised hardware originally built for lisp which turned out to be useful for Smalltalk. The CIA was the first big user: Smalltalk was good for printing propaganda leaflets. Peter Deutch got Smalltalk working on tectronix. Digitalk then created their own version (digitalk methods and Smalltalk/V) thus it became accessible (only in 1990 did Xerox/Parc make its stuff available.) Berkeley Smalltalk added generational scavenging (Dave Ungar from idea by Baker), crucial for performance. Some Java implementations still don't use this, so have poor GC on small object creation. Then came GemStone and (with OTI) Smalltalk/VMac. In 1984 OTI built Orwell (so called because it was never to be sold) which we now know as Envy. OTI's initial interest was to built Smalltalk embedded systems, working with Tectronix; their oscilloscopes (late 80's) were the first use of an OO VM in shipped product.

- (1984-1994): lots more companies appeared: Instantiations Team/V, KSC, Object People (Toplink, Objectshare WindowBuilder). VisualSmalltalk, VisualWorks, Envy Smalltalk, Smalltalk/X.
- ANSI was a battle to get Xerox/Parc to release the basic collection classes, etc. OTI worked with IBM to make Smalltalk look safe to conservative companies. (Technically, it turned out to be work they knew how to do because a mainframe is like a large embedded system; programs must be small to be registered, etc.) Also because everyone else was being not-paid working in UNIX while IBM paid them to work on the AS400; lesson, go where money is and where you're wanted.
- 1995+: Strongtalk, OTI universal machine, Smalltalk/X Java extensions, PPS jigsaw.

Dave started in Smalltalk in the early 80s through wanting to introduce OOP teaching to Carleton university; three professors wanted to learn OO and decided Smalltalk would help them converge. With few resources, they did world-class research. Their students liked Smalltalk so much they convinced Dave to start a company, which seeded OTI, Object people, Object Time, etc., (3-4000 jobs). To keep programming in Smalltalk, they had to make VMs efficient for real applications (they were running in 10Mhz machines - very slow screen repaint, etc.). Their first virtual machines were built in C but the C compilers got smarter and stopped

putting values directly in registers when asked, so they built a framework which built VMs from processor descriptions. Everyone else (including Parc) said you can't do embedded Smalltalk so they were very successful - no competition. Plus they were then ready to scale up to other systems.

Unlike Java programmers who imagine they understand threads, they realized that processes were not really objects, therefore they invented Actors (clients, servers, workers, couriers and notifiers) to encapsulate threads of control. This let them do complicated concurrent system design in an OO way. They added

- Transactors: a persistent transaction framework using federated middle-tier transactions (other frameworks for these are ill-done today)
- Agents, Avatars, etc.

They also built distributed VMs.

Dave then reviewed some current commercial hot topics:

- Just before the IBM purchase, they were looking at large memory environments (512Gb RAM) in which the database is just the archive and the image is the business (MUD). There are some interesting problems here (multi-user shared images, etc.) and Dave still thinks it has commercial relevance.
- Trusted VM: needs capability protection within VM, GC security-aware, VM verified, security policies as behaviours. Lots of value to anyone who can solve this hard problem.
- Grid and MPP: MassivelyParallelProcessor systems (every object has its own processor e.g. show many stars and have them all twinkle at same time because each has own processor, give every plane its own processor in traffic control system, give every room a processor to control its temperature, security, etc.). They did something in 512 Fujitsu MassivelyParallelProcessors but used reflection and never made performance OK. IBM's BlueGene (drug design) is MPP: how will people program it? Someone who can provide the answer will have a market.
- Meta-modelling and meta-programming environments.
- Tools for research: doing research in Java is cruel and unnatural. If I already knew what type this should be, I wouldn't need to do research on it. If types mattered, Excel would force you to provide detailed types for all its cells on opening.
- Reverse engineering: aspects are very important in reverse engineering, not in designing. An aspect refactoring browser is the next tool we all need.
- Simulation environments for eLearning: it is a great opportunity but squeak won't get us there in its current state.
- Application-oriented 'thinking machines'

Lastly, Dave talked at length about what he calls the 'Sapphire' (i.e. better

than Ruby) opportunity: Scripting Smalltalk. People need languages in which they can think and assemble things quickly. In perl, Javascript, etc., people do trivial stuff and call out to external programs. Dave got lots of e.g. biologists to use his frameworks because it doesn't have perl, python, etc., syntax baggage, etc. Dave says we need a simple ASCII Smalltalk with Java-like class and package structure, proper modules, namespace and components, multiple IDE support (Smalltalk-80 *and* Eclipse *and* Visual Studio.net *and* Emacs; you must have backward compatibility with the brain-dead input solution), and proper lexical closures and continuation. It should also not ignore functional and vector languages (e.g. APL or J). Much time in self and hotspot is spent making brain-dead libraries run fast; vector language features would give better libraries that didn't need this.

PHPs power is that it lets you use all the unix and C APIs, so it's used despite a syntax that looks like you threw up on the page. Sapphire must have Linux and Widows API calls. It needs strings, XML, threads. It needs a Zope-equivalent weblog (Python succeeds because of Zope): wiki + webserver. It needs really good libraries: Dave claimed Smalltalk names are silly: collections should be made and changed later to what's needed. This is what VB does: make a collection then tell it to be keyed, ordered, whatever. Just because we have a refactoring browser doesn't mean we should always use it as the solution. Don't hardwire names so much. Instead, use factories rather than classes as in VB.

Sapphire should get rid of the image by making it a cache: code lives outside, you just work in cache. All image-based languages have problems: they keep all old history. Make image saved state of work, not saved structure of program.

Obviously, SmallScript.net and F-script (MAC APIs and array programming with selectors as objects) are two approaches to Sapphire. Dave welcomes divergence at this point; we must diverge to reconverge after we learn what changes are good. "Respect the best and embrace the future". (Response to later question: yes, GNU Smalltalk is also an exemplar; Dave is pleased that project is being done though he disagrees in general with using C as basis.)

Dave listed some products OTI built in early 90s: PDAs, phones, Oscilloscopes. He used students (they don't know it can't be done) on hard problems (always have a customer with a need and therefore a context for your problem). The path to commercial success is never clear; follow the customer. Envy resulted from, 'Either you solve the version management problem or we cut your funding'. Smalltalk is a much better language for modelling the world than UML and much better for scripting than python, ruby, PHP, etc., but it must meet them on their own terms.

Smalltalkers must also consider presentation. For years OTI, when asked, 'What do you use', replied, 'We use C with a small scripting language on top'. 'Could you do it in C++?' 'Sure, it will just take you 3x longer; I'll prototype in Smalltalk and then you pay manufacturing costs if you like.'

Q(Reinout): python, ruby, PHP are open source; sapphire must be too? Yes, at starting.

Q. how talk to customers? The future is in programmers with domain knowledge so you can have dialog with customer (programmers need to 'shut up and listen' to get domain knowledge). Mere programming is going to go off-shore (Brazil, China, India, etc.).

Q. Delta/V? DeltaV avoids CVS problem of copying things. Eclipse has option for method-level saves (must turn it on).

Q. commercial meta-programming? IBM and others are doing meta-programming work to support AOP in middle-ware to weave together web services, security, etc. (see recent AOP conference).

Q. A German car manufacturer wants us to build them a customised UML tool but they always say 'Write it in Java'. How would you handle this? I would tell them you must start with a rapid prototype to ensure that requirements are understood. I would then do the tool in Smalltalk and get it working. I would make explicit the cost of the Java 'back-end', and that the overall cost is competitive with a pure Java solution. If they still want it in Java once they see what the Smalltalk prototype can do, they pay the extra as indicated. Hire appropriate people to relieve you of most of the tedium of this so you can spend some time on your next Smalltalk project

Smalltalk in Large-Scale Enterprise Architectures, Rob Vens
Traditional Smalltalk apps are client-server, fat client and GUI-intensive. Rob believes he has identified a Smalltalk opportunity in a different area.

A year ago, he did consultancy for a company working with the Dutch emergency services. They have many applications holding information in databases and talking to foreign databases. These applications needed to be secure and hardened. Their response times are human (one second) and life-critical. Rob had a team of 25 migrating old systems (some very old, e.g. CODASYL databases) that were no longer maintainable. They had to be rebuilt, although it was politically correct to say they were updating them, not rebuilding them. Another item of political correctness was that the final result had to be in Java. Therefore they prototyped in Smalltalk and implemented the final result in Java. (I characterise this development process by paraphrasing XP's 'make it run, make it right, make it fast' to 'make it run, make it right, make it politically correct'.)

They used Rational Rose and implemented their Smalltalk prototype by hand from the diagrams. (Rose' meta-model is wrong so generation would be wrong. Stephane has a UML meta-model in Smalltalk; Rob will talk to him). The final result w as orderedto be in J2EE. They used reverse-engineered VisualWorks adapters, which Rob saw as a key enabler. They had to use traditional Java, not J2EE, because of thread/process issues. The system currently has 300 Java classes and Rob is very worried how it will scale up to thousands of classes as it must in the future.

Rob began by telling his amorphous team stories about how to manage complexity. He had to persuade old-timers that they could work in the brave new world and ex-students that what they had learned at college was mostly irrelevant. His process was

- convince the audience that what you are about to tell them is something they can learn
- tell the audience that what they are going to do is different from what they are used to

His aim was to 'move fear out of the room'. His project manager, whom Rob regarded as the best project manager he's yet known, was unpopular in his company because he did not do things like the other managers. He never organised, only delegated (very similar to OO). He also provided coffee and cakes a lot (I was reminded of the Oxford statute that any student can demand 'cakes and ale' during an exam, though only if they have remembered to include a sword in their examination outfit).

The project was very successful. This was quite a surprise for the company: working for the government, they were used to 'we're late, well, that's not good but it's not a disaster; the government will pay us more money to complete'. Rob's was the first ever project of this company that was on-time and in budget. The system went into production and is running with *no bugs found* since september last year! Rob can't explain the complete absence of bugs except that he was working with a dream team (I will say here what he was too modest to say in the talk, that his team-building work described above might have had something to do with his team quality).

So far, Rob had presented a 'hire smalltalkers and prototype in Smalltalk' success story, even if it was not a 'deliver in Smalltalk' success story as such. However, this was one of 60 projects done by the 700 developers in the company. Its success caused the company to rethink how it did things. They have decided to migrate all their other projects into a new large scale architecture for doing such projects. Rob believes that Smalltalk is the ideal language for implementing this architecture. He wants to use Smalltalk as an Enterprise Application Integrator.

Conventional EAI wraps existing systems and has a message-broker between them, but message-brokers are poor at business logic. In hub and spoke designs, business logic in is in the hub, publish and subscribe behaviour in the spokes, where adapters implement the spokes. Security, especially, on many levels is an issue. Several Dutch users of such systems report that it takes millions of money and 1.5 years to implement such systems. Rob used CBL for the publish-subscribe components and found it much cheaper (despite many fights with CBL consultants who thought Rob's team was misusing the software; they wanted CBL in the hub doing business logic). Smalltalk is what you need in the hub.

Smalltalk is very suited to the business logic component, and because it is only 5% of the system by a defensible way of counting, it is politically manageable to use Smalltalk there.

- language and problem domain closer than anything else Rob knows

- business domain component has other concerns than the rest: flexibility and extensibility

Their business model is an OO model (a 'Roger Rabbit' model, i.e. active objects that take initiative; in the film 'Roger Rabbit', teapots and cups are active along with people, i.e. cup drinks itself not drunk by person, doors open themselves not opened by person; similarly things that are active in the real world may be passive in the model). It is written in UML, implemented in Smalltalk, and attempts to be an exact replica of the business in software. (Example model use case: policeman confronts suspect, needs data on whether suspect is known to be dangerous, how to handle, etc. This model is replicated in the Smalltalk.)

For modelling OO systems, there are three alternatives:

- process modelling: problem is, sequence diagrams don't scale
- data modelling: problem is, can't handle processes
- distributed agent modelling: active objects with own thread of control. This is Rob's choice.

Business processes unfold in a backward-chaining process of objects delegating responsibilities. The process model is a 'pull model'. (Search the web for this phrase to get papers or read the book 'Out of Control', which describes how, at an HCI conference of 3000 people holding rods with red and green bulbs watching a ball bouncing game, a flight simulator, etc., people used rods to control the game and, after 10 minutes, they had lots of balls bouncing, could make a plane take off, etc. Rob's deduction is that a model of lots of simple active objects is best.)

Three months before release they had an enormous performance problem (J2EE and other issues): response was 1.5 days not 1 second!!! As the delivered system was not in Smalltalk, it was very hard find the bottlenecks. In the end they solved the problem but not by tuning Java. Usually J2EE systems have performance problems in the Java components. They had a well-compartmentalised Java thanks to their Smalltalk prototype and had most of their problems in database communication. The lesson is that Hub business logic is not performance-critical.

(Ernest Micklei's 'how to connect Smalltalk and Java components' part of this talk had to be omitted.)

Development of a Document-Management-System with VisualAge Smalltalk building a Windows-, Web- and WAP-Client, Marten Feldtmann

The DokWorks project was a document management built in VisualAge 4, 5 and now 6. Marten reviewed the decisions made five years ago when the company was founded and how marketing and technical issues have evolved since then.

Five years ago, the company was founded by a German steel company. It has 2 sales people and 3 developers. As time went by, they focused on a

more general market than steel, fortunately as the German steel market collapsed 3 years ago. When the company started, Marten already had experience in VSE and VW2.5.2, in which he had written some election result display software (see it in November). He had some problems with Versant and VW2.5.2 in this project, so decided to use VA in DokWorks. The steel company wanted to calculate project management information overnight and see reports in the morning. They had 500 plots in their current system and did not know how to manage them (file selector obviously hopeless). They wanted a document filing and retrieval system. There is a standard for this, ISO 10166-1. He had to read it 30 times to see what it meant and it is 12 years old.

The marketing idea was to sell a black box (linux server) with database and server, native windows client, and (in future) a WWW interface. Five years on, their other idea of a linux client market is seen to be fading.

Their first GUI was very like a VA or VW IDE. It had many windows and used menus not button icons. The customer said, 'Ugh!!!' Their third and final GUI was windows-like. It had one overall window containing subpanes, and button icons. They now have a Document Management System with yellow pages, email archive, versioning and rendering.

In 97, VW marketing from ParcPlace was dreadful. He really liked Envy (in the election project, the last code change was always 4 hours before the election program went on the air). Platform portability was very important. VW ODBC support was poor then (is it better now?). Multiple images on one computer were giving them a lot of problems (and still are in VisualAge today).

Hence, he switched to VisualAge: IBM was a good name to use when reviewed by their contractors. It had Envy and WindowBuilder. They also considered Smalltalk/X because it was a German company. It had Linux but a poor GUI.

In 97, Versant cancelled their VW-support therefore the project froze. They did the project with 3 people in 8 months which would have been impossible in Java but now they have no route forward (his manager is very unwilling to use an OODB because of the Versant experience and reluctant to use Smalltalk for the same reason and because developers are less common). He thinks GemStone is a very interesting product and considered it for DoK but was demotivated by Brokat's handling of GemStone/S: the price is high and there is no OODB standard. (He remarked that standards seem to be Java-specific today, which prompted discussion. Smalltalk can use JDBC drivers. I believe VW provides such drivers.) He therefore thinks of relational databases. PostgresSQL gave him problems by crashing often; he now knows the process you have to run regularly to fix this. Adavas-D was attractive but going up in price. They therefore used SAP-DB. To connect to the RDB they considered two products. TopLink is very expensive, did not give you full source and now has major management issues. POLAR is a German product: it was cheaper, poorly implemented and had a bizarre database constraint feature

(company has now vanished).

He decided to use rational rose but soon moved to using it as a drawing tool only. They soon realised generating from rational rose had lots of errors.

He decided not to use fast machines and forbade programming against local database. Developers disliked this but the program ran much faster at the customer. He had a 'my image is my castle' problem with some in his team who felt free to change system methods in the base image, leading to upgrade problems. He enforced 'you are guests in VA base image; no change without discussion'. Thereafter upgrades were easy. He changed five methods in VA4, only one in VA5, giving a one day update from 4 to 5, except servlets and that was because VA changed their protocol.

Marketing: they found themselves being more and more focused on windows. All the companies who had said, 'We're interested in Linux', revised their story as time passed. Hence after 5 years, they are using just POLAR designer and Visual Age with WindowBuilder, the TotallyObjects socket tools and other VA utilities.

Offering XML was very good for marketing but very time consuming:

- strange or buggy VAST5 framework
- creating XML much faster in VisualBasic due to its wizards

and when it was done, the solution was not customer friendly. It proved better to import Excel spreadsheets. They couldn't find examples of serious use of XML. Most medium-sized companies cannot produce XML data anyway. (This might change in the future.)

The RDBMS affects the implementation. Great standalone algorithms are useless due to DB access times. GUI browsers show 80% of useless data. It needs a lot of analysis and is hard for programmers to accept; they tend to think in terms of navigation. He now always thinks in terms of objects *and* tables. His rule of thumb is that a notebook page switch gives time for two SQL statements, so use notebooks. Use SQL logging to show problems. The amount of data transferred is also a problem; try developing against a database located on the internet! You have to strike the right balance between navigation and queries.

Windows gave them DLL hell, DDE problems, etc. They met a 'must look like Windows' requirement only to have the customer remark, 'Oh, does Windows 2000 look like that?' Windows NT FTP server had serious caching problems.

WAP phone development: his manager came round one day and said '...I have heard...'. Marten ridiculed the idea that usable data could be displayed on a phone screen but his manager kept repeating what someone had told him, so he had to hire student to do 8 weeks on WAP. The student did a reasonable job but WAP is very unusable except for very specialised applications. It is a critical, floating and unstable technology (very hard to debug), although the speed is OK if all else works. The development tools

were neither stable nor conforming to the standards and in the end even his manager found the result unusable.

Various issues caused his manager to asked him several times, 'Do we *have* to use Smalltalk?' Smalltalk programmers are expensive, tend to have 'strong' personalities (consultants), tend to want home-work and/or part-time work. The new VAST selling organisation (IBM passport advantage) is very Java-oriented. It takes them three days to call back with, 'Yes, actually you are right, we *do* sell VAS.' A customer wanted the source code in case Marten's firm ceased trading one day, but they did not want Smalltalk source code, they wanted Java source code. VAST licences are expensive and the third party market is poor (are Smalltalk customers like Linux customers who won't buy tools?). The third party products are not always up-to-date; sometimes the last update was released in 2000. VA lacks the wizards that VB has for e.g. XML generation.

Summary: Marten's impression is that the larger Smalltalk vendors are moving to the upper right of the market, depending on a small number of rich customers with demanding requirements. Are Dolphin and Smalltalk/X the hope for younger developers. Meanwhile, his company seems to be moving out of Smalltalk, e.g. they've defined Python as the scripting language for DokWorks.

Q. if you had to do the same project with the same resources today, what would you use now? Marten would still use VA with an RDB. Partly, of course, that is because he is experienced in that technology; if he had only Java-experience and was considering Smalltalk de novo, he might choose differently. As it is, however, he has won several competitions against Java programmers, and has to balance the Smalltalk difficulties described above against the certainty that, given the same resources, he would have a much inferior product to sell at the end of the day, if he completed at all.

Extending Smalltalk: Languages, Metaclasses and Aspects

David Simmons - Invited Speaker: SmallScript for AOS & .NET, evolution and advancement of the classic Smalltalk Language and Runtime Architecture

Dave believes classic Smalltalk is very good at a range of problems (therefore SmallScript has full backward compatibility) but needs to ease user adoption by supporting features from other paradigms.

SmallScript began before the .Net team approached Dave. It runs on two entirely distinct virtual machines, Dave's own AOS and .Net. He has a cross-jitter that maps between the two (and should allow him to match to Java, albeit with much slower performance). In practice, it's a static cross-compile but it is designed to be dynamic (e.g. compile on first call), and to have pluggable jitters. (Today, .Net makes it costly to compile very small code increments. Microsoft wants and plans to change this.)

The first thing he learned was that half of what makes Smalltalk good is the tools. His first year on SmallScript was very educational in that respect since he lacked those tools. Scripting languages perform poorly in today's

environments because those environments are adapted to static languages. Doing SmallScript on .Net both made these features obvious and influenced Microsoft (ongoing) to solve them insofar as Dave has not already done so within SmallScript.

After explaining why he wanted to add the various features, he described how to use it. Dave coded and ran simple scripting examples in a typical scripting text environment. (Q. why backquote? Dave uses the backquote to escape strings that his Eval

```
Eval [code] or just [code]
```

will parse for the same reason that the RB uses it in metacode expressions, because it is not used, or almost never used, in Smalltalk.) He then built a small program, again writing it in the text environment. SmallScript source is a variant of XML called AML; it can support multiple languages.

So far, he's just done what any Smalltalk can do, albeit in far smaller space (4k examples). SmallScript supports additional syntax for messages (paren form and array form as well as classic selector form; you use which you want). DLL hell arose from the lack of strong naming. One vendor's installation could mess up another because things could not be distinguished. SmallScript supports strong naming. Dave created a DLL, created a C program that linked to it, and ran it. The total size was 21k, the DLL was 4k. You can run this Smalltalk with no need for large image, start-up time, etc. Dave added version resources, icons, etc., making it 12k in size. These were the things Dave really cared about.

Q(Claus) beside the DLL you created, what else do you need to run? You need AOS.dll (currently 1+Mb, should be 1Mb in final version). This file can be put anywhere on your machine and it will be found correctly (thanks to strong naming).

Something Dave loves about Smalltalk is that it is a dynamic language. Change is happening all the time and dynamic languages handle this fact better than static. Objects are an abstraction we put on this to handle it better. Thus what an object is is crucial. SmallScript objects have:

- properties: these are extensible
- by-ref fields: named slots and indexed slots. SmallScript has optional typing. In a dynamic language, the type that matters is the type of its behaviour, not its structural type.
- by-value fields: Java's primitive types, C/C++ struct, Smalltalk bytevars. It's really an array of one-byte characters (with any encoding you want, not just string) or two-byte unicode characters. These are for interoperability: get data structure from the heap or from malloc and handle it here.

Everything is conceptually an object. All inter-object operations occur through messaging. Only an object can reference its fields without messaging. Everything is dynamic and subject to change except an object identity. Non-smalltalkers complain, 'Messages can have side effects'.

Dave replies, ‘Running out of memory can have side effects. Understand encapsulation. You must trust the encapsulated object you are calling, not treat its behaviour as your responsibility’.

In classic Smalltalk you start with a large image, write your app, then extract what you want to give to other people. This model is flawed. Dave wants to build applications by saying what you need during development time, building up from small pieces. Thus he has designed what he believes is unique dynamic module system, which depends on the fact that everything is a message.

He has an adaptive garbage collector. Smalltalk threads are green (non-pre-emptive) but in Dave’s scenario, e.g. Smalltalk called from C and back, you must have the same characteristics as your caller/callee. Thus he uses native pre-emptive threads with native exceptions. For example, a memory violation in called C can be caught and handled in Smalltalk (allows powerful memory handling).

He has architected SmallScript to allow all kinds of experiments. The MOP is visible and extensible. You can create an annotation, attach it to a class, reflect on it and use it.

Dave has dumped the classic byte code instruction set VM approach. Java has a terrible instruction set. .Net isn’t much better. In a common language architecture (many compilers generating code with many different semantics and characteristics) you don’t want one optimising compiler removing stuff another wants. Your instruction set should capture language semantics. SmallScript set understands what closures are, what fields are, it can see annotations, etc. Dave will optimise against that instruction set which, because it understands the semantics, can do intelligent things.

He supports an extensible predicate-based binding architecture. The hardest part of his design was making this clean and efficient.

He also supports dynamic multiple inheritance: AbstractMixin, PureMixin (behaviour only, not structure) and Interface (concrete structure with aggregate structure). Via interfaces he can, in effect, add fields to a Class that is defined in a module he does not own (as if an Envy app could add instVars as well as methods to an extended class) while ensuring that multiply inheriting from two classes with a common superclass does not duplicate structure. Interfaces are lazy: the aggregate structure is created when referenced, not before.

Modules are distinct from namespaces. Modules are simple. Dave could never have built SmallScript if he had not identified first principles. If your module system is complex, you’ve not got it right. Dave has had namespaces in his smalltalks since 1991 but only now has he got it right. A modules is a unit of deployment, packaging and other non-runtime things. A Namespace is a unit of access, security, visibility and other runtime things.

Classes are one of the visible named entities. Access means access to a class or to a variable that a Namespace (which is a class) contains. Classes are units of behaviour (obvious to Smalltalkers) of meta-data (he puts meta-data operations on the meta-class so they can never clash with class methods) of structure (how are its instances held in memory) and of privilege and scope (every class is a namespace). Dave has no system dictionary: classes are namespaces and the root namespace is just another class. Dave also has a meta-namespace system (he would need much more time to explain it) that lets you run two versions of an application in the same image.

Dave's next demo showed how a SmallScript program can use other language tools. SQLLite is an SQL library written in C. Dave got its C API and DLL and documentation. The DLL has 33 different entry points. Many current scripting languages have reams of code to access tools like this. Two hours after he began studying his download, Dave had what he now showed us. He declared the functions he needed of the 33 (simple one-liner per API call) plus four convenience functions for easier calling. He then declared an Exception, with generated getters and setters and an annotation providing an alternate name for the message (because the keyword message doesn't translate to C). This in turn motivates him to use +, his alias of Smalltalk's comma concatenation operator, in this code, so he can use the paren form, with its commas, without having to add an extra pair of brackets to each use. These alternates make it easy to port; he can just dump a C header file into SmallScript without rewriting things. He also can force all inline accesses to be getter/setter accesses declaratively. He needs to refactor this often (for reasons to do with e.g. patterns for deciding when external object proxies are GCed) and does not yet have the RB so needs to refactor it declaratively.

He showed multi-methods (one took table index as parameter, one took table column name). He gave the exception a callback and showed the marshaller that supported it (he quoted getting 10 times better performance this way than a JNI system: 40,000,000 trivial calls/second). His test used a SmallScript implementation of printf to tell stdout that the test is running, while creating a table, and writing and getting two rows. For every row the query returns, the database will issue a callback with parameters for that row's columns and values. An Event onQueryCallBack wrote the rows to stdout.

Having browsed the code, Dave built it (overall 11k in SmallScript) and ran it. Then he ran it again, trying to create an already existing table, to show the exception being caught and handled (in stdout in the example but it could have been caught and handled by any debugger; of course, the debugger would have to understand the exception which might be raised by C or by SmallScript in this example, and in any language in the general scenario).

Dave next demoed selector namespaces. Two namespaces ('French' and 'English') were imported by the test class in that order (order matters) and used to show how scopes worked. Time prevented Dave from showing how

this lets you do security; see my StS2002 write up for the details.

Q(Andy) SmallScript should be called LargeScript? Good point. The idea was you *could* write small scripts and scale up naturally.

Q.UI? Dave demoed some UI manipulations: draw text in 360 degrees, etc.

AOP in Smalltalk, Kris Gybels, Robert Hirschfeld
I missed this talk. See their slides on the wiki.

MetaclassTalk, Noury Bouraqadi, Computer Science Laboratory, Ecoles des Mines
He has done many experiments on multiple inheritance, aspect-oriented programming, mixins, types, etc. Each time he dives into the Smalltalk meta-layer to make the framework for these experiments. Now, he has built a meta-framework that will let him do these experiments.

Smalltalk is the best language in which to do this but even Smalltalk has obstacles:

- metaclasses are implicit (no name) and in a parallel hierarchy: you create the class and the system creates and assigns the metaclass
- evaluation mechanism hard to override

MetaclassTalk (formerly NeoClassTalk, formerly ClassTalk) is Smalltalk with explicit metaclasses. It also allows control of program execution: instVar access, message invocation, etc.

The root is Object subclass: StandardClass (has Object and itself as instances). By default, each class controls the evaluation procedure for how instance memory is allocated, how instances are created (#new = #allocate followed by #initialize).

If anA sends foo: value1 bar: value2 to aB then Class A handles the invocation and passes it to class B which processes it. Thus if you make your class a subclass of metaclass TraceClass which redefines #send: and #receive: to do

```
Transcript show: ‘sending...
```

followed by

```
super send: ...
```

this will trace execution. Usually, you will make TraceClass an instance of StandardClass.

MetaclassTalk is implemented in Squeak. He demoed creating a MessageCounter metaclass and a MyObject class, with a simple method, whose metaclass was MessageCounter. After showing MyObject had normal behaviour, he edited MessageCounter to have an instVar count, incremented whenever instances received a message. He then initialized MyObject count: 0 and showed it all works. A slide listed a few of the many patterns that could be implemented in MetaclassTalk.

Q(Michael Prasse) advantage over method wrappers? Method wrappers let the receiver (only) change the behaviour of its methods (only). MetaclassTalk lets the sender and receiver change the behaviour of method invocation and instVar access. Of course, it changes the behaviour of all its classes' methods. One will still use wrappers for single method changes.

He showed using mixins to let ReadWriteStream inherit from ReadStream and WriteStream. His current implementation generates the mixed class and dynamically reflects changes but only for classes in the image when the change is made; the generated mixed class will not inherit passively from its non-superclass. He then used them to mixin a LazyAllocation metaclass that did not allocate memory for nil-valued instvars, and a BreakPoint metaclass that allowed breakpoints to be set. Q(Stephane) lazy allocation dangerous in multi-threaded environment? Yes, if stateful; his threads are stateless (I think - may have misrecorded the discussion of this).

Aspect-oriented programming in MetaclassTalk uses the natural split that the aspects are Metaclasses and the application functionality is provided by classes. A metaclass SimpleAspect supports adding aMixin to aClass, plus #weave and #unweave (AOP jargon for method wrapper install and uninstall). Specific aspect subclasses of this override #weave to add specific mixins.

MetaclassTalk is implemented in squeak 3.0 and will soon be in 3.2. They are working on improving performance (currently very poor as they interpret everything) and extending the library metaclasses. Their long-term plans are to build a complete new Smalltalk kernel with explicit metaclasses

Q(Reinout) did it require extending the VM? No, not doing so was a requirement.

New Model for Smalltalk: the Trait Model, Nathanael Schaerli, University of Berne

The trait model goes beyond mixins and multiple-inheritance (which can have unexpected effects) to achieve reusability. In a typical class, methods have interrelationships (e.g. they call each other or affect the same instVars). He demoed with a circle class whose instVars are divisible into geometry, colour and whose methods are in these protocols or are in comparison protocol (only uses geometry), etc.

Smalltalk protocols are merely descriptive and can't be reasoned about. Nathanael defines these groups as Traits, first class entities you can reason about. A trait is completely behavioural (no state) and provides services. It also requires services (e.g. Circle comparison requires Circle centre and radius). Two traits can contain the same method name: such conflicts must be resolved manually, e.g. they can both be the same method (defined in class). When calling a method, you do not refer to a trait, so cannot say 'give me method X of trait Y rather than just method X'. However you can use a trait while excluding one if its methods, thus forcing the choice of the same method in another trait you also use.

Traits are not classes: they have no state and no inheritance. They differ from Java interfaces because they have implementation, not just selectors. A class is a compound of states, glue and traits (plus methods outside traits, e.g. those with direct instvar references).

He demoed (the implementation is in Squeak). As you add methods to a trait, the system computes their requirements. He created a new class, gave it a trait, showed the default behaviour for requirements (threw error) and then converted them to requirements (colour changed to show class now usable. He then gave the class three traits. Colour coding showed conflicts, which he then resolved in various ways (use this trait's method, use that trait's method, reimplement a third method in using class, etc.). Reused method cannot be edited in reusing class, only in original. He cannot at present add two traits that have the same method and have each trait call its own method in its own code.

Dave Simmons argued for trait ordering: it ensures that conflicts always have a default correct solution so the user does not *have* to resolve them (but of course they still can).

I felt that he needed a real 'customer' to progress his work. I know only one case where you really need something like multiple inheritance in production code. It is needed to implement the Object subclass: Class' pattern. I will document this pattern on the meta-programming patterns page and send a copy to the speaker.

F-Script: Smalltalk scripting for Mac OS X, Philippe Mougin
ESUG is sponsoring work in F-Script. Alas, I had to miss this talk to catch Jan's. The slides are on the wiki.

Open source

GNU Smalltalk, Bonzini Paolo
The Squeak Demo, Stéphane Ducasse
Swiki demo, Samir Saidani
Smalltalk/X presentation, Claus Gittinger
The Smalltalk/X version on the ESUG CD handed out at this conference is two years old; the next version of the CD will have the up-to-date version. Save for this snippet of information, I missed all these as I was attending the parallel thread. I hope information will appear on the wiki.

Miscellaneous and Impromptu

Terminal Widgets on Objects, Ernest Micklei
Ernest has had Repetitive Strain Injury. He wants fast access to simple programs. He wants to explore mixed (pixels and ASCII) interfaces. He delivered an application to a company and found they used the keyboard short-cuts, not the mouse, because it was faster. He therefore wondered if character-based UIs were all bad. He wants to simplify UIs.

He has therefore produced an app (in VA, but designed to be portable) to let him have character-based applications with objects behind them. He demoed a character display widget. A widget displays an aspect of an

object in a defined region of a window. A region is defined by a rectangular array of characters. It has a controller for keyboard events, selecting, etc. A simple framework of classes lets you build terminal-like menus, displays, scrolling text, etc. He then wrote a(nother :-)) class browser, a Windows widget, etc. A top container can show, hide and (in future) stack windows. Border highlighting shows focus between widgets, highlighting shows focus within a widget.

TerminalForm displays a 2D TerminalGrid of TerminalCharacters, unlike the usual approach where a single object displays all characters. The InputController was hardest. As text is typed, lines must scroll, wrap, whatever. He did this character by character; Sames recommended handling the whole text and mapping it to character positions last.

He was impressed with the linux terminal widget: program completion, etc., all done with keyboard, no mouse. He therefore wrote an object terminal widget, that views a namespace, where the root namespace is Smalltalk. It executes various linux/unix-like commands, e.g. ‘edit’ method in Smalltalk is invoked by ‘command_edit’ in terminal, plus any quoted text is treated as pure Smalltalk.

He is almost finished porting it to a SmallScript module. Remaining questions:

- how to design character-based UIs that are still OO
- is anything beyond TELNET needed for client-server architecture
- what more widgets are needed? buttons? dropdowns? (don’t want to mimic Windows)

Discussion: Claus sees many applications for this once it has full terminal emulation. There are many commercial terminal emulators but they don’t have the toolbars, etc., of this. He stressed transparency must be added, e.g. to see alerts. Aaron suggested using it on small devices, e.g. after porting it to Pocket Smalltalk.

Ernest’s final demo showed an app with text and graphic widgets but he agreed that transparency is needed to do more with this.

Fun with Lego Mindstorms, Alexandre Bergel
I missed this due to a clashing talk. The paper is on the wiki.

Talking to users, Andy Berry
I missed this due to a clashing talk. The paper is on the wiki.

Portable Smalltalk Format, Peter van Rooijen
I missed this talk but arranged with Peter to get a description of it from him on Friday afternoon, so as to include it in this report. In consideration of the issue described in section ‘ESUG Board Ruling’ below, I did not do this.

Solving the XP Legacy Problem with (eXtreme) Meta-Programming, Niall Ross, eXtremeMetaProgrammers
I gave a repeat of my Smalltalk Solutions 2002 talk on how to add XP to legacy Smalltalk projects to interested parties on Tuesday evening. See my StS2002 report for a summary, or my paper and slides for details (both available from www.whysmalltalk.com.

Argo Framework, Michel Tilman
Argo is a meta-data framework (in VisualWorks) for creating web-enabled tools to manage documents, processes, and other administrative support needs of large organisations. It is particularly strong when the organisation needs its tools to be built quickly and changed frequently. At StS2002, Joseph Yoder praised it as still the most advanced meta-data work he knew.

Michel and I went through it on Wednesday afternoon. I hope to help Michel develop it and find opportunities to use it. A department where I used to work spent 100 times the effort Michel's tool would have taken in setting up web-enabled administrative tools for documents, contacts, projects, etc. As we were repeatedly reorganised, these tools aged and died, replaced by others. Many organisations are in the same position today.

Other Discussions

Smalltalk Applications
Adriaan van Os has built a meta-data-driven system for an agricultural research institute attached to the Agricultural University of the Netherlands. The system lets them handle their data and workflow. He demoed by creating a small model of Camp Smalltalk (people, events, projects) with their connections and workflow. A demo can be downloaded from www.soops.nl

Fransisco Garau, Daniel Pfander and Vincent Disselkoen are working at a Swiss insurance company, Die Mobiliar. An attempt to reimplement key functions in Java failed last year, so Smalltalk is currently re-popularised in the company.

Dirk Roeleveld now resides in the UK but till recently worked with Graham McLeod in the South African company Inspired that sells a knowledge management product Archie, written in VisualAge. He still acts as technical researcher for them.

eXtreme Programming Styles
Sames approach to XP involves more tests than most of us would use. He writes a test that verifies that a class he plans to use exists before creating it. He also writes a test that verifies that its superclass exists. Legitimate refactoring browser operations would break these tests, arguably a serious violation of XP's testing-and-refactoring philosophy.

Sames has two arguments for his approach:

- He is designing a framework. The classes are going to be used by many. Although he could refactor their names all he likes at present, when the framework is released, their names will be part of its interface and casual changes, however safe in the image in which they are made, could break customer code that relies on the framework once they are released.

  I grant that this makes perfect sense *for specified interface classes*. By writing a test that asserts that the class exists, with given name, you precisely capture the fact that the framework's overall behaviour includes the promise to provide a class of that name, not just a class of any name with the same behaviour. The test is thus informative. However Sames, and he tells me Ron Jeffries too, would write such tests for *any* class they create. This I disagree with. A legitimate refactoring should never break a test. If it does, the test is in error. Tests should only ever forbid acts that are not legitimate refactorings.

- Sames second reason is that a code reference to a non-existing class will create a global in Undeclared. Can one be sure that a test (or an earlier act of yours) will not populate this variable with some value that another will manage to use instead of falling over?

  This is not an empty concern. I have myself suffered bugs because what I though were declared ClassVars or classInstVars (though never a class in my experience) were in fact Undeclareds. However I regard a test on Undeclared as a far better solution than tests that legitimate refactorings break.

General Conversations
Why 'Pollock'? The project was a successor to one called 'Van Gogh' so Sames picked the name of a more recent artist. He admitted that he knew less about Pollock when he chose the name than he did now, but claimed there was 'something there' in Pollock's pictures; I suggested it was probably bits of the tortoise to whose tail Pollock attached the paintbrush :-). We amicably agreed to disagree about the merit of Pollock's work.

Christian asked me about 'Local Senders/Implementers' in the RB. This issue is mentioned in my CampSmalltalk Custom refactoring project pages. In some Smalltalk IDEs, 'Local Senders/Implementers' means search in self, sub and superclasses. In the Refactoring Browser, 'Local ...' means search in the environment currently being browsed. Every RB UI views some *Environment instance which is its filter on the overall Smalltalk dictionary. If you search for (global) senders of #add:, say, the resulting environment can see only methods calling #add:. A 'Local...' search for implementers of #initialize, say, invoked from that browser will therefore find #initialize methods that send #add. This is more powerful than the standard behaviour but the coincidence of names is unfortunate. Many people try the RB's 'Local...' commands, don't understand why they don't behave like the standard ones, and give up on them.

I've several times heard second-hand stories of Smalltalk-using firms concealing their development process. This ESUG I heard a more first-

hand story. Alison and Aaron's company entered a 'quality development process' competition for their industry a while back. When it seemed clear that they would win, the company ordered their team to withdraw lest the resulting publicity alert their competitors. Starting out years ago as a small (intra-state, I think) company, they have now grown so big as to be immune to take over (largest in the U.S. in their field, I gather, or one of the largest) and this has made them more relaxed about publicising their process (thus, for instance, Alison's ability to present this year).

ESUG

ESUG has a new web site. It also has a free CD; 1500 will be burnt and made available to various conferences, e.g. ECOOP. It has a student volunteer programme, a free books online programme, and a universities programme. They offer 17 Smalltalk packages and 5 Smalltalk immersion packages, the latter costing 12000 Euros. If you have any contacts to universities interested in upgrading their OO courses, let ESUG know.

This year's conference presentations can be found on the wiki at http://csl.ensm-douai.fr/esug2002/.

ESUG Education Symposium

(I only caught the last few minutes of this.) We believe Smalltalk is better. How would we show this to a computer science student? Alfred Wullschleger uses SortedCollection as his base example when teaching (he's taught both Smalltalk and Java 1.1). It is instantly clear that Smalltalk's is easy and Java's is strange. Andy Berry suggested that we each write a few paragraphs on 'Why Smalltalk is better than Java' and email it to Stephane. (My effort is in the appendix to this report.)

Re-election of ESUG Board

At the start of the final day, the ESUG board (Stephane Ducasse, Roel Wuyts, Rob Vens and Joseph Pelrine) presented themselves for re-election and were supported by a forest of hands. A round of applause expressed the audience' appreciation of what ESUG has done to help the cause of Smalltalk.

ESUG Board Ruling

The board were then obliged to perform a tedious duty. Peter van Rooijen, a frequent attender at ESUG and present again this year, has not paid this year's attendance fee. Repeated, courteous, requests from each board member and from Isabelle to do so having been ignored, the board were left with no choice but to ask him publicly to pay the fee, as everyone else in the audience had already done. This being again refused, the board announced their decision that Peter van Rooijen is banned from all future ESUG events and activities (including any Camp Smalltalk that is ESUG-related; other CS organisers may decide for themselves) until he settles his just debts and offers a suitable expression of regret for the delay.

Having reported these facts, I will add some personal comment. I have in the past had several interesting discussions at conferences and by email with Peter and found him a distinctive personality (like most smalltalkers

:-)) with competent Smalltalk ability and variable ability to work with others. How the current situation first arose I do not know. None of the reasons Peter appears to be offering seem remotely reasonable to me. I speculate that a burst of temper initially caused by some half-forgotten circumstance is now maintained by that well known human behaviour, a determination never to admit error. However that may be, Peter has knowingly both received a benefit and refused to pay for it, an act that in ordinary language is called stealing. I will be very pleased to delete this section of the report, and to resume our occasional discussions, the moment I hear that Peter has settled the account. The longer he delays, the more I (and I suspect others) will feel obliged to describe his act, whatever its motive may have been, in ordinary language.

On a somewhat lighter note, I remark how much more amiable the ESUG board members are than I was in my Oxford University days, when I was part of the university territorial army unit. We did a sideline in helping organise (and provide security to) many events in Oxford, and we offered our customers what we thought of as a wide and comprehensive range of solutions to non-payment situations ('crashers' as they were called). However, "Let him stay but tell him he can't come again", was not on our list.

Conclusions

My fourth ESUG and as good as ever. I look forward to next year.

- Positive statements about Smalltalk web service frameworks.
- I'm pleased the K-Infinity / COAST framework is having commercial success. (I predicted it would go places in my past ESUG reports.)
- Some interesting remarks about Smalltalk opportunities and issues in the 'Using Smalltalk: Experiences and Opportunities' section talks.

APPENDIX: ESUG Education Symposium Follow-up

Smalltalk: the Language without (unwanted) Baggage

Note: This is my reponse to the request made in the education symposium for a short explanation of why smalltalk is better, aimed at an audience of computer science students with little commercial experience. (Ideally, they would have had a lecture on XP before hearing this but that is not essential.)

Everyone thinks their language is better. They can't all be right, but equally they can't all be wrong. So it's worth hearing and considering their reasons.

Why Smalltalk is better than its main commercial rivals, C#, Java and C++, is best expressed by the term Smalltalkers use to express the most obvious technical difference between them. Smalltalk is a dynamically-typed language. Its rivals are commonly called 'statically-typed languages'. But a Smalltalker will often call them 'stiffly-typed languages'.

This isn't simply a piece of abuse (though it's certainly that :-)). It expresses succinctly the feeling you get if you build and maintain real

systems first in Smalltalk and then in C#, Java or C++.

- Firstly, the rival language seems ‘stiff’. Like a bad employee, it resists your attempts to make it do something, and resists more your attempts to make it do the right thing.
- Secondly, much of this ‘stiffness’ seems to be related somehow to its static typing.

Smalltalkers will talk for hours to anyone who will listen about what the relations are between static typing and poor productivity. I plan to talk for a few minutes, not about *what* these relations are, but about *why* they are.

I must begin with a confession. When I was young and starting my career as a software engineer, I was a fool. You, I’m sure, are all clever enough to recognise instantly that this seemingly humble confession has two not-so-humble implications. Firstly, that as I am now old-*er*, I must be wise-*er*, and secondly, that as you are young, you may be as foolish in your opinions about what makes good software engineering as I was at your age. However that may be, I know that when *I* was starting out, *I* accepted uncritically a lot of very silly ideas about software engineering:

- that errors cost more the later in development you find them, so you should spend a lot of time up-front finding them all in the design phase
- that, ‘Why aren’t you coding yet?’, was a wicked question asked by bad managers, which should always be ignored
- that ‘Coding from a design is like walking on water; it’s easier when it’s frozen.’

These were truisms of software engineering gurus when I started out; by many, they’re still believed today. I believed them because I was stupid enough to think that I was clever. That is, clever *enough*; clever enough to know early in a project what I was trying to do and how I could do it. The process of learning that I never did was long, embarrassing and painful. The process of learning that my team leaders and managers never did either was long, infuriating and painful.

One day, when I had learned this lesson but not yet thought to learn from it, I was taught the discipline of eXtreme Programming, which you may know. It says the way to deliver good software is *first* make it run, *then* make it right, *last* make it fast. To the obvious objection, why not first make it right and then make it run, XP replies that that shot is not on the table. Noone is clever enough to make it right without a running system to study. The only real alternative is first to make it *wrong* and then to make it run. ‘Wrong’ does not just mean explicit error. It also means building features the customer doesn’t want that use up time and get in the way of features you then discover *are* wanted. ‘You won’t need it’, is an XP motto.

The third part of the mantra, ‘last make it fast’, is saying the same thing. The optimisations you haven’t already got by making your program ‘right’ will be ones that make your program more computer-friendly and less human-friendly. They will trade flexibility for speed. In short, they will make your program ‘stiff’. Since your program will not be right till well on

in the project, this trade must be made as late as possible. XP guru Kent Beck expresses it thus, 'There are two rules about optimisation. Rule 1: do it later. Rule 2: see rule 1.'

Now we can get back to our subject. Statically-typed languages are 'stiff' because their very design violated these rules. The type system is an optimisation built into the basics of the language by designers who thought they could work out in advance what was needed. Smalltalk's designers told themselves, 'We won't need it.' (It's no accident that the theory of eXtreme Programming was invented by Smalltalkers.) Because Smalltalk is very flexible, researchers have since written acceptably-performant utilities that add static typing to Smalltalk. They haven't caught on because, as everyone who writes Smalltalk finds, you *don't* need it.)

Just as XP doesn't say what features of a particular stiffly-built project will inhibit its success, only that there are sure to be some, so I haven't tried to illustrate what precise features of C#, of Java and of C++ cause each to be less productive than Smalltalk, only to show you why there are many such. Listing them all would take a very long talk. There are several good papers on the web, but it's better to find out for yourselves. Get adequately trained in Smalltalk and in one of its rivals. Do a non-trivial project with a real customer that isn't yourself, first in one and then in the other. See for yourself.

> ***Note:*** those who were in the symposium will now see why I responded to Andy Berry's suggestion that we all write a paragraph with a request to be allowed a little more space :-). If anyone knows how to compress the above thought into a paragraph, such that a student who lacks the relevant experience will nevertheless understand it, I would be glad to see their version.

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

---

* End of Document *
