Smalltalk Solutions 2003, Toronto, 14 - 16 July 2003

I spent the preceding weekend with friends at Oakville, 20 miles west along the lake from Toronto. It struck me as a very pleasant place; if Procor were hiring (see Val's talk), I think a Smalltalker could do worse.

A visit to the hockey hall of fame on Tuesday night was fun for the food and discussions. I also learned a little about hockey. My wife's description of her schooldays had indicated that in English schools (where hockey is the standard girls' game, as cricket and rugby are the boys' games) it is played according to Clauswitzian principles; after you have destroyed the enemy team's capacity to resist by wielding your sticks so as to maim or incapacitate several of them then you can score as many goals as you wish. (Joanne Rowling's descriptions of quidditch matches gives the style; perhaps she played in a similar team in her schooldays.) However the hall of fame displays suggested that transatlantic hockey players used their sticks primarily on the ball rather than their fellow players.

Cincom also stood us dinner on Monday and ice cream on Wednesday.

Style

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

Author's Disclaimer and Acknowledgements

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

Summary of Presentations

I have sorted the talks I attended into various categories:

- Applications and Experience Reports
- Refactoring and Testing
- Agile Development Processes
- New Frameworks
- Vendor and Smalltalk BOFs
- Porting and Miscellaneous

after which I list Talks I Missed, describe Other Discussions, note some Follow-up Actions and give my overall Conclusions from the conference.

As there were often three and sometimes five parallel programme tracks, plus ad-hoc discussions, I could not attend, still less report on, half of what happened. (Some of the choices forced by the schedule were painful; should I listen to Don Roberts and John Brant on Refactoring or Avi Bryant on Seaside?) The talks' slides should be reachable from the conference website (http://www.smalltalksolutions.com/). James Robertson's blog posts for the relevant days may cover some talks I missed.

Opening, Alan Knight of Cincom and Allen Davis of STIC
More than 140 people pre-registered for the conference and some more registered on-site. This is down on last year but good in the current economic circumstances and given the SARS cloud hanging over Toronto (I know some people cancelled for that reason and suspect there were others). Alan thanked the attendees for braving the dangers of Toronto. As someone who believes we're all far too risk-conscious these days, I will remark (pointedly to any who stayed home for that reason; of course, many stayed away for other reasons) that I found crossing the road in Toronto far more cause for concern than SARS, given Ontario's interesting 'cars can turn right against a red light' rule (at least with 'priorite a droite' in France, it is the driver who practices it who assumes the position of greatest risk).

Allen thanked the conference sponsors and Joy Murray (STIC), Andrew Ignatow, Alan Knight and Jim Robertson (Cincom), Jason Jones (whysmalltalk), Ginny Ghezzo (IBM), Monty Williams and others I could not type fast enough to record; my apologies to whoever I missed.

Exhibitors
The Cincom stand raffled a GPS (I think it was; I didn't win it :-/) and were handling out VW7.1 and OS CDs, etc.

The Smalltalk world is (too :-/) small, still it was a coincidence when I met Lisa Almarode who works for GemStone and is married to Jay, whom I last met at Lake Windermere in 1990 when he presented on GemStone and we ran an OODB BOF together. Alas, I gather from her that while she is keeping the Almarode name in Smalltalk, Jay, though still regretting Smalltalk, now works in Java and does not expect to have the opportunity to return. At least GemStone has fully returned from its former ParcPlace-like state of two-three years back under Brokat. As with VW, you can't keep a good product down; strange that some managements tried.

Totally Objects and I have agreed to restart the UK Smalltalk users group. Expect to see some web pages shortly, gradually developing into a site with useful UK-relevant Smalltalk information: talks, users, resources, etc. David Pennington glanced at a map to plan his trip to Toronto. Accustomed to British distances, he thought Connecticut looked quite close enough for Toronto to be a short drive from an airport there. Some 500+ miles and six+ hours later he arrived, vowing always to check a map's scale in future.

Silvermark's Test Mentor addresses widgets by name if they are named or by generated name if it can, but by path from enclosing subcanvas or window if duplicate names make this necessary. It would doubtless be

possible to make it always use the logical path approach (i.e. as precursor to revising the generated widget-addressing test code to be logical feature invocations, so less brittle).

Reiling provides utilities for GemStone DBAs: self-restarting daemons, etc

Applications and Experience Reports

Procor Experience Report, Val Balovnev, Procor

(The beginning of this clashed with the end of a competing talk so I missed a few minutes.) Procor is a rail company. (Special offer: come to Val's talk and you'll get a free ride. When you get to the talk you learn it's on a tanker car as that is their business :-)). Procor is based in Oakville, Ontario. It has Canada's largest private rail car rental fleet.

In 1996 they moved from Enfin Smalltalk to VAST. Every fortnight they have a problems and answers session for all Smalltalk developers. They also have a 30 minute tips meeting every week. They push design patterns and Smalltalk books on all developers. They have standards for web app appearance, etc. They have four teams of 3-4 each and a couple of experts.

They have 40 VA apps, of which 26 are web-based (intranet), one on internet. All apps are specific to a line of business. Some are also used by their U.S. affiliate (Union Tank Car company). Why 26 web-apps, not fewer? There are advantages (independent simultaneous development, Line-of-Business specificity) and disadvantages (duplication of business logic, integration difficulties).

Their repair shops, etc., are widely scattered so web apps were attractive (needs no client installation, etc.; the limitations of web connection was also a plus, not a minus in their environment). They use iPlanet web server, DB2 and VAST, and have some in-house frameworks for security, cascading web menus, etc.

As their web app suite grew, the need to share business logic became more important. They have left server workbench and web services for future exploitation, concentrating on the last of the following possible reuse approaches:

- put business and mapping objects in every app: this raises versioning problems
- pass user to relevant app whenever reuse needed, passing login ID, etc., in URL: this would need encryption and pushes the limits of what can be passed in the URL
- Use apps as servers: App 1 requests info from App 2, gets back info, display in App 1 UI. This is useful for integrating legacy apps but is still a URL-based request, so needs complex URL data formatter
- share objects using XML and MQ: this is the route they have chosen and are developing. Each app talks via an MQ queue to an XML app sitting between the web apps' queues.

The user can pull data by logging in and navigating. They also push data: user defines requirements on login and they email the use with this data daily, or provide it whenever they login, fax it, whatever. The key business value of this is that it reminds the customer they exist.)

They use HTML, XML, JavaScript and CSS on their front-ends and use their imagination to exploit Smalltalk's power on the server.

Q. Web browser conflicts. Most customers are intranet, so can be told what web browser version to use. External web app is simple on client, complex on server. They have many problems with the back button therefore much data put in session data. (He will be going to Avi Bryant's Seaside tutorial.)

Smart PVS, Vincent Disselkoen, Swiss Mobiliar
(I missed the start of this talk as it clashed with another but had background from talking to Vincent last year at ESUG.) VAST rule-driven insurance system. User supplies data (e.g. on web), data sent to model, model fires rules, rules determine what offers and further questions are now asked user.

Rule systems of this complexity are intractable although much checking can be done. Example:

- User asks car insurance, system takes no special action
- User identifies as sports car, system offers special cover, related offers

Instance variables seemed not well suited to capturing these relations (e.g. roles: did this person own, insure or steal the vehicle? :-)). Thus they built a system that modelled object nets generically and dynamically: <aPerson> owns <aVehicle>, <aCover> insures <aVehicle>

They have put their meta-modelling stuff on-line, not for commercial use but for study. Vincent overviewed how it handles relations, roles, etc. They have a `path:` navigation algorithm that walks over relationships to find things, so `self path: insuredObject` might start at a FireCover, work to a comprehensive cover, then to an insurance, thence to an insured car.

Data driven rules systems have complexity traps. Firstly, what language should be used for rules? If you don't use Smalltalk the product design department might be happy with some domain-specific language but the interface problem will be costly and changing as new features appear. So they chose Smalltalk and provided a graphical interface to help business users. Secondly, the rules and attributes differ in each specification so how to model methods and instances. They modelled dynamic Smalltalk. They could have specified them as strings and use the runtime compiler but it is 10 times slower than anything else. They could have made parse trees executable but that is very dialect and dialect-version specific. So they pre-compile to simplified parse trees (by sending `precompile` to constructed expressions). They have a 'Smalltalk in 12 easy classes' framework (BlockExpression, Assignment, VariableExpression, etc.) with which they can build smalltalk on Smalltalk. (I was reminded of Michel Tilman's similar framework for VW, in the Zork-Analysis package.)

Vincent showed how this worked in a simple example, building up a simple parse-tree-like object graph running to completion (detailed examples on slides). It runs 30% slower than native smalltalk. They use this to build dynamic classes with dynamic attributes. They chose to make using the framework quite distinct from using native Smalltalk. By contrast, I tend to make them the same, as does e.g. Michel Tilman in his Argo framework. So Q. Why not use DNU to integrate static and dynamic? (Much discussion ending with) if Vincent were the only programmer he would want this but as they have a turnover of people, often less experienced smalltalkers, he sees much value in forcing them to call `path:` to get a dynamic method to work on an object, so they don't get confused between static and dynamic.

Dynamic expressions and semantic nets worked well for them. They found writing the compiler for it was the hardest task. SmartPVS uses 100 business classes and there are 2000+ dynamic classes for their insurance products. Most of these are linked and putting it in a diagram is unusable (too large) so must be managed otherwise. They are stored in normalised tables in the DB (since 1994). DB2 COBOL (with a decent DBA) handles this fine. Since they don't want 2000+ changing tables, they hold the data in a meta-schema: a table for classes, a table for relations, a table for data.

They have data-driven security: they enable which actions the user is allowed to do based on their profile (a dynamically-modelled object just like their products). SmartPVS sessions can be long-running and may be reactivated or abandoned. They recognise a client from the IP address if they are on the intranet, or by object identity but that is performance costly. They are careful to configure each session and attach data to sessions to avoid mix-ups. SmartPVS offers a transaction per main business object, e.g. each partner and each contract mentioned in a session would have its own transaction. A single transaction for the session would be too restrictive; the user would like to work on a contract and e.g., add another vehicle then abort that without aborting the whole contract, to which they may have spent time adding data. SmartPVS has full responsibility for data quality: every host call to the DB is just commit or abort.

They communicate with the database via MQ (he skipped the example for time reasons; it is on the slides). From Smalltalk, he would rather use XML protocol. From COBOL, they could not find the right tool. They looked at tools that convert existing DB WS stores to XML but all the tools 'nearly work' (e.g. one had a convert-EBCDIC-for-numbers problem, another had another problem, etc.) so as they just had 50 calls they wrote them by hand. From an interfacing point of view, code generation is static so not ideal.

Lessons learned: do *not* put business logic in interfaces or have transactions across system boundaries. Use the Bart Simpson pattern over RMI and MQ. Use active rule systems with dynamic GUI. Use business objects as the anchors of your transactions. Marry OO graphs to normalised DBs.

Q. Download costs? The performance is like opening a 15-page word doc over the net, so its no problem for the system's acceptability. (Server start time is minutes as it reads all the meta-data but the users doesn't see that.)

Q. Resynching deltas? They use data-propagator (IBM tool) and almost never see a problem. The user can explicitly request a reload but they find the system very robust. On-line over wireless (narrow pipe) means they will keep this feature, otherwise they might be prepared to drop it. Deltas are checked coming back and if there are any problems then it is checked (e.g. sometimes someone makes a claim before their policy is complete!)

Q. Your dynamic framework is only 30% slower than Smalltalk; is that surprising? We certainly were, but the figure is reasonably verified.

Q. DB / OO understanding? They were very lucky in their original DB designer who was smart enough to understand the whole system and create a good (meta-oriented) schema.

Discussion: they find having VAJ in a shared Envy repository with their Smalltalk incredibly convenient for having a lightweight cross-testable Java front-end to their system connected via RMI. They don't want to lose that (so are a little concerned about IBM's WSAD enthusiasm). I mentioned Sandstone with their tiny VAJ wrapper to their VAST system; Sandstone don't care whether VAJ evolves or not since it exists and can wrapper them (and, being old and lacking latest Java features means it will work on all browsers, the old ones as well as the latest versions).

WebLogs and RSS, James Robertson, Cincom

A blog (web-log) is an on-line journal. To James, the point of a blog is that it does not have to go through the marketing department. Other companies (Oracle, even Sun but not IBM yet) use blogs for this purpose. James' and Dave's code is in public repository. Blogs are becoming common on technology, politics, hobbies. The Blogosphere is also where everything discovered 20 years ago is being reinvented, sometimes less optimally.

James' wrote a blog program as a demo of the VWwebtoolkit. His first thought was 'How hard can this be?' Gradually he discovered more and more things that a blog can do (or not, as it chooses). He soon discovered that he needed to edit misspelled posts, that it takes a long time to get readers, that he was the only one posting. Blogs are unlike Wikis in that they tend to have a single voice and to be massively cross-linked.

Most blogging systems store stuff as HTML and archive them in separate directories, fast to load but slow to edit. James stores stuff as BOSS. BOSS schema migration takes care of shape-changing.

(I remarked that, given the way the code works, having a daily BOSS file influences how much you see when you go to an archive post; some blogs move the reader to a old post within that week or fortnight of posts, which can be helpful for seeing its context, or just for tempting them to read what you posted the day before or the day after. When I looked at this, I altered the code to make the BOSS increment configurable.)

CSS was a mystery to him at first so he laid out pages in tables. This was wrong. He now uses a template.

He has a basic BlogEntry object that has evolved to handle comments, etc. When he started, he did not know VWwebtoolkit servlets too well. He should have used a get servlet but started with page Smalltalk code which grew and grew. His strongly advises to look at servlets first and avoid ssp pages until you find you need them.

At first he had ~5 readers per day. Then he got email asking him to add an RSS feed, something he'd never heard of. He learned (that RSS has its own politics so RSS formats have some surprisingly optional tags and there are modules that duplicate these optional tags, etc., etc., all making life harder for BottomFeeder and similar newsreaders :-)). RSS is a syndication format, needed because while some people go to a blog's web page and see if it has new stuff, others have a news aggregator that gives them a summary of which sites have changed and how much. At first, he added an RSS feed by building an XML document by adding tags, which is the stupid way to do it. Later, someone told him about SAX drivers and he used one. He wrote a small bit of code to parse RSS off the web and put it in the repository. Then Dave wrote a UI for it and so BottomFeeder was born.

(www.hebig.org/blogs/archives/main/000877.php has some aggregators.)

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

A blog increases its exposure by registering with various sites. You can also support trackbacks and pingbacks. In the blog world, XML-RPC is what people use, not SOAP. Roger Whitney (San Diego) did an XML-RPC VW app which Jim ported from 3.0 to VW7 and made work with servlets.

The other pattern people use is REST. It is simple, mostly just a URL, but sadly, few people use it. Pingback and Trackback *were* different (former was autodiscovery) but now they are both autodiscovery and there are four distinct but same-name-using forms of trackback (HTTP Get, HTTP post which are complex and bad, CommentAPI which is better, and a fourth).

The web toolkit had no way of doing REST as it did not expect anyone to reply with raw content added to the middle of the post, so he had to write a method #rawContent to do it. It is a multi-path method only because he wanted it to work with several versions of the toolkit. On any one version, it will only go down one route.

Cincom use RSS feeds for bug tracking, the internal and public Store DBs, the Wiki,... Some Microsoft projects use blogs for internal communication.

JWARS Performance Tuning, Donald McQueen
JWARS is a military simulation tool written in VAST, used by the DoD to run scenarios, e.g. if half my satellites are lost, how many aircraft will complete their missions. Donald has described JWARS at past conferences. This talk reviews their performance experience.

Setting OldSpace to the maximum they needed halved their overall time required. This is worth doing on a packaged images as well; it reduced a sub-detected-by-sonar run from 4 hours to 21 minutes.

They used GDK from OTI but without the internal cache and transaction support. They also use packaged images and Oracle. Server Smalltalk has a limit of 64k for object transfer; they increased that to 8Mb. All scenario data is stored as BLOBs. They found Abt was caching the last 10 SQL requests but not reusing buffers when retrieving BLOBs; they halted caching of blob retrieves and arranged creation and reuse of adequately-sized buffers. (10 megabyte BLOBs degraded the object dumper a lot.)

Early on, Smalltalk was 10 times slower than C/C++ at floating point, so they looked at wrapping floating point calculations but found they were not frequent enough to justify it. JWARS incorporates some validated C code for global terrain and coordinate systems, and for distributions. They did these as platform functions at first, then reimplemented them as primitives and they were much faster.

Donald showed a simulation of their 'Iraq 2010' scenario which (as he remarked :-)) had been 'overtaken by events'. Red forces seemed to flight more effectively than Saddam's did in reality and sent a force to invade Kuwait leading to a major battle. They model behaviour (concern for casualties, aggressiveness, etc.) and doctrine (e.g. soviet tank doctrine differed from western).

He ended by describing a few of the comical teething troubles they had in JWARS early days.

- A tank transporter was ordered to stay 30km in rear of HQ. When the HQ retreated, the transporter moved into enemy territory to stay in 'rear' of where its HQ was now facing. When the tank transporter reported seeing a Scud launch, they realised it was closer to the enemy than tank transporters are supposed to be.

  The first cut of the satellite code reused the aeroplane code to 'fly' the satellite, so satellites were requesting permission to take off from airports before they would orbit. Likewise, their first cruise missiles flew to target - and then flew back to launch point again.

Q. RDB issues; did you consider using GemStone? GemStone did not handle all required platforms and (so Donald was told; he was not there when this part of the project was being set up) were not too responsive to suggestions that they expand to cover the remaining ones. Technically, he would far rather have used an OODB.

Q. How has your success affected DoD willingness to use Smalltalk? DoD is a big place and we still sometimes get asked, “When will you rewrite this in Java?” Our collaborator, CACI, are now readier to propose Smalltalk work in government contracts but it still has low overall visibility.

Refactoring and Testing

Secrets of the Rewrite Tool, John Brant, Don Roberts, The Refactory Inc., www.refactory.com/Software, brant / roberts @refactory.com
The refactoring browser was first released in 1994 so is about to have its tenth birthday. The core of the RB is the rewrite framework and this is exposed to the user.

A refactoring is a transformation with several preconditions. This tutorial talks mainly about transformations as preconditions are fairly easy. (An example of a not-so-easy precondition is extracting an instance variable to something you reference. Proving this is a refactoring requires proving they are in a one-one correspondence, which is very hard. However, if you created the class in an immediately prior change then you know that the post-conditions of the class-creation refactoring satisfy these otherwise unprovable pre-conditions.) The refactoring framework supports

- custom refactorings: generic rewrites that the RB does not currently provide
- bulk transformations: your project needs to change a project-specific pattern to a new form
- changing layers: e.g. build a new DB layer, find and change 17,000 references to old layer
- migrations: e.g. synchrony’s VSE to VA tool uses the rewrite framework.

Parsing is key. A method body is free-form text conforming to a grammar, a set of rules. Don showed these rules in BNF. Equally, these rules let you derive the derivation tree, with the method text being chuncked into the leaves of the tree, i.e. the terminals (selector symbol, varName, etc.).

John and Don simplified the derivation tree into a parse tree which, for example, has only a single class for method name, with unary, binary and selector just being a data state of the method name instead of further branches in the tree, and other terminals being instVar values of their owning non-terminal nodes.

John and Don had to think about comment placement and formatting which most parsers don’t care about. In the latest version, wherever possible, they do replace-in-place to avoid reformatting the method.

The rewrite tool is a parse tree matcher whose language is a superset of smalltalk, which makes for both ease of use and ease of reuse. The rewriter can search and it can also replace (i.e. rewrite). One use of search is Smalllint (now renamed Code Critic). One use of rewrite is to do the refactoring browser’s refactorings.

Pattern matching is accomplished by unification between the pattern and the code. The input is the tree to search and the pattern to find. The rewriter tries to find an assignment to the variables which matches and returns the found boolean plus this set of assignments if found. Matches can fail though incompleteness or through inconsistency, e.g. `X := X` pattern matches `foo := foo` but not `foo foo` (incomplete) or `foo := bar` (inconsistent). A rewrite then takes the matched variable assignments and outputs them in the requested rewritten form.

First easy error: the rewriter is not a textual search and replace tool, e.g a variable replace will not replace a method name call. (Use Vassili's regexpr utility if textual search and replace is what you want; sometimes, of course, a purely textual search and replace is the right solution.)

John showed it finding and rewriting smalltalk code, displaying the changes inspector. (This inspector gives you a moment to pause and reflect, letting you bail out if the rewrite has found much more or much other than you thought it would, or rewritten it quite unlike what you expected.) He then showed changing some of the smalltalk into pattern variables and rerunning the rewriter, displaying the proposed changes in the inspector.

- Simple variables: `‘foo` - matches any variable or pseudo-variable
- Simple literals: `‘#foo` - matches any literal, nil
- List or subtree variables: `‘@foo` matches any subtree, and can be thought of as ‘match anything’ (any var and any literal)
- recursive search - `‘‘@foo` will look within what it has found, e.g. `‘‘receiver parent` will match both cases in `x parent parent`, whereas without the second backquote only the outermost case would be rewritten. Rewrites that (could) match expressions with blocks in them often need recursive search.

To match an entire statement, use a period: `‘.statement` will match a statement. Thus

```
‘.duplicate.
‘.duplicate
```

should match two identical consecutive statements that are the whole body of a sequence node. However as soon as we get beyond a within-statement expressions, we are matching sequence nodes. Matching two statements within a sequence node requires

```
‘.@beforeStatements.
‘.duplicate.
‘.duplicate.
‘.@afterStatements
```

Because the . makes the tool build a sequence node, you *must* provide the ‘zero or more statements before and after’ code, unlike the expression case where it could match an expression within a longer expression. If the sequence node has temporaries, the above will not match. You must use

```
| ‘@temps |
‘.@beforeStatements.
‘.duplicate.
‘.duplicate.
‘.@afterStatements
```

which will match two duplicate statements within any sequence of statements. It can be tricky to match sequence nodes; Don admitted he usually took two or three goes to get his expression right.

(Q. Have that template appear when the tool comes up, as a method template appears in standard smalltalk? Some discussion, including of adding it to the Custom Refactoring project’s right-click menu.)

They then showed renaming methods, noting that the change inspector only showed the new method as the old had been renamed.

Q. Anyone use this for code generation? Usually not so much for raw code generation as for bulk code alteration. Will Loew-Blosser’s paper at OOPSLA 2002 described using it to replace one database layer with another. Will made 16,000+ changes using 200 rules that he scripted to change all DB access code over a weekend for a system that had to be up and running again on Monday (database access times were reduced from 40% to 2%). Aaron Hoffer made a similar number of changes at Washington Mutual, and nearly lost his job. The changes worked fine but his managers told him that 10,000 changes to the code base in a day rang every metrics-based alarm bell they had. Don wrote rewrite patterns to change a large system from old to new ANSI-standard exceptions; it took him half-an-hour including writing the patterns.

You may want to match methods with a specific number of arguments, e.g.

```
‘@receiver ‘keyword1: ‘@arg1 ‘keyword2: ‘@arg2
```

or any number of args,

```
‘@receiver ‘@keyword1: ‘@arg1
```

or a mix of specific and general args,

```
‘@receiver at: ‘@arg1 ‘@keyword1: ‘@arg2
```

which will match any call of a two-parameter method whose first keyword is `at:`, noting that in e.g.

```
(conferences at: #SmalltalkSolutions put: self agenda)
  at: #RBtutorial ifAbsentPut: #excellent
```

only the outer at-method would be matched because `‘@receiver` does not request recursive matching; write `‘‘@receiver` to match both. Likewise `‘‘@arg1`, `‘‘@arg2` would search for further matches in any matched args.

John remarked that he suspects an issue in the specific keyword number matching. I agreed; browsing the code I had found a case where I thought

only the first keyword would be checked. We have refactored this code in the Custom Refactoring project and believe our release fixes the issue.

Programmatic matching uses a curly-brace-delimited smalltalk block:

```
‘{:node | block must return boolean: match expression?}
```

For example, to match any variable whose name begins with RB, use either

```
‘{:node | node isVariable and: [RB* match: node name]}
```

or, using the standard meta-language to do the isVariable check,

```
‘var ‘{ :node | ‘RB*’ match: node name}
```

(The Custom Refactoring project has added wildcard matching to expressions. If the Custom Refactoring code were loaded - and you were using underscore as the wildcard character - you could do the above with

```
‘RB_
```

Obviously, block-matching remains very valuable in more complex cases.)

Another example: ANSI says that 3 - 4 should have a space before the four whereas VW does not require this. A rule to find all such cases is

```
(‘@a - ‘#literal) ‘{:node | node selectorParts first
start = (‘#literal start - 1)}
```

The parentheses above (i.e. the first pair of ordinary brackets) get parsed away but define an expression to which the block applies, just as, in the example further above, the name matching block applied to the preceding `‘var` expression. The expression matches the minus sign and so has to be a message node. John put a self halt in the above to show how they contrive to bind `‘#literal` in the block to its value in the dictionary.

Search blocks can take two arguments: the second is a dictionary, which communicates between search and replace. You can reference the matched pattern variables (e.g. `‘var`) and/or just stuff something (#foo in the example below) into it. For example, to add the letter a to

```
SEARCH
‘var ‘{ :node :dict | dict at: #foo put: ‘a’ , node
name}

REPLACE
‘{:dict | RBVariableNode named: (dict at: #foo)}
```

RBVariableNode appears because, to replace, you must return nodes, not just text from blocks. A replace block must return a node just as a search block must return a boolean.

The rewrite tool can be used to restrict the scope of a rewrite to just those packages, classes and methods you select (in VW7; in earlier / other

dialects, select applications / categories and classes, and, with Custom Refactoring code loaded, protocols and methods).

Any rules you add to the Smallint classes are automatically integrated into the Smalllint tool. Look at ParseTreeLintRule on the class side for examples of those rules that are based solely on parsing. Any method that appears in the five rule categories on the class side of ParseTreeLintRule will appear in the Smalllint tool (trivial hack on superclass' method will let you add a sixth category that will also be recognised). John demonstrated by adding a rule to turn isNil ifTrue: to ifNil:.

ParseTreeTransformationRule works much the same. Each item within these classes is an array in which the first value is the pattern for what to find and the second is the pattern for how to replace it. You can add to these lists or add your own rules.

Q. How find an expression that accepts one of four methods in given place? Use a block or write four rules (or load the Custom Refactoring project and use an OR separator - available from the right-click menu - between each of the four matching expressions).

Change objects: when you change the system you should have an object that represents that change. That's what the rewrite framework does. All subclasses of RefactoringChange know how to execute themselves. They also all implement asUndoOperation which returns the inverse change. They also effect the transactional behaviour. These changes are added to a CompositeRefactoryChange that is only executed after all is prepared.

To demo this, they wrote a script to create a CompositeRefactoryChange. If the changes are related, e.g. because performing them all will move the system from one consistent state to another, then a composite change is better than doing each change individually because the latter means you must undo them all individually if necessary.

```
changes := CompositeRefactoryChange named: 'testing'.
changes defineClass: '.... class def text'
changes defineClass: ....
....
```

Don explained that they have the RB's model, containing RBClass objects, and real classes are created after you execute the changes (alter RB model); and then effect the model into the image. Because the rewriter has this transactional model, it can (almost always) rewrite itself. The only code they cannot safely refactor is some code in these change classes. (They wish all of VW would use change objects; accept something in the VW debugger and suddenly the RB's undo goes grey since you can no longer trust that your undos are safe.)

They then wrote a script creating a composite change that renamed all their classes from RB* to BR*. John then called `primitiveExecute` on the composite instead of `execute` as the latter calls `primitiveExecute` and

then adds the change to the RefactoringManager for effecting into the actual code. He then inspected the changes. Later, he did them and went on using the tool, as an example of the Refactoring Browser refactoring itself.

Lastly, John discussed using the tool in migration. The model helps this by using SmaCC (their generic parser utility) to either change and then export code without first recompiling it, or to import and change code before recompiling it. SmaCC is three times slower (when just parsing smalltalk) than the RB parser. SmaCC can do error handling much better than the RB.

They are adding a new refactoring: Extract Block.

Q. Binding References? Matched as literals.

VWUnit, a Framework for GUI Testing, David Buck, Simberon
David did some work for a company whose software manages bonds, etc.: when to issue coupons, make payments and so on for trades, swaps, etc. A significant number of people spent 2 weeks every release just looking at screen snapshots and comparing them to the current release.

Screen snapshots did not capture order of entry issues (e.g. enter USD then CND and get conversion rate was not the same process as enter USD then conversion rate and get CDN, and could induce changes further down the line of the overall process due to rounding issues, etc., but looked the same on the screen), so snapshot pages got covered with little notes. This had to be fixed. They also wanted to use XP for UI testing. They were already using SUnit for non-UI and now wanted tests that would match exactly the screen values that were currently being used in the screenshot by-hand comparisons. The test framework also had to handle pop-up dialogs.

They liked SUnit and so built a framework around it. They had to run tests in a given order since some tests created states for other tests (e.g. test on swap requires trade and instrument to be already there). Thus they overrode buildSuite to create an ordered suite. Another problem was the fact that tests tended to save to the database, thus second run would encounter the object already there from the first run. The solved this by adding unique timestamp suffixes to object names so reruns would not clash.

Their apps had many levels of subcanvases within subcanvases. They found these hard to navigate so decided to locate items on the screen based on name only, e.g.

```
harness rightOfLabel: 'Name:' enter: 'John Doe'
```

(Reinout has added a feature to locate by name or id.) The method allLeafWidgets gets the collection from which they search in the above methods.

Widgets are sometimes are not there yet (screen still opening) so they try a few times before conceding failure. There are various kinds of methods:

- locating methods: locateListBox (finds first, useful on simple screen), locateLabel: aString, locateRadioButton: aLabelString, locateTopLeftWidget, etc.)
- they fast flash widgets as they are located to verify behaviour (with 20 ms delay flash instead of standard 120ms flash to speed tests; this was an example of the 'profile, don't guess, about performance' rule; he thought allLeafWidgets was the problem till he profiled)
- Action methods: press{Radio}Button, pressCheckBox{On/Off}, atTableCell:enter:
- Combination methods: these combine location, navigation and entry: rightOfLabel: aLabelString enter: aDataString
- Group Boxes: withinGroupBox:do: can help to find a widget whose name is reused in several places.
- Dialogs: handling dialogs was hard; the following works but may not handle dialogs that themselves popup dialogs

```
harness performAction: aBlock onDialogPerform:
anotherBlock
```

- Tables: VWUnit supports tables but not datasets. They will add cell-by-cell comparison. To write a test to check a table, start by checking against an empty array. Run test, it fails, do cut and paste of table in failing assertion, now test passes; it's an easy way to get the table.

They can attach to new windows. They can print out the screen snapshots they test against by calls to self printScreen during the test to provide an audit trail for the end-users. They can scroll down when printing but the scrolling is not yet in the version in the Open Repository as it needs work for datasets, etc.

The can replace the test harness with a reporting harness which rewrites the smalltalk calls to user-oriented reports. As it happened, the users were more interested in the screen snapshots. They can also export to excel files to do file comparison. They use `performUIBlock:` to do all UI calls (I was reminded of my `inDialectUI: aBlock` for Custom Refactoring UI tests.) They now have 100+ tests and the whole suite runs in 2 hours.

David would rather have written non-UI tests than UI tests but his client wanted UI so he did the work for them.

Limitations: it can't tell whether a widget is or should be enabled or disabled. He wants to add dataset support, cell-by-cell comparison, better locate by name of widgets, menu testing support, ...

David ended with a demo. He ran a test case, a screen appeared, ran through widgets and closed. The test tool updates with the result as each test runs. You can halt runs, etc. (The tool's UI was roughly SUnitBrowser-like. David has given it an 'abort run' feature. I have also found such a feature useful, so added it to SUnitBrowser). Dave also showed the report generation, and changed the flash time to show how that affected speed.

Q. How do you validate? We have locate...:check: methods.

Q. Notebook tabs? We handle our Notebooks but not VW ones and ours are not in public version. Locating is the hard issue.

Q. Excel macros found by location? Some thought on this. They could have a script that captured the values to be checked.

Q. I asked David why he overrode buildSuite instead of building the ordered TestSuite he needed. David said it was just easier to use class names. (I think that the fact that TestRunner is not set up to accept Suites naturally tends to point people away from using them when they first encounter SUnit and the habit lingers. My tweaks to TestRunner to make it accept suites are in the Cincom Open Repository, in the StarBrowser / SUnit Extensions. I believe TestSuite construction is the right approach.)

Q. I suggested handling Dialogs by using method wrappers (this is how I handle dialogs in my own tests). Wrap the appropriate dialog method with one that instead gives the effect of its being clicked and closed, etc.

Q. The case sounded like one suitable for my deep comparison test approach (see my talk at last year's Smalltalk Solutions), albeit the scale was smaller so did not compel such an approach. David had noted my talk but was concerned about two issues.

- Resilience to additions: would adding a widget fail the meta-test whereas an ordinary test that ignored it would still pass? The latter was the behaviour they wanted. (I think default customising new widgets to not compare should be straightforward.)
- Widget comparison time-delay: as noted, they had to retry the comparison a few times and wondered if the meta-approach could handle this. (I thought it could without difficulty, but will consider.)

Q. Maybe open window and write to graphics context that does nothing, so you do all comparisons without the time delay of actually displaying. Interesting idea; Dave will think about this.

Q. Why not use Test Mentor? Dave wanted them to use test mentor but the last bureaucrat would never sign off so he had to do it under the covers. Dave thinks they would have been much better off with Test Mentor.

Testing Web Applications, Colin Putney, Whistler
(I enlivened the start of this talk by upsetting a water jug over the chair next to me, which made life interesting for Bill Dargel when he arrived and sat there while I was plugging in my computer and so momentarily distracted from warning him.)

Colin had what some Smalltalkers may have thought of as God's own job, working in the ski resort at Whistler, where they have a 20cm rule; if the snow exceeds 20cm, work does not start before noon. Except that I suspect his rates took account of the job's fringe benefits, it would seem ideal work for a skiing Smalltalker.

He wanted to test the application logic behind the web page. Separating the presentation from the logic began with cgi and developed to putting code inside html. This is not enough for what Colin needs. His framework reads in the template and processes it. It reads the source into a DOM, does XSLT transformations on it and serialises it back to the browser.

The Unit testing framework makes assertions about a syntax tree for html. Colin showed some html and its parse tree for a simple count-incrementing element. He uses macros, marked by @ ("@macro-name" in html). He did not intend to use the MVC pattern but it fell out so naturally from the work that he accepted it. The macro receives the tag and replaces it with an href (e.g. to a method of his Squeak Seaside application).

He has an @loop tag that creates an invisible parent node for the loop and thence outputs html for each cycle. The loop is a lexical scope for variables. Of course, one could have a Smalltalk loop that generated this code but the aim is to have the right separation between Smalltalk and html as these are written and edited by different people (smalltalkers and designers). The @loop construct lets you avoid writing code to generate html that a designer can just write. He tends to store templates in class-side methods.

Pages are served by the process: template is parsed, macros are expanded, html is served. In the testing case, the third stage is replaced by assertions on what is served. Tests can

- check that actions are bound to tags (checked via verifying that something somewhere in page is bound via callbacks)
- check that action has required effect (do action then check via verifying that returned value is somewhere on page)

Colin demoed both the above cases on a simple increment method: that it was bound, that the right initial and incremented values were on the page at the right stages. His tests simply verified that certain literals were anywhere on the page. The view classes have do: methods to iterate over their contents so tag-based tests can be quite detailed.

Q. (Niall) I have found that testing UIs via tests that exercise model functions through the UI are much less brittle than test point-and-click. This approach carries that idea to the extreme in that you don't check anything about the page's appearance, just that the right methods and values are in the served html. Should it be supplemented by simply printing (or otherwise capturing) the pages and making user look at them, the idea being that if the user likes the look of the pages then your tests, if all run green, ensure that nothing else can be wrong?

A. (Colin) The intent of tests enabled by this approach is to verify that the dynamic state of application matches the state of code. It says nothing about whether the html is white on a white background or similarly unusable for purely visual reasons. Any such bugs would have to be caught otherwise but, as you suggest, should be trivial to find and fix.

Agile Development Processes

Are you Agile or are you Fragile?, Scott Ambler, Ronin International

Scott is a Toronto-native and long-time Smalltalker, though he has also done work on the dark side (Java). He give a similar talk to this at a database conference and some people walked out, they were so unwilling to even think about agile. Scott began by showing a picture of Mo (the Iraqi Information Minister) and asking if anyone in the hall sometimes felt they were working work for him :-)). A little picture of Mo appeared on all the subsequent 'XP doesn't work' slides as Scott's comment on the quality and honesty of some of the arguments he was rebutting. He sees a lot of lies about agility on the newsgroups; false assertions made by people who haven't a clue and have no clue that they have no clue. He aimed to present what he thinks the real story and talk about what else beyond XP is needed.

No bureaucracy does not mean no modelling. Agility does not mean code hacking. There are lots of low-end developers hacking code and saying they're doing XP. IEEE computer last month had an excellent article giving evidence that iterative minimal-bureaucracy development *does* work. Design is so important to agile developers that they do it every day, not in a swiftly-forgotten phase at the start. XPers do a lot of work planning in the planning game; they just don't waste time drawing an MS project gant chart because that's useless bureaucracy.

75% of project teams are less than 10 people so naturally most XP teams are less than 10 people. This is not evidence that XP, any more than traditional methods, has a problem scaling to large projects.

Agile methods focus on software. CMMI has lost this focus. It is preached by the software engineering equivalent of accountants who can't use a spreadsheet, i.e. by software 'experts' who haven't coded for years. Try finding a computer book these days that talks about *not* burning the money, *not* wasting the budget, i.e. that talks as if the money that pays for all this process were *our* money. Scott would bet his own money on an XP team against any CMMI level 6 six-sigma team. He does not think six sigma enthusiasts would do the same. He always asks six sigma method experts, "Do you develop software at home". Most do not but a few say yes: he has never found one that does six sigma on their own software at home. By contrast, plenty of XP enthusiasts (myself for one) use it at home.

CMMI six sigma appeals to bureaucrats needing to justify their existence but less to people with real work to do. It is possible to work that way but it usually fails. The Standish Chaos report states that project success rates are dropping: 65% fail blatantly (project is canned outright or its use is never attempted), 85% fail in fact (not valuable or not used). People either aren't doing traditional methods properly (i.e. those who must do them must dislike doing them too much) or traditional methods aren't working.

The agile manifesto says that we must value one set of things over another.

- Individuals and interactions are more important than processes and tools. People working well as a team and with stakeholders matter more than a process.

- Working software matters more than comprehensive documentation.
- Customer collaboration matters more than contract negotiation. An ironclad contract is no use without a good relationship with stakeholders; you will fail. (You'll have a good excuse when the failure goes to court; is that the game you want to be in?)
- Responding to change is more important than following a plan.

The agile manifesto also has 12 principles; he talked about some of them. Mary Popendike, "A changed requirement late in lifecycle is a competitive advantage, *provided* you can act on it." Out-sourcing and other non-agile methods deny you that advantage, but make life easy for bureaucrats.

Agile software development is not code and fix. It is not an excuse not to document. The software industry is arrogant and assumes the stakeholders are morons. It spends their money on documents we tell them we must write. In an agile method, you tell the stakeholders that it will cost $50,000 to write a system overview document. Let them decide whether to make that a requirement or not; don't decide it for them. No more "create a software overview document because RUP says we must do that, hold a review because RUP says we must do that." Always ask, what value does that add. Justify these things. (And, I would add, try to ask those who will actually get the document's value, or lack thereof, and be paying for it.)

Agile is not an excuse to ignore enterprise concerns. Enterprise people (the database, internal customers) must change their ways to work with agile otherwise the developers will ignore them, as in fact happens today under traditional methods. Never believe IT people who say they must work serially, not iteratively and interactively. They can; they just choose not to.

Out-sourcing is happening because the business community are sick of being burned. Now they will out-source and be burned at 25% of the cost, which is at least a step in the right direction. This decade will see that shake-out occur; you will be agile or out-sourced.

There are several agile methods:

XP emphasises the planning game and the importance of tests (including customer/acceptance tests). An XPer's day starts with a quick paired design session followed by testing, coding and refactoring, with some integration (rerun of general test suite) checks. XPers talk to stakeholders frequently. There is not much of a niche for paper-pushers in this process.

SCRUM is a project management/project coaching method (see also remarks in Joseph Pelrine's talk). It presupposes a project backlog, a pile of use cases or other list of things to do. You pull 30 days worth of highest-priority items off the stack and sprint for 30 days implementing them (using XP or another method; SCRUM does not say). It is a rule that there are no requirements changes to what you're implementing in that 30 days (change requests can be put on backlog stack). Every month-end there is a go/no-go decision for the team - i.e. the stakeholders decide whether to pay

more or less budget to the team - and the backlog's content, and that contents' priority ordering, etc. are revisited.

SCRUM is for managers, not paper pushers; it asks for decisions, not reports. It has the problem that it does not help fabricate long-term annual budget estimates. In traditional methods, the accountants know that their next-year budget is in fact a fiction but they want to play the game of computing it, so may object to SCRUM because it interferes with the game.

Dynamic System Development Method (www.dsdm.org): this may have value for a very pro-CMMI organisation because it has some of the traditional ceremonies while being agile-ish.

Feature-Driven Development (www.featuredrivendevelopment.com): very like XP but with an explicit domain modelling activity up-front to drive the feature list (plus the model is further developed over time). The explicit model may make FDD more palatable to management.

The **Crystal Family** (www.crystalmethodologies.com, Alastair Cockburn): a family of methods; you pick the one that suits your project.

The **Agile Unified Process** is a marketing attempt to fit agile into RUP. The good side is that one can get close to doing XP in RUP. The bad side is that RUP speaks to bureaucrats so the people who want the RUP don't want agile and those who want agile don't want RUP. Thus a project of 10 people will allocate two people to the role of blocker - they do the RUP training, attend the meetings, create the stupid documents. He asked how many have done that (many hands) and how many have been caught (no hands). Not only do management imagine this stuff lets them manage, they're too stupid to catch us when we fake it. (He wrote an article on this a few months back; much feedback on it.) Have you been on a project where there was a big requirements document that developers didn't know existed or never read (several hands). This is the disconnect; traditional methods would work (not very well) if they were followed but they are not.

Agile Modelling (www.agilemodeling.com): on a project some years ago they designed a great architecture and then went shopping (for large sums) on vendors' products to be the architecture's platforms. They could not get these products to work together. You must prove it with code, not architecture. Agile modelling just makes this point within other methods that may do some modelling.

(I saw the same thing happen in a major project at a former employer, supposed to be the basis for a key new offering. The third-party products' licence fees made every seat very costly, the system had a huge footprint and the parts were hard to make interwork and even harder to debug, both technically and administratively since every vendor began by claiming it was one of the others' problems. Many months of up-front analysis and design of this important system had failed even to notice this might be an issue. Once construction began, it soon became obvious to those involved but the plan was made, the analysis done, the design set: the perceptions of

the skilled architects working on the project became a background noise of muttered warnings and moans that the project's CMMI process existed to suppress in the name of 'plan what you'll do, then do what you planned.')

When you have 20 configure-controlled models, you have 20 models to update when a change is proposed, a powerful incentive not to propose changes no matter how obviously needed. "How many people live on a street that is not on their road map" (several hands went up; I had encountered the phenomenon the day before travelling around Oakville with friends, and noted it as a difference between America and the U.K., where it would be most rare). Scott lives on an unmapped street but he will find his way home tonight. Q. How does the fire department find your house? "It's the one that's on fire!" (and the police? - "they manage" :-)). Barry Boem has spoken on the issue that traditional methods have risks as do agile methods.

Test-Driven Development, like agile modelling, this is a method fragment, not a complete method; it just says how to do TDD.

Agile Data Method (www.agiledata.org): how can data and enterprise developers work on an agile project? Operations people must be helped to get on board. ADM just defines principles to help this. (A fortnight ago, he talked to some data people who were convinced that there were less than 100,000 object developers in the world. Such 'living in 70s' people exist.)

Why does agile work. It is focused on quality every day, responding to change every day, working with stakeholders every day. You need people skills and some humility. The challenge is to get stakeholders to show up. For three generations, the IT industry has trained stakeholders to turn up at the start, sign off that we can burn this much of their money, then go away till acceptance test. Stakeholders must also be open-minded as we will be showing them our actual work state, not telling them all is fine till delivery.

Agile can shoot itself in the foot. So can its rivals. Agile shortens the feedback loop. If you believe that software is all about communication, then you want to work in the way that uses the best form of communication. Formal documentation is notoriously the worst form of communication. Standing round a whiteboard is better. Pairing on code is best. Even the best-run review is still a mistake. If you've invited people who are qualified to give feedback now, why were they not asked for their opinion earlier. If their opinions were not worth canvassing earlier, why are they asked now.

Software is like swimming; it's dangerous to do it alone. The heroes are the most dangerous people on your team: at best, they write good code no-one can understand, then they leave. (I was reminded of Edmund Burke's remark, after a lifetime's experience as a very skilled legislator, that, "I never saw any law that was not much improved by the comments of people far less able than those who had drawn it up.", from 'Reflections on the Revolution in France', quoted from memory.)

Specialists (e.g. a specialist use case modeller) are a serious problem in the

industry. Such specialists put information in their tools whereas it should be in the code. Have no specialists that can't read code and put information there. A requirements analyst that can't express information in code is like an accountant that can't drive Excel. (Scott has been in a meeting where sales people keen on web selling proposed delivering real physical products "as email attachments".) XP requires a degree of team work: the 'uber-techie' with no (bearable) personality is not wanted.

The 'Extreme Programming Perspectives' book has some success case studies and others are appearing in magazines. XP today is where objects were in 1990. It took a generation for objects to be accepted; XP will be the same. Scott works with researchers and notes that traditional methods have very little proof that they work. XP can get little funding for research; he knows frustrated researchers who would like to get their PhD studying the effectiveness of agile methods but can get no funding.

Why does the agile alliance work so well? Because most of them are smalltalkers.

Q. Selling to management? CEOs are as frustrated as the people in the projects. Their direct reports don't feed them agile for the reasons above. They are tempted to compromise (agile in RUP) as discussed above. Often you must wait till the company has shot itself in the foot several times before they will try XP.

Often middle managers prefer a larger team; it gives them more visibility. People who are not good at producing find political ways to justify their existence. To be visibly successful right away means that the politicians try to get involved. SCRUM states that 'the project manager's function is to be a shield, protecting the team from incoming flak'. Also, success can raise the level of expectation too much for others in the company. Scott worked in a team in a company that had not delivered in three years. His team delivered and were shown the door the day they delivered.

Q. If agility is where objects were 15 years ago, will we, 15 years hence, be using not agility but Javility: some dumbed-down version? Maybe; there's sure to be some, and indeed we're seeing some now.

Remote Development Panel, Dan Antion (chair), Avi Bryant, David Buck, James Foster, David Pennington, Terry Raymond, Niall Ross
As I was in the panel, I recorded others' points as best I could and my own not at all. Thus mine are added post-hoc from memory (and probably read more eloquently than they sounded at the time :-)). Dan thought design tasks were more difficult to do remotely than development tasks. He asked each of us to summarise our remote working experience. Thereafter, Dan introduced topics and we gave our views on them.

David Buck (Simberon): had a smalltalk developer at a remote site (i.e. his home) and needed to share code while writing Elastolab. (Also in POV-Ray project, remote experience managing source code). Remote end-users are hard since you need feedback (Chicago to Atlanta end-user was hard).

James Foster: two years ago, he was doing healthcare Smalltalk work. The company asked him to move which he refused. Instead he and colleagues got hired by an Atlanta company while working in Fargo North Dakota (which is pretty remote :-)). His experience is that a development team can be distributed. All his customers tend to be remote (e.g. hospitals in 17 states) so when he became a remote developer there was no change in that.

Conventional wisdom is that you must partition tasks and give some to the remote office. By contrast, they have been much more integrated into the overall team. James spends more time talking to people in Atlanta than to people in Fargo; they have intentionally arranged the tasks that way.

The Corporate Datacentre in Atlanta is half-an-hour from the main office in Atlanta; they're remote anyway :-).

Avi Bryant: does much remote Squeak consulting. His clients need source control systems for Squeak and he also pair-programs with them in Squeak, plus he participates in the Squeak open source activities which are usually remote. The open source world has large projects, e.g. Squeak has 50 people working on bits of it (OK that's unusual). Real design discussions take an hour face-to-face or a few days of emails; the chronology, not effort, can change significantly.

Niall Ross (eXtremeMetaProgrammers): mid-90's, I ran a team which at one point needed a particular combination of skills (Prolog as well as Smalltalk, etc.) which we could only source in someone who had to live 200 miles away. Most reluctantly, my then employer accepted hiring this chap to be on-site for two (consecutive) days in every fortnight, otherwise working remotely. He

- had to learn how to work remotely
- had to teach me how to run a split local / remote team

It was a very useful experience and I've built on it over the years. If I had to name one thing (as the panel progresses I'll mention several), it would be Keyboard-Video-Mouse-sharing tools. We started with NetMeeting (which I still like) and I have also used SameTime (Lotus notes add-on), pcAnywhere (not so good for this) and (recently) Citrix. I have provided a write-up of a remote pair-programming session using KVM-sharing which (I assume) will appear with the conference slides and papers on the web.

David Pennington (TotallyObjects): Dave runs the ultimate remote company as they are in the UK but 97% of business is international (mostly in the U.S. but he has also had customers in Sweden, etc.). Dave has been working remotely for Dan Antion's group for three years. He met Dan for the first time at a conference last year and has met him for the second time on this panel. Biggest remote working problems are hardware; 'It doesn't work on AIX connected to IBM' but TotallyObjects have neither machine so cannot reproduce error. Client uses Oracle, so TotallyObjects must grapple with installing Oracle. Dave noted the possibly advantages of KVM-sharing with his major clients, but if a client has only bought a £100 utility from TO then they will not arranged any KVM-sharing with TO.

Time zones are another issue. They work with clients in New Zealand; forget to send that last email of the day and you've lost an entire day in getting an issue resolved. Public holidays can delay you similarly because, "Oh yes, it's the fourth of July over there". He can talk end up talking on the phone late from home and thinking the next day, "Just what did I say to them after finishing that bottle of wine circa 23:00?"

A Swedish client just specified a DB problem and let TO solve it, which worked well. Had the client sought more detailed control, that would have made remote working much harder.

David Buck's pair worked one day on Elastolab each week, whereas David was working at a day job and on Elastolab in the evenings, so it was hard to coordinate. If his pair got stuck at 09:00, he stayed stuck till David was free. He found that remote working needed extra documentation, e.g. design documentation so his pair could read it when he was not there, and also standard processes (start up, etc.) had to be documented. This could be seen as an advantage.

Niall and Avi both commented on the advantage that more things get captured when you are working remotely. Avi found it helpful that a design discussion carried on by email let you revisit those emails to see who said what and why. Niall noted that if a diagram was needed to illustrate some code or design idea, it had to be drawn in a tool (Visio was the one we often used) instead of on a whiteboard, so was available to look at later.

James pointed out that time difference and space difference are not related. On a typical site, some people come in at 07:00, others at 10:00, so a degree of time-zone separation can be compensated for by different start-times and/or seen as another example of them. Only when the gap grows large is remote working creating a new issue. Terry Raymond (Crafted Smalltalk) in U.S. works for Soops in Holland. He starts work at 06:00 (noon their time) and uses IRC and email but not (yet) KVM-sharing tools.

Security rules can make problems for remote working. James' customers are hospitals and some believe they cannot show vendors any of their data so it is, "describe your screen please" when he is trying to fix problems for them. A KVM-sharing tool would be blocked by this requirement.

Face time: David and Dan worked with very little face time (met last year and at this conference). David noted that the first meeting with a customer after three years of working for him can be terrifying. Having met up, they use AOL instant messenger more and David is readier to pick the phone up to Dan than before he met him. Dan is the only one of David's regular programming clients whom he has ever met (he's met some who've bought his software at StSs) Niall said that remote XP is very doable provided you have met (and occasionally re-meet) the people with whom you remotely pair-program. I think it important to have their face in your mind and some knowledge of their temperament, but noted David and Dan's experience; this need could itself be a matter of temperament, theirs and yours. James,

like Niall, has always had the pre-existing relation. However you can get to 'know' someone from a blog or frequent posting on newsgroups. Terry visits Soops in Holland every two months for a week. The visits really help.

Next, Dan raised the question of configuration management. In Smalltalk, you need to pass code to your co-workers and sometimes need to pass your helper classes that live in your development image and enable what you're sending them.

Squeak's lack of a good version control system was a problem for remote development. Avi has worked on this. Nebraska is Squeak's KVM tool: it is very good and allows everyone connected to have their own cursor.

Niall found Envy quite good for export-import but hopeless for remote access due to its very chatty protocol. His team always ensured the KVM-tool shared an image that was local to its Envy. Because Niall had been in the habit of talking work home at weekends for years before the team began KVM-enabled pair programming, a process for how code could be worked on in local repositories and reintegrated with the master had already been worked out. It is straightforward to work out such a process (e.g. naming conventions that allow tracking of who changed things and from what base) but important to do so. My limited experience of Store suggests it will also be good, and better for working remotely from the repository ('better' here means viable; Envy was not viable). As regards the KVM-sharing itself, early Java-Swing conflicted with NetMeeting; Niall has not seen any other problems. James has found VA/Envy doable.

Usually, the client supplies any code TotallyObjects needs to do the work. When TotallyObjects sell their tools on the web, they make new releases only every year and a half, so purchasers find reloading and reintegrating acceptable. However TotallyObjects have to make loading their tool releases easy because those clients that they supply to directly may get a new release once every two weeks. Backward compatibility is similar.

David Buck found source code management the easiest problem to solve. Store in VW is designed for remote development. Postgres was easily internet accessible; no problem. His practice was to define an interface and then asks his co-worker to implement it; this avoided code clashes.

Soops publish and merge 3-4 times a day and Terry merges in the same way the others do. Cincom people try to avoid treading on each others' toes so have few merge problems. Niall found that the merging process was much the same as the local one because the remote person often pair-programmed with a local person by sharing the latter's machine, so the code was in the latter's view of the repository anyway.

Niall was asked to describe his KVM-sharing pair-programming process in detail (see Niall's paper for an example). The first need we found was for everyone in the team to have a headphone set for their telephones. These are cheap (tens of £s) and make an immense difference to the ease of the process. Non-headset handsfree phones pick up too much background

noise (especially from the machine, which they will be near) and also cause it (bad when several people in a bay are all working this way). Ordinary phones quickly cause neck-ache.

Using the computer to send the voice instead of a phone-line is possible and with ADSL and upward may be feasible. However good voice quality and rapid remote response are important. Only Smalltalk-specific tools like Tukan / Coast can work with 64k or less for the KVM-sharing. NetMeeting and pcAnywhere must have 100K+ (so ISDN is OK) for the KVM-sharing (I've been told VNC needs even more), so you must use a phone-line for the voice unless you have ADSL or better connection. Another significant motive for using a phone-line for voice is that if the weblink is lost, freezes or falls behind, the voicelink lets you discover and remedy it. The only motive for using the computer to send the voice, but it can be a compelling one, is cost. (SpeakFreely and picoPhone are good IP telephony tools.)

Easy swapping of control (either by having two cursors or by NetMeeting's double-click-to-take-control feature) is essential; pcAnywhere lacks this. I also like being able to share just the application I want, not the whole screen, and having the shared screen in an overall window that I can minimise. Again, this favours NetMeeting and SameTime over pcAnywhere (I've yet to explore the range of configurations Citrix offers).

A final point that took us by surprise but has been a great discovery is that KVM-sharing is also the best way to do pair-programming when you are *not* remote. If two (and sometimes it is three or more) people cluster around a single screen at a single desk then the screen, mouse and keyboard are (at best) at the right distance and angle for one of them, so that one becomes the principal programmer and the other is marginalised. This effect is enhanced if the machine belongs to the nearer programmer, as is often the case. Once we became fluent in KVM-sharing, we gradually realised that two people could sit at adjacent desks, share KVM and work exactly as if remote, except that, being in easy talking distance, they did not of course need to put on their phone headsets. This was an easier way to work. Each person could login as themselves on their own machine, see their own environment, have keyboard monitor and mouse at the right distance for them, and then share the Smalltalk IDE and pair-program. I now work like this routinely and find that people who pair with me soon get used to it and find they like it. It also improves remote working since local and remote pairing now feel much the same and use the same tools (except the phone).

James noted that pcAnywhere was designed for remote support, not shared working, and was not good at the latter. NetMeeting is windows-ubiquitous (but it does not work over a Cisco VPN). Citrix is good and may well be already present in related activities the client is doing.

Lastly, Dan asked how we should justify remote working to management.

James: remote development is bad but the alternatives are worse. If someone good is moving, remote working may let you keep them. If a deadline has to be met, remote working may let you meet it. Arranging to

work in the same site may just be too costly. Niall agreed. I once had a sudden deadline dropped on my team just before the Easter weekend. Getting the team to work on-site for four holiday days would have been impossible as regards our families and pre-arranged activities. By contrast, pair programming from home for selected hours *was* possible and thanks to it we met the deadline and won considerable applause. Someone quoted Sir Winston Churchill, "Democracy is the worst form of government - except for all the other forms." Remote working has its drawbacks but it may still be by far the most cost-effective solution to the problem of getting the right people working together in the right groups at the right times.

Dan will put his notes on www.nuclearInsurance.com/ftp.

New Frameworks

Croquet - A Collaboration Framework, David Smith,

Croquet is simply an example of David's real concern: innovation and the nature of innovation. Dave bought his first Mac in 1984 almost twenty years ago. Today that's what we have but faster and with more colours. What is new about what MS has given us? The answer is, nothing. Smalltalk, laptops, windows: it all came from Xerox Parc decades ago. ('Please restart your computer: failure to do this may result in lost revenue to Microsoft'.) Croquet came from his and Alan Kay's frustration about that (and Andreas Rob and Dave Reed, co-inventor of TCP/IP). Croquet is build entirely in Squeak (OpenGL at the bottom).

The internet is wonderful in what it could be, boring in what it is. It should be a broadband phone call. As soon as Alan et al got basic two dimensional stuff working in the old days, how to use 2D quickly became obvious. They think that Croquet will similarly drive the discovery of what to do with 3D.

He showed us two croquet screens, his and Mike Reuger's, showing a three-D world with mirrors, people (Alice and the white rabbit were his and Mike's avatars) and other objects that could be moved, resized, etc., by either of them. He spun the objects, reflected them in the mirrors, etc. Mike walked around too so we saw many things from two sides.

Dave Reed worked out how to make these objects collaborate. It is not a protocol. Objects understand messages and when objects are in the collaborative architecture they can collaborate on any message by default.

Other objects are portals: URLs represented as windows into another world. Dave went to Mars through such a portal and drove the lander around, etc. (BTW, look at the performance of this; Smalltalk's performance is so much no longer an issue !!!) Dave has only been an Smalltalk programmer for two years and the ability to modify while running is superb. Dave looked back at earth through the portal and grabbed an object on earth while viewing from Mars.

Dave raised and lowered a flag while Mike spun the platform Dave was standing on. ("Can't pull flag back instantly because it's a simulation and just like real cloth it would tear - ours just goes unstable and it's not pretty).

He went into an underwater world and Alice (his avatar) turned into a fish, as did Michael (White Rabbit) when he entered. This is a kind of security model. He then looked at web pages under water (eliciting the inevitable demo glitch: they were not using a web connection and the page was not cached on Mike's machine). He then drew a new fish in the Squeak paint package, inserted it into the world and 'pumped it up with air' to make it into a 3D fish. Andreas created this world in a few hours. This makes Croquet a powerful design tool, especially as you can work collaboratively (and remotely) with others.

He showed us a flash movie playing in an object. He then walked into a gaming world (fountains playing, impressive ruins, etc.). Alice walked up and along the high aqueduct to audience cries of 'Don't do it' from all and sundry (sundry were those who cried 'jump' instead :-)).

Croquet has internet telephony (built in Squeak). Snapshots are worlds into which you can jump, and jump back (like favourites); you just snapshot where you are.

He is working on a (kind of) 3D portal; a doll's house miniature version of the larger space. (Alan Kay, "A point of view is worth 40 IQ points.")

"Innovation is not dead but it's really been put on hold." A year and-a-half ago, Croquet did not exist. Three people (Alan just paid for it :-)) built it. Dave thinks this or things like it will make the next few years more interesting than the last twenty.

Q. At Stamford you said, "You hoped that this would grow to economic adequacy". This needs more than people building cool stuff in it. To change the world, it also needs an economic story. Dave agreed. They have done things to allow it to be secure, able to prevent bad code going to bad places and ensuring that if you use it for a phone call noone can listen in. Also, if you want to use what someone else has done, how can they charge you for it? This needs an economic model.

Q. What is on what machine? Teatime is their collaboration architecture and is still being completed (due this month). The demo was peer-peer; the worlds were on both machines.

Q. Where do permissions live? On the object: object owns its access state.

Q. Have you seen other 3D world avatar systems: Black Sun, etc. Dave has been doing 3D worlds for twenty years. He wrote the first 3D adventure game, 'The Colony' in 1987. He has always been convinced that 3D had to be more than just the worlds, just pretty pictures. He is disappointed with what people have done. The multi-player games are the best but in them the most complex thing you can do with a person is blow them up, which is fun but he doesn't think it scales :-).

Q. Pictures replace code? No, text programs are an efficient form of communication. Pictures let you do things quickly but not deeply.

Q. Can the avatars interact? Not in this example but the avatars are just objects like others so interaction is easy to program.

Q. Ten years hence, what will the clerk in a retail store see? Maybe the store, where the stock is. Doubtless there will be other apps; for sure, anyone would rather have apps written in this than in Windows.

Q. You've done this with Smalltalk's 30-year-old object model. Do we need more? Smalltalk has been wonderful so far. They are looking at parallel architectures and feel no obligation to maintain the status quo. Smalltalk lets you transcend itself; you can use it to rewrite itself. So they will do that as, if and when the problems require it. To date, they have changed very little. Teatime required compiler modifications and the scripting required some deep changes.

Q. Will the OS disappear under the impact of ideas like this? Dan has said OSs are bad. Everywhere Squeak (and a little 3D) will go, Croquet will go. OS will become less important.

Q. Will it be easy to shoot yourself in the foot in this? Alan's eToys work with children (Smalltalk was originally to teach programming to children) helps us build a system that avoids this. They have worked with non-programmer artists and found that when objects know *how* to do things then non-programmers can tell them *what* to do.

AOStA, Eliot Miranda, Cincom
Eliot started with the 'It's nice when things just work' car add. Smalltalk has optimisations (ifTrue:, etc.). To make it viable for 80's hardware, they cached native code. Thus each send site has a memo of the last send they performed. You get an in-line cache of what method was right for that receiver class last time so you need not do the lookup again for that receiver class. However you've also collected type information on your program.

Traditional static typers use type information to optimise their compilers. This does not map to Smalltalk; exploratory programming and concrete types do not mix. Another issue is that primitive types are costly to type check. Peter Deutch added go-faster primitives (assume type is right) but it was not clear how to generate them until self. Self had no direct slot access and no inlining so a simple in-line cache did not work. Self could not infer much information by looking at the byte codes. Thus Polymorphic Inline Caches were introduced (see Eliot's earlier talk, described at length in my report on ESUG 2000 in Southampton) as self has more polymorphism than Smalltalk (but Smalltalk has quite enough to make PICs valuable). The first call still does the full lookup. Later calls use the table (which will build up to some maximum like 8, after which you fall off the end and do the lookup as you would for a first call on that receiver).

Now you have type info for all of the sites except when the table overflows, which is rare (< 2%). You can now use that type information to build optimised methods (needs dynamic deoptimisation for debugger and for

redefinition). The self VM that used this shrank from 26,000 LOC to 11,000 and gave much more predictable performance. However the compiler still showed pauses and had a large cache.

Mid-90s, the Animorphic team solved this in applying it to Smalltalk and showed a prototype in 1996 at OOPSLA. This took 5 years of effort and Eliot just does not have the time to repeat this effort (original was lost due to the sad pause of 'DarkPlace DodgyTalk'). However Animorphic's coder is now available to be looked at and some good relevant books have been written. Hobbes (Smalltalk in Smalltalk) also gives information. Thus it would be nice to do it and to do it in Smalltalk. This gives the rationale for AOStA. Eliot and Gilad Bracha will try to do it in Smalltalk.

AOStA moves the VM's complexity and the housekeeping into Smalltalk. Bytecode -> Static Single Assignment info -> type analysis -> faster bytecode. The result is faster because of inlining, optimised primitives (no type checks), using registers as temps, and unboxing floating point operations (see e.g. discussion section in my ESUG 2000 report).

The send site counter used to be in the method prologue but this check of the callee prevents capturing calls from different sites. A simpler choice is therefore to count branches and it is faster. The optimiser must not change the code cache as it is counting so it must freeze and thaw the cache while counting and while applying the result of counting. The performance hit is 15% (10% from loss of static type prediction and 10% from counting when each is done individually, so Eliot doesn't understand why it only causes a 15% hit when you do both, though of course he's pleased about it).

Bytecode to bytecode compilation is easy since bytecode is well structured. Java dynamic optimisers have a much harder job to show gains due to primitive types; the result is that Smalltalk has much low-hanging fruit to pick. In-line primitives can speed up at: calls and instvar initializing by 3.5. Loops can show factor 10 gains.

Eliot showed a loop calling at: in smalltalk and in bytecode. You must check whether the receiver is immediate, is a pointer and whether the index is SmallInteger or not, then you must add the fixed field size to the index, etc. The whole takes 50+ instructions in an x86 machine. Eliot then showed the optimisation. The VM presents the type information as a literal array of branch and send data (bytecode number, then call number or class, method. This and the Static Single Assignment form guide the optimisation.

SSA form eliminates variables in favour of always knowing the type of any value. Reasoning over the SSA of his example, Eliot showed how the VM could deduce the possible values of a variable and so eliminate a type check. The correctness of a type on a call can be trapped for much as a PIC checks types (e.g. VM deduces receiver must be an array; traps if not).

You don't want to unbox floating points aggressively lest you thrash between implementations. Instead, the work is done on intermediate representations. The idea is to eliminate the unboxing of a + b in an overall

a + b + c and just unbox the final answer. The code generator will not generate the intermediate stack at all (except for the debugger when needed) but map it directly onto the floating point registers. Generally, complex heuristics are needed to avoid thrashing between bad (borderline?) guesses about what can be optimised. These can be handled much better at the Smalltalk level.

AOStA puts the optimising compiler in Smalltalk, etc. If the optimised guesses prove to be wrong they must revert (deoptimise) on the fly to the old PIC form. All this can be done in Smalltalk using contexts; we just give them methods to let them render themselves as deoptimised. (Test that deoptimising is perfect by partially evaluating the deoptimised and then partially evaluating; values on stack must be the same.)

The optimising is easy to turn off, so that you can do training runs, save and then do e.g. an animation run. You can also instruct the system to optimise infrequently done things which matter to you. You could patch the optimiser dynamically (e.g. download someone's latest from the open repository) and carry on.

Eliot started this October 2002. No floating point is yet done, no dynamic deoptimisation (except toy examples) has yet been done. Eliot has got far enough to discover that implementing a compiler in Smalltalk is a joy and doable in the existing commercial context, which doing it in C++ was not. He has had interest from GemStone, Dan Ingalls, Paolo (GNU) and would be happy to hear from others with compiler smarts.

Seaside, Avi Bryant
(I missed Avi's tutorial - John and Don's conflicted - but got a 15 minute summary from Avi on the last day.) Seaside separates the concerns that make web application development difficult.

- Firstly, it separates time concerns: it lets you express the straight-through logic of your app (the route that a 'know what I want to do and how to do it on this site' user would take) without being distracted by handling all the 'go back, go sideways, make mistakes and reverse' behaviour that users actually do.
- Secondly it separates space concerns. It eliminates much of the need for frames by letting components work in pages without being distracted by what they are a component of.

Task and Component are the two classes you subclass from to make most of your seaside components. They have the basic protocol (go:, etc.) you use to exploit the continuations.

Avi mentioned enough commercial applications to use up most of my fingers counting them: I recall an airline booking app, a university on-line CV app, an on-line store, Lucas' work (see my report on ESUG last year), Colin Putney's work (see his talk above), and there were others. Avi mentioned that while he had no reason to be concerned about it, it should be noted that Seaside apps to date had not been huge volume ones (airline

was a small airline, not EasyJet, etc.) A large number of simultaneous users in the middle of a large number of forward-back open continuations will cause a noticeable memory footprint (e.g. 100 users x 100k = 10Mb).

Cees de Grout offers high-end smalltalk hosting. Avi told me Colin Putney is considering offering low-end hosting; contact him if interested. (Bytemark offers virtual Linux hosting for £15/month but setting up base Squeak or whatever and restarting your app after crashes is your problem.)

Vendor and Smalltalk BOFs

Store BOF, VW7 Cincom

Alan Knight feels the fundamental Store decision (to use an RDB) is good but the implementation reflects the problems ParcPlace was having at that time.Various issues were mentioned: reloading a bundle over a slow connection is noticeably slower than loading it the first time as it has lots of 'is this later / changed' checks. Changing packageItemCount to comment out forRequest ..., replacing it with a simple ^1 makes some loads go much faster (but you lose your progress dialogs).

The merge tool should be cleaner, e.g. merging classes where instvars have changed. John Brant had bad experiences with the merge tool initially therefore rarely uses it (the effect was to make it look as if all methods had been changed). Jim Robertson advised not marking things integration-ready unless you mean it. Invoking merge from Store reacts to that. Travis has never had a problem with the merge tool. Jim agreed that once you learn its limitations then 2 or 3 way merges are OK. Don Roberts has found that one way of invoking merge was good and another was bad. Sames said the problem was the tool's failure to present a clear model of its limitations and way of working. I agreed to check the current 'Store usage patterns' Wiki page(s), add to them as needed and prompt others to record their experience. Cincom want to use the refactoring browser's tools for store, not the old browser. Thus they are refactoring the store tools' application models to allow this. Meanwhile, there is much inconsistency in the store windows functions. (Metaclass definitions are also going away.)

Some people asked Cincom to use community Store add-ons. However, user code tweaks to Store that work in one configuration may fail or have great performance problems in others. It takes much effort to verify them on all platforms, so cannot be done outside the standard release cycles.

Travis wanted to morph user names when publishing to distant repository? It just causes confusion if his work appears in the open repository under confusing usernames. He sorts his work under project names like 'xp', 'pdp' in his local repository where everyone knows who those names signify and they have a useful sorting function. When he exports to the open repository, these names should all be mapped to tgriggs or similar.

Vassili asked whether we should have categories as a final expandible level within packages. Most people were against or indifferent, very few in favour. Some proposed dropping categories, either entirely or converting them into searchable keywords.

The replicator can be used to move bundles with version information between databases without having to go through the image. John Brant pointed out that it's more performant to replicate from the far-away machine to the near one than the reverse, as the process is chattier to the replication source than to the destination.

Travis pointed out Diana Merry-Shapiro's useful utility (Method History, in the open repository) that assimilates Store and change lists to show all method versions you wrote in the current image and all in store in single list, giving an effect similar to what Envy did. It also has useful comparison features and is a good place from which to launch merges. (John Brant: FYI non-appearance of this utility's arrow widget on Linux is Linux bug.)

Travis also recommends the TOI configurations' package ('Store Extensions' would be a better name for it) for lineups: it uses the prereqs of a no-content package to capture lineups. By contrast, Bruce Badger and I think lineups should be bundles for reasons to do with test repeatability. I want to be able to load the exact set of packages on which I ran the tests and then versioned last week, or last year. Only a line-up that captured those exact versions, not merely the bundle and package names, would be acceptable. I am also conscious that the current tools are set up to display and load bundle contents well and prereq chains much less well. If pre-req line-ups are to become practical politics, they must be integrated with the UI tools; might it be better to address the reasons why bundles currently do not make ideal line-ups (and meanwhile maybe accept their imperfections instead of those of prereqs)? Bruce and I also felt that if a no-content prereq line-up was to be created, it should be a helpfully-named bundle (which people expect to be a line-up and to have no direct content), not a package.

Several people proposed giving Store an object model mapped to the RDB by an O/R mapping tool (a Store for GLORP package is in repository). This would let Store run as a server (and/or on GemStone), so deal with high-latency links, etc. (N.B. the issue is latency, not bandwidth. Store is less chatty than Envy but when link latency reaches 300ms, Store operations become unhappy.) Also you could put policies on the server instead of having to distribute a policied image, have the server run relevant SUnit tests, etc. You could also have the RDB only on the server, not the client. (Desire to BOSS out properties, etc., to swap between images.)

John Brant, I and others mentioned the need for useful blessing comments, which few provide, the difficulty of reaching the package comment when browsing unloaded items (can find it but it is slow) and the general problems which these facts create for people trying to find things in the open repository that might help them. I rely on the user name and my knowledge (from my conference reports, etc.) of what people's interests are to deduce the likely intent of most open repository pundles. Without some years' experience of the Smalltalk community, I would be lost.

Documentation: VW documentation is noticeably improving as releases go by. Store for Envy users (Alan's talk) is on StS2002 CD (get the CD from STIC; free to members).

Store is better than Envy. It is much better at remote working (Sames keeps Pollock in synch in three repositories, which would be harder in Envy). It is much better at package refactoring than Envy. The Store diff tool is similar to the Envy diff tool (noting that its menu layout is annoying). Its merge tool is better (not that that's saying much).

It would be nice to have an easy install script for Store, so that people doing local work or evaluations could just get started without having to wrestle with PostGres installation or suchlike. Several people pointed out that Store for Access is extremely quick and trouble free; several others remarked that they had not known that. It was agreed that a Store for Access quick-start script for single users on Windows (made very visible in documentation and/or start image) would be a sensible addition.

A read-only connect to the open-repository could also be provided easily, linked to some kind of registration, perhaps, but very easy for the interested user to set up. (However John and my earlier comments about needing a good understanding of the Smalltalk world to find things in the current repository are relevant to assessing the value of doing this. Maybe we need to discourage uncommitted users until this is fixed.)

Q. Can we explicitly delete bundles? There is a method that does this which is nowhere exposed; you must invoke it programmatically.

Q. VW7.2 will have a runtime image. Can Store be headlessly-scriptable (no popup dialogs)? The value was noted.

Visual Age BOF, Ginny Ghezzo, John O'Keefe, IBM
(I only caught the part of this that occurred after the remote development panel.) WebSphere is important to IBM and you can develop in VAST and integrate into WebSphere. WSAD is an IBM-strategic development environment (Eclipse plus some things) and they are building Smalltalk in WSAD so that it will appear in the list of WSAD languages. (Some people memorise the acronym via mnemonic, 'We Sad' as in either 'we sad people to use a less productive language than Smalltalk' or 'we sad people to use Smalltalk in Eclipse instead of in the far technically superior VAST' :-))

IBM is still marketing VAST. By contrast, VAJava has been withdrawn from marketing. At a certain date, VAJ will no longer be sold. This indicates the different approach; VAST is alive. WSAD does not have an officially-announced longer support date than the same Dec 2005 that has been announced for everything else, including VAST. Obviously, WSAD is strategic whereas VAST is not but VAST has its revenue stream. If you are confident about Smalltalk's productivity (as we all are) then you should not fear that lack of extra dollars pumped in from IBM's strategic funds will prevent VAST from progressing to new releases with fresh features.

If any IBM sales people don't know about VAST, tell them to contact Ginny Ghezzo. Also if you think WSAD is missing things, tell IBM. Likewise tell IBM about any problems you see in J2EE and Java. Always be specific about what is missing, etc., and your business problem that it impacts. General emails moaning about Java and J2EE are of no relevance.

IBM want to use Eclipse to deliver a single industry standard tool for building new tools. They open-sourced it to show it was not a locked-in solution and because they hope it will move into markets that are too small for them to be interested in but in aggregate will give value and coverage. Making Smalltalk work in it gets Smalltalk visibility in language lists that IBM will be publicising and might also occasionally let non-Smalltalk customers accept Smalltalk utilities for specific functions. Meanwhile the group clearly felt that anyone who had ever encountered VAST would continue using it and WebSphere, not move to WSAD and WebSphere, a view that I felt Ginny and John were not disputing though it was doubtless proper of IBMers to leave its more emphatic expression to others.

Rumours that SUN may not survive the next two years abound; I heard them before I came to this conference and they were circulating there too. I had some quiet discussion about whether demise of SUN, and possible consequent change in public perception of Java and J2EE as SUN creations, might change the emphases of IBM's approach, with possible benefit to Smalltalk. Watch this space.

STIC Discussion Session
Perhaps put a list on STIC of conferences at which it would be useful to make Smalltalk presentations, roughly classified (decision-makers go to this one, relevant Smalltalk app could be presented at this one, etc.). There are several smalltalkers (myself for one) who could present but who are sure to miss the deadline or fail to select a good conference unless advised.

There is the audience of decision-makers, there is the audience of non-smalltalk programmers and there is the audience of smalltalkers whose morale need to be raised. Only a small proportion (people quoted one in five and 3 in 47 of smalltalkers at their companies) read c.l.s and come to Smalltalk Solutions.

STIC could be an umbrella for a local Smalltalk user groups attending a nearby conference. The local group could attend and present something relevant that mentioned Smalltalk. STIC could review it and let them use the STIC name to give them greater cachet. STIC could also add demos and interest detail to news releases. It could maintain a group blog.

Proposal: calendar of events on STIC website. People email details of events, etc. (e.g. local user group presentations, Smalltalk at conferences). As well as being useful, this would keep the STIC site active. (Peter Lount suggested added more stuff to smalltalk.org as well.) Is there a risk of duplicating whysmalltalk? Should that be the frequently changing site that holds the calendar (and alerts you if STIC or smalltalk.org has new stuff)?

Update FAQs: some FAQs are quite old. People could email suggested additions and changes to the FAQs and STIC could vet them and insert them. Ask the current FAQ owners to pass their authority to STIC.

STIC could coordinate: when anyone is visiting a location near a local Smalltalk user group for work reasons, they could enquire via STIC whether there is local interest. The visitor might as well give a talk as sit in a hotel room of an evening. Equally, STIC could let vendors know that potential audiences existed in certain cities.

Some people teach Smalltalk. The local XP group is often involved with local schools and are not committed to particular languages; that's a good way in. Squeak eToys is good. David Buck has neat VW code to animate a puppy dog, controlled from workspace, also ray-trace code defined in Smalltalk, etc. STIC could make such presentations available.

Negative and very inaccurate press from e.g. Gartner group, can be very harmful; David Buck described a case where Gartner told a company that Smalltalk was dead, no new vendors, no new releases. Five (count them!!) Smalltalk vendors released new versions within a year of that report but Gartner stated that although they were wrong on specific points in the remark, they would not back down because it was their corporate strategy. The only reply is to be sure to make management know of other analysts; there are some that are very positive about Smalltalk. STIC should find quotable phrases from friendly analysts so that people in companies can guide management to those analysts. (BTW, Gartner wrote reports only a few years ago saying Smalltalk was the wave of the future.)

We should ride the dynamic language wave that is coming. Python, Ruby, etc. - *and Smalltalk*. Arguing about the past interests noone outside the community but riding the dynamic wave to say that Smalltalk is the best dynamic language has potential to interest many people (several people quoted remarks heard at Python groups, etc., in support of this).

Java started by claiming to be a simpler, more like Smalltalk, C++ ("It's better than C++ and while it may not have everything Smalltalk does, *it's good enough*" was the phrase with which its supporters closed many an argument). However Java has become more and more verbose till Java 1.5 adds generics and now it is as verbose as C++. The old argument is no longer true even in its own terms.

Could we restart The Smalltalk Report? It does need serious money and paper publishing is dropping. Paper still has a certain cachet, but several doubted the value of this. Monthly or bi-monthly would be costly but could we do it annually? Make it the StS call-for-papers plus articles. It is hard to do (Squeak tried and it fell off). Bruce proposed making it a digest of stuff; its easier to cut stuff out than create it. He also suggested rotating editorial duty (has worked in other language-specific user groups).

Porting and Miscellaneous

Using and Abusing XML and XSL, Joseph Pelrine, MetaProg

Joseph is in charge of SUnit. Keeping it up-to-date on all dialects was what prompted him to get into this area via the Rosetta tool. He has now resuscitated Ginsu (configuration management tool; see Joseph's talk in my ESUG 2001 report) into Rosetta. Rosetta uses XML to rewrite exported code from one dialect to another. Ginsu lets instance variables have additional stuff e.g. type constraints as needed by GemStone.

Joseph has used XP for longer than it existed (old friend of Kent and Ward) and has been using SCRUM for years. He finds SCRUM gives him all the things he found missing in XP. There is a regular SCRUM meeting at which everyone answers three questions: what have I done, what am I doing, what problems do I have (I was reminded of the Imperial government in India in the Victorian age. It had a very small proportion of bureaucrats and each official had only one form to fill in at the end of each month on which they said what they expected to happen next month as regards tax collection, law and order, etc., and described what had happened in those categories last month, comparing it to what they had predicted on the previous form). Joseph told the joke about the chicken and the pig who planned to start a restaurant. Let's call it the, 'Bacon and eggs' says the chicken. "No way", replies the pig. "You'd be involved, I'd be committed". In SCRUM meetings, only pigs (the committed) can speak, chickens can't even squeak (and no making faces).

XML separates data from presentation of that data: it defines and structures data into custom formats. The price is that you must transform data from one format to another and that is the purpose of XSL. XSL is an XML-based language so it can also be manipulated. XSLT is the transformation-describing language. You can also restructure XML in other ways (by using SAX or by doing DOMs in programming languages).

- XSL Formatting objects describes the formatting semantics.
- XPath (hideous to read) is the syntax for selecting nodes in an XML doc.

This course is about using XSLT and XPath. It uses tags (e.g. <html>), elements (everything between two tags inclusive), attributes (e.g. < table = ..., space delimited). Entities (e.g.   non-breaking space, &le less than sign.), CDATA and PCDATA. See his SUnit.ExampleSetTestCase on slide (slides are on Joseph's website www.metaprog.com). It shows a test suite result and the times each test took to run.

XML can be well-formed or not; many browsers tolerate very bad XML (missing tags, un-nested elements, ...). Use www.validator.org to see if a URL's html is well-formed. Joseph has tested back to NetScape 2.0 and verified that browsers will not usually break on well-formed html (!!!) but NetScape 4 has a bad reputation and he notes it is an issue to bear in mind.

A DTD or a schema is an abstract definition to which a document can be required to adhere. DTDs are not XML-based and have quirky syntax but are easy to write. Schemas are XML-based but less easy to write. Joseph's

rule is to think about how he would approach parsing it. If he would have to worry about scanning then he uses a schema. If the tokenizing is trivial then he thinks it safe to use a DTD. XMLspy is the tool he uses to generate a first-cut schema which he can customise to his needs. (Read these notes with Joseph's slides as they have reference data for DTD layout, etc., that I have not attempted to reproduce.)

There is as yet no published SUnit or JUnit DTD. Joseph reverse-engineered a DTD from some JUnit stuff and has applied this to SUnit. Thus we can generate SUnit html reports that look just like JUnit ones. JUnit DTD pumps out 50 system properties (what version of this, that and the other you're running on) which he has still to add. Errors have their info plus some CDATA (currently a stack dump). PCDATA means the field must hold character data. His slides summarised the regex-like rules for element DTD definition (one, zero or more), alternatives (this or that), defaults, e.g.

```
<!ATTLIST square width CDATA "0">
<!ATTLIST payment type (cheque|cash)>
...
<!ENTITY DonaldDuck SYSTEM someURL>
```

Entities can be external (as in URL-ref example above) or defined in the DTD. Often entities are defined in a DTD referenced by others so that when they change they change only in one place. It can be hard to track down a definition.

XML Namespaces deal with element name collisions. XSLT relies on namespaces.

```
<SUnit xmlns:sunit="http://..some URL...">
```

XML Tools: there is a big industry:

- XML Spy is Joseph's favourite (from altova.com). It is not cheap (various levels: $400 for version he uses). Joseph demoed by writing S# in XML Spy; S# has a DTD so the tool told him what type of thing he had to put next, etc. It looked very pleasant to use; popup menus prompted you with the list of things you could have at each point.
- XPath Visualizer (Dimitry's tool): Joseph demoed by opening on a file and typing in XPath expressions; the tool highlighted what was selected. He strongly recommends it for checking XPath expressions if you ever need to write any. (Limitation: tool only runs on IE.)
- There are various XSLT Tools:
  - — Xalan came from IBM alphaworks lab and is now on apache.org open source. The Java version, XalanJ, is included in standard zip and has its own weird way of writing XSLT docs. The C++ version, XalanC, lags behind XalanJ
  - — Saxon and MSXML are other XSLT tools
  - — XSLTProc is Joseph's preferred tool. It is very fast and is the only one that supports the XSLT1.1 spec. It is written in C. The 1.1 spec

lets you output multiple documents which is the key reason why Joseph uses it, since e.g. ObjectStudio has each class in a separate file, as does S#, so Rosetta must be able to do file -> many files

Joseph writes tests to verify users of his utilities have base methods needed.

```
self
  assert: (anSUnitClass selectors includes: #aMethod)
  description: 'SUnit requires the method aMethod on
class ', anSUnitClass name, '. Please create it with
the following implementation....'.
```

is an example. (This is part of Joseph's Common Base Smalltalk work.)

As an example of a DTD, he then went through Rosetta's concepts.

- Cluster is what he calls a composite CM concept (it is a Cluster in VSE and Rosetta, a Bundle in VW, and is similar to a config map in Envy)
- Package is the leaf CM concept (a package in VW, an Application or SubApplication that has no SubApplications in Envy),
- Subsystem
- Program
- Module specification: timestamped as digits only (collates well in files), exporting dialect, moduleType (one of the above list items)

Rosetta constructs an XML Dome which can handle forward references.

XPath is used to define paths within an XML document: ways to get to some place and ways to find things. It has a library of standard functions. It is a major element of XSLT and is not written in XML. Joseph tried to avoid XPath when he started but when his tasks got complex he had to use it (and found that the comp.lang.xml has some good people who will comment on small examples). XPath can't group (e.g. group all my stuff by class) and he had to work around that.

XPath uses pattern expressions to select nodes in an XML document:

- /roottag gets you to the root element tagged
- /roottag/tag gets you a nodeset containing all the tagged elements within that root
- For example, you might use /catalog/cd/price to get all prices of CDs in a catalog. You can also skip the intervening tags and just type //price to get all price elements in the document, whereas /catalog/cd/* will get all nodes within CDs in the catalog, and /catalog/*/price gets all price elements in items in the root catalog item.
- Predicates are shown in square brackets. They let you select nodes, e.g. /catalog/cd[1] (think cd at: 1), /catalog/cd[(last()] (think cd last, there is no first() function, just cd[1]), catalog/cd[price=10] (think of this as a cd select: [:each | each price = 10])
- using alternate symbol | you can select several paths.
- using @ you select attributes, //cd[@country='UK']

XPath syntax is SQL written by aliens on a bad day.

Stepping to a location start from an axis (a relationship between nodes: am I facing up, down, left or right when I start stepping) and a test (what nodes am I looking for) and much else (see slides). It is best understood by mapping it into smalltalk: ancestor = superclass, child = subclass, descendant = allSubclasses, following = sibling, self (is the same word, thank god!! :-)).

- child::cd selects all children of the current node that are cds, while ancestor-or-self::cd gets owning node or start node if either is a cd.
- /descendant::cd[position()=7] gets the seventh CD in the document
- there are various unix-like abbreviations (. for current node, .. for ancestor node, etc.)
- all requests are sets, whether obviously going to return a single object or not. If the object requested is not there, an empty set is returned
- order of predicates matters: [5][@type = “classic”] means select classic items within the item at position 5 within the current position whereas [@type = “classic”][5] means select classic items within the current position and then select the 5th such item

Node set equality is in fact an anySatisfy: check while not-equal is a negated allSatisfy:, so node sets can be simultaneously equal and not-equal. XPath has many library functions and lacks many others (e.g. no substrings function - you must call substring recursively). Watch out for when indices start from 1 and when they start from zero: cd[0] but [cd=1].

Never put attributes in XML directly; put them in Cascading Style Sheets. Because the browser does not know whether <table> is data or furniture you need XSL to explain to a browser how to display things.

(Many sites have a page with little on it because they’re checking your browser’s capabilities; then they route you to pages you can handle, or make appropriate remarks to please turn on Javascript or whatever, or just make remarks about how old your browser is :-). Joseph’s site has a hide page trick: see the remarks if you need to, see correct first page otherwise).

XSLT uses XPath to find nodes that match a pre-defined template. Nodes that don’t match will remain unmodified or, if that path is never visited, ignored. Joseph showed XML to display HTML of package manifests from packages held as XML, building up the example stage by stage (see slides). He set the stylesheet, as for any HTML doc, then gradually added more and more to an xsl... select statement, using the following:

- An xsl:template has the rules for what to do with matched elements: what elements should it output, etc.
- xsl:value-of gets the contents of an element (e.g. package’s name)
- xsl:for-each is looping construct (e.g. for-each select=”...pre-reqs...”)

- filters are like predicates: [@moduleName != ‘Kernel’] avoided showing Kernel in the prereqs as it’s not interesting (other operators are =, &lt, &gt, can use < and > for these when embedded within quotes)
- xsl:sort is not supported before MSXML 3.0
- xsl:if for control flow (also not supported before MSXML 3.0)
- xsl:choose and xsl:when (xsl:otherwise for default block) for multiple alternatives
- xsl:apply-templates is how you apply other templates’ rules to the selected elements or their children, so you can structure your templates, call them from within each other
- xsl:call-template calling a template by name like a function with parameters

With all this built up, the web page now presented the package list effectively.

VW XSL is the implementation of an old spec, so does not handle the more complex things. (Joseph sent a bug report to Steve last year. He does not know when the fix is planned). VA wrappers Xalan. Dolphin does some XSL things. Any Smalltalk could wrapper XSLTprobe (it is a C library and fast). So far no Smalltalk does correct full XSLT in Smalltalk.

Joseph’s preferred book is Michael Kay’s XSLT Programmer Reference. It has many many examples you can just cut and paste to reuse.

Variables in XSL are really weird to programmers. They are constants, placeholders, not variables; new value => new variable.

Rosetta will convert between all Smalltalks except S#, where it can translate to but not back again. Rosetta has a cross-dialect semantic model plus transformations between dialects. It offers:

- namespace annotations: watch out for accidentally putting subclasses into the superclass’ namespace when that’s not what you want (it also has selector namespacing support though that is little needed currently)
- custom transformation blocks: for example it uses an RB rewrite rule to exchange Squeak’s backarrow and standard := assignment
- splitting of a single source file into many when a dialect requires it

The output dialect must have a satisfactory package concept (thus Ginsu is provided for Squeak) and a small dialect-specific preload module. The input dialect needs nothing. To choose between package concepts, go to the stylesheet and change the chosen concept. The default configuration aims to produce something that will load in a vanilla image so does parcels for VW7 not packages. Currently, Rosetta has seven dialects fully done, four dialects (GNU, #Smalltalk, SmalltalkMT and S#) mainly done and one dialect (GemStone) still to be done (discussion of this led to the conclusion that GemStone should be very easy to get to the mainly done stage).

Joseph ended with some Rosetta demos. A typical scenario is moving stuff to all dialects and then suddenly realising you forgot some minor thing. His first example was realizing you have recategorised some methods: e.g. you reclassified methods in the 'Camp Smalltalk' protocol to several sensibly-named protocols in SUnit. XSLT template can extract className, selector, category and do tree diff to see what you forgot to port and to which dialect.

Demo 2: We looked at the XSLT for transfer from Rosetta standard to VW7 and to AOS.

Demo 3: output tests results and display as HTML.

Q. I noted that Rosetta spits out .dialect tags; sometimes these must be changed to import, e.g. .vwxml must be changed to .xml to import into VW7. Joseph explained that the original tags help track what is what when doing a push to all dialects, outputting files to a single directory. It could be useful to have right tags if doing many exports to a single dialect.

Q. Do as succession of individual transformations? Possible but much harder to track cross-references.

Q. Rosetta in Envy? The commercial add-on does various extra things, e.g. converting code not loaded in image.

Joseph closed by thanking Michael Liepert who got him started on XSLT, Alan Knight, Dave Simmons, Dino Rosati, Camp Smalltalk and ESUG.

One Font, Two Font, Red Font, Blue Font, Travis Griggs, Key Technology

Travis overviewed emphases and runs in VW currently and looked at problems. He stressed he was not addressing the pixel font-unmatched issue (mention was made of a magical factor 1.14 that solved this).

He showed a simple script that wrote text, changing its size, colour and font en route. There are lots of classes and methods you encounter before you get the job done:

- Display Scanner scans for each run change
- default FontDescription is copied and modified: it has a dual role as the description of what you want and a description of what there is (can be)

There are two kinds of Fonts. There are raw character glyphs. These are finite in number; you describe a font and it must find the nearest match. The other kind is glyph decoration (SyntheticFont in current architecture). These wrap the device font and are potentially infinite in number.

Travis got into this years ago when he added super and subscripts while working for an engineering firm. It worked by asking for super/subscripts in the FontDescription, then (via a case statement) the synthetic font asked the FontDescription if it must super/subscript, etc. and acted appropriately.

Then, developing the PDP tool, Travis wanted to put breakpoints *in* the text, not on the side like other debugging tools. Hence PDP wanted to add emphases too, to show where its probes are. Then along came the RB highlighter and it added emphases too. The RB subclassed rather than extending, otherwise the approach was very similar. Travis felt this was approaching 'path criticality'. Everyone was hacking the same method:

```
CharacterAttributes>>newWithDefaultAttributes
```

and you couldn't load all the utilities because the patches overwrote each other. There were also conflicting patches to SyntheticFont and lots of boolean slots being added. Travis therefore created ExtraEmphases. While doing this work, he learned to save his image often (every method) since on every error debuggers came at him at lightning speed.

The package is in the open repository. There are only two base overrides. SyntheticFont is obsoleted.

He solves the `newWithDefaultAttributes` issue with a class side properties dictionary and one line of code in the method that checks and acts on it. He solves SyntheticFont overload with a new class DecoratedFont (a third layer, the other two being adorned font and warped font). Each font encapsulates its own logic: it decides whether it should apply itself to the underlying font or not.

FontPolicy>>findFont: was changed from a case statement to an (I assume; I didn't actually hear him use this word but it sounded like it) visitor pattern that traversed the subclass tree and asked each class if the visitor wanted it. ExtendedFont's properties dictionary works like that of AnnotatedMethod.

Currently there are 10 special fonts implemented: coloured, jagged, back colour (colour square behind e.g. number in cell, ...), leading / trailing insert, substitute (e.g. replace smiley's text with smiley graphic), underline, strikeout, etc. Travis ran a script that applied random emphases, producing some pretty (unreadable :-)) examples that changed every time he ran the script (just to show it was all real).

Travis then invited us to choose an example font to add from a list (but not the asterisked ones as he found a bug in the width-handling last night he will fix). We chose blink and created a Blink class with a 'colours' Array. After some coding, example text in the font iterated through the colours based on the millisecond clock value when sent the blink method. (The behaviour ran off a loop since, as he remarked, forked animation in a totally event-driven system is hard to do without a high-level loop for everything else to run off, c.f. most games programs, Squeak's high-level loop, etc.)

Q. Composite fonts? Needed when two or more glyph sets must be used to build up the text. He has done nothing with composite fonts as he's not convinced of their sanity. "Maybe they make sense in Japan."

Q. Is this integrated with all else in open repository? With PDP and RB highlighting, yes. He can't comment about anything else.

Q. Could this be used to e.g. hide comments, move them to the side of page, make them hyperlinks, etc.? Yes, it could be so used. (Sames: Pollock will support hyperlinks).

Talks I Missed

To make it easy to see whether I have reported on a talk or not (and to prove my point that I could not attend half the conference :-)), here is a list of all the talks that are not reported on, even at second hand, above.

One talk was added to the schedule at the last moment, **Scalability and Performance in a Smalltalk/Oracle Environment**, Laurie. I missed it but mention it so anyone interested and otherwise relying on the agenda to know what was discussed will know about it. One talk had to be cancelled, **The nature of certain enterprise systems**, an experience report by Mathieu van Echtelt of *CosmoCows*, as Mathieu could not make it.

In rough chronological order, the other talks I missed were:

Using PostgreSQL from Smalltalk, Bruce Badger, *Openskills*

Graduating to Objects, DanAntion, *ANI*

Black Knight: Eclipse as a Smalltalk Development Environment, John O'Keefe of *IBM* and Eric Clayberg of *Instantiations.* See the VA BOF report above for some of what was covered.

The Multi-Process UI in VW 7.1, Terry Raymond, *Crafted Smalltalk*

SmallCOM/X demo, Josef Springer, *JOOPS*

From CVS to shining CD, Joseph Pelrine, *Metaprog*. See my ESUG 2002 report for some of what was covered.

An open-source acceptance testing application, Sean Morrison, *OTPP*

Is it slow...Hobbes, Vassili Bykov, *Cincom*

Is it slow... Elastolab, David Buck, *Simberon*

Time To Market with Smalltalk Technology?, Peter Lount, *independent consultant*

User group panel, moderated by Bob Nemec of *Northwater Objects*

Delivering Smalltalk natively on .NET with S#.NET and S#.AOS: Competing on a level playing field, David Simmons of *Smallscript* and Joseph Pelrine of *MetaProg*

An Application Framework for Developing MVC web applications with the VisualWorks WebToolkit, David Shaffer, *Westminster College*

ReStore - Relational Persistency for the Discerning Smalltalker, John Aspinall, *Solutions Software*

Adding Strong Encryption to the MSN Instant Messenger Network, Anthony Lander, independent consultant

Smalltalk in an Autonomous Underwater Vehicle demo, Jon Hylands, *HUV*

Preparing For Pollock - The Future of the VisualWorks GUI and **Advanced Pollock: Under The Hood, Getting Your Hands Dirty**, Samuel Shuster, *Cincom*

HOP: Multidialect Object Persistency framework, Giorgio Ferraris, *Elevensoft*

A new MacOS X Smalltalk, Dorin Sandu and Mark Suska, *Ambrai.com*

#Smalltalk: An open-source Smalltalk for .NET, demo, John Brant and Don Roberts, *Refactory*

O/R Mapping in Smalltalk, Alan Knight, *Cincom*

smallCharts - its development and design, Christian Haider, *Smalltalked Visuals*

I missed this but Travis Griggs did attend the presentation. It was well done and neat to look at. But the name may be a bit misleading. He had thought (as I did) that smallCharts is a graphing framework. It is not. It is an application used by the print industry to create the stock trend reports you see in newspapers and such. So the presentation was a sort of an experience report. Christian did a good job of presenting, and from what Travis could tell, a good job of writing a cool app.

It does do a fair bit of very specific plotting. Christian and Travis talked about this for a while afterwards. Travis also do a put of on screen XY plots (BG doesn't work, because it can't support independent X series from line to line, just like Excel). One of the things that comes up in any of these discussions is how (unfortunately) frameworky any plotting package has to be. Everybody wants their charts to look just a little different. Labels this way, lines that way, etc. There are some common themes. Autoscaling for example is kind of a black magic. It would be nice if a good extensible framework could be derived, but Travis just doen't know how.

Test-Driven Development, Dave Astels, *Adaption Software, Inc.*

Performance Tuning in GemStone/S, Norm Green, *Gemstone*

Clean-Slate Smalltalk, Brian Rice, : *LGOS Research and Development*

Other Discussions

Pollock discussions with Sames: he has to complete tree and tab control; most other widgets are now done and resizable. It will be usable for *some* tools in VW7.2 (if you are happy with Windows-only feel; doing all feels is straightforward but tedious so Sames will do all widgets in one feel first) but expect it to be 18 months hence before you should use it generally.

Michael Ruger told me about the tweak framework. Think of it as Seaside for UI interactions. Seaside lets the web app designer put multi-page logic in a single method without having to manage interacting state models for the user's back and forth and sideways possibilities. Tweak, by using a similar approach, will let the UI app designer put e.g. drag-drop behaviour in a single method without having to split it across on-mouse-down do start, on-mouse-up do end, worry about what the user might have done in between these, state models. Michael suspects David's 'next month' may be 'before year end' since both teatime and tweak must be in there. The Croquet download has been pulled from the site (too big at 90Mb and not yet ready). Before year end may be the time to look for another download.

As David Pennington and James Foster remarked in the Remote Working panel, you can get to 'know' people from newsgroups and remote work. Then you meet them at conferences and discover what they actually look like. For some reason, this Smalltalk conference, far from my first, was one where I put names I knew to faces I previously had not known. I had not met Ginny Ghezzo before; she was merrier and younger than I had imagined the manager of an IBM product line. I had met the creators of the Refactoring Browser before but realised this conference that John and Don have a good sense of humour. Various other known names (Travis Griggs, Terry Raymond, Bruce Badger, Avi Bryant) I will now recognise if I meet them again. (By contrast, Colin Putney looks just as one might imagine the guy who landed the Squeak skiing job would look :-)).

Kapital-specific Discussions

I talked to the Refactory and to MetaProg in detail about such tools and techniques they could offer as would help with the VW5i-to-7 port. (I also got background information and pricing info on TestMentor in case its UI testing abilities were wanted.) Some of the technical detail is in the main report above. This sections notes some additional points.

Syntax: this is the easiest part of the port as VW5i and VW7 are close, Envy constituting the main cause of difference. Both the MetaProg and Refactory tools use custom refactoring to allow

- refactor and export without recompiling
- import and refactor before recompiling

The MetaProg tool also has some support for exporting from Envy to Store without having to load the code.

Elimination of polling UI: we could buy a refactoring from the Refactory for a given polling to non-polling pattern at an unalarming cost and see if we liked it. If we did, we could buy another and maybe test and bounce it

back to them with exceptions until it handled the complete pattern. We could then buy a third and so on. We could carry on in this way for as little or as much in the way of refactoring help from them as we liked.

We could aim to end up with refactorings for the entire polling UI in this way. However John and I have both noticed in similar past projects that these things tend to split into an 80% of refactorings that can be codified reasonably straight-forwardly, a 16% of refactorings that require serious thought but are doable and a final hard 4%. If the few, considered as a percentage of patterns, are also few, considered as examples in code, then a point comes when coding the change as a refactoring instead of as a test for what a human has rewritten ceases to be economic. We would do better to refactor those individual few methods by hand and ensure that our process alerts us if the start method has changed.

Unused Code: the Refactory could sell us unused-code-detection tools. The unused-class-detection is very lightweight. The unused-methods-detection is less so; its performance cost could make it safely usable only in test runs.

FDP-porting: MetaProg are available to pair-program with their considerable experience of Smalltalk CM tools. The FDP tool appears reasonably separable from the Kapital app so can be studied remotely. Obviously we have the ability to do this entirely in house. However, as it is seen as the hardest part of the port, extra help is worth considering on two grounds:

- simple time and resource: staff porting FDP are not doing other things
- best-match porting: pair-programming with an experienced external smalltalk CM tool builder may help get the best match of FDP behaviour to Store abilities (greater than Envy in some areas, less in others).

UI Testing: Silvermark's Test Mentor costs $3,500 for 5 or fewer seats and adds circa 10% for running on two non-trivially-different versions (e.g. VW5i/Envy and VW7/Store).

I talked to Joseph Pelrine about Rosetta and about Smalltalk configuration management tools, and to the Refactory about porting patterns and about finding unused code in large systems. To find unused code, John Brant has a lightweight class-instrumenting trick usable in performance-critical runs.

Follow-up Actions

Niall To Do:

- Get Method History and ExtraEmphases from the Open Repository
- Drop changes-from-base-7.1-only package of our Custom Refactoring work to the Open Repository so people can more easily see our work.
- Ask Diane Severide about current and possibly upcoming Store construct(s), how they will best map onto other CM constructs.

- Maybe look at the change inspector's display of renamed methods in the Custom Refactoring project. Can we display the old method name?
- Write-up my 'meta-class subclasses class' pattern and submit to Joe Yoder's meta-patterns page.
- Find means to make my deep comparison framework available to interested parties.

Conclusions

A pleasant mix of the immediately useful (e.g. for me, the refactoring and porting talks), the relevant (Vincent's meta-programming system and Scott's agile methods overview) and the fun (Croquet, Travis' fonts). The UI and web testing talks and frameworks show other people's solutions to an area I'm already working in and I may well use some of the ideas.

- Dynamic languages are a renewed focus of interest and a Smalltalk opportunity.
- If SUN's problems become sufficiently visible, Java and J2EE may be re-evaluated in the marketplace as the task for pundits becomes to explain their sponsor's failure instead of their success.

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