Smalltalk Solutions 2002, Cincinnati, 22 - 24 April 2002

‘When the world ends I want to be in Cincinnati; things happen ten years later there’: Mark Twain (quoted by the mayor in a welcome speech that had three good jokes and, better still, was not too long). I can report that the conference was a lot more interesting than Mark Twain would have expected and, while the end of the world may come late in Cincinnati, the long-overdue end of the Java phenomenon may be happening early there. (Frequent changes between thunderstorms and pleasant weather also made my visit to Cincinnati more interesting than Twain’s quote would suggest.)

In the UK, we sometimes have the impression of the USA as the land of political correctness and cynicism. I’m happy to report that, in Cincinnati at least, these trends have clearly not yet overcome all more old-fashioned views. The conference opening (combined with that of the co-running Cincom Control User Group) featured a patriotic medley of american songs well-presented by the Cincinnati school of the performing arts. Being sufficiently old-fashioned myself to wonder whether the 1776 rebellion against the crown was not a most questionable innovation, I nevertheless found this a pleasantly different way to open a conference.

A visit to the impressive Newport aquarium on Monday night was fun. Cincom stood us dinner on Tuesday 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 Inc (send any comments to nfr@bigwig.net). It is as accurate as my speed of typing in talks and my memory of them afterwards can make it. My thanks to all to the speakers and participants whose work gave me something to report. Thanks also to the conference sponsors: Cincom Systems, CG E&Y, Gemstone Systems, IBM, Object Arts, Synchrony Systems, Inc., Precision Systems, Reiling Consulting Corporation and Totally Objects.

As there were usually six and sometimes seven (!!!) programme tracks, plus BOFs and ad-hoc discussions, I could not attend, still less report on, half of what happened. (Some of the choices forced by the schedule were very painful; should I listen to Dave Simmons on Smallscript, Eric Clayberg on advanced VA programming, or Don Roberts and John Brant on Refactoring?) Slides of all talks should soon be on the conference website (http://www.gosmalltalk.com/). John McIntosh has notes on many of the talks I missed; see his report

http://wiki.cs.uiuc.edu/CampSmalltalk/
Smalltalk+Solutions+2002+Trip+report.

Summary of Presentations

I have sorted the talks I attended into various categories:

- Applications
- eXtreme Programming and Testing
- Patterns and Frameworks
- Development and Deployment Platforms
- Miscellaneous

followed by Other Discussions, Follow-up Actions and my Conclusions.

Opening (shared with Cincom Control Users Group)
After the rousing opening (see above), Tom Nies speech stressed Cincom's support for collaborative web-enabled business models. Next came a philosophical Keynote (Think Naked, Marc Marsan, Marco Polo Explorers). Kent Beck says an extreme programmer needs a modicum of courage and conviction. Marc's talk, amusingly if not very deeply, made the same point. His equivalents of the 'smiley (I agree with you so you can stop talking)' and 'rat-hole (off-line this discussion)' signs my team uses were spheres with faces on them which he encouraged us to throw at people who came out with OTT ideas. The spheres were soft but the message still seemed a trifle mixed. He advised thinkers to see others' reactions to way-out ideas as invitations to explain further and said that no concept should be sacred when thinking out of the box. These are common ideas but in my opinion wrong: a window lets you see through a wall but if you also see through the objects outside then you're blind. Thinking achieves nothing unless *some* concepts are sacred. I ducked out to see the exhibitors' area.

Exhibitors
Totally Objects had a great Smalltalk Solutions offer (their entire VA toolset at 70% off) and had a good show; some very positive comments about them made by clients.

I helped the VA people pin up their banners and talked to Mark about how the VA packager maps CompiledMethods and how to package method wrappers.

I discussed meta-programming frameworks with Paul Baumann on the Gemstone stand. A Gemstone-oriented consultancy (Rolling Stone - Gemstone administration tools, etc.) also had a stand.

I won a CGEY T-shirt by providing three questions to their Smalltalk quiz, interspersed with three from Eric Clayberg, which set me a challenging standard. (However as Dr. Johnson said, 'You should never be embarrassed in exams because the greatest fool can ask more than the wisest can answer.', so I think those who won T-shirts by *answering* three questions achieved most.) CGEY's Smalltalk consultancy arm is XP-oriented: they like to run contracts with clients according to XP's tenets.

I failed to win the GPS that Precision (Smalltalk placements) were offering. Perhaps it's as well; my yacht has two already.

Synchrony provide tools and consultancy for migrating between dialects.

Object Arts (Dolphin) had a stand but I never caught them at it.

I talked with Mark Foulkrod of Silvermark about what instrumentation TestMentor adds to VA UI widgets, and whether my deep comparison tests could invoke it.

The Why Smalltalk stand projected slides on various Smalltalk projects during breaks (exhibitors area was also coffee area).

Applications

MarketPlaces Trader, Reinout Heeck, SOOPS

This system, implemented in VW, provide one-click trading of stocks in real-time. (Can be dangerous - it once raised 1000 orders per second due to someone resting a book on keyboard - but warnings protect you.) Supervisor function monitors risk exposure and margin for each trader. The system presents stocks, to-do-list for day, portfolio (separate subsystem as required by Dutch law), status of trades, options, etc.

It was first developed as a value-add to raw market information. A leased line between premises and customer, essentially an ethernet connection plus lots of encryption, transfers the trading information. (Satellite dish is weakest link; some parts of customer's system designed to work on deltas; bird flies past dish, lose delta).

The system permissions need to be flexibly configurable; people on holiday or sick need to hand their trading portfolio management permissions to others and every now and then there is a panic to get things done that needs lots of extra permissions.

Their system took 3 hours to start up early in development due to the time needed to pull data from the customer's AS400 system (later cut to 20 minutes). Smalltalk's ability to develop without restarting was therefore very useful to them.

Their first client had a very complex model: client and server shared object model. The second was simpler. The third needed complex communication between client and server, leading to complex timing issues that they solved. They are now working on a trading solution (called SpotLight) for their fourth client, power exchanges. The current power market trading system is very slow (trades can take 10 seconds). They duplicated that system analysis model in Smalltalk and are iterating towards the delivered solution; the client is enthusiastic. A power trade is displayed as graphs and data, validated and colour-coded. Smalltalk also makes deployment easy; the traders used to need smartcards, hence CD for smartcard software upgrades, another for PC software upgrades, etc. Now, Smalltalk VM is installed once, then upgrades are easy.

They started developing SpotLight in VW7 but went back to VW5i.4 with back-ported RB and other tools because 7 was not stable with respect to Store at that time; they hope to deploy in 7. Their code is written in English (previous power-trade system was written by Spanish company; babelFish does not work on 3 letter acronyms in its database).

They use State Replication Protocol (from Gemstone, written by Paul Baumann) to encode data for passing between traders. Updated objects send changed messages to clients; timing issues about not changing while client operating on needed care. Reinout made the server single-threaded (sounds odd but right for this case). The server encodes an update as SRP and it puts in a client's queue, all in single thread. For their scale and given that Reinout's team would have needed training in multi-process issues, he found single-threaded the simplest thing that could work. (May change later, to scale; his team has now learned a lot).

Reinout then handed over to Danny Doog, an independent day-trader, to demonstrate the system. Danny dialled in to Amsterdam. His day-trader firm is small (only 3) so does not use the system's risk manager feature (a firm of 20 would use it). Start screen showed three traders, each with own fee, trade clearing cost, etc. Danny selected some stocks, examined the market depth (open price, top price, and many other trade data summary statistics). Stocks whose trade distribution becomes too abnormal get frozen for five minutes, then are reopened. Danny works with limit orders; the alternative order type is too dangerous. At the appropriate depth of viewing of stocks, he showed how to place orders by clicking on the appropriate trade statistics. When the trade is made, a screen appears and his profit statistics are updated (unfortunately it was just a demo with dummy stock market data being supplied by Reinout's team over the phone line to Holland; Danny didn't actually make $8000 during the demo). Eliot mentioned that JP Morgan's system paid for its entire $15 million development cost in the first 40 minutes of operation.

Q. Make it more graphical, less tabular. Danny explained traders are familiar with the tabular layout.

Q. Use automatic market trader. It is against Dutch law to do so. A SOOPS client has considered using such a system and making it legal by having a human acknowledge each trade by pushing an OK button.

Q. How long for newbie to learn. A competent trader learns it in 2 seconds.

Web-TCM Translators workbench, Georg Heeg, Georg Heeg Objectorientierte Systems
Web-TCM is multi-lingual web site used by Translators.

Microsoft has a localisation process. There is internationalisation (i.e. base preparation), personalisation and localisation (VW thinks these are the same which is not so). Web-localisation wants to offer every culture and every language. This needs languages, characters (e.g. cyrrillic), date formats (US and European differ), etc.

Translation Memory tools store human-translated phrases in repositories. When a variant of the text is found, matching phrases are automatically translated, with fuzzy matches for variant phrases. Translation memories were not previously available for web-localisation. Web-TCM provides one.

A Web-TCM system has one source and many target languages. The original web-site is processed by a segmenter that extracts phrases, keying them by ids in the web pages, working on XHTML (HTML parsed to have a cleaner structure). VW parses XHTML into a parse-tree. The segmenter transforms this into TM segments. Thence it generates .ssp files which reference the original language segments in the TM.

```
<%lang := request anyParameterValueAt: lang.
tm := ...>
```

When the segmenter much fragments the text, it is hard for developers to understand, so a UI lets them invoke ‘Merge with next’, Merge with next two’, etc., on too-granular segments. The UI presents all items not already translated through matching, or fuzzy matching, with already-translated phrases. Summary statics of how much has been translated are presented. Users can select segments, see all language translations of it, click to see where the phrase is used in the web pages (first version was written in VisualWave; done again as ssp).

The system leaves all state on the client; the Web-TCM server is stateless and entirely re-entrant. Persistence is handled simply by BOSSing out the TM dictionary of ordered collections. The system is small: one core class for retrieval system, some others for editor system plus helper servlets for ‘Merge with next’, etc.

It took Georg two weeks to do all this. He started the Smalltalk server in June 2001, restarted it in September 2001 after a power failure. Otherwise, he delivered 55 versions of the package at Anhalt university with the translators using it and there were no problems; he concludes that VW + Store is stable (FYI, the server uses development image, not run-time).

Web-TCM is the first TM tool that supports a virtual team, an obvious value since translators are unlikely to be co-located (c.f. Microsoft’s TM has to resynchronise TM’s from all parts of the world).

Q. Special handling for phrases of very different sizes? No, just tell web page designer never to use fixed-size frames (and currently no support for translation to Arabic or Hebrew right-to-left languages).

Several people commented that this was a good example to publicise. The domain is generally understood and Smalltalk's strengths come through.

Decision Support in Smalltalk, Jerry Blinten, Caesar Systems
Jerry's background is engineering in the petroleum industry. This is the first Smalltalk system he has built (in VSE; I was surprised to learn later from Jason that Cincom are still selling two new VSE licences per month, plus existing). The upstream (i.e. extracting oil from the ground) petroleum industry is risky. Exploration and development decisions are based on indirect measurements requiring some level of interpretation. This technical uncertainty, added to the very large capital investment requirements, plus economic and geo-political issues, creates a challenging environment for decision makers.

PetroVR competes with spreadsheets used in an interactive team environment. Hence it needed to be intuitive and fun to use.

- Partly it is a project management tool (typical project management gant-chart-like graphics, etc.) Each branch of the decision tree can have its own business model with its own constraints.
- Partly it is a simple mathematical modelling tool (c.f. Didier Bessier's work). Data values are generalised to statistical summaries (in any units) which are correlated via monte carlo methods, to produce summary output statistics. These can then be subject to sensitivity analysis (to see which variables are most strongly correlated).
- Partly it is a simple flow/connection process model (usually superimposed on a geographical or schematic background).

Scripting support allows users to make their own analyses. An equation editor lets them develop their own economic models (e.g. tax structure effects). BP has 20+ licences and think they have reduced cycle-time by 50% through using it, mainly because it improves communication.

He has been careful to separate decision support and business model aspects so he thinks he could drop another industry's model into the tool.

Building Video Games in Smalltalk and C++, Anthony Lander
I missed most of this. See www.groovemonkeys.com/anthony for some information. Eliot hopes to provide Anthony with the means to show a very convincing performant 200-frame-per-second game running on VW for OOPSLA.

eXtreme Programming and Testing

XP Release Planning and User Stories, Ann Anderson, Chet Hendrickson, and Ron Jeffries.
Game: plan project three times in three styles. Task: do essential software to save $2(arbitrary zeros) for company. The software needs 45 features; plan which features to do in which month.

- First style: high executive says do all features in 6 months: says nothing meaningful re their relative priority, value, etc. Refuses to say more as months progress and not all features planned are done. Refuses to accept any plan that does not meet deadline. We don’t know feature difficulty. This style guarantees everyone gets fired at 8 months.

- Second style: this time we were told the cost to do features but not the value to deliver features. Plans made better progress but still unsatisfactory.
- Third style: we were told both cost and value.

The results were plotted on a spreadsheet; for a single release, releasing in the second month gave maximal returns over the first year. Releasing every month would give better returns, of course. In short, early release of a project that has achieved high value per development month is better than late release of high overall value.

Feature cards with costs and values on them let customers and managers think about priorities whereas without them, all they remember is the promised delivery date. Hence, always have feature cards; they are so easy to pick up and move, the most uncooperative manager will be unable to resist picking them up and moving them around. It can help to give people a finite counters (poker chips, sweets, ...); this especially lets multiple stake holders vote together about a feature’s value.

Interdependency discussion: how to show feature value when e.g. three features all need the same piece of infrastructure work so whichever is done first will cost the additional effort of doing the infrastructure. Suggested solutions:

- Ron, Ann and Chet: score each story as explicitly two figures A + B where A is cost to install infrastructure, paid once for first story to be done, shown on all three story cards that need the infrastructure.
- Kent Beck: stub out incomplete infrastructure while doing each story, so show cost of these three stories as A/3 + B = single figure on each card (assumes all will be chosen and that stubbing is feasible; Kent told Ron he thought you can always stub out interdependencies and that card values should be kept straightforward to let the customer make all business decisions).
- Niall: make the customer value stories in long-enough time increments; developers then sequence them to produce maximum value in each iteration, leveraging their understanding of how one story may enable another. In short, *customer* assigns value to stories but *developer* assigns stories to iterations.
- Dirk (if I recall aright): get values from customer, produce 3-4 alternative technically well-sequenced plans, ask customer to choose between them.

Adding Developer Testing Mid-Stream to the Development Process on the KnowledgeScape(tm) Software Project, Randy Ynchausti, EIMCO Process Equipment Company

(Randy kindly moved his talk to my session so we had an entire session about introducing testing to existing systems.) The KnowledgeScape software uses genetic algorithms and neural nets to provide real-time, adaptive process control for industrial processing plants. At first, marketers, management and developers failed to work as a team in this project. Marketers promised things the software development team could neither deliver nor communicate this fact to managers. Their process was waterfall-like, resulting in significant post-delivery rework: internal customers implicitly were saying ‘Build me the wrong thing and then I’ll tell you what’s wrong with it’.

However Randy then took issue with Kent Beck (explicitly, when I raised the point) by demanding that software process be predictable and statistically controlled, the opposite of XP’s philosophy. To achieve this, they introduced a revised process in which developers used forms to record estimates and actuals of the time they spend designing, coding, testing, etc., and of the number of lines of code they write. This data was used to argue back to management but management was not allowed to use it to demand that estimates meet actuals. Thus it was administratively useful to the team although they did find collecting it interrupted their work. Randy claimed their single programmers produced code as good as pair-programmers would have done in the same time elapsed.

A unit-testing process was integrated into the development cycle 3 years ago, approximately mid-way through of the KnowledgeScape software, when it already had several hundreds of thousands of lines of VisualWorks Smalltalk, Visual Smalltalk Enterprise, GemStone Smalltalk, and Visual C++. They went for 1:1 correspondence between unit test code and production code, i.e. not so XP-refactor friendly. They explicitly aim to reduce time spent coding in, or after viewing, the debugger and while their process permits writing tests before code, it does not at all require it. They also test GUI classes (another, less controversial, departure from XP as Kent advises it).

I would have spent a lot of time debating the differences between this process and XP - had my talk not been next :-). I definitely remain a fanatic XPer, but were Randy’s company recruiting I’d consider working there.

Solving the XP Legacy Problem with (Extreme) Meta-Programming
Niall Ross, eXtremeMetaProgrammers, Andrew McQuiggin, HECM
If I dumped all my detailed slide notes into this document, I would wildly unbalance this conference write-up; see my slides and their accompanying detailed notes on the conference web-site for details. Very briefly:

- XP depends on a synergy: the same tests that speeded development of a feature show whether later changes are refactorings with respect to it, i.e. whether they do or do not change that feature’s (desired) behaviour

- Legacy systems break this synergy: either refactorings must be confined to new code or adequate tests for the legacy must be written

Creating a full set of tests and assertions is a hard task for a large legacy system. It is usually straightforward to create basic tests that use stored or example data to drive an instance of each legacy product type into each state; the problem is how to show that a given change is or is not a refactoring with respect to these tests. The assertions an XP project writes to develop its code may not detect every non-refactoring but assertions quickly thrown together post-hoc to XP-ify code written years earlier offer much less confidence (or take much more time).

Our solution began by noting that, in an OO system, the behaviour that a would-be refactoring might disturb may be adequately represented by a subgraph of the objects produced by running a test: e.g., a view layer subgraph capturing what the user would actually see, a business logic validation layer subgraph capturing the system's view of the test's validity, etc. Hence a set of basic tests that captured (a deep copy of) a chosen subgraph for a given product in a given state, along with two utilities:

- A deep comparison framework that allows very flexible customisation per class compared and per comparison run.
- A test framework, subclassed from the SUnit and SUnitBrowser frameworks, that holds multiple test results and comparisons between them.

enabled the process:

- View test result, run selected tests in start (e.g. latest released) state. As the released version was conventionally-tested, the subgraphs captured represent acceptable behaviour.
- View new test result, effect refactor (by coding in same image, by loading already coded into same image or by exporting test result to new image as supported by test framework) and rerun (all or some) previously-run tests. The subgraphs now captured represent post-refactor behaviour.
- View a comparison result for these two test results. Invoking 'run' on a test now runs a deep-comparison test on its two run copies held in the two test results being compared. A green result means the tests had the same outcome and deep-comparable subgraphs. Yellow means they had the same outcome but not comparable subgraphs. Red means they did not even have the same outcome.

As only a small proportion of products are meant to change in each three-monthly release, we expect to see comparisons passing for all others' tests; failure provides timely warning that a change is not a refactoring. I demoed our system, running three tests, making a small change, rerunning the tests against a new result, then 'running' them against a comparison result that showed examples of all three comparison outcomes.

The rest of my talk described my comparison meta-program and asked what others were in use and whether Smalltalk would benefit from common meta-program patterns, protocol and/or frameworks. Some, but not all, meta-data patterns are documented (see Joseph Yoder's talk), but meta-programming patterns seem even rarer. I had benefited from studying Paul Baumann's DeepCopy framework and the Refactoring Browser's ReferenceFinder framework (and had found it easy to retrofit my customisation strategies to subclasses of them), but my impression was that they were little used. John Brant pointed me at the GFST package in VW which also has some meta-programming behaviour but agreed that these things were not prominent in the public domain.

I also raised the issue of how key instVars for tests fit SUnit's philosophy; products defined in meta-data naturally suggest meta-data-defined tests.

Tricks for Testing Your Smalltalk Project Efficiently, Mark Foulkrod, Silvermark
They have VA and VW Test Mentors and also a Java product (written in Smalltalk with tiny Java part). I missed most of this talk. The part I caught surprised me by seeming slightly pre-XP in its approach (I've no reason to think the tool is). The talk's style may be appropriate to the assumptions some of their clients still start from but in a Smalltalk conference the basic philosophy of XP should be either assumed without long discussion or explicitly argued against if the speaker so wishes.

Patterns and Frameworks

A Multi-Process Smalltalk Agent, Bob Nemec, Northwater Objects
Many applications need to load data automatically, trigger scheduled actions, and run unattended, handling errors and not letting one task block others. Bob has encountered this requirement in several jobs and evolved ideas to handle it.

His agent started as a file monitor that read pending files and moved the read files elsewhere. Next, he wanted to do this regularly: polling would have been easy but he knew he wanted to develop it further, so added scheduled events. He had a choice:

- complex scheduler, simple events
- simple scheduler, complex events

Bob chose the latter design. Every minute, the scheduler checks what events should run. Events decide what they do when run.

The scheduler must trigger behaviour, not do it, so it uses not while loops (could block) but a semaphore-controlled queue which calls e.g.

```
deferPerform: #checkForNewFiles answer: [:each | ...]
```

to put the event in an AsynchProxy whose queue passes it to a process that will run it. It is important that the answer block not do non-trivial work; if it needs to, make it put another event on the queue. It’s also important that, as well as semaphores for adding and removing from the scheduler, you also use a mutex semaphore to protect modifying the queue, lest adding and deleting overlap.

Example: request a quote and five minutes later get the quote you requested (should be done by then) by adding #sendQuoteRequest event to scheduler queue for time now and #getQuote event for time five minutes from now.

Early on, they recognised that something had to hold these monitors, scheduler, proxies, etc. They implemented a standalone agent with an ini file for e.g. quote delays for the various systems, etc. Later they also had an ini file for the schedule itself.

Next they developed it into an update monitor to invoke actions on their GemStone database and their financial application to do, for example, end-of-day processing. They separated the GemStone and application (e.g. FTP quote request) schedulers because the two had different requirements; the former had to signal support staff pagers, etc., if end-of-day processing failed (aside: their pagers wouldn’t work with the email program the manual said would, but did work with TotallyObjects’ socket set). All messages to GemStone went as events to the GemStone process which alone could signal GemStone. To avoid the UI locking up, they made it go through the GemStone process as well (took a little time).

The debugger can only debug one process at a time so can’t debug multi-process as other process may still be doing something. Hence, they wrote trace logs, essential for understanding. These developed to three kinds: Error log (details of serious errors) trace logs and status logs (trivial notification of what happened). At first, he wrote text messages to the status log but then switched to just providing the method call and parameters as a debugger would.

They added an Agent window to view the agent; add event to tell window to refresh every minute. They also had a scheduled GemStone abort every five minutes if nothing done by user (thus updating their database view). Likewise they scheduled closing of long-open files. In principle, the error count can control behaviour, e.g.

```
errorCount > threshhold ifTrue: [self reinitialize].
```

but in fact they have so few errors they react by hand. Other counts can ensure the system does not, for example, generate a huge phone bill by sending millions of request. The system runs for weeks at a time.

Lessons learned:

- don’t fork and forget processes; keep the process in case it dies.
- Don’t try to do too much at once; it’s hard, so get each increment working before next.

- Don’t try this outside Smalltalk; it’s too hard to do in inferior languages.

Bob praised TotallyObject’s tools: ‘Their stuff just works.’

Adaptive Object-Model Architecture: How to Build Systems That Can Dynamically Adapt to New Business Requirements, Joseph Yoder, Refactory, Inc. (www.refactory.com)
Multiple axes of change create the requirement for meta-data. Data is easier to change than code, so put as much of your domain model as possible into (meta-)data (a.k.a. meta-model, reflective architecture, meta-architecture).

Joe attended the first meta-data workshop at the University of Illinois in 1998. Work has continued since then; www.adaptiveobjectmodel.com has papers, including the slides of his talk (more recent than on conference CD). He has worked with Michel Tilman and many others (see list on his slide). He has noticed that the architects of a system with Adaptive Object-Models often claim this is the best system they have ever created, and they brag about its flexibility, power, and eloquence. At the same time, many of the later developers of such systems find them confusing and hard to work with. Such architectures are powerful but their style needs to be grasped.

Ralph Johnson’s meta-data rules:

- If something is going to vary in a predictable way, store the description of the variation in data so it’s easy to change.
- Never build a framework until
  - you have to
  - you’ve done three examples (the rule of three)
- Anything you can do I can do meta (to be sung :-)
- Meta is better (to be pronounced in an upper-class English accent :-)

Meta-models usually arise as domain-specific frameworks, often in financial domains. Thus public domain meta-model frameworks are rare. Patterns for meta-data have been written up: TypeObject, Strategy, etc. The various people working in this area have outlined some 30 patterns used in meta-models

Joe presented a vaccination tracking system as an example in which to demonstrate various meta-data patterns. The system used a TypeObject pattern for the various types of vaccines and for the acts of vaccination. Usually meta-data types will have varying properties, handled by the Property pattern. TypeSquare pattern concerns the commutativity and consistency conditions that must hold when the meta-model has closed loops of relationships between types.

> ***Note:*** We discussed the importance of having a naming pattern for the meta-data class/instance relationship to avoid confusing it with the Smalltalk

> class/instance relationship. Naming patterns suggested included group/member, operation/knowledge, and dynamicClass/Instance (as against static, i.e. Smalltalk, class/instance). I argued that method names should also use a consistent preposition, e.g. choosing ‘of’ as the preposition would mean writing all methods with meta-model class parameters in the style ....<selectorName>Of: aDynamicClass...

Fowler’s Accountability pattern is one write-up of the Entity-Relationship pattern. (When working with database people, ensure they understand that ERs in the meta-model include abstract types; they do not map one-one to database tables.)

Explanation is a key issue. Some people just don’t get the meta-idea, however often explained. I mentioned the meta-class subclasses class pattern I developed for my system. Joe agreed that it, with visual tools for displaying instances (which almost any such system would have), would be a good way to help newbies understand what meta-data was about, and asked me to write it up (he thinks no write up of it in public domain today).

Strategy is the pattern that assigns behaviour to the meta-model. A strategy can be a business rule. These can be logical - simple primitive rules and compositions via AND, OR, NOT - or numerical (one paper on their site gives an insurance calculation example). Joe remarked that these can become complex (I agree - indeed I question this way of building the meta-algorithm; I recommend meta-programming). He then talked of building interpreters to execute the rules. I raised the alternative of generic algorithms implemented in Smalltalk and overriding the compiler e.g. to execute rules (as we have done in a production system). He agreed that interpreters could become complex; they were best when there were many and changing rules simply composed and without large transitive closures.

Medical observations, where research advances and policy about what data is collected on certain patients in certain states changes frequently, is a perfect example of a domain area needing to be handled in meta-data. (In one contract, his clients were so delighted at the flexibility of this that they started wanting to use it for all their data, which was inappropriate.) In the example system, the strategy pattern applied a simple validator which checked whether an observation was within an acceptable value interval.

When this system was built, the UI people found it took four weeks to build a dynamic GUI layer to display the first model, thereafter two days to show the second model, two hours the third, etc.

The meta-modellers created meta-data editors. Theirs were meta-data specific but Joe agreed that editors that could browse both instances and meta-instances would be good for system editing and also for explaining meta-data (and he has seen systems that did). The whole system took five years of effort but that includes UI, persistence layer, security, handling politics with the client (who was a governmental organisation), etc.

Models and data require version-control to maintain consistency, else, for example, an observation valid when it was made might appear invalid due to a change in the validator's meta-data. The history pattern handles this and is written up on the website. A change to the meta-data has to be tracked as a release, needs XP test cases, etc., just as if it were a code change.

He then showed three other examples:

- Insurance: rules were values composed by simple arithmetic (Ralph and Jeff Oakes have good paper on this system on the web site)
- Document workflow (Argo: Michel Tilman, Martine Devos): the fullest and most powerful example of meta-modelling he knows (I agree). It handles complex constraints, business rules, dynamic UI creation, scripting rules for system events, etc.
- Objectiva telephony billing example (Francis Anderson). Accenture used to take 100 years of effort to build telephony billing systems for customers. Objectiva did the same in two years effort. (Some info on-line from Joe's website but no paper as info is proprietary.)

Don't used meta-data for:

- error and warning messages to user
- relationships between actual Smalltalk classes
- variable inherent to the design

because you don't want to re-implement the Smalltalk language.

Development and Deployment Platforms

Visual Works Web Services (VW User Session)

I missed some of this and due to the crowd (standing room only by the time I arrived) could not type up the part I caught; my write up of Alan Knight's ESUG talk covers some of the same information (navigate to it from http://scgwiki.iam.unibe.ch:8080/SmalltalkWiki/117). Alan also talked about work that uses a meta-model to keep customer web interaction information in context.

Visual Works BOF

Some people like Store, some don't. The key is not to assume that you use Store the same way you use Envy. VW 7 will come with documentation on this. Store prereq tools will be available in the RB (or Joseph Pelrine's tools). Store lacks an equivalent of an open config map; bundles require versions so are not an equivalent. James Robertson and Alan Knight agreed that they now realize that Store does need such an equivalent. Petr (of Fractal) remarked that his group had already implemented unversioned bundles precisely to achieve this. (They would be happy to provide this and other things they've done over the last two years for demo but would like some pay back if it was shipped with VW and used. James remarked that an arrangement similar to what is already done for the RB, for Terry’s debugger and perhaps also in future for a demo version of Silvermark’s Test Mentor, might be possible.)

VW likes short release cycles because they can’t go too far down a wrong direction in six months and because short release cycles means less time spent maintaining patches. James wants to build releases from current valid features, while teams implement features as fast as they can but not committed to any given release. Most (not every) patch is on the Cincom Smalltalk wiki. They want to add ‘get all patches’ to Store but it’s not due for the next release.

James would be happy to put VW’s SUnit tests on the CD. The VW developer programme is semi-by-invitation. It can’t get much larger without overloading Cincom resources.

VW’s customer base is 60% on 5i3/4, 25% on 2.5/3.0. The ObjectShare confusion means sometimes customers contact them whom they didn’t know about (records lost in ObjectShare hand over).

In June 2002, 5i.2 with Envy will become legally unavailable for new purchase. (IBM OTI group now working on Eclipse so IBM VA group is now also supporting Envy, hence their dropping of Envy support for VW was unavoidable and they asked way too much to sell it.) James thinks VW is better off without Envy since they plan to eliminate stripping in favour of loading parcels into a runtime image and that would be hard in Envy.

Petr remarked he pays so much more for VW licences in the Czech republic than in the U.S. that buying licences at the conference had paid for his attendance. James told him to

- talk to Jason Ayers and check whether it’s due to different tax rates
- if it’s not, tell his sales rep to fix it; there should not be any non-tax-induced difference

A set of stories wanted but not currently being worked on by Cincom people could be posted and Smalltalkers with free time could work on them (maybe use VisualWorks Wiki IRC as a means of Smalltalkers communicating).

Q. 5.4 then 7, what happened to 6? We noticed we’d only ever released on a prime number, so decided to continue doing so.

SmallScript - Building Modularized Applications, Servers and Components for Integration with Native Host Services, David Simmons, SmallScript Corp
Smallscript is a superset of the Smalltalk language (N.B. not of the standard Smalltalk frameworks; reproducing them is not the goal of Smallscript, although Dave trusts this will probably happen, sometimes

from Smallscript Corp, mostly from third parties). There are 10 times as many scripters in the world as all other programmers. Scripting languages are by definition dynamic languages. Smalltalk was the best so he started from there but wanted to support all other dynamic scripting languages' features. Thus he wanted a common dynamic language architecture.

Smallscript runs on two VMs:

- Smalltalk Agents VM: now in its fourth generation. The Agent Object System has a unified object model (richer than other STs). It emphasises deployment via mainstream industry standards.
    - Example: building DLLs and COMs is hardish in many Smalltalks (Smalltalk MT being an exception) and it is hard for other STs to be servers; they mostly just consumed DLLs/COMs.
    - Example: the Unix/Linux piped text style of scripting is hard to do in other Smalltalks.

  Even good things in ST (keyword syntax, owning the UI, ...) impede mainstream acceptance.
- .NET Platform: the AOS.ON.NET Enabler assembles smallscript to .NET with almost all features. Two are missing:
    - No allInstances calls (a security-oriented limitation).
    - No dynamic Smalltalk-style development. The .NET unit of modularization is the assembly. This static-derived concept means that changing a running application must freeze the whole assembly that contains the class you're changing. This blocks dynamic Smalltalk-style development. Thus the AOS hacker and Wiki hacker won't work on .NET (there are some workarounds). Microsoft knows this is a problem and would like to fix it, but it gives them serious issues with their design.

The AOS to .NET compiler gives the ability to build in AOS for .NET.

Dave rebuilt Smalltalk from the ground up in AOS to make it more modular, able to have small runtimes (2k example) and to scale from trivial scripting up to complex. Dave's focus is to built tools for .NET that happen to be built in Smalltalk, not to build another Smalltalk language. That's how he thinks Smalltalk will grow.

Smallscript has many features.

- Interfaces are first class mixin and aggregation entities.
- Optional typing for marshalling and multi-methods
- Static languages are more fragile than dynamic because the latter assume you'll add code while running whereas static assume not. Dave has selector namespaces as additional help to letting many scripters add code to the same webpage... This also lets sandboxes be built easily. Modules and Namespaces are distinct.

- Smallscript's VM performance compares with VW's, both of which dwarf most scripting languages performance. Scripters can be persuaded to use Smalltalk if they can be shown this.

Object: Smalltalk objects have slots, bytevars, ... Some dynamic languages are not class-based; instances can be given methods and slots dynamically. Smallscript objects have:

- Basic properties: can it be moved around in memory? Can direct reading/writing of instvars be managed by another object? Is there a manager?
- From the object's viewpoint, named slots and index slots are much the same. By-ref slots (named, indexed) are stored in the object.
- Foreign function interface transparent across languages because object model is aware of its types. ST information about an object can be distinct from where the object actually lives in memory: ST-heap, host-heap, C++ new/malloc, Thread Local Storage (lets objects have different structure in different threads).

Characters are virtual objects. An object property is whether it has one-byte chars, two-byte chars, etc., and what set these chars live in. An object has no single permanent memory representation. The VM may restructure an object during its lifetime, e.g. if an interface its class inherits from has an instVar, so is not pure behaviour, then assigning to that instVar causes that object to be restructured by aggregation to have a slot for the instVar.

Class: in smallscript, a class *is* a method dictionary (keeps classes lightweight). A smallscript class is a unit of behaviour, of meta-data, of privilege/scope. A class is a unit of structure but this must be distinguished from the others. Smallscript has single inheritance of field layout but multiple inheritance of intrinsic aggregation and interface composition. This is how Dave strongly believes SI and MI should be used.

Public/protected/private are bogus ideas. It is use, not implementation, that determines these because an object interacts with many frameworks, not just one, playing different roles in each. Every smallscript class is a namespace so can have its own view of other classes. As a namespace, a class is a container of named fields and a scope for message selectors. Classes inherit scope SI via single superscope and MI via importing. Protocols are first class objects and can be run-time enforced.

Modules are the unit of packaging and deployment. They contain code, media, manifest version and security data. In 1993, QKS fitted on two floppies. Within five years it had grown to many megabytes and every package had to drag it along. Dave decided to build Smalltalk by composition, i.e. declaring what pieces your module needs to run, not by decomposition, i.e. strippers / packagers that will usually include too much. Mainstream languages already had standards for this information:

COFF basic format plus PE, ELF, CodeFragments, etc., so Dave used them and benefited from their utilities.

A script is an independent unit of executable source, editable with text-editing tools and auto-generation tools, and accessible to human beings (thus SIF and XML are not scripting languages). Scripts assume they are substitutable for binary code, both semantically and in terms of performance (e.g. can you afford to launch VM for each script execution when 50 scripts are chained together). The Smallscript VM has millisecond start-up.

Dave demoed script writing in Visual SlickEdit, a language-agnostic IDE, of which there are several e.g. Mac's CodeWarrior, to show Smallscript's ability to work with such.

```
[stdout cr << `'nHello world']
OR
[stdout cr nextPutAll:`'nHello world']
```

(The second ` is a back-quote escape to say, 'Run nHello world.')

```
wstsc -eval:"(the time is" + Filename.now.asString)
alert (windowless vm eval exprn)

Compiler cmds: ...
Project name: MyApp

Function [<$entrypoint>
main
  stdout cr << `'nHello world'.
]
```

You could call main by any other name in the above as it is the entrypoint annotation that tells the program 'this is the function you start at'. This script produced a 4k .exe file, plus 1k .rdb debugging file, not needed to run (after the usual demo hiccough: where did it create those files?). The files contain binary and source code. All classes, apps, etc., are strongly versioned; the same user name for two apps will not cause problems.

Next Dave showed an example that launched its own UI,

```
Compiler cmds: ...gui...
...
IconResource id: 0
  ....   \IconFiles\....
```

creating a 10k file.

The current demo license is free, undistributable, and no distribution of apps. The AOS.NET and VBscript folders in the demo release are empty: first .NET Smallscript release will appear mid-July along with VB script implementation running on it. Installation is straightforward (typical windows style). Currently there are 700 tech previews out there with 20 active contributors of utilities (several of which are in the latest demo distribution). There is an issue of which DLLs (6 or 7) should be distributed with it (everyone has six, but for the last fortnight Dave has been working on the latest .NET using 7). The AOS.dll is currently 1.5 Mb but the release should be 800-900k. This file must be distributed with your apps (or auto-download over web when needed). All else is development support, mostly written in Smallscript. All but the .NET extension is and will be free. Dave xpects the .Net extension to cost ~ $2000.

Currently, these is no visual debugger (they use Visual Studio’s debugger). This is the first release with a GUI system so now they could build a debugger. The documentation provided is work in progress. A script dumps out classes and methods to HTML giving a simple viewer. The browser, written in Smallscript, lets you browse Smallscript code but is also a framework for browsing the registry, files, etc. The current samples are examples of esoteric language features, not of typical code; don’t use them as indication of the usual degree of code complexity.

The process of going from source to executable code: Smallscript is written in AML which is XML with flexible syntax to make it acceptable for humans. Thus pure XML will compile fine but AML is easier to write, e.g.

```
Class name: Example
{
  Function [
  selector
    ‶... body ...″
  ]
}
```

The `selector ‶...body...″` part can be written exactly as you would in your smalltalk IDE and in fact is the Smallscript part; the rest is declaring things about its context in an XMLish way (thus the Smallscript compiler front-end is a powerful XML parser and can be extended to handle other things, indeed has been to show icons, files, ...). (His DTD for AML is out of date, unused by the engine; he may write an up-to-date DTD sometime.)

Joseph Pelrine has written utilities to convert VA, VW and Dolphin to Smallscript. He’s used it to convert regex and other utilities. Thus in theory one can develop in a rich IDE and then run on .NET. In practice, use of frameworks in the rich IDE might limit what you can move. However the business layer could probably be moved and then called as DLL from the GUI layer. Moving to Smallscript automatically makes you a COM component.

```
Compiler cmds: ‵-target:dll...
Module name: MyDLL

Function [<$dllexport=callback>
EITHER main(hwnd, hInst, <LPTSTR> lpCmdline ...
OR main: hwnd : hInst : <LPTSTR> lpCmdline ...
OR main: hwnd with: hInst with: <LPTSTR> lpCmdline
...
  lpCmdLine asString alert.
]

Function [<$dllexport>
greeting
  stdout cr << ''nHello!'.
]
```

The above main is calling the standard windows rundll32 which needs parameters hwnd, hInst, etc. Type info is provided for the argument that this main actually uses but not for the others as they are unused (but rundll32 insists on their being there). Calling or being called by external code situations are the only places where you must provide type information. The types (classes) and converters you write in Smallscript.

```
Module name: client dll: SimpleDll
EITHER [greeting()]
OR [self greeting]
OR [client.greeting()]
OR [client greeting]
```

Note that if greeting had had keywords we could not have invoked it in Smalltalk syntax *from the DLL* since the outside world does not understand keyword selectors. (Dave agrees that there are many ways to write this and we need to find the best patterns for reading and writing code; his examples in folder sample are Dave's experiments, some of them showing very bad examples of how to write smallscript.)

```
RunDLL32 SimpleDll,main ...
```

Calling this you don't have to provide the unused parameter types. Callouts to C code, etc., are minimal penalty since the VM is designed to be multi-threaded.

Currently, the frameworks are basic language infrastructure (MOP), numerics, collections, specifiers, GUI and others. There are also the .NET compatibility frameworks.

The GUI framework has been Dave's focus for the last three months, and using it to build the Smallscript browser. The browser is deliberately like the MS explorer and accesses items via URLs. Modification is easy. Sensible defaults make it easy to browse many kinds of items. It was built in one week and is currently largely the infrastructure from which a richer browser will be built. Dave showed its code - typical menu creation methods, etc. A platform-independent layer wraps platform-dependent widget creation code. Dave uses platform widgets instead of emulating. Dave's code mixes paren-forms and keyword forms, e.g

```
Button()
  topLeft(10,10,100,32);
  caption: ‘Say Hello’;
  when: #onButtonRelease do: [‘Hello’ alert].
```

as he finds one or other quickest and easiest.

Button() is a way of writing Button new. Dave showed the method on constructor protocol in Behavior that aliases () to new:

```
()
  tail:return(self new)
```
instead of

```
()
  ^self new
```

for performance; the VM replaces the alias with the actual in the method frame.

Event specializes Method to provide a method that lets another object than the receiver handle it. Assertions are innate to the language and minimal cost. Smallscript supports declaring C structs and other external types with minimal typing.

Dave demonstrated interfaces.

```
  anObject as: anInterface
```

lets you see an object as if it were an instance of an interface of its class. An example showed how the interaction of interface precedence ordering, interface implementation and class implementation provides acceptable MI resolution. Then he demoed selector scoping: an instance of a class with methods scoped to different modules responds to in-scope methods, raises doesNotUnderstand for out-of-scope methods and took the earliest precedence implementation where different scopes had rival implementations.

Sandboxes have secure methods which call their insecure equivalents. Smallscript systems can have many sandboxes and can modify the sandbox on-line, plus, unlike their static rivals which must check everything up-front, they have zero up-front cost since you only check what the user calls, i.e. pay the cost of the secure-calls-insecure call. A simple demo showed a three-method sandbox. Dave also demonstrated that Smallscript, like QKS, has full block closures, unlike Squeak and Dolphin.

Types can be declared as values or expressions in the type algebra, which includes self-reference but is not a completely general ‘any expression returning a type’ system (Dave avoided that so that the types need not exist during the type analysis epoch). Using this, multi-methods can be defined:

```
* <LargeInteger> multiplier
  ...
* <Float> multiplier
```

. . .

He also has implemented an ABIA (Around, Before, Inner, After) framework. The inner method is the standard Smalltalk method. The other three stages are usually unused but always available for use: execution is in order Arounds then Befores (from outside in) then Inners (called method and any super calls) then Afters (from inside out). There is no performance cost for the framework's presence as it is effected on the fly as needed. Example: a utility framework exposes method #foo and a user overrides #foo to do stuff then calls 'super foo'. Later, the utility framework needs to do an additional action before #foo is called. In other Smalltalks, we would use method wrappers but selector namespaces and multi-methods conflict with these so Dave achieves the same effect using an explicit ABIA mechanism: an around method for #foo defined in framework becomes the first action done when #foo is called, even from users' code.

Miscellaneous

J2EE for Smalltalkers, Alan Knight, Cincom

When J2EE was first conceived, its goal was world domination, particularly pushing out microsoft. Its means were supposed to be by providing easier distributed application building ('3-tier systems should be 3+ tear systems') and a web power builder for VB programmers. The idea was to have transparent mechanisms for distribution, transactions and connection management. Everything was very Java-centric, which is why some parts have been supplanted by web services which are genuinely portable. Smalltalkers need to know about J2EE. It has *some* things that not all ST dialects have, you may have to coexist with it, and you may have to give reasons for not using it.

J2EE is a bundle of elements (JDBC database, messaging, etc.) most of which are neither new nor interesting. JDBC is just another not-quite-standard (but no less than others) database driver that you get from the vendors. In Smalltalk, the vendors also have drivers but they are less standardized and not always up-to-date. (One of the useful things Frost could give would be the ability to run JDBC.)

Transaction service JTS is identical to CORBA Transaction Service: not-perfectly-distributed transactions with two-phase commit. Smalltalk has CORBA Transaction Service so JTS is unneeded.

JMS is a messaging service: guaranteed ordered transactional delivery. The Smalltalk equivalent is binding to proprietary (e.g. MQ) services that lack a portable API. (Alan admitted this isn't his area of expertise; some vendors claim to have equivalent Smalltalk stuff).

JNDI Java Naming and Directory Service maps to various services of which LDAP is by far the most important. LDAP is not language-centric so JNDI is unneeded. The Smalltalk equivalent is CORBA bindings to LDAP, DNS, etc.

Connectors give standard interfaces to outside systems. There are few uses and few definers of them.

All these are fairly minor and trivial. The interesting J2EE stuff is servlets and beans.

Servlets replace cgi scripts. The complaint was that cgi required forking a process for script, etc., so doesn't scale. Servlets have simple lightweight protocol, giving flexible control for programmers and automatic header parsing, plus Java is easier than Perl. However it is still awkward since pages must be built by programmers: a minor page change impacts all code. Smalltalk equivalents are VW5i.4 web toolkit, whitecap (VW + apache) Wiki works (has servlet-like protocol). Some similar projects are Swazoo/AIDA, Commanche (squeak), VA web-connect and others. Servlets are like commands, not like e.g. Swazoo objects that know how to render themselves.

JSP, Java server pages (copied from ASP) try to fix servlet problems by templates - HTML containing code e.g., <%/= Person name%>, compiled into servlets at runtime. Smalltalk equivalents are VW5i.4 web toolkit, Tsunami (custom tags only), Squeak, etc.

Servlets and templates have problems (e.g. loops syntax is not good) but together they make up a reasonable web presentation layer.

All these are fairly lightweight things. The thing that really drove the J2EE story was Enterprise Java Beans. The EJB spec is 600-pages and growing. ('People have been researching component and aspect architecture for 20 years: along comes EJB and demonstrates it's ignorant of all that work': Eliot Miranda.) It was very strongly hyped despite being a joke technically. Currently, 1.1 is the implemented version, 2.0 is new. It is now going downhill; people are realising its problems. It's complexity is fractal; equally complex at every level.

EJB was meant to be a server-side component model, an answer to microsoft's. EJB has no relation to Java beans; there are no features in common between the two. EJBs are server-side RMI that divide development into roles: writer, deployer, user. Beans are server (unshared) and entity (componentised domain objects). Session beans are stateless (like CICS, sort of) or stateful. Stateless beans are scalable. (NB performance and scalable are not the same. Performance means go fast. Scalable means do many things without dying. These two things can be opposite, e.g. performance likes caching in memory but scalable systems may not since memory may bottleneck. Many scalable architectures, e.g. CICS, are stateless.)

Entity beans are explicitly persistent and shared. The persistence has a very RDB flavour; the EJB people are explicitly hostile to OODBs (EJB and JDO teams do not get on). Entity beans are always passed by reference (even when local) but non-beans are always only passed by value (i.e. Java serialization mechanism, again even when local call). The spec does not permit optimising this, though some implementations do. Related beans can see local or remote but never both, so your code cannot be polymorphic. This is because Java methods insist on having exceptions in their signature and remote and local exceptions differ (maybe you could cheat by making local raise remote exceptions but Alan is unsure whether that is possible). Deployment results in XML files specifying security, transactions, etc., etc., which exceed the size of the code by a large factor. EJB has had 6 public drafts each of which was declared to be the last. The problems are many. Assignment semantics take 18 pages and must be read to be believed!

Alan overviewed three kinds of component models:

- Widget-like: e.g. java-beans: these are event coupled with strong user/author distinction and no visible-to-user inheritance, and use reflection to view and edit.
- Service-like: e.g. session bean, COM/MTS. These have very little coupling (usually one layer deep), little extensibility and a strong author/user distinction.
- Domain Objects: e.g. most business objects: these have strong relationships with complex interactions so are hard to reuse. The author/user distinction is weak, with heavy use of inheritance and polymorphism. This is the model EJB is trying to componentise but it has none of the characteristics that let the other two models work.

There are very few implementations: Alan tried to find examples with little success (one dumped EJB between Alan's finding them and his giving an earlier talk). You must restart server after loading/upgrading. Typical debugging instructions start by telling you how to insert print statements in your code, a real blast from the past. Paraphrasing XP's motto, EJB is 'the most complicated thing that will never work'.

For a workable development process, they advise you write fine-grained non-bean business objects and wrap the beans (session, no entity) at the end. So why did you pay all that money for EJB? Deployment is very complex.

Threading was supposed to be made simple by enforcing bean-thread isolation (only one call in bean at a time) which means you either do pessimistic locking per entity or make per-user copies of the bean. TopLink did transitive closure copy on write and were told that was a spec violation; they had to do it for read as well!!! Loopback calls (B calls A calls back to B) are not allowed. Re-entrant beans are strongly discouraged in the spec.

Persistence: Alan could have talked for hours without exhausting the problems.

It's a truism that distributed transactions cost 100 times normal ones performance-wise and that 98% of the time you don't want them. EJB forces you to have them always. Further, while some of the distributed features are useful (e.g. pooling), others buy you little. Clustering is also a problem.

Today, prominent authors are saying, ‘Don’t use entity beans.’ Session beans are unnecessary though fairly harmless (typically, servlets and domain objects are sufficient; a session bean between the two buys little). A major customer found a factor of 200 slowdown between using TOPlink without and with entity beans.

Smalltalk has no equivalents to EJB, thank God!!! Automated failure for sessions and declarative transactions are its only interesting ideas and they’ve been done much better, albeit not standardised. The value propositions of early Java lay in standardising things that were simple and already well understood but gratuitously incompatible (e.g. JDBC). In EJB they tried to standardise something that was not simple and failed hopelessly.

In Smalltalk terms, servlets and server pages are useful; perhaps not the best but viable and familiar to many people. (Alan knows people in Cincom who are frightened by Smalltalk IDEs but very happy to read the Smalltalk equivalents to asp syntax.) Never use EJB, even if you’re implementing in Java. Session beans add extra infrastructure, give little, do little harm. Smalltalk could use a standard LDAP interface.

Alan ended the talk by taking us through a slide giving a hypothetical answer to why not use EJB.

Q: Corba 3? Early Corba 3 was meant to be J2EE-compatible. Annick Fron says it has changed since then. (Spec is even thicker than J2EE.) Alan’s vague impression is that existing CORBA is being much used, new Corba ideas are being little used and may not catch on.

Q: Ever recommend Java? No.

Q: Client has J2EE. Can a Smalltalk application be added to this? VA has Java RMI. VW does not but everyone except WebSphere has a Corba interface.

Q: What’s the next hype coming up now EJB is going down? Web services, but J2EE is not helped thereby. IBM and Microsoft, not SUN and BEA, are benefiting from this. Martin Kobetic will be happy to explain that web services are more complex than Corba and superfluous, but they are language-neutral and Smalltalk-friendly (any XML-talker friendly) so much better for us.

Lastly, Alan described the absurdity of assignment semantics in EJB 2.0. EJB 1.0 has no relationship semantics (as I complained two years ago when Nortel wanted to use it). EJB 2.0 has hidden relationship enforcement semantics. Thus, for example, making an Employee-has-Address relationship have 1:1 cardinality means that if two Employees marry and one has their address assigned to be their spouse' address, the said spouse' address is invisibly set to nil.

Keynote: You have to either laugh or cry, Ron Jeffries
Ron had a summer job in 1962 in Strategic Air Command, where he learned that everyone who worked there was sure they would not survive a nuclear war (enemy would make sure of that), thus giving them what he regarded as a very healthy attitude to the idea of starting a nuclear war. There he learned to write fortran programs (but did not see the computer that ran them; his clearance wasn't high enough for that. Later, he got cleared to see the computer). It's often no fun working for the government. They have strange rules, forms to fill in, accounting, etc. His first boss told him, 'You have to either laugh or cry.' Likewise, you have to laugh or cry about Smalltalk's history.

Ron has programmed in a startling number of languages not (almost) including Java. There are only two languages worth looking at: Lisp and Smalltalk. Lisp makes him think in new ways. Smalltalk makes him think in simple ways. His Smalltalk experience began by building a wonderful system (generic tax system) that did not deliver enough in time to prevent its company going bust.

Smalltalk gave the world:

- Objects
- Refactoring
- XP (Kent Beck's formalising of what Ward Cunningham does naturally)

However Smalltalk also gave the world C# and Java (but not C++; why does C++ exist?). Smalltalk also gave the world Ruby.

Smalltalk has created programs that probably could not be translated to other languages and certainly could not have been created in them (much financial work he has seen but also graphics systems for pipe scheduling, etc., etc.). Smalltalk is best at doing complex things simply. We need to communicate this by creating a rich on-line community, talking to schools and universities, etc.

His motto used to be Quality Will Out. He still believes this but now realises he wants Quality to win in his lifetime. Forces from Microsoft and Sun and IBM have moved the market toward technical mediocrity (put charitably, they pursue different goods from those Ron as a techie cares about). Smalltalk's own forces (farces?), such as ParcPlace, ObjectShare and Digitalk, harmed our cause.

He partnered with another programmer to write an extended set theory product. All 6 people who bought it said it was wonderful. He should have partnered with a marketing guru. Quality needs money and marketing. It also needs (and Smalltalk gets) commitment. The objective of many of us is to go on programming in Smalltalk. Educators (e.g. Ralph) want others to program in Smalltalk, to get what we get. Ron would like Smalltalk to rule the world but does not know how to achieve this. We would all like Smalltalk to be recognised. But Ron thinks the real question should be, what would be better than Smalltalk? He sees Ruby, .NET and above all Squeak as pointers.

Other Discussions

We swapped so many stories of Java projects that failed dismally (the once-much-touted Java phone that never managed to make a phone call being my contribution) that Jason suggested setting up a JavaFailures web page. However I think it is becoming no longer necessary; Java is on the way down, fashion-wise. .NET is the new thing but as Smalltalk is a listed .NET language this is not creating the same political problems for those of us who must persuade managers to let us use Smalltalk.

James told an anecdote of someone who, after 20 years of defending static typing, got bitten by the Java primitive types problem, complained bitterly about it, was told, ‘In that case you want Smalltalk’, and stopped arguing for static typing. However, while the primitive type issue can be a useful way of getting people to listen, of course James’ acquaintance was really still confused; Java could have eliminated primitive types the same way Smalltalk does while retaining static typing. I described how hard I had found it to explain to people unused to Smalltalk the real reasons why static typing is harmful. Show the absurdity of having to cast an Object extracted from a collection, or having aSpecificType.clone() return Object instead of this.getClass(), and even capable people reply ‘So what! Typing a cast doesn’t take long.’ Explain they might not know what cast to do and they reply, ‘But I must; I’m about to call a method on it.’ To them, the prison walls of static typing are the edges of the imaginable world; they have never been beyond them even in thought. Primitive types get in the way of what Java programmers know they want to do, so they’re a great talking point. By contrast, I had to talk for hours to a capable researcher keen on XP (whom I had hired at a non-trivial hourly rate) to get him to see that, yes indeed you might want to write a framework where no single type value could possibly be the right return for a method and a rational and likely refactor would require writing new casts on every call, most of which would probably be in code you didn’t own. And when he finally saw it, he ventured that, as it took so long to explain, and he’d never met such a case before, might it not be one of those issues that are correct in theory but too esoteric to be worth bothering about in practice :-/ !!!. Jason remarked that people usually accept Smalltalk because they’ve found they can do something in it they’d failed to do elsewhere, not because they understand why.

Over dinner, Eliot described seeing Don Roberts and John Brant in their tutorial refactoring awful code with a truly bizarre Fortran-derived implementation of log base 2 (print it as a string and study the size of the resulting WriteStream collection!!). At one stage in the rewrite of this dreadful method, the test failed as it should, but then passed when run a second time, causing much alarm. The cause: Alan Knight's fix that silently makes immutables mutable on a certain exception. It was suggested that it should do so more noisily, if at all.

I asked Don and John which refactoring rewrote a method to be called from one of its parameters. It's 'Move to component', which combines 'extract method' with what I wanted. John explained that the implementation of ReferenceFinder was VA-oriented; if he had not been keen to keep the same code in both dialects he could have written a VW version with a single dictionary instVar for backlinks. However as I've found what he wrote easy to specialize, perhaps easier than one that had a tighter implementation, I'm not complaining.

Follow-up Actions

To Do:

- Jason Ayers suggested that the VW7 Refactoring Browser documentation include mention of the Camp Smalltalk custom refactoring project pages so that people can use them for seeing how to write custom refactorings and adding their own. Contact Bruce Bowyer about this (and add more refactorings to those pages).
- Email Ann Anderson and Petr Stepanek a copy of my XP-rience talk.
- Write-up my 'meta-class subclasses class' pattern (Joe Yoder has seen it used but he thinks it's not written up anywhere).
- Email Paul Baumann my deep comparison framework (he thinks he may have a use for it in his SRP framework).

Requests:

- Ask Joseph Pelrine for his resumable test cases utility.
- Ask Jeff Odell about the checking of resources when running in SUnitBrowser (I hoped Jeff would be at StS but he couldn't make it).

Conclusions

Good talks and good conversations. I look forward to next year.

- Seven parallel tracks is too many. I missed many talks I longed to attend because others clashed with them. On the other hand, it's not easy to say what could have been done to improve things. We might have opened with the songs and a 'Welcome' from Tom Nies, then started in promptly. We might have had another day. We might have had repeat slots for popular talks. Or maybe I should apply Jane Austen's maxim about holidays - 'Delightful in being much too short' - to Smalltalk conferences and just accept that there will always be more going on than I can capture. (I hope the remaining talks will appear on the website soon.)
- Java is going down. Loads of it will live for ages but the political problems it used to give anyone arguing for implementing in Smalltalk are now diminishing as its flaws become more admitted and more admissible.

- Meta-data patterns are recognised and some are documented, but not all (e.g. Michel Tilman's Argo remains an example of what could be done rather than a model of what is being done). Meta-programming patterns and frameworks seem even less available than meta-data ones (Paul Baumann commented that I was one of few who had used his deep copy framework). I shall see what I can do in this area.

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

* End of Document *