ESUG, Ghent, August 30th - September 3rd, 1999

This conference was a lot of fun, with lively and enjoyable discussions during and outside the talks. Peripheral bonuses were that the weather was superb throughout, and it seems impossible to get a bad meal in Belgium.

Author’s Disclaimer

This conference summary was written by Niall Ross (nfr@nortelnetworks.com), a senior researcher with Smart Network Technology, Nortel Networks, Harlow, U.K. It presents my personal view. No Nortel Networks view is expressed or implied.

Style

I occasionally write ‘we’, ‘our’ when contrasting work going on in Smart Network Technology, Nortel Networks, with work being presented.

Summary of Talks

Tutorials

Pocket Smalltalk, Carsten Harle, Senior Consultant, Germany
Pocket Smalltalk is a Smalltalk implementation for the Palm Computing platform. It has a 24Kb VM and full access to the PalmOS. The talk discussed the implementation and gave programming examples.

Palm pilot has 512k to 4Mb RAM, 2Mb flash Rom but extreme memory limitations as dynamic heap is 16-32k; rest is in protected databases. Applications need to be in the 30-70k range. Smalltalk gives the most functionality per kilobyte as so much is in the virtual machine (in a database); C on PalmOS is a real pain.

The speaker first tried to port Squeak Smalltalk to Palm Pilot but

- Squeak is too big: min 0.5Mb (PocketST is23k - 4500 lines of C - with 18k of base class libraries)
- Squeak is heavily optimised to 32-bit; PocketST has 16bit object table.
- Squeak does all the window painting itself (PocketST uses PalmOS GUI functions)

The cross-development environment runs on PC in dolphin smalltalk (ports to VA, VW ‘heard of’) and can simulate the device, except for PalmOS access. It occupies 600K and is available on www.pocketsmalltalk.com (open source from March 1999). Example application, Dungeneers, has 100 classes, 9000 lines; 40k code, 10k graphics, 23k VM = 73k.

To save space, symbols are replaced by small integers so one can’t get a symbol from the user and ‘perform’ it (ordinary ‘perform’ works O.K.). There are no method dictionaries, just pointers to compiled code for method. These two features are why there is a development environment on the device itself. There are also no class instance variables (speaker agreed this was a limitation) and no exception support; both of these may appear in a future release.

The speaker used a palm pilot emulator on PC to demo. He showed an application, edited it, produced a stack trace (perhaps not quite deliberately but it was impressive to see on the palm pilot), corrected it and showed it working.

The speaker stressed that a 4000 lines-of-C PocketST VM compares very favourably for size both with the existing unofficial Java VM on palm pilot and (even more so) with the one that Sun has promised but has yet to deliver. "Everyone knows that Java for Palm Pilot is coming; no-one knows that a smaller faster Smalltalk for Palm Pilot is already here. This is a general Smalltalk problem.[1]"

Currently, three $5-each games are the total commercial portfolio of PocketST. The speaker felt that the business value of PocketST would be not so much in development environment (tools companies are finding both Smalltalk and Java IDEs hardish to sell) but in writing cheap web-delivered applications. Several companies already live on these ($5-10 each - download, try and register) but they have very little functionality as it's hard to write and fit into the space in C. With PocketST, real applications for e.g. banking access become feasible and much faster than via loading NetScape or similar.

During the conference, the speaker ported a game from VW3.0 to pocket smalltalk and demoed it on the last day; this took him only a few hours overall.

Building highly configurable and adaptive frameworks, Michel Tilman, Senior System Architect, Unisys, Belgium

Building complex applications in a world of ever-changing needs requires appropriate techniques. Meta-data (data about data) and active object models (explicit object models that are interpreted and may be changed at run-time) are increasingly applied to build highly configurable and adaptive frameworks.

This tutorial built on earlier presentations at OOSPLA, ECOOP, BSUG and the Illinois Meta-data Pattern Mining workshop. By means of a few smaller examples and a larger case study (Argo project), this talk explored:

- patterns for building these systems
- trade-offs: bootstrapping a highly reflective system requires the user to be focused on what they are doing; easy to get initialization conflicts.
- how to obtain good performance: only looked at issues related to their use of Oracle RDB (which I judge a serious impedance mismatch to their problem)
- the relevance of the (reflective) facilities of the Smalltalk language

1. but see article earlier this year in Dr. Dobbs journal.

The Hartford User-defined product Framework is for financial (e.g. insurance) service definition. Without meta-data, it would have needed 10,000 classes to handle all the features. They used the variable state pattern, the strategy pattern (using blocks), component type (a list of a components attributes, similar to our ‘sources’ dictionary in our meta-classes), etc.

The Argo framework concerns management of Flemish schools in Belgium (an extensive description of the Argo framework will appear in a three-volume book on object-oriented frameworks from Wiley Computer Publishing, 1999). The customer was moving from a control-oriented to service-oriented culture and knew there was a reorganisation coming but didn’t know what it would be. Hence they needed to store meta-data as well as data in their database, plus tools that dynamically interpret the meta-data. Meta data is list of object types with properties, associations and rules (and events, to drive the rules). Associations have two ends (one can be hidden to preserve encapsulation) and are objects in their own right, to allow for association attributes, etc. Association instances and types are traversable by the tools so that a UI widget that holds an association can get the list of possibles, etc. (e.g. find all who play role X in department Y).

The same tools are used to view the meta-data and the data. The scripting language is Smalltalk, extended to support database queries. Properties are descriptive (slot-based) or virtual (in database). The implementation uses the variable state and proxy (database access) patterns. Methods and Rules are stored as compiled code in the database and accessed by doesNotUnderstand (slow; they are speculating about Just-In-Time generation of classes and methods; by questions after the talk I satisfied myself that this was at least in part a problem caused by the impedance mismatch of their database as against e.g. GemStone).

Their goal is to bootstrap the framework, replacing the initial set of hard-wired tools by objects configured within the system. This requires more reflection; the meta-data must be self-descriptive. Rules must be applied to meta-data as well as data so that much of the innate behaviour could be provided by meta-rules.

Argo was built on VW2.5 with VisualWave. It took 2 days to port to VW5i. Overall it consumed 4-5 years of 2 people.

They use a relational database, Oracle, as their database, with an elaborate two-layer bridge (abstract OO model, abstract relational model) to make them as database-independent as possible. Much of the talk concerned how to overcome the problems of the impedance mismatch caused by this choice. They flatten inheritance in the RDB tables (therefore many sparse tables; standard Oracle problem with OO schemas.) They have a hierarchical cache. They use hierarchical transactions with inherited locks, commits/rollbacks. (They looked at design transactions but haven’t yet needed them.) Much of the talk concerned issues more or less directly related to handling the impedance mismatch to a non-OO database (so reducing its interest to me, as we use GemStone/S OODB).

Their meta-model is less integrated with their model than ours is. Generic classes, Type, Property, ..., are used to describe the meta-data independently of the model(s) of the data. This may be why making the meta-model self descriptive is a task instead of being an immediate fact. Overall, I felt we were ahead of them in meta-data design.

The Classification Browser, Koen De Hondt, MediaGenix, Belgium
There are many browsers in Smalltalk. Often they use their own APIs and repository, duplicating development effort, resources in system, etc.

The Classification Browser is based on a general model to organise software entities, called the software classification model. Classifications can be considered enhanced categories with which classes can be grouped in arbitrary, user-defined ways. Apart from the support for organising software entities, the Classification Browser also provides integrated support for browsing senders and implementers (e.g. capture browsing history), and for browsing acquaintance relationships. Classes and methods can be viewed in different ways, so that the developer can choose the most appropriate way of browsing for a given browsing/development activity. The browser can be used for reverse engineering; it is able to produce UML diagrams. It can also be used to manage software evolution, by tagging all methods changed by a refactoring with a record of the refactoring's motive.

Classifications are virtual (given by the Smalltalk system being browsed, not user-definable) and user-defined[1]. One can acquire items of the same name in user-defined classifications. The tool distinguishes them by showing their virtual classification by an icon next to the name. Icons are also used to distinguish user classifications from virtual (e.g. to distinguish a view of a class of which only one method, found in a search, is shown, from a view of the whole class). The browser shows classes and metaclasses in the same virtual classification list as some classifications cross the class-metaclass divide. It is well designed to avoid creating vast numbers of windows.

They have integrated Smallint (but with rule selection and code marking still to do) but not yet the refactoring browser (should be done soon). The speaker has written one refactoring. It took him three days to understand how the refactoring engine worked, then two hours to write the refactoring.

The demo was of understanding a new application by using code comparison, root class navigation, etc., ending in generating a UML diagram in Rational Rose. It looked good.

In response to my question, the speaker felt that the 'left hand side' of the tool could be used for general user-driven classification browsing on any basic (virtual) model. The classes 'ClassItem' and so on would just need to be changed to return the new virtual elements. It was not clear how pluggable the right side

1. The distinction is eminently sensible but this usage of the term 'virtual' struck me as counter to standard database usage.

of the browser was; at present, it expects Smalltalk objects and brings up browsers on them. I stressed that the classification problem he had solved applied in other areas (not least our own meta-modelling tool) and a pluggable framework that solved it might find itself used in many Smalltalk systems as well as to write those systems.

Advanced Topics in ENVY, Joseph Pelrine, Daedalos Consulting AG, Switzerland

ENVY is an excellent OO code configuration tool, used by most Smalltalkers and by VisualAge for Java, but the documentation leaves something to be desired, especially as regards extending the ENVY environment. This talk provided an in-depth look at the ENVY system API, uncovered some interesting things, and provides numerous examples of tools and scripts which can immediately be used to increase the productivity of an ENVY site. Generally, it aimed to show how one could work with ENVY not against it.

- Layered architecture: the slower to change the lower the layer; only references to the same or lower layers.
- Start with one map per layer. He recommended not using required maps, but admitted this was debatable; I couldn’t follow his argument.
- He recommended not over-using subapplications; this I agreed with.

Test prereq and packaging: speaker uses eXtreme Programming on two week iterations. He always has a minimal delivery package test by the end of the first iteration on any project.

It’s annoying that moving a class definition or extention to a new (sub)application loses that class’ history. Alan Knight is working on a hack for this. The book and website contains a number of tools to check packaging and prerequisite checks, poor initialization (e.g. of Globals), etc. Prereqs are direct, implicit (transitive closure) and indirect (class references method of other class that is in extention of that other class that is not in prereq is hard to catch).

Library Specs are classes convertible to arrays or strings. Each spec holds app, subapp, class, version ... These are the basis of many of their tools, for example their three-way differencing tool that lets you merge two streams. This ties to their ‘release clean classes only’ command that stops you releasing an edition that clashes with a rival stream; instead you need to browse three-way differences between the last common version, your version and the rival version.

Envy window classes are poor, as are many other parts of its structure. It has lasted a long-time (so good marks) but has much that should be cleaned up but, because of the need for backward compatibility with the repository structure, has not been. They have built subclasses of EtChangesBrowser and made it work in VW (almost) and in VA (for last half-year). (They do most of their work in VA; will port to Cincom VW ‘real soon’.)

The Envy Error Reported is documented in the preferences workspace. You can subclass and override the error reporter with your own (see slides for details). They do this to hook error 49 (‘bad prereq’) into Kent Beck’s testing framework, plus ‘nil Globals’ error.

The Envy application manager can be overridden e.g. to enforce company standards for version name incrementing algorithm. Theirs is <release aimed at> < release> <date> <writer’s initials> (using EmUser>initials).

User Fields can be attached to ConfigMap, (Sub)Application, Class, CompiledMethod, EmUser. They can only store strings: objects are stored by serialization. Local UFs apply to a version of an item. Inherited UFs apply to the item BUT this is achieved by coping them to each new edition as it is created, so a change will vanish if you load a version from before you made the change. You can use user fields for application attachments if you want to wrap .zip files of documentation with release versions (N.B. there is no delta on user fields so this can bloat the library). They have written a transaction controller to let you add multiple fields in a single rollback-able transaction.

They use the Envy API and prompters to write scripts, e.g. adding a user, replacing the old user with them in apps, etc. This uses a useful changeUserTo:aUser while:aBlock method. They also use many other ill-documented methods (see slides).

They have developed checkpoints to let people move open (sub)applications and config map editions (classes must be versioned, subapps must be released) between images. The original database still has the open editions. The one the edition is moved to has versions but their version names relate to the edition timestamp. They will add code to purge checkpoints.

Vendor application extensions is a new feature that lets vendors extend the Envy applications with new menu items, etc., with visible:aBlock (like enable: and toggle:) evaluated at run-time to avoid your menus becoming too tall.

They have added refactoring browser menus to the applications browser and made ENVY-relevant refactorings, e.,g. MoveMethodRefactoring (for utility methods that don’t reference self, super, instvars, etc.) does type checking (to guess which class you’d want to move it to) and offers a choice of which Envy application to move it to.

Questions:

- speaker tolerates releasing open application editions into config maps; others suggested this was a problem as others could release classes into your edition (I only allow open editions in personal developer maps)

- storing objects in user fields: one can be tempted to hold onto your image because browsers store history (last method visited, etc.) and classes have class instance variables (e.g. for meta-data) which you don't want to lose by re-initializing; perhaps user fields could store serialized objects - but these objects would have to reappear with the correct mutual cross-references (and of course one could only store objects of classes tied to the storing application or config map by prereqs or similar)

See www.daedalos.ch or www.daedelos.de/~j_pelrine/ for Refactoring Browser, etc., downloads. Get the book 'Mastering Envy/Developer' for the rest (hopefully out by OOPSLA in November, should definitely be out by December).

Ultra Light Client, Sanjay Madhavan, OTI, Zurich, Switzerland
The IBM VisualAge Ultra Light Client (ULC) is a new architecture to address the fat client problem. The principal idea is to run applications on a centrally controlled application server. Only the presentation part of an application is run on the client's desktop. The presentation part is implemented by a universal user interface engine (UI Engine). This occupies 250K on the client machine.

ULC was developed by OTI in collaboration with Credit Suisse and UBS (Swiss banks). It went on general release in April 1999. These banks found their clients were getting fat, hard to test and distribute, and the desktop bloat left inadequate room for business applications. They also wanted to use some Java on the client slide while also using Smalltalk on the client and on the Server. They claim that the Smalltalk and Java APIs feel like Smalltalk and Java respectively, even though they support exactly the same feature set.

Thin client development is non-trivial:

- where to make the split
- distributed object programming issues
- naive approaches don't scale
- Java applets get big just as quick as any other language
- keeping client and server deployment in synch

Their answer was ULC. It produces thin, easily distributable, low bandwidth client applications.

The talk demoed building and running a ULC Smalltalk application. The style was very similar to that of VisualAge for Java (as one would expect since VAJ is built on VAS).

Extreme Programming, Kent Beck, Daedalos Consulting AG, Switzerland
Kent spent many years becoming a better and better programmer. At first, this helped his projects but a point came when being a better programmer didn't make his projects get any better. So he looked at software engineering and, as someone who lived and breathed code, decided that most of it was wrong. Some of it is

wrong because of discipline-envy; it borrows from e.g. civil engineering the idea that because the cost of change is higher as time goes on, the big decisions must be made early and frozen.

As a very competent Smalltalk programmer, Kent knew that this idea (that the cost of change necessarily goes up) was not true. He could make the same change after a month or a year to his programs for much the same cost in effort and footprint. The standard cost of change curve is wrong for well-planned Smalltalk programs. (Kent programs at 1/3 of his Smalltalk speed in Java but believes his programs still show this same stable cost of change curve. He knows a few people who program in C++ and think their curve is like his. Smalltalk is the environment in which maintaining such a curve is easiest but it can be done in other languages. The key to writing such programs is testing, refactoring and pair programming.)

This discovery was what set him on the path to eXtreme Programming; everything else in it follows from this. XP is the extreme form of the iterative lifecycle; each cycle takes two weeks. The lifecycle is:

- Stories: testable features written by customers and quickly (days, hours) evaluated by designers for whether they are meaningful and implementable. This is not what the designers commit to doing.
- Releases: a few iterations-worth of stories that make business sense together.
- Iterations: a set of stories that are completed (ready to go into production) in one-to-three weeks (e.g. phone system would be able to make one call, payroll system could print one check, ...)

This reduces the cost of each decision and teaches designers (more quickly than any other way could) how to make better decisions. The process is:

- collectively, designers estimate time for team to do story-set for iteration
- collectively, designers list tasks within iteration; individuals sign up for tasks, individually estimate them (e.g. 1-3 days)
- write test case for task; the test case will prove next day, next week, next month and next year that that task was and still is finished
- (re)write the code till it passes the test

At first Kent wrote tests after coding. Then he remembered reading a very old book that said to take the input tape (remember tape?), write the output tape for it, then code till the input tape provides the output tape. When your code passes all the tests you can think of then you are done; you can’t write code to pass tests you can’t think of. If you find more conditions later, you rewrite your code then. Tests are written in Smalltalk but are unlike standard Smalltalk; no loops, no conditionals. (A constraint language integrated with Smalltalk might be of value - c.f. the rules-based language we implemented by subclassing the Smalltalk compiler.)

The rule is that customers can’t change their story inside an iteration (but they can point out bugs to fix); in Wall Street, one might have to relax this rule as the market rate of change is too fast.

Pair programming is an add-on, not an essential part of the method. Use it if you want to progress as safely as possible. Kent also keeps 3 x 5 cards (a notebook would do as well) in which he notes down ideas as they occur so he can finish one problem before thinking about the next. The XP programmer personality (especially if not paired) needs at least a modicum of courage and conviction (Kent recommends the book ‘Crossing the Chasm’).

After this description, Kent demonstrated by writing a simple webserver. He wrote his first test, putting it in the same category as the code (he was about to write). Then he kept running it, writing code in response to the errors it threw up, till it passed. Then he wrote his next test, and repeated the cycle.

It is common to expose more of the class to the testers than generally. Kent put such additional accessors in protocol ‘private’ (another solution mentioned was to put them in a test-specific Envy application, as we do).

[I observed that the interface (called TestRunner) to Kent’s SUnit testing framework gives a significant usability improvement to the raw testing framework I pulled off the wiki web server not so long ago; SUnit (with TestRunner) is now available from the XP wiki server.]

To sum up, the natural state of a software product is in production, being evolved, and simultaneously in use, being maintained. If you implement a feature that it turns out you don’t need, you lose money and opportunity. If you’re uncertain, just wait. (Kent is here using the Elizabethan motto: ‘To enjoy the benefits of time’; must suggest it to him.) He explained this by an analogy between stock options (the right, but not the obligation, to buy at a future time; the cost to buy remains fixed whereas the value of the stock will only be known for sure at said future time) and stories (the right, but not the obligation, to implement a feature at a future time; the cost to implement remains fixed whereas the value to the customer will only be known for sure at said future time). Just as stock options are more valuable when the future is uncertain, so XP delivers most value in just those cases where the standard lifecycles have most difficulty: when the customer isn’t sure what they want, how much they want it or what their priorities are.

XP was first used five years ago; that system is still in use. Its ‘brochure project’ is C3 at Chrysler (40 years effort) which is still delivering stable quality code; they have glitches and they make mistakes but they recover from them. An example that it does work for projects that do end is the VCAP project at Ford: 4-5 people spent 18 months building a system that was then cancelled for upstream reasons; they wrote a 10 page booklet on how to run the test cases and listing interesting places to start in reading the code, and put the code to bed; then customers gave them money to finish the project and they found it easy to restart and do so.

Lastly, two important quotes:

“One of the great things about the web is that it’s trained our users not to be so picky. They’re now accustomed to seeing UIs that look like garbage and change without notice.”

“Being able to refactor is vital to surviving change. GemStone/S is so good because it alone supports migrating live data in an evolving schema. GemStone/J (their Java product) can’t do it and all other database vendors have never thought of it. GemStone don’t realise why their product is so good.”

VisualAge Tips and Tricks, Greg Hutchinson, Object Oriented Ltd., Switzerland
This tutorial focused on some useful tips and tricks to help developers make their coding easier and more reusable. It also explained some less-known functionality provided in Visual Age for Smalltalk.

The Eavesdropper Pattern: occasionally the widgets that are supplied with your favourite dialect of Smalltalk do not have all the functionality that you may desire. Usually your choices are to subclass the supplied parts, or decorate (or wrap) the supplied parts. The eavesdropper pattern is an alternative to the above. It requires the class on which it is eavesdropping to generate sufficient information via an event handling mechanism, e.g. dependency mechanism, that you can control those aspects of its behaviour that the eavesdropper wants to alter (see his article in The Smalltalk Report).

Greg demoed setting an eavesdropper on a window widget to ensure that it could never be resized smaller than its initial size. He then showed how to use the windows clipboard for cutting and pasting objects between widgets in a single image. Lastly, he talked about rendering, a subject specific to VisualAge.

Experience Reports

MOO Design Fest, Ivan Tomek, Acadia University, Canada
A MOO is a client-server text-based virtual environment emulating real world features such as places (rooms, building, etc.), actors (people, machines), and objects and combining them in a virtual universe. It allows its users to build new locations, instantiate objects, navigate in the virtual universe, communicate with others, and extend the environment by creating new object templates. MOOs have been shown to be useful in education and in computer supported collaborative work, and they are very popular as gaming environments. In a sense, virtual environments can be viewed as a generalisation of existing networked environments that provide new forms of scoping, communication, navigation, customisability and extendibility. Up to now, however, they have not proved as popular as their potential would suggest. This seminar aimed to suggest an initial specification and work out a suitable design.

I felt the talk would have had to be better presented to achieve its declared aim of significant audience input. The speaker used heavy text slides instead of class diagrams and other formal problem specs. Although seeking advice, the speaker did not seem to me quick to take it (but guess who offered the most advice :-). The speaker had chosen to use event-driven communication for all objects, not just avatars, which seemed a poor fit to the problem being modelled. He did not have a good description of what a MOO could do that web, email, netmeet, etc., cannot already do.

Smalltalk implementation of selected numerical methods, Didier Besset, Switzerland
Numerical data analysis seems largely ignored by most Smalltalkers. It turns out, however, that implementing numerical method in Smalltalk is very easy because of its much larger potential for mapping mathematical objects directly into computer objects. Since 1987, the author has been using Smalltalk to develop large applications using statistical analysis of data as a basis for decision support. This talk presented a general framework for computation by successive approximations, plus numerous examples of concrete implementation of this framework: zero finding, integration, finding the eigenvalues of a matrix, least-square fit. It also looked at vector and matrix operations and showed how to implement matrix LUP decomposition (to solve linear systems of equations and invert a matrix).

The slides contain many code and design examples. The speaker mentioned BT’s use of cluster analysis to detect a long-distance phone fraud. His key message was that OO can be used to express mathematical problems very elegantly; the more polymorphic the language, the more elegant the representation, which is why he uses Smalltalk.

Smalltalk is slower than C but the greatly increased ease of writing the analysis methods means he prefers it for all cases except when performance is absolutely key. However, he has analysed performance and noted a puzzling feature. For complex floating-point calculations (the worst case) Java is five times slower than C and Smalltalk is 8 - 10 times slower. However repeated profiles tell him that almost all the time difference is spent not in high-level Smalltalk but in primitives, e.g. Smalltalk’s exponential primitive is 10 times slower than C’s. There seems no reason for this. Have the suppliers simply not optimised these in the believe that Smalltalk is not used for heavy numerical calculation?

SmallScript 3D, Ernest Micklei, ELC, The Netherlands
Smallscript 3D contains a set of classes for representing all kinds of 3D objects, operations on these objects and some nice exporting facilities for rendering them using non-Smalltalk programs such as POVRay, Quake[1], VRML or a simple Direct3D viewer (or Java, for which he has written a builder; this was more complex than a rendering program’s builder). As Smalltalk is good at scripting but other tools are better at rendering, the idea is to let Smalltalk script and these other tools render.

The process is

- script: Smalltalk

1. a PC game whose viewer format is public domain

- objects: Geometric 3D hierarchy
- builder: VRML builder, POVRay builder, ...
- elements: mesh, ...
- writer: VRML writer, POVRay writer, ...
- file: various .extns corresponding to renderer type

The demo, which used VRML for rendering, showed some of these scripts and scenes (endless stair, Escher knot, animated mandala, VR building generation from construction diagrams, moving elements multiply-reflected in multiple angled mirrors), and also let the speaker spin his slides around in 3D (quick to spin, slower to realign :-). The speaker gave a detailed view on some interesting objects and concepts used:

- Smalltalk Interfaces
- Kind Dispatching: generalized visitor pattern uses name of receiver's kind (usually class or superclass) to notify requester. These double-dispatch-style methods therefore group all VRML writing on VRML object, all POVRay writing on POVRay object, etc. Like the standard visitor pattern, it assumes that the node structure, in this case the 3D geometric hierarchy, is fairly stable.
- Optimising Collections

As the system makes Smalltalk a VRML (or whatever) server, it is a way of presenting any smalltalk program on the web.

Software in the next millennium, Andy Berry, UK
After an interesting, but debatably relevant, 'historical context for software development' overview, the speaker arranged a brainstorming/snowballing session to identify what we want for the first few years of the new millennium:

- Software that writes itself
- Software you can paint or carve (like an artist)
- Computers do everything; we go surfing
- Never install software; it's just there.
- Want to configure multiple items automatically but also want to customise items => original product and customisations must be written in reflective language that automatic configurer can reason over. (No prizes for guessing who suggested this one :-)
- Software that educates its users

The session was fun but no particular conclusions were reached.

Research Work

Duploc, Matthias Rieger, University of Bern, Switzerland

Duploc is a tool to detect duplication in source code. Duploc compares source files line by line using simple string matching and displays the results in a two-dimensional matrix, where each dot stands for an exact match between two lines. Thanks to image reduction techniques, Duploc can display matrixes of arbitrary size, the only restriction being the memory capacity. Duploc also produces textual reports about the duplication found. Its basic of comparison is the line. Its unit of compared code is usually the file, i.e. it compares a file with itself and compares pairs of files to detect duplicated code but it can also compare classes, Envy applications. etc. It has a pre-parser that removes ‘noise’ (whitespace, no-effect code, etc.). It has limited ability to spot changed variable names in a repeated sequence (could be better). It takes two hours to write a parser for a typical new language.

Duploc was inspired by genetics work looking for duplicated DNA. It uses DotDot displays to spot duplicated code. The user can click on any interesting-looking sequence to see a vdiff-style code display. They can also tell the diagram to hide sequences they have inspected and decided are not interesting. The demo example was run on one million lines of C++ code (provided by an industrial partner). They found four implementations of object factory, 20 implementations of lists, an obvious cloned class, refactorable constructors, etc. They are working on using duploc output in a tool for semi-automatic refactoring.

Duploc is implemented fully in VisualWorks2.5/3.0 and is distributed free under the GNU license from http://www.iam.unibe.ch/~rieger/duploc.

CodeCrawler, Michele Lanza and Stephane Ducasse, University of Bern, Switzerland

CodeCrawler is a tool for reverse engineering of (large) object oriented systems. It combines graphs and metrics to achieve a graphical display of source code. The generated graphs are interactive and serve as a base for information extraction which is useful for the intuitive understanding and the detection of problems of the displayed system.

CodeCrawler is based on MOOSE, a language-independent metamodel developed at the university of Bern, Switzerland; currently, it can analyse Smalltalk, Java, C++ and Ada. It is part of the European FAMOOS project. The display of graphs is done using the functionalities of HotDraw, a graphical framework developed by John Brant. CodeCrawler is written in Smalltalk using the VisualWorks 3.0 environment.

A graph is a set of metrics, a layout and an ability to interact with the graphs. He showed a few graphs: system complexity tree (display classes, no. of methods and LOC for each; can get names, etc., by interacting with the nodes using the mouse), system hotspots, method efficiency correlation and coding impact histogram (compare method length distribution between classes). There are two mode of use.

- Problem detection: look for and spot and click on outliers; the big ones, the small ones, the strange ones (c.f. my metrics research in the 1980's).
- Program Understanding: browse the graphs and interact with them to understand the system.

CodeCrawler is a single tool, whereas most other reverse engineering tools are in fact an imperfectly integrated set of tools with limited interaction ability. CodeCrawler is not a product, not supported and the language for programming in it is a little idiosyncratic. It is also language-independent, which limits what it can do. It has been applied to a single large case study; a 1 million line commercial C++ system. He demoed it analysing its own code.

One questioner asked about increasing the number of axes of display and suggested using smell !!! (I've heard people say, 'This code stinks.' ... :-)

Product Presentations

VW 5i, Simon Townsend, ObjectShare, U.K.

[VisualWorks, plus its entire development team, has been bought by Cincom; ObjectShare retains their consultancy business and the right to use VW in that business. I had an interesting talk with Sherry Michael of Cincom who was at the conference. She was very positive about the change. I also feel fairly positive. Sherry intends to investigate using the classification browser in Cincom's development of VW.]

This talk started by handing out free cups, pens, stress-reliving penguins and VW for Red-hat Linux and NT discs, which made it instantly popular.

VW5i is the biggest revision since VW was built from ObjectWorks.

- VW 5i Personal: cheap entry product.
- VW 5i Enterprise: main product, released in March 1999.
- VW5i OpenTalk (see other talk): due for release this year.

Secondary products:

- VW ST plug-in: for applets, exactly like Java. Delivers small efficient applets over the net (e.g. 'hello world" takes 3 kilobytes). Unlike Classic Blend, which maps to Java, this add a Smalltalk plugin to your browser. It is better suited to intranets as security is not much currently (e.g. can do parcel that remove's client's windows installation). Uses DST CORBA or URL for communication. Works with Netscape and Internet Explorer on PC. Plug-in is small 1.7MB run-time image (can be downloaded from OS web).
- VisualWave: eCommerce application builder.

- Distributed Smalltalk 5i: adds IDL to DST generator.
- RoseLink 5i: adds StORE support.

Basic language changes:

- NameSpaces: no more class abbreviations (of course, have to think about NameSpaces when importing legacy code and designing). Also gives user-definable source code protecting and ability to make things private. Class variables and pool dictionaries have disappeared (but not, of course, class instance variables); they are replaced by statics - leaves on the NameSpace tree; this is the major change that tends to need a little work when porting (fortunately, we rarely use either).
- StORE: Smalltalk Open Repository Environment. Low-end alternative to ENVY. Workable team-tool. Requires Oracle personal or better as standard (but also works with GemStone, Sybase, Informix as uses internal database abstraction layer). Is inter-operable with ENVY; not intended as ENVY competitor.
- OpenTalk: new generation distributed computing product
- XML: all source, help, etc. is in XML, plus there is an XML parser

Performance improvement: faster on all platforms (e.g. non-local block returns) but also VW runs 40% faster on Linux than on the same NT platform. UI updates, including hierarchical lists. System browser is more NameSpace and parcel focused and so slightly different to use. XML viewer. Automatic install.

VW5i is integrated with Envy (this took much work). Refactoring Browser for 5i ‘is being worked on’. For 5i, you must read the documentation. It is a non-trivial change.

At the end of the talk, all were invited to email stownsen@objectshare.com with what they are doing with VW and what they need (but the Cincom change probably means we should now email someone else). Simon stated that several key commercial (financial) customers ‘use Smalltalk for competitive advantage so won’t let us, their supplier, talk about it.’

VisualWorks 5i and Opentalk, Roland Wagener, Georg Heeg Objektoriente Systeme, Germany
The speaker introduced himself as ‘the only VW ambassador in Europe’ (apparently there are lots of them, all in the States), The presentation consisted of an introduction and a demo of OpenTalk with VisualWorks 5i.

OpenTalk is the next generation of Smalltalk messaging, build into Visual Works 5i. It is an extensible, pluggable development architecture, initially supporting a pure Smalltalk to Smalltalk messaging model. It is user-expandible, and will support disparate protocols such as COM/DCOM and RMI. OpenTalk allows for fullbore remote development (browsing and debugging of code). OpenTalk is

- Smalltalk IDE: as before
- Distributed System Architecture: build a distributed application

- Distributed application framework
- Distributed Smalltalk IDE: build these applications in a distributed way; remote debuggers, browsers, etc.
    - — lets you analyse exceptions in a run-time image which can be very hard to reproduce
    - — lets you call support, say ‘I’ve got a bug’ and have them reply ‘O.K. I’ll connect to your image, start working with it, etc.’; c.f. netmeeting, but they have their image with their debugging tools to use.

OpenTalk is a flexible, extensible and reliable open system, follows CORBA, and is easy to understand, use and extend if you know Smalltalk. Currently, they have the Core, ST <-> ST (strictly, VW5i <-> VW5i) using TCP/IP and multi-cast IP, and IDE. Later, there will also be CORBA, DCOM and JavaBridge.

The idea is that, in a network of computers, each with its Smalltalk IDE and image, you can program all the images remotely. There is no IDL; your class library is also your IDL. Remote objects are created implicitly or explicitly as weak references to local objects. Their identity is preserved at the level of the image. Some objects (Immediates, Numbers, Strings and UninterpretedBytes) are passed by value, others by reference, classes by name. This last is because a class must be in both images; the speaker noted that this was not true for GemStone and so the statement needed changing. Further, the method passInstvarsAs: anArrayOfPassingStates sets the passing state (I asked, “Does this override if instVar holds class?”. The speaker was unsure). Various passing policies were described (e.g. identity instVars never change and so could be passed by value). Agents are stateful or stateless, resident or mobile.

The speaker thought that remote garbage collection works O.K. but was unable to speak from his own experience. There is a big difference between Distributed Smalltalk and OpenTalk. DST makes Smalltalk meet CORBA spec; OpenTalk creates a very clean class library, refactored from DST, for distribution. The speaker has used the alpha and beta-1 versions so could not comment on performance.

Demo showed IDE. Browsers, inspectors, etc., show you which image this class, instvar, etc. is in via user-settable colours, titles, etc. Original browser is split into Browser (local) and BrowserAgent (can be shared by many Browsers). Code compilation (accept, do it) is done in the IDE image for the target image (which might be a run-time image without compiler); this is the major difference between OpenTalk and standard (e.g. CORBA) distributed objects which always required the target to have a compiler. Speaker closed with simple ‘Have a nice day’ demo; text written in one image, echoed to other.

SilverMark Testing Mentor, Mark Foulkrod, SilverMark, U.S.A.
SilverMark was formed in 1996 by former IBM Visual Age employees. SilverMark's Test Mentor (email info@silvermark.com for slides, www.silvermark.com) is an automated testing solution developed specifically for VisualAge Smalltalk and VisualWorks (2.5, 3.0 now, 5i is scheduled, but all must

have ENVY) that automates test creation and execution at the GUI level, as well as testing at the model level[1]. After a worthy but obvious 'testing is good' discussion[2], they presented Test Mentor, focusing on how best to apply it to the development process, based on their experience deploying it on many major Smalltalk development projects. It can record and playback UI interactions and web connection transactions. The tool takes your test case to where the recording should start, records at your convenience, then continues.

The test editor has classes within the Envy library system:

- test suite (variant: file iteration runs suite for each row of data in a file)
- test case/scenario
- test step: various subtypes - script, scenario (use existing step), UI recording
- test verification: assertions to check state (GUI support for comparing UI display and internal state)
- test result: results are stored into ENVY.

(The tool can generate tests from UML Designer, whose future status is in limbo now IBM is working with Rational, and from Rose, but one must define use cases or interaction diagrams in these tools for this to happen.)

They ship 95% of their code (all but licence mechanism) so it's a very open and customisable system. Their limits are the limits of Smalltalk; if you can call out to it from Smalltalk you can include it in your test (plus some limited OLE support). Andy Berry stated that he downloaded it, wrote tests for a real smalltalk application within one hour and gained value from using it. They have an integration layer with Kent Beck's Testing Framework that automatically copies tests written in it to their tool.

A VisualAge for Java version is being considered (no timescale).

Gemstone products, Martial DorÈ, Gemstone, France
GemStone/S was called an OODBMS. Now they call it 'multi-user Smalltalk'. It is explicitly not presenting itself as an OODB any more; it is an application server linking to Smalltalk, Java, CORBA and C (there is some de-emphasis of their GemBuilder for C++ product). You can use GemBuilder for Java to connect GemStone Smalltalk to GemStone Java, or just to Java (you can drive any feature of GemStone/S from Java), so offering a route to preserving essential Smalltalk capabilities in applications where some groups demand a pure Java environment.

The talk was a clear standard presentation of how GemStone works; for details, see their articles via www.gemstone.com or smalltalk.gemstone.com. (Also see the quote at the end of Kent Beck's article above.)

1. claims to be the only such
2. including mention of the conventional cost of quality curve that Kent Beck argued against in his talk, though of course their tool is well suited to support his method.

ObjectStudio on the Middle Tier, Sherry Michael and Thomas Haaks, Cincom
ObjectStudio provides an environment for building complex, n-tier systems (talk focused on standard three-tiers: presentation, business logic, database).

The presentation discussed the ObjectStudio Transaction Framework, synergies with their Smalltalk Request Broker, Internet Client connectivity, and the initiatives currently under way to re-architect ObjectStudio as an Application Server on both the UNIX and NT platforms. Various improvements were presented; ObjectStudio uses native OS threads for its Smalltalk threads, remote debugging and console support. They concluded with a demo of their initial release, now in the final stages of testing, running on Windows NT and enabled by BEA Tuxedo.

Unfortunately, given the interesting look of the tool and Cincom's importance to the future of Smalltalk, I had to listen to this talk with half-an-ear while doing some urgent work of my own.

Additional Talks

TOPLink, John Tobin, The Object People
TOPLink is a tool that lets you use relational databases (e.g. Oracle) as object databases. As a strong believer in the impedance mismatch problem, and as someone who agrees with Kent Beck's remark quoted above about the importance of schema evolution (and thus of GemStone-like OODBs), I think one should avoid using relational databases wherever possible. If, however, one is caught in one of the many legacy or business contexts where one has to, then (paraphrasing John's remarks to suit my own prejudices :-) it appears that TOPLink might be a good way of hiding this shocking fact as far as possible.

The business case for adequate reflection, Niall Ross, Nortel Networks
Companies (e.g. telecomms companies) whose product and service portfolio is rapidly expanding and diversifying find themselves compelled to build systems that mix meta-level and class-level programs. Since, in a rapidly changing world, you can never solve the meta-problem for all of your problem domain, your solution must either be wholly at the class level (and so large and slow to change to match your rapidly changing business environment) or a mixture of generic code handling the problem at the meta-level and specific code handling those classes you can't yet wholly deal with generically. Continually:

- new business cases with short windows of opportunity will introduce new specific code
- existing cases will be recognised as examples of a more general meta-pattern and converted to data handled by the refactored meta-program.

This observation is just the application to meta-programming of the general rule that refactoring an Object-Oriented system tends to move state downwards and behaviour upwards[1].

1. See e.g. Ralph Johnson's article on the Refactoring Browser in the Smalltalk Report

If the above is accepted then it follows that a real system using meta-programming will have an ever-moving boundary between its class-level and its meta-level (to which the class-level is data). Using an example based on the DeepCopy/DeepEquality meta-algorithm, I showed that the meta-program would require the ability to write type expressions in its method declarations (this ability is called type quantification or F-bound polymorphism in the jargon of the subject) if it were to interact seamlessly with the specific parts of the system. If it did not have this ability, every shift of the boundary would not be a refactoring; it would break the system and require significant rewriting.

This is not a problem for Smalltalk, which has implicit type quantification. It would also not be a problem for a language with the ability to provide explicit type expressions in its method signatures (indeed the added formality might be a bonus - or, there again, it might not). But for insufficiently reflective languages such as Java, the necessity of e.g. the clone method returning a type value (Object) instead of a type expression (this.getClass()) blocks integration of the class and metaclass parts of a mixed system using DeepCopy (which any such system will)[1]. More complex meta-algorithms will hit the same obstacle.

It was noted that my case was a particular example of the general difficulties of writing generic frameworks in Java. One listener confirmed my type theory from his own analysis. Another had heard of a research project investigating F-bound polymorphism for Java but had no reference (presumably not Gilad Bracha’s GenericJava as that only offers parametrised types; possibly Rémi Douence and Mario Südholt’s MetaJ). Annicke told me of a recent conference where, in a talk on Real-Time Java, the speaker (without using the term) confirmed that blocks were essential, describing a framework for them in Java and indicating the fundamental VM features that would have to be added to supply it.

(Someone else mentioned the stack-frame problems for blocks in Java. A literal block can be created in a method and passed in a call to an enclosed method. However, if it is defined in one method, stored in a variable and later called in another method, it may have vanished from the stack by the time it is referenced.)

Smalltalk Advocacy

There is concern over the contrast between Java’s visibility and hype and its inferior (to Smalltalk) capabilities in many areas.

Annicke led a discussion about what should be added to a suitable Smalltalk web site; job offers, who trains in Smalltalk, project page list (the proposal was to use Ralph Johnson’s page but make the project’s country location visible early in template).

1. Will LaLonde and John Pugh outlined DeepCopy and DeepEquality solutions in JOOP (September 1994). Paul Baumann presented a detailed DeepCopy solution in The Smalltalk Report (March/April 1998). Paul observed that his work showed what Java could not do, but his article did not present a compelling business need for it.

Many people stressed the need to publicise the well-verified 3:1 effort and time advantage of Smalltalk over Java to create any given level of functionality in almost all domains (verified by formal metrics studies and experiment contests as well as personal testimony from experts). I argued for the need to counter the argument that Smalltalk might be three times quicker than Java but there were nine times as many Java programmers as Smalltalk programmers so any given level of IDE, VM, etc., capability would arrive three times faster for Java than for Smalltalk. There are two kinds of counter-argument to this:

- examples of what had to be done for good business reasons and yet couldn't be done in Java, as opposed to what was merely quicker to do in Smalltalk (e.g. see my talk)
- examples where the Smalltalk capability did arrive first and better (e.g. PocketST on PalmTop as opposed to Java from Sun on PalmTop).

In arguing a business case to a manager, it was not only not advantageous for a techie to stress that they'd rather write in Smalltalk. It was positively (and, to a limited degree, understandably) grounds for suspicion that technical preference, not business need, was the motive for Smalltalk advocacy. As someone who chose to write the initial version (10,000 lines) of a system in Java and was forced to switch back to Smalltalk (for the reasons I describe above), I was free from (just) suspicion on this point. It was important for Smalltalk advocates to realise that, "I could write this three times faster in Smalltalk", could sound like, "I'd rather write this in Smalltalk", to an inattentive manager. The argument is good but a little care to avoid this misunderstanding when presenting it would help acceptance.

Conclusions

An enjoyable conference with a higher ratio of good to bad talks than many I've attended. I look forward to next year (which might be in the U.K.).

- Smalltalk is still very much alive. There are many domains of use; the most common seems to be financial applications managing complex fast-changing products, which is of interest to those developing telecomms service management applications. (Although some of those attending had telecomms clients, the mention of BT in Didier's data analysis talk was the only interesting telecomms reference.)
- (From discussions more than from the one talk on it) Meta-programming is the way to manage complex, fast-changing products (and Smalltalk is especially suited to meta-programming).
- Extreme programming is a radical improvement on existing concepts of software engineering (and Smalltalk is especially suited to it).
- A union of the Classification and Refactoring Browsers would be well worth looking at, especially if the latter's classification framework were pluggable.
- From my discussions with Cincom people at the conference and my own observations, the sale of VW is good news. Cincom seem committed and seem to know what they want to do.

Other discussions

I spoke to a couple of possible recruitees. My talk offers a context for other attendees should they see our project mentioned on job-offer Newsgroups.

I had several interesting meta-data discussions with Greg Hutchinson, Sherry Michael and other attendees.