CS11 and ESUG 14, Prague, September 4th - 8th, 2006

Arriving in Prague on an early flight (5 am get-up :-/), I got on the airport shuttle bus, only to discover that the currency of the Czech Republic is crowns, not euros. I approve their individuality and independence on this issue (which may not last much longer, I fear). And since a courteous fellow passenger offered me a fair rate to exchange a euro for my fare crowns, I was not tempted to moderate my admiration by any considerations of mere convenience for foreigners like myself.

The hotel had large labels saying accommodation, restaurant and other facilities but none giving its name. Fortunately, the heaviness of my luggage persuaded me to try it anyway since it was in the right place, otherwise I might have walked far before turning around.

Style

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

Author's Disclaimer and Acknowledgements

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

There was much activity in the Camp Smalltalk room, only some of which I learnt enough about to summarise (with possible errors) below. Inevitably, my notes treat my project in much more detail than others.

Likewise, the talk descriptions were typed while I was trying to keep up with and understand what the speakers were saying, so may contain errors of fact or clarity. I apologise for any inaccuracies, and to any participants whose names or affiliations I failed to note down. If anyone spots errors or omissions, email me and corrections may be made.

My thanks to:

- Katerina Barone-Adesi, Adriaan Van Os and Bernard Pieber for working with me in the Custom Refactoring Browser project
- Katerina (again) for looking over my shoulder while I was typing notes for this report and catching several errors
- the speakers whose work gave me something to report, the conference sponsors and the ESUG organisers
- the local organisers from Fractal and the student volunteers for ensuring that everything was present and working when required, and for loaning me a spare keyboard when mine had problems.

Summary of Projects and Talks

I begin with Smalltalk Admin talks and my Camp Smalltalk 11 project summary, followed by the talks, sorted into various categories:

- Coding, Testing and Analysing Smalltalk
- Application Frameworks, Experience Reports and Packaging
- Web Frameworks and Applications
- Languages, Dialects and VMs

followed by the Academic Track. I close with Other Discussions and my Conclusions. Talk slides are reachable from http://www.esug.org.

Smalltalk Administration

ESUG Activities, Stephane Ducasse

Stephane introduced himself ("I am Stephane Ducasse") and was met with with great applause ("You saw the same talk" :-). He thanked the sponsors and the local organisers (Jarmila, Petr, Martin; a bouquet was presented to Jarmila). This year, the conference has 12-15 first-timers in 90+ attendees (100 last year). The conference again has student volunteers helping to run it (an innovation of a few years standing which has been very successful). The research track papers will be published in an LNCS special edition.

ESUG is collaborating with netstyle to offer free Seaside hosting. There are now 200 accounts. They have a server to share lectures (structure being developed). They offer 100 euros for Smalltalk articles published. If you know French and want to translate articles to German or English, talk to them. There is an ESUG DVD. If you want to put something on the DVD, talk to ESUG for next year.

The SummerTalk programme gives students 1000 euros so that instead of taking a summer job they can develop something in Smalltalk. Items developed this way include:

- new Squeak compiler
- fast loading package
- RDF framework for Squeak

There will be a second SummerTalk. If you have project ideas, tell them.

They want to extend the Seaside hosting, offer videos and lecture server on how to use Smalltalk, launch an intern student programme that will send students to companies (also helping with visas, work permits etc.). If you need support to present Smalltalk at conferences, they will be glad to help.

Sponsors for all this are of course welcome. If your company would like to sponsor, talk to them. Sponsorship can be at various scales; for example, Christian Haider's company smalltalkedVisuals sponsored one of the awards this year.

Next year ESUG will be at Lugano. If you want to be the local organiser for 2008, tell them.

STIC Update, Bob Nemec, netstyle.ch (www.netstyle.ch)
www.Smalltalk-Central.com has been created by Mark Roberts with a six person beta-test team. It aims to be the central resource site for Smalltalk current technical information.

www.SmalltalkSolutions.com will be managed by Thomas Stalzer (of OBJECT dynamics Software GMBH). It will hold presentation archives, online submissions, surveys, feedback. It will be powered by VA.

STIC has a new treasurer, Bob Cherniak. Bob will manage www.STIC.org where you can pay your annual membership (and please do :-).

There have been many Smalltalk logos over the years. Bob at first hankered after the balloon but the knights of the square bracket crowd have a logo that is easy to customise, easy to fit into places and so on. So [|] is it.

In 2007, Smalltalk Solutions will be part of the IT 360 conference. It is a decision by STIC to aim to be part of large conferences; it is better for advocacy. The IT & Linux world conference was a first try and we are learning from that how to do it. For example (point raised by Reinout) we will have a keynote speaker who mentions Smalltalk. However it did show the word Smalltalk to the 100,000 people it was marketed to, quite a rise on the 250 organisations of the previous year's Smalltalk Solutions.

How to do an ESUG presentation, Stephane Ducasse
Make the point of your talk clear: its subject and its message. Make just one point per slide. Use pictures.

Use large fonts. If you can't read your slides when the laptop is on the floor and you are standing then your font is too small.

Be active on stage; move around. Change voice tone (for example, Joe Armstrong varied his style a lot in the Erlang talk). Have eye contact with everyone in the room, not just three people in one corner.

We are all foreigners. Speak slowly and do not assume your listeners have a perfect understanding of idiomatic English.

The last slide is the most read; don't just have 'Questions' and blank.

Squeak News, Marcus Denker, Stephane Ducasse
Marcus introduced the Squeak Foundation. It has 7 board members: Tim, Marcus, Cees, Bert, Yoshiki, Craig and one still to be elected. It promotes squeak and helps communication between Squeak teams; visit it at http://www.squeak.org/. There are lots of teams; join one and have fun doing Squeak. If you want to do things and have time to do things (doing is harder than joining), just ask.

Squeak now has bug tracking. There are 230 open bug reports for you to work on. You can get an account on https://bugs.impara.de and fix bugs, test that a posted fix works, or report new ones. 3.9 has *lots* of fixes.

Squeak 1.1 has been released under APSL2.0. APSL2.0 is open-source certified. You can now write products on Squeak without alarming your lawyers as much (but there is still work ongoing to transit all of Squeak to a clear licence state).

Squeak 3.9 has pragmas (as in VW). The SqueakLand 3.8 and Smallland forked versions have been merged back into 3.9. The SqueakLand 3.8 fork is mostly to do with Unicode support bugs discovered by people in Japan and China who used Etoys. The Smallland was work done in Spain. (Alas, Smallland does not yet use 3.9.)

Squeak has a new look which is faster and more professional in appearance. Anthony Lander has built a new compiler based on the Refactoring Browser. It uses a snag-based parser and enables closure. It has a structure you can understand and they find students can grasp it and experiment with it. Inlining took 2 hours to understand in the old compiler, two minutes in this one.

A ToolBuilder UI abstraction lets you build tools that work in Morphic and Tweak and all the GUI frameworks. It is not yet used in the tools but the test runner and Monticello are (moving to?) using it.

A service architecture has been added so you can add tools to browsers more easily. Romain Robbes is adding the Refactoring Browser features as services to all tools.

Squeak 3.9 has (Roel's?) change event notification system.

3.9 tools: a closure compiler, Monticello and SqueakSource. Tweak, Croquet, Pier, SqueakSource and Squeak itself are managed in Monticello. The OmniBrowser (see Colin Putney's Squeak Tools talk at Smalltalk Solutions 2004) is a framework for building new browsers. Use it for a day and you will never use the old browser again.

Q. ToolBuilder is OmniBrowser? No, ToolBuilder is a cross-GUI framework abstraction, OmniBrowser a tool-building abstraction (yes, the names invite confusion).

And there are a lot of other tools: Traits (he forgot to put it on the slide), SqueakMap, Shout (coloured typing tool) and eCompletion (uses RoelTyper, so is more intelligent than you might expect), the RB, the new SUnit browser (much finer-grained test running than the old raw browser) and Christo (a coverage browser).

Refactoring interaction with Traits is still to do (anyone interested?).

Future work: they really need a test server. They are shipping with 20 broken tests because noone runs tests that take 50 minutes. (Actually, I do; you can always start the slow test suite before going out for a walk, or as the film starts, or last thing at night, and so need never be twenty-four hours from having run everything.)

Stephane then reviewed some Squeak projects.

Tweak is a new UI framework which is being used in Sophie (see Michael's talk) and Croquet; visit www.opencroquet.org (version 1.0 is released). Other projects are Spoon, Seaside, St-Exubery (JIT in Squeak), Chronos and Pier. Use www.dabbledb.com to show people a nice seaside application (that has passed $2 million). For ESUG-sponsored hosting, go to http://seachart.seasidehosting.st.

Spoon is a micro-VM for squeak. www.netjam.org/spoon

Squeak 3.10 is currently high in talk and low in activity. One vision is more cleaning: split Etoys out of the base, clean Morphs, integrate the Sophie packages, etc., and shrink the image: Pavel's 3Mb image sets a target.

Stephane stressed that if people take time to do good things, these should be integrated. Integration takes a lot of time and a lot of energy but not doing so demotivates. Traits can be flattened (i.e. removed) so we should be bold in experimenting with using Traits.

Q. What other visions are there? Still talking about them.

Squeak BoF
This was held in the very noisy student's club/pub in the basement, surrounded by bemused students. Squeak 3.9 gamma is out and 3.9 is virtually done. Plopp is Squeak's first commercial product that you can buy on a CD in a colourful box shrink-wrapped. (Of course, there is plenty of commercial Seaside hosting using Squeak and some in-house Squeak projects.) Monticello 2 code has been released as 'use at own risk' for the moment; development continues.

Alexandre and Noury paired on RealTalk, looking at sensor setting. Karsten Kusche talked to Marcus Denker (ByteSurgeon) about an oddity he sees when decompiling literals. He has programmed in Smalltalk and Objective C a means of fudging the stack momentarily so the debugger opens on the failed assertion line, not several methods further down. Lucas is working on a new testing framework based on assertions. If I understood correctly, this too is looking at opening the debugger in the most user-friendly place for failed assertions.

As it was too noisy for a regular BoF, everyone else had a beer and talked.

Awards Ceremony, Noury Bouraqadi, Suzanne Fortman
Noury announced the awards; Suzanne presented the prizes. There were 61 votes and the results were very close, ranging from 22% for the first to 8.5 for seventh candidate. They tried three different algorithms to combine the first, second and third votes but they all gave the same rankings:

- 1st prize Plopp: see talk (Sophie was also an entry)
- 2nd prize Mondrian: see talk
- 3rd prize SqSquare: EToys over the web

Other entries included SqueakBot, a statistics tool, DakarTesting (see talk), UbiquiTalk (see talk) There were 6 entries at Kothen in 2004, 9 at Brussels in 2005, 11 this year. Of these, 6 were written in Squeak and 5 in VW, and 6 were from academia, 5 were commercial. (Any code is eligible, whether free or commercial, provided it is separable from its background system.)

Camp Smalltalk 11

Camp Smalltalk 11 ran for Saturday and Sunday before the conference, and during the conference breaks, afternoons and some evenings of the five conference days.

The Custom Refactorings and Rewrite Editor Usability Project

I spent most of my CS time on this.

Katerina Barone-Adesi and I paired on the Cascade refactoring: rewrite selected statements to a cascade if all have the same receiver, warning if the receiver could have side effects and extracting last-statement return and/or assignment to the whole cascade. If the whole method is selected, rewrite all viable top-level subsequences to cascades. This was completed and will be in the next project release.

We noted that the menu text ‘Create Sibling...’ leaves a user guessing whether to select the parent class that is to become a child, or the child class whose parent is to become its sibling. The former is the right answer but people avoid the refactoring because they are unsure. ‘Convert Parent to Sibling...’ would unbalance the menu width, and might suggest ‘convert parent of selected class’ instead of ‘convert selected class’. We need to think of a name that will make people comfortable with this refactoring.

We also noted that the MenuAugmentationGenerator in VW processes the first pragma in a method. (Marcus Denker later told me that Lucas Renglii had already noticed this implementing menu pragmas in Squeak, for ease of code compatibility with VW.) This looked like an error, induced because you must check whether attributeMessages == nil so it is natural to fall into returning nil or an item from the method instead of a collection (possibly empty) of items. We rewrote Menu>>testForMenuItems:with:forMenu: and Menu>>augmentFrom:to:menuName:for: to fix this; I will offer the code to VW dev.

Meanwhile, Adriaan and Bernard upgraded the VA release from VAST 6 to VASmalltalk 7, remedying bugs and dependency issues.

We also looked at ‘extract with holes’: the ability to select text for ‘extract to self’ or ‘extract to component’ and then select within it those parts that were not to be extracted. In the model layer, this is ExtractToMethod composed with a number of ExtractToParameters on the extracted method. The cascade work on multiple refactorings of the same method had shown the issues to watch. In the UI, adding a pane to the extract dialog window showing the text being extracted, and a button to prompt ‘unextraction’ of selected text to the parameter list, would let the user choose the ‘holes’. Alternatively, we could switch the user into a select-only-mode code tool to complete the refactoring. Work will continue after the CS.

Lastly, Kat and I paired with Adrian Kuhn on how to write custom refactorings for Moose metadata methods. We wrote the first one, after which he had got the idea of how it was done and was ready to show the rest of the project how to do more.

Other Projects

Much more was going on, both during the weekend and in breaks in the conference. Work was done on Moose and on Seaside. Unfortunately this year the camp smalltalk room had a hard closing time. (We had tried to sort this during preparation for the conference but could not. I think the problem was an attitude legacy from the communist days when having a resource unused because the pre-planned procedure did not fit the requirement was quite normal :-). This time was late enough during the weekend but too early during the week. Some groups carried on in the main hall, whose benches had power points, but more would have been done if the room could have been left open for longer.

Coding, Testing and Analysing Smalltalk

Coding Dojo, Joseph Pelrine, MetaProg

The Coding Dojo is an idea for learning Smalltalk by doing it. Joseph will program with Jan and hopefully reach the end of his time just when the talk is getting interesting

Why a Dojo? Well great musicians become great by practising. Having talent and knowing the theory is good but nothing replaces practice. How do you get to be great in sport. Practice on the dojo is essential. But in software, we take programmers trained in the theory and start them on projects right away. Would you be operated on in hospital by doctors who had never practised? This habit is part of the reason software is so bad.

The Dojo idea was invented by Laurent Bossavit with Dave Thomas. Martial arts experts have katas: set formal moves you do over and over. Rantouris are free-sparring exercises.

Software developers do not practise, so they learn on the job and that is where they make their mistakes. Practice will pass on knowledge; this is especially valuable for Smalltalk. You will know a good practice session because you will leave knowing more than before. Find a place and time in which you will not be interrupted and focus on the essentials of the kata exercise. Remember to look for feedback on every major decision.

Joseph is used to speaking in front of people but is still a little uncomfortable coding in front of people. That is why Jan is here. (Jan did not know about this half an hour ago. :-)

Kata no 9 from Dave Thomas: implement a cash register to total prices for buying apples, bananas, cherries and dates. The pricing rules change often.

```
co := CheckOut new: pricingRules.
co scan: item.
...
price := co total.
```

Jan suggested that pricing rules can be a dictionary and the register just encapsulates a counter.

```
...
price: itemString
  | co total |
... itemString inject: 0 into: [...
```

How many people know how to use inject: into: without thinking about it (surprisingly few hands). Now lets use a Bag; how many have ever used that? (This also got few hands, but though one of them was again mine I was less surprised that few people had used a Bag.)

Now we moved from slides to the browser and promptly Jan learned not to leave the VW default create initializer options selected when creating a class in the wizard. He removed them and carried on; yells from audience 'you're on the class side'. The first test was written.

```
| co |
co := CheckOut new: pricingRules.
self assert: 0 = (co price: ‘’).
self assert: 50 = (co price: ‘A’).
self assert: 80 = (co price: ‘AB’).
```

Brief XP recapitulation: if you do not understand the task enough to write a test, you do not know enough to code it. The best time to discover problems is before you start, the worst is when you end. Having omitted `new` he saw 'does not support variable size allocation' - certainly could teach a newbie Smalltalker something while explaining what that message meant; we forget our classes are divided into fundamental types.

Jan then created his class and wanted to run his test. Second learning experience; must rename test method to test... (This is JP's fault; he told Kent to find which methods were tests by starting them all with 'test'.)

Very XP-style development got us from `self new` up to a CheckOut class that did something. Programming by intent: if you can express in a few words what you want to do but cannot yet program it in detail, write meaningful messages and the code in the debugger at each point.

At this point a truly weird effect occurred: `price:` was not understood in the test, but was when run repeated in debug. What do you do when you see this: *don't panic*. Close the debugger, garbage collect, restart, use IRC, ... Jan had saved the method while the debugger was open and the method cache had not flushed; needed to force garbage collection to make it flush.

Discussion: at what level you should test. Joseph chose to test at the interface of the class. He could have tested individual method functions. He could have tested the functionality the user wanted at the highest possible level. (The last is my preferred approach.)

Very XP-style coding got the next line of the test passing with hardcoded values but the line after that forced generality, as it had to handle prices of items. (I found myself thinking that this extremely XP-style of coding

might be slower that one that coalesced some small steps into one, but it might also ensure fully general tests.)

How many of you are now wanting to get up here and do this yourself? That is when we do rantouri exercises. I was the first to volunteer. Jan was now the 'watching' pair while I typed. We added an instance variable, justified ourselves, deleted it due to audience comments, then re-added it.

```
price: aString
  ^ pricing at: aString
```

Jan stepped down and Reinout came forwarded to code with me. Pricing needs to be improved; it was given some pricing rules. There was discussion of the input format: `item.priceitem2.price` was displaced by the more smalltalk-friendly `item->price`.

Use of a String or an OrderedCollection of associations quickly switched back to the dictionary discussed by Jan. I coded a 'one-char string to value', corrected to 'char to value' by Reinout, since the pricer will get a string and iterate through its chars via standard collection processing.

I kept wanting to talk (what a surprise :-) about interesting aspects of this way of coding:

- generality versus speed as a function of the level at which you aim your XP tests
- the delay of using an unfamiliar keyboard versus screen-sharing your machine to the adjacent machine of your pair-programmer (using NetMeeting, NetOp, VNC, etc.).

but such interesting digressions are not really on when the time limit for each pair is just two minutes. The cry went up from the audience: 'Start coding!!!' :-) My time expired and Bob Nemec came forward.

Reinout and Bob Nemec: Bob likes to add intermediate variables early when debugging XP-style; he finds it speeds understanding.

Bob and Alexandre: Bob finished his work and resumed in the debugger; the test passed (applause). "Writing good unit tests is one of the most challenging parts of writing software. (What we just wrote were not good unit tests.)"

Then Joseph made the problem more complex. What about quantity-related prices. Bananas are two for 45. Apples are 3 for 130. So our pricingRules cannot be a Dictionary. This is an excellent example of Niall's rule (Hey, I'm being quoted !!) that static typing is the ultimate example of premature optimisation. We must change the type significantly but dynamic typing and polymorphism let us avoid changing much else.

Google for "PragDave kata" for more info.

Q(Bob) Can kata's be recorded, made available as exercises, as patterns? Good idea. Some of the katas are interesting in other languages but trivial

in Smalltalk so documenting that is also advocacy.

Scrum and XP, Rowan Bunning, Wizard Information Services
A rugby scrum is a good metaphor for the software practice of coming together, interacting, then breaking away and being very flexible till the next scrum. Since Joseph Pelrine has already given excellent SCRUM talks at past ESUGs (see e.g. his talk at ESUG 2005), my notes below concentrate on Wizard's experience.

Rowan gave a three part talk:

- explain Scrum usage at Wizard.
- 15 minute sprint example.
- How to sell Scrum and challenges in using it.

Wizard info services is one of 5 Wizard businesses. Four are run by someone who is a champion of scrum. Over the last few years they have changed from bespoke development to product development.

Scott Amber spoke on Scrum in his StS2003 talk 'Are you agile or are you fragile?' His short mention of Scrum caught Rowan's attention. They tried a few practices in their EzyXML project. Then Joseph visited Australia for the first course in that hemisphere. Wizard management were very positive about it. The first Scrum project was CommunitiesOnline. They also used it in non-development activities, in the WizDom2 projects, then appointed a Scrum mentor and now it is their normal practice.

Two week iterations and avoidance of the architect/developer disconnect are a great help. Smalltalk is good for rapid development so it works well with Scrum. Business decision makers feel empowered; they can prioritise the backlog and they can pull the plug on projects. Developers feel empowered as well; management interaction is structured and lightweight, so they can get into new areas without needing reams of documents and without interruptions. Scrum is used by many other organisations.

Scrum is for project management. XP gives them their technical development practices. Upfront planning processes are about making a plan and then trying to make reality fit the plan. Scrum is continually inspecting and adapting to reality. It slices through everything you do. You do top to bottom slices (spike through some DB, middleware and UI).

Scrum allocates roles

- product owners prioritise.
- scrum masters remove impediments and promote scrum values. They chair meetings
- a scrum team is 5-10 people

The daily Scrum meeting is a quick standup where everyone says what they did yesterday and what they will do today. Scrum sprints are two weeks at Wizard, can be 30 days in Scrum theory. They are careful to define what is done and not show the customer things that are half-complete at the end of

a sprint. The spend 2-4 hours in review and planning meetings at the end of each sprint. Retrospectives (looking back to see what you have learned) are integrated into these meetings. Task allocation is often left open in these meetings.

Testing is done on the development server. Bugs are classified and are added to the backlog if not too serious for the sprint goal.

He showed the Scrum tool they use and talked through its panes (see slide). CommunitiesOnline was coded in VA using as many Seaside ideas as were available. Suppose the backlog requirement was to reduce repetitive moderation tasks in CommunitiesOnline. Suppose a way was to let people select feature groups randomly. He opened the testing tool, found the relevant tests and added a new one (just a selector and comment).

Scrum always has something near to delivery so if the customer's budget dries up they are not left with nothing.

Challenges. The domain experts must be available. You must think in vertical slices so you can show core functionality to your customers. You must have comprehensive test coverage. Some customers demand traceability from individual methods right up to what they asked for.

Q. Internal use has gone very well for you. External? Yes, they would be very willing to do so and just need a customer likewise.

Testing for Real, Niall Ross, eXtremeMetaProgrammers
(Kat wrote up my talk as I gave it; these are mostly her notes.)

Niall's client for this work was a Swiss company who provide back-end management of insurance. Lifeware don't sell insurance, their clients do. They provide the backend and system support, servers running images, etc. The clients gain the technical advantages of a mature generic system that can be quickly evolved to their needs, and the financial advantages of pricing that is risk-free, since Lifeware charge by the number of policies their customers sell, and predictable, since they offer a complete solution so their clients' IT management has a very precise idea of costs. Lifeware use eXtreme Programming, especially test-driven development.

Niall first preached XP to his team in 1999. They asked if it would scale; he said it would scale at least as well as any other method in use at that time. Test-driven development is wonderful; Joseph explained why in his talk. "So everyone using XP lived happily ever after ... well, up to a point."

XP tests have certain characteristics. They tend to cluster around normal and initial states. They tend to describe the *intended* use of a system, testing for what the system should do, and for some obvious things it should not do. Real users don't use the system as they 'should', but as they 'have to' or are compelled to; they use the system in entirely unanticipated ways.

The obvious way to deal with this is to write better tests. However a test

must be something you can actually write. There is a combinatorial explosion of possibilities and configurations: insurance contracts have complex lifecycles. There is also a vast amount of unexpected usage (Niall discussed an example requiring multiple views of time: error correction may need to be done and audited at one time but calculated as if done when it should have been). Statistical analysis of the many real contracts in their databases and the many contracts constructed by their XP tests shows that the two distributions are significantly different.

What is a test? Niall's slide showed definitions taken from the literature but he observed that no Smalltalker could believe in real tests preserving this nice structure; real tests quickly evolve into a mix of operations, assertions, etc. People who use XP to maintain large systems in real use gradually evolve a domain-specific language for expressing their tests. This speeds writing the tests, and it also gives you tests that are readable. Lifeware developers consider this readability very important.

- At Smalltalk solutions 2002, Niall spoke about generating XP tests from domain objects for the fixed legacy part of the system used by a UK insurance company who had discovered XP only after their system was already large. In that case, the main need was to ensure that new developments did not break existing requirements.
- Lifeware had different needs. They already had XP tests for their whole domain. They wanted to use objects in their databases to generate tests written in the same readable Smalltalk style as their existing ones.

The problem was not one of lacking tests for any individual operation: the client's strong XP focus meant they always had a test to generate any simple contract data. The problem was combining these simple fragments to create and exercise a contract matching a real contract from the database.

Human domain-expert knowledge is needed in the first stage to annotate these simple fragments with expressions that extract the values they set from the objects they create. This is usually easy for the domain expert (and the side-effect of a code review can have incremental benefits). All else is done automatically by the test generator:

- Helpfully, audit requirements mean a contract's history is already available as a sequence of events. The generator gets the contract you have chosen from the database and extracts its event sequence.
- It maps each event to the annotated test method that creates events of that type. Mostly this is class-based - a given event class has a given method - but it may depend on the specific data values of the event
- It then partially evaluates the code, treating the event object as 'self':
  - Conditionals on the path of evaluation are evaluated and recursively inlined on the result.
  - Loops are expanded to the number of calls the event has for that loop. (And all possible Extract Code to Existing Method can then be run to make the code in the loop tests more readable - a sort of anti-inlining; this is an option currently).

  - — Parameters are evaluated and the result written as literals, or as `storeOn:` expressions where the result is not a literal

- The result is then inlined into an overall ‘event’ method for the test being generated.

Niall walked through these stages for a code example (see his slides) and then demoed (the image has 13k application classes and nearly 12k tests, of which a significant percentage are generated).

Principle uses:

- generate a test from the existing data of a real contract and add it to the XP suite, thus supplementing the hand-written tests with tests whose lifecycles are more complex and better reflect the vagaries of real use, but are as easy to debug as hand-written tests
- to write XP tests for new features, generate a test from a suitable existing contract and then edit it by hand, thus save time typing and get a more realistic initial set-up

The work required Niall to extend the Refactoring Browser into a partial evaluator of code. He also needed to clean-up InlineRefactoring and InlineToComponentRefactoring, to make them more programmable, and to add some additional refactorings: InlineEvaluableIteratorRefactoring and InlineEvaluableTemporaryRefactoring. The readability refactoring ExtractCodeToExistingMethod may appear in the custom refactoring project as it is generally useful.

Future work: although he did this work using the Refactoring Browser (because he knew it best: “When what you have is a hammer, everything looks like a nail.”), he would like to make parse tree fully evaluable - and more easily evaluable - by using an abstract Smalltalk grammar. The Zork Analysis tool already provides an evaluable grammar for VisualWorks and SmallTyper offers a similar one for VisualAge.)

Niall closed by quoting a remark Kent Beck made after a remote pair-programming visit to the client a few months earlier: “Extracting realistic tests before beginning implementation is becoming as addictive as writing tests by hand was when the code and data were simpler.”

Q(Bernard) Does it mean that the data I choose needs to have no errors? The data is a real contract; it ought not to have any errors - you should be getting real objects out of your database that you trust. If you find errors, it’s probably good to become aware of it. But generally, you expect the contracts to be well-formed.

Q. How much work to generate one test? This is a press-a-button procedure. (The generator is not optimised for speed and may be a minute or more generating all the methods for a complex, many year contract.)

Q. Where do the tests live? In the test suite along with all the other tests that are run on every integration of code. Generated tests that are not then hand-edited live in special ‘Generated ...’ categories to show they can be

regenerated and overwritten. Tests for which the generator was just used as a wizard to kick-start what was then hand-edited are moved to standard test categories to show they should not be overwritten.

Q. Does generation give you a combinatorial explosion? Statistical checks indicate that currently the generator can generate tests for 70% of the contracts (to do more needs more annotations to be written), but far fewer are generated. People select contracts for generation to increase coverage, or because they (or end customers) feel they reflect interesting unexpected behaviour, or just as a way to speed up writing the test.

DakarTesting, Karsten Kusche and Damien Cassou, Georg Heeg
A new code tool for the VW7.4 Refactoring Browser lets you view application code in the left window and test code in the right window. A middle pane lists all the tests relevant to the method or class you are viewing (deduced from names and/or chosen by user and captured in class-side method of generated class). Dynamic protocols group the passing, failing and erroring tests (also those referenced in the class-side method but unloaded). The tests are grouped in TestSuites.

Q(Stephane) can this be integrated with a coverage tool, so that you could navigate between tests and the methods they call? Not considered at the moment - they were focused on test-driven development going from test to application code - but an idea they could consider.

Q(Joseph) could this be reimplemented without pragmas so that other dialects could use it? Discussion: something very similar could be ported.

The dynamic protocols were ported from Squeak and are useful. (Speaker asked anyone who knows a lot about decompiling and bytecode to help him track down bug: a sender is not being found.)

A Coordinated Testing Framework, Carsten Harle, straightec GmBH
In an ideal world, test cases would run quickly, be fixable by any developer, and always run on published code. In reality, we have broken test cases. So how do we handle them.

Broken code (JP mentioned the issue of distinguishing genuinely broken from test case not passing) can be part of the XP process. New tests are expected to fail initially. Tests can be written to demonstrate known bugs or desired improvements.

Their test server tool tracks test results and code versions. Developers register with it and then execute tests with a test client which communicates to the server. Now everyone can see what tests are passing or failing against which versions of the code. He showed the test client UI. Its tabular layout shows all the tests; any can be selected and run. This starts the test execution engine (their internal one but any other can be plugged in, e.g. TestRunner). He showed a test that had passed but showed red in the next column indicated that the developer did not have the latest published code loaded. Failing tests also have a cause displayed e.g. the

DNU and message or the assertion description text.

Developers can exclusively checkout test suites from the server for repair. Such checked out suites are not run automatically when other developers 'run all'. Other developers can see who has checked it out and so coordinate their work.

TestCases can be put into automatic execution mode. Ten computers running test clients can be set to look at the server and run the next test until all have run, checking it out while doing so, so each runs a separate subset of tests. (Same checkout so if long run shows early fail you can check it out and the auto run will not rerun it; either you do not check it back in before auto finishes or you do and it is rerun and you see if your fix passes.)

You can therefore choose not to run your tests locally but just to publish them and have the server run them in the background.

Often you see a failure and the first thing you want to ask is when did this test last pass. The server can tell you that immediately. They have views to show statistics of numbers run, completed, etc.

They have clients for several Smalltalks and for other platforms (Java, Eclipse, .Net) in case your system must work with them. They support SUnit and other test frameworks. Work in progress includes coverage.

The current code is too integrated with their image to give away. They are reimplementing and expect to offer it open source, perhaps charging for things like the Java plugin.

Q. Make the display a tree widget? They have a basic tree view (two-level: class and its tests) and are thinking of going to a general tree view as soon as they decide what the higher level categories will be. (The obvious ones are the class' package or bundle in VW7, and equivalent CM units in other dialects. I often build TestSuites programmatically, so would like this framework to accept test suites generally, with package, bundle, application or whatever being just particular ways of getting such suites, as indeed is the case for class already in basic SUnit.)

Code Optimisation, Adriaan van Os, Soops

Adriaan started with a cartoon of a donkey and cart, and a car. Adriaan was impressed with Travis Griggs' talk at Smalltalk Solutions; this talk is much influenced by it.

Adriaan has worked at Soops since 1995. Demanding Soops projects include one that must build forms from rule-based warehoused data (economic research app), and another that does time-critical calculations to do power trading. Optimisation requirements are very hard to predict in advance (including whether there will be any need for optimisation). Optimised code breaks Smalllint rules and is harder to refactor. So the strategy is to concentrate on the design first. If there is a performance problem, analyse it, then solve it using various tips and tricks. Then test it!

Analysis can use Time millisecondsToRun:, the (Multi)TimeProfiler and the (Multi)AllocationProfiler. For inspiration, try the RBByteCodeTool and the RBDecompiledTool. He demoed the TimeProfiler. When using `millisecondsToRun:`, run it many times, e.g. `1000 timesRepeat:`, and watch out for large integers, allocation and the garbage collector.

Special selectors: see the defined opcode pool class for specialized opcodes that are written if syntactic requirements are met. This cannot be overwritten.

Optimised selectors: will be transformed if syntactic requirements are met. See the MessageNode class. These cannot be overwritten. Adriaan showed the `ifNil` transform.

Primitives: write your own.

Use `and:` instead of `&&` even if you have the argument ready because `true and: [true]` is faster than `true && true`. Although the VW compiler will warn you, `aBoolean and: [aBlock value]` is *not* faster than `aBoolean and: aBlock` but `timesRepeat: [aBlock value]` *is* faster than `timesRepeat: aBlock`. The compiler inlines `self do:` inside SequenceableCollection but not outside.

Inlining is one way of reducing the number of message sends.

Keep your blocks clean. Speed is increased if you declare variables in the innermost scope and especially if you avoid assigning in a block to outerscope variables. It also helps to avoid returning inside a block.

Numbers: use higher generality first (10.0 * 10, not the other way around). Use doubles. Avoid fractions. Avoid large integers. Avoid coercion.

Collections: avoid intermediate collections and repeated iterations by using appropriate iterators, e.g. `aDict keysDo:` instead of `aDict keys do:` (avoid keys anyway if possible). If you need more speed, implement others such as `select:collect:`, `select:do:`, and `collect:select:`.

```
anOrderedCollection firstIndex
  to: anOrderedCollection lastIndex
  do: [:idx | ...
```

used to be faster than

```
1 to: anOrderedCollection size do: [:idx | ...
```

but it is slower in 7.4.1.

Use the correct collection type for your expected size and use. John Brant uses RBSmallDictionary because it gives faster lookup for 6 or fewer elements. Note that `aColl asSortedCollection first` is a lot slower than `aColl fold: [:a :b | a min: b]` - avoid unnecessary collection-handling code and move static code outside the iterator block.

`ifTrue:ifFalse:` costs. Maybe you have too few classes. Put the common cases first in case-statement-like lists.

Make caches fast to lookup and simple to manage.

In GemStone all the previous points apply. In addition, having objects on disk makes using identity preferable to equality. Reducing round trips is a major way of improving performance. Garbage collection of shared objects is harder. Use the specialized collection types.

Adriaan then showed some examples of performance-assumption-checking tests. When you change your dialect, your version or your platform you should reset your assumptions. His `slower:than:` method addresses the GC, allocation and repetition issues he mentioned. He ran some of these tests including a cascade-is-faster-than-list one (I was amused to see that the refactorings Kat and I had been writing in the parallel Camp Smalltalk were in fact optimisations).

Dotted reference is slow and `Object.DependentsFields yourself` is slower than

```
obj := Object.DependentsFields.
obj yourself.
```

Adriaan concluded by saying that design should be done first, performance optimisation last; don't guess about performance. Inlining, caching and choosing the right collections are the easy wins.

Painting Objects with Mondrian, Michael Meyer and Tudor Girba, Software Composition Group, University of Berne
This is Tudor's third time at ESUG and, as usual, he has already demoed his tool to lots of people by the time his talk is due. Tudor talked and Michael drove the tool.

Mondrian is about visualisation. People who see such talks divide into those who love images and those who say 'Why visualise? The browser is good enough." Is one picture really worth a thousand words. He showed a standard architecture diagram: it was a picture and it had a thousand words (in all its class boxes' names, etc.) and it did not communicate very well. Then he used colour, outlines, connector lines and dots to show how quickly we can spot patterns visually and on how many axes. So one picture can be worth a thousand words.

Moose is a big thing on the Smalltalk CD that alarms people. Let Mondrian show what Moose really is. He showed a thousand words (Moose' selectors) in which you can't spot any patterns (Georg promptly spotted that no selector began with Z). Tudor then changed the sizes of various words; now we had visual categories of words (the big, the medium, the small). Then he applied a grey scale and ordered them from darkest to lightest. (made an almost-English phrase :-). All this did little but it was all done in Moose: Moose lets you do quickly any visualisation you want.

Their GraphViz tool draws nodes and edges. He brought up the Moose browser and selected a Moose model showing classes, methods, call-sites, etc. He inspected a classGroup: the 38 classes in the model. A simple script

showed a simple graph of the inheritance hierarchy

```
classGroup each directSuperclasses do: ...
```

Simple Smalltalk code defines the nodes and the edges. The nodes have no name, no behaviour when you click, etc. This is what standard graphing tools do: they 'kill' the object and present it dead and stuffed. You are at a distance from your data (i.e. a lack of reflection; there was a relationship but now it is history). Mondrian is a scriptable reflection of the data. Its metaphor is of painting a view. Its style has much in common with Seaside.

```
view := ViewRenderer new.
view nodes: classGroup.
view nodes: classGroup forEach:
  [:eachClass |
  view nodes eachClass methods.
  view gridLayout].
view edges: classGroup mooseModel allInvocations
  from: [:each | each invokedBy]
  to [:each | each anyCandidate].
view edges: classGroup
  from: [:each | each superclass]
  to: [:each | each].
view treelayout.
view open.
```

He built up the graph adding each line (of the middle group above) and executing to show how the view changed (usual demo hiccough at one point, a DNU quickly fixed).

Q. You are re-executing the workspace; can you tell the view to re-layout? Yes (loved the question). Tudor has created a MondrianEasel, a pane within the three-pane Mondrian browser. A quick self-reminder of some of the code found Tudor the `classShapeWithlabel:` method.

```
view classShapeWithlabel: [:each | each name].
```

but now the method look is wrong so

```
view nodes: classGroup forEach:
  [:eachClass |
  view labelShape label: [:eachMeth | eachMeth name].
  view nodes eachClass methods.
  ...].
```

which ended up with a standard-looking detailed class hierarchy diagram.

Tudor always changed the model; he never wrote UI code. He wants to see which visualisations help and which hinder. In the above, `view` is like Seaside's `html`; you keep rendering on it.

Next, Tudor looked at interaction. Models are installed, like spec methods, as a method on a class, taking parameter `view` and converting class references (e.g. `classGroup` will become `self` if you install on the instance-side of ClassGroup).

```
view nodes: classGroup forEach:
  [:eachClass | eachClass viewBluePrintOn: view].
```

but we only want to see it for the class we select

```
view interaction onSelect:
  [:eachClass |
  view2 clean.“this code will be different soon”
  eachClass viewBluePrintOn: view2.
  view2 evaluate].
```

Then he made them popup on right-select, view on select, and in the view2 made the fly-by-help show the code of a method if you moved the mouse over it (easy code but more than I could type and listen to at once).

Q.(Jan) Change the model through the view? Yes.

Q.(Grit) UI to change model? They are looking at providing that?

Q.(Niall) how do I use this? The beta version is on the ESUG CD. A new version will be released soon.

They have an RB code tool in which you can preview how layouts will appear in the edit as you define new layout methods.

Application Frameworks, Experience Reports and Packaging

Dynamic State: a dynamically defined state model not based on the State Pattern, Alfred Wullschleger, Swiss National Bank

Alfred has been in Smalltalk since 1992. He worked on OVID at Fides Informatik (in production for 11 years) and the OASE project at SNB, in production since 1999. OASE gets financial statistics from Swiss Banks and companies. It runs on GemStone/S and VisualWorks 3/Envy.

Users of the system want easy configuration of their workflow, so they wanted a non-class-based configurable state model with easy communication with other system components. Their solution was CxStates. It uses the ANSI Event model as its implementation pattern.

CxBaseState defines a state with a table of legal transitions to other states. CxTransitionEntry in this table has a symbol identifying the transition. CxActionSequence and CxAction communicate with the outer system. CxStateEnsemble defines the complete state diagram with all legal transition symbols.

There are two kinds of actions:

- preTransitionActions execute within the current state
- postTransitionActions transition to new state, then execute an action in that state.

The message receiver is inserted dynamically during the transition; it is not (usually) knowable before. So the receiver begins as nil, is set when the transition executes and can be delegated to polymorphically. The CxTransitionContext holds the receiver and the arguments for the declared action. `CxBaseState>>doTransition: aCxTransitionContext` does all this and returns nil when illegal, false when inhibited, and the new state otherwise. A special CxCondition action is used to inhibit and returns a boolean.

As the real model is very large, he demonstrated something simpler - a coffee machine with 6 states and 7 transitions, such as #stopGrinding and #startBrewing. The defining code was a many-line method but easy to read. He showed the machine running and increased the temperature till it was ready for coffee after which the next states ran. All the states reported themselves on the transcript intelligibly. He then did it again and was not fast enough for the machine so was trying to move to an illegal state, showing the inhibition.

Q(Bernard) Where do you use it? Statistical data from banks are received and processed. They have a state machine to handle the message arriving and being run through all these transitions. Each message has its own state, distinct from others. A complex machine handles the states of received, late, received with errors, being processed, returned for update, etc. There are 3 other applications of the model. Typically their applications were released with 30-40 states. Now their customers are asking for more states to handle more conditions.

Q. Have you used this for device management, as in your example? No, that was just an example. (Bob) could you simulate a machine with real-time constraints with timing constraints? Current use is in non-RT environment so not yet tested for such use.

Q(Niall) non-polymorphic return (nil, bool, state) did not impact code? No need for null pattern? They started by always returning a state, then found a need to distinguish illegal and inhibition returns; they have not yet seen an impact on code (the return handler is simple, single point in the code).

Cryptography, Martin Kobetic, Cincom
(See my reports of Martin's related talks at Smalltalk Solutions 2006 and ESUG 2004.) Elliptic curve mathematics is being used for cryptography on small devices that cannot manage the standard public/private algorithms. In these standard algorithms, the so-called 'secret' keys are not in fact secret: they must be shared between those who must communicate. The 'private' keys are indeed known only to their owners.

Symmetric algorithms are 32-bit-register arithmetic operations. Public key algorithms are more complex as they are derived from known hard computational problems such as prime number factoring or discrete logarithms. The mathematics is not that hard; on the contrary, it needs to be conceptually simple since for security you must be able to reason about all possible modes of failure.

Symmetric cyphers use a secret key that the two parties must share. If there are more than two parties you need more keys unless you are happy that everyone can read everyone else's mail. Sharing keys conflicts with the need for security. Asymmetric algorithms have a public key to encrypt and a private key to decrypt. Thus anyone can use your public key to encrypt something and only you can decrypt with your private key. It also means you only need one public key to receive messages from many people.

RSA was released in 1977. Let the modulus value n be the product of two large primes p and q, let public key e be a prime relative to p-1 and q-1, and let private key d be 1/e mod (p-1)(q-1). If M is an integer, less than n, representing the information to send (in VW, use ByteArrays), then anyone can send you $C = M^e \bmod n$ and you can recover $M = C^d \bmod n$.

You therefore need to generate keys with these properties. e is often set to 65537 (which is smallish but performant and safe provided your other values are set to correspond, i.e. d will be much larger). The smaller your e value is, the faster the algorithm will be. The larger your message is the more secure it is; very small messages can be fairly easily cracked by taking the e-th root. Modular reduction is what gives security. When padding a message to avoid this, repetitive structure is a bad idea and weakens the security. RSA is implemented to pad as required. An attacker cannot factor p to its primes but the attacked text must be many factors of p or an attacker can just try the first 1000 powers to see if they get a match.

He inspected examples:

```
keys := RSAKeyGenerator keySize: 512.
alice : RSA new publicKey: keys publicKey.
msg := 'Hello World' asByteArray encoding: #utf8.
msg := alice encrypt: msg.

bob := RSA new privateKey: keys privateKey.
msg := bob decrypt: msg.
msg asStringEncoding: #utf8.
```

(At this point the usual demo hiccough occurred. Martin's slide framework DNUed when he executed the code in the slides; quickly fixed.) The cypher is likely to be 64 bytes long since whatever the size of the original, raising it to power 64 is likely to leave it at that size.

If public key is so great, why do we have symmetric. Because it is 3-4 orders of magnitude slower. So people use the public key to encrypt a one-time symmetric key, e.g. `key := DSSRandom default next: 40 ..` and then attach that to their symmetric-encrypted data. This is called key establishment. It can be done by RSA.

```
key := DSSRandom default byteStream next: 40.
msg := 'Hello World' asByteArray encoding: #utf8.
msg := (ARC4 key: key) encrypt: msg.
alice := RSA new publicKey: keys publicKey.
key := alice encrypt: key.
bob := RSA new privateKey: keys privateKey.
key := bob decrypt: key.
msg := ((ARC4 key: key) decrypt: msg) asString.
```

It can also be done by Diffie-Hellman http://www.ietf.org/rfc/tfc2631.txt. DH cannot be used for encryption or signing but it can create a common shared secret between two parties over an unsafe channel. You need a large prime modulus p (usually > 512 bits) and smaller q (usually > 160bits) and generator g (order q mod p). Private x is random between 1 and q - 1. Public y is $g^x \bmod p$. Public y' is the other party's $y = g^{x'} \bmod p$. The shared secret is then $y'^x \bmod p$ which is the same value for both parties.

```
gen := DHParameterGenerator m: 160 |: 512.
alice := DH p: gen p q: gen q g: gen g.
ya := alice publicValue.
ss := (alice sharedSecretUsing: yb) asByteArray.
bob := DH p: alice p q: alice q g: alice g.
yb := bob publicValue.
ss := (bob sharedSecretUsing: ya) asByteArray.
```

It is a perfect forward secret in the sense that the DH establishing key can then be discarded. If this is done then even someone who captured the communication and later gets hold of alice' and bob's computers cannot read the communication.

Diffie-Hellman can be done offline with one party publishing their public value where anyone, including the other party, can see it.

```
bob := DH newFrom: gen.
yb := bob publicValue.

alice := DH newFrom: gen.
ya := alice publicValue.
ss := (alice sharedSecretUsing: yb) asByteArray.
msg := 'Hello world' asByteArray.
msg := (ARC4 key: ss) encrypt: msg.

ss := (bob sharedSecretUsing: ya) asByteArray.
msg := ((ARC4 key: ss) decrypt: msg) asString.
```

Q(Carsten). Why do it this more complicated way? DH was in the public domain from the beginning. The RSA patent only expired in 1997. Even now, you may be talking to a library that only supports DH. Also, DH has the perfect forward secrecy property.

The most important use of signing digital signatures is to see if anyone has tampered with the content; you can't prevent it so you need to spot it. It can tell you who you are talking to. It can provide evidence for non-repudiation (which you should consider the value of in a given legal or social context).

Hash functions are used for data finger-printing. They must be unable to reconstruct the original data content and yet *very* unlikely to collide. A hash function is a compression function. MD-strengthening is one such. A risk is that if someone knew the hash function and the original crypt, they might be able to append without knowing the original decrypt. Thus some noise is usually added; this is called length extension (there may be undetectable brief interrupts in your music files caused by this).

MD5 was designed by Ron Rivest in 1992. It chunks the message into 512 blocks and increments a 128 bit digest. It was broken in 2004 by a Chinese group who discovered how to generate collisions within hours of seeing the message. A collision is not a complete break but you should now not use MD5 for any new work. Researchers are working hard on new functions.

```
(MD5 hash: 'Hello' asByteArray) asHexString.
(MD5 hash: #[1 2 3 4 5] from: 2 to: 4) asHexString.
(MD5 hashFrom: aReadStream) asHexString.
```

SHA was designed in 1993, broken quickly and evolved to SHA-1 in 1995. This was broken in 2005, though not as badly as MD5 (they reduced collision detection from 2^{80} to 2^{69} which does not yet allow collision-generation within hours). Its variants SHA-256, -380 and -512, released in 2002, are harder but researchers are seeking new cryptography with basic structure differences.

```
input := ‘Hello World’ asByteArray readStream.
sha := SHA new.
sha updateWithNext: 5 from: input.
sha digest asHexString.

sha updateFrom: input.
sha digest asHexString.

input reset.
(SHA256 hashFrom: input) asHexString.
```

Q. API is the same across languages? Martin would expect Java’s API to read similarly to the above modulo language differences.

You can use RSA for signing (this is rather stronger than handwritten signatures which are fairly easy to forge). You hash the plaintext and encode this digest. Then you encrypt the digest with your private key. To verify, the recipient decrypts the digest with their public key. They decode the digest and hash the plaintext. Then they compare the digests. The API is straightforward, e.g.

```
| keys alice bob sig |
keys := RSAKeyGenerator new keySize: 512.“or larger”
alice := RSA new privateKey: keys privateKey.
sig := alice sign: 'Hello World' asByteArray.
“You can inspect sig asHexString to see what it looks
like. At the far end”
bob := RSA new publicKey: keys publicKey.
bob verify: sig of: 'Hello World' asByteArray.
```

Digital Signature Algorithm (DSA) resembles Diffie-Hellman with somewhat more involved mechanics. Their p has been revised down and then again up over the last few years. At present they are generating keys of length 15k. Their q is fixed at 260 bits (so we do not specify it below).

```
keys := DSAKeyGenerator keySize: 512.
alice := DSA new privateKey: keys privateKey.
sig := alice sign: 'Hello World' asByteArray.

bob := DSA new publicKey: keys publicKey.
bob verify: sig of: 'Hello World' asByteArray
  “this returns a boolean”
```

SuperModel, Stefan Denove, MediaGenix
MediaGenix has 26 people of which 16 are Smalltalkers. It is a 100% Smalltalk Dutch-speaking company with 14 years of ST experience. A MediaGenix product is WhatsOn.

The product uses an XML model, a storage model (Object Lens) and a domain model. The write-once paradigm pushed them to SuperModel. He opened a code browser. on a PSIMusic class, showing the methods that the

old approach had: `addXMLSpecsTo:`, `addDBSpecsTo:`, etc. Now, `buildSuperModelWith:` has a single definition of a class' contribution to the model. (Standard meta-model definition style, like Glorp et al.)

He added a field-defining line to the method and demoed the appearance of virtual accessors for it, and the visual code tools view of it. (Usual demo hiccough at this point; mistyped line caused DNU #type - presumably as in 'you can't' message from machine :-). He upgraded the database with the new field. Continuing demo hiccough - debugger revealed domain attribute had not been set.

Q(Niall) The Argo trick for accessors (Michel Tilman et al). The virtual accessors are not actually there; just seen in model. It's a new approach, so not quite Argo's old solution, but influenced by it (another ex-Argo person is now at MediaGenix).

Q(Carsten) Do you have GUI builder for your model? We have a builder to add layout and similar info to the model. Default rules enforce a consistent look and feel annotation.

Sophie and Plopp, Michael Ruger and Gird Schuster, Impara
This talk was titled 'Building End-user Applications' but everyone knew it as Sophie and Plopp, not the names of the presenters (though they could have been; I guess that would make Gird 'Sophie' and Michael 'Plopp' :-) but the names of the Impara products they demoed. Michael stressed the importance of robustness, system integration, etc. to end-user applications; people who build applications in Squeak must think of this because Squeak by default does not. For example, Squeak pre-loads all fonts, which you do not want. It dumps everything in C:\Squeak instead of being a 'good platform citizen' and locating things appropriately for each platform, etc.

Another goal is making things look really good; they have two full-time designers just doing that. It is a discipline. Cross-platform is also hard. Lastly, the packaging must be slick and common: install/uninstall, double-click application, double-click documents on all platforms.

They have some 10 people working on these projects including students. They use Monticello and Subversion (which has been very productive for them) for code control. They use Mantis to track bugs, features and notes. (They were sceptical of Mantis at first but have found themselves using it more and more.)

Then Gird opened Plopp, a painting tool aimed at children. Very attractive sidebar tools let her choose colours and brushes and start painting in 2D. She drew a cartoon icon of a girl and then converted it to 3D (quick and slick). She spun her and gave her hair, cloned to give her a friend. Then she chose a background from a palette, set its colours, drew a street and rotated it to have perspective (sand, road, pyramids). The figures palette now had the shape she had just created (flipchart visual icon for prior versions). Sound effects help the child see the computer is doing something whenever an operation took a few seconds (only one or two step were not immediate).

Children can take pictures (camera icon) of what they have drawn and email a postcard of it or make it their PC background.

Having demoed, she opened slides to summarise. Plopp has no menus and no help files (it uses audio help). It was tricky to implement audio help: they had to schedule the audio correctly, ensure it had the appropriate priority of execution and solve sound manager buffer issues.

Kids forget to save and have a hard time with standard filesystem tools so everything is always saved and versions are kept. The challenge is to find which states are sensible to save.

Colours are not as simple as they seem. The needed a real-world-equivalent colour model that children could understand. They settled on a subtractive colour model with a paint tube paradigm. They do not currently handle the full colour space.

Kids tend to have low-end machines and/or their parents' old ones. They needed it to run well in that technical environment. For output they had to provide the printing support that Squeak largely lacks. The postcards needed to protect against spam (they use a salted hash).

Visit www.planet-plopp.de to learn more.

Michael spoke about Sophie. Impara hope its paradigm will become the future of reading. Bob Stein proposed TK3 at Voyager in the late 80s (some hypercard involvement). TK4 became Sophie. Bob talked to Alan Kay and became infected with Smalltalk. The project is open-source (MIT licence). Its initial funding (from the Mellon foundation) has just ended: five developers, two designers and one visionary (Bob). They release 'softly' (i.e. tell noone :-) on September 15th; see dev.sophieproject.org.

Sophie aims to have no key modifiers and yet everything is to be one click away. They want to maximise the effective use of the screen; no wasted space. They use halos and huds to give minimal mouse travel. They have (almost :-/) no modal dialogs.

The small common core architecture lives under an application architecture that provides the UI. Every UI function is part of an extension (to control feature creep). The core architecture contains the tree structures, timelines, triggers. They have their own units 127 cTwip = 1 twip = 1/20 pt = 1/1440 inch = 1/7200 mm. It uses Rome and Freetype for fonts, XUL and CSS for styles, XML for storage (they version 'everything').

Michael then demoed. Opening a document scanned the fonts, then 'that's what I call maximising screen real estate' (and indeed there was a very small topbar only). The demo was in the developer version so a debug menu accompanied all the others. He viewed a Sophie-book in the page, a picture whose elements had halos from which he could resize elements (at this point, he had the usual demo hiccough - a DNU while using a halo). He brought up Resource icon lists and added to the book, demoing the

usual word-like text functions, making text into buttons to show images.

He created a timeline. Timelines can be given various resources and will display them at the appropriate points in synchronism as the timeline runs after being triggered (by user scrolling, by user clicking on button, etc.).

They have to fight to keep themselves using drag and drop and not to stuff more and more in menus.

Q. (Christian) Why not do your talk in Sophie? In two weeks Michael would have but he was not quite convinced enough today.

GemStone Notification, Alfred Wulshlegel, Swiss national Bank
You install a static exception on Exception with a block that does whatever you want, then signals System. A class creation protocol variant maps to the GemStone error number.

```
Exception
  installStaticException:
    [:theException :cat :num :args |
    “do here whatever is needed”
    System enableSignaledObjectsError]
  category: GemStoneError
  number: (ErrorSymbols at: #rtErrSignalCommit).
```

You want a single error notification system that is conflict free (every session must be able to commit independently and you can have as many notifiers as you wish).

On the event side, objects that are committed and want to tell other objects raise TpzNotification events (Tpz signifies Topaz). The event handler side uses TpzNotificationReceiver to get the event and do the actions. Thus generation and handling are wholly separate.

TpzNotifications are identified by a symbol name and hold an array of size `System maxSessions`, where the TpzNotificationElement for each session is placed. This has a value and a session id from which we can quickly lookup session values.

```
TpzNotification>>tpzNotify
  (sessionValues at System session) tpzNotify
```

keeps count of notifications. Array index lookup is fast enough to let 1000 sessions use this without impact.

Handlers are session-based so they are stored in slot 20 (user slot) in the session state object. A TpzNotificationReceiverCollection class holds them and the notification exception. Each Tpz..Receiver is activated by its notification to start a process. `registerForNotification:` initializes the loop, puts itself into the collection and executes for the first event (the one that activated it).

When a notifier is signalled, the System is asked for all signalled objects, to which it is forwarded. There are usually few signalled objects so

`notifierSignaled:` and `hasSignaledObjects` are fast enough. You may have 1000 sessions but for an event only 2 or 3 signalled objects.

Each receiver is a subclass of TpzNotificationReceiver and it holds a notifierSymbol and the identifier. The `executeFrom...` methods are where the actual work is done.

Alfred then demoed a single arrangement that notified 5 seconds after its object committed. Installation took a few seconds (a wait ensures no lost events as you install). He GS-Printed his example and saw an array of results with no notifications - because he had not committed anything. Then he committed and ran and saw them.

Q. Use in your own system? In users' sessions, they add stuff to reduced conflict queues. Its process must be notified when objects are ready for it. When the user commits an object (e.g. an RcInput) that notifies the process that something in the queue is now ready.

Rethink Smalltalk, Matthieu van Echtelt, Cosmocows
Matthieu founded Cosmocows with Wouter Gazendam. Wouter built a development environment on top of VW providing business administration systems to their clients. The clients need model and data persistency (especially financial data), audit support (how was this invoice created?), authorisation, transaction support (one change impacts several databases - multiple concurrent changes to model and data). They want to update their model and asynchronously update their data to match the new model.

Usually these issues are not solved in the beginning. The system grows exponentially and solving these problems causes an exponential growth in (or return to) productivity.

Cosmocows goal is to resurrect some design principles.

- A single individual must be able to understand the entire system. This is very difficult in a multi-tier business world of XML streams, HTML and Javascript, with OO used to manipulate data and SQL to store data.
- A truly OO system provides automatic storage management.
- Every component the user can reach should be able to present itself in a meaningful way. Today, this can mean observation over the internet in a web browser by multiple concurrent users.

Matthieu showed three implemented solutions, all of which sort of started from scratch although the third was strongly influenced by the other two. He opened his web browser and logged in to the application. He obtained a form and typed in some data. This was a form for the model in which he could name entities, choose types from a list, etc. The web form also has a test block where the behaviour of model changes can be exercised.

He created a Person class and added some value slots to it, relating Person to Organisation, etc. The appropriate buttons and fields appeared in the main-level forms as he altered the meta-level. You can nest schemas inside schemas. This affects layout. He nested Person inside Organisation and

Persons became an expansion on the Organisation page rather than a call to a separate page. He showed form and list layouts.

Q. Where is data? Data and meta-data (the model) are in the same database.

Q. (Stephane) The meta-model is describing itself? Yes.

He added a set name action, activated by the user.

That was the first solution. Managing 500 schemas like this was not ideal. The second solution was to let the user see appropriately-detailed class and method data by rendering the model description into expressions for all system layers (SQL, OO, HTML, ODF, Javascript). This tended to break Model-View-Controller separation in a few cases which therefore required specific overrides.

The third solution was to have a web-based IDE to develop models and another to administer them.

He started a current model (the demos above were on a trivial demo model) and had the usual demo hiccough (he had to reset the connection). He loaded the model and some data and showed the various navigation options to understand the model. Then he looked at layout and styling, how they controlled the look of the forms they generate.

They were influenced by Joseph Yoder's and Ralph Johnston's meta-data work, and by Magritte.

Q. Graphical meta-model editing? They would like to but have not had enough time. (They would have to do it over the web, not just in VW, but Javascript could make it doable. He is surprised his customers have not asked for it.)

The announcements framework was easy to integrate because they were using meta-modelling. Changing the base technology is easier if you meta-model.

Packaging Freeware/Shareware/Smallware, Rob Vens, Sepher Software

Rob was an organiser at the first ESUG conference at Brest in France in 1993. He was a smalltalker 'from birth'. His dayjob has no Smalltalk; he uses it in evenings and early mornings. Recently he wrote some tools (to manage his personal finances) when he could not find ones doing what he wanted. The apps convert proprietary financial formulas used by specific institutions into standard ones, etc.

He thought about offering them as freeware/shareware. After some hesitation he did so. Rob decided to share his experience of doing this. He knows of BottomFeeder and wondered if any others had done this.

So, if you are one person with little spare time, what are the problems? You

will have a very irregular release rate. You have an anonymous user community (no registration or tracking of them except that one of the users is yourself), who drive all changes, i.e. the work is bug driven and feature driven. Sometimes email traffic is heavy; at other times weeks go by with nothing. His aim was to support as many people and platforms as possible, so he releases localised versions. He uses the RuntimePackager with no significant stripping. For Windows he delivers a standalone executable.

Packaging: he starts from a virgin image, loads his source and then uses RuntimePackager. He loads via file-in (parcels seemed to do something that he had not time to look into). He uses VisualStudio, 7-Zip (public domain), Xara3d (cheap) and Axialis IconWorkshop to do icons for applications, and ResHacker of course for the windows standalone executable.

He showed the standard directory structure on his computer: folders docs and packaging, latter has subfolders for each platform he supports. He uses a simple script. (BottomFeeder uses a script driven from a Smalltalk image; he will move to that.) His is just

```
call compress
callBuildLinux86
callBuidlMacPPC
....
setVWHome
```

There is no data; everything is in the image saved once when he packages. Boss creates any objects required (‘lessmi’ is Dutch for ‘readme’).

Distribution is done via his website www.sepher.ni; choose your platform and download. He usually maintains a Dutch and English version of his web pages. At the foot of the page is a download list.

Q. What use is VisualStudio? On Windows, he needs to install in a folder that a user specifies. He needs to do things in registry. He knows there is software in VW to do this but he is accustomed to VisualStudio. Platform compliance is always hard. Windows was the most difficult.

None of his code is signed with any digital certificate so users get warnings about ‘unsigned’. (Anyone using CACert? Apparently not; Jim uses the Cincom website.) He is looking to solve this. (If you are handing out points, help him collect his hundred. If you don’t know what this means, don’t worry about it.)

He tries to maintain a consistent bug and feature report but the page on the site is out of date. Jim uses sourceforge just for the reporting. Rob is thinking of moving there for that reason. A mailing list did not work (Niall: because people are concerned not to see too much in their mailbox; when they have a problem then they want to mail? Rob, yes and also sometimes the follow up involves confidential information exchange) so people email and he replies. It is important to reply within the same day otherwise you don’t get a reply to your reply; the person who emailed you has moved on.

The code is published, partly so people can check the app is doing nothing secret with their personal finance data. You can download it from the repository on his website with a copyright warning. Reinout recommends the MIT licence; it is short and easy to understand; use for anything,. original writer keeps copyright, user cannot charge royalties for using it. Currently, Rob uses an old NC license from VisualWorks. If users get profit from an app but Rob does not then Cincom will not bother Rob but they might ask him for his list of users to contact them.

Shareware likes to make some money; usually there is a donation button. He gets almost nothing. Is it worth the hassle? Yes, because it was very easy to put that donation button on his website. Legal hassles? not till he make real money. Jim agreed that when you make enough money to care you will be changing your process model anyway. Meanwhile, Jim is happy to see this kind of thing happening.

Web Frameworks and Applications

The Art of Seaside in 10 steps, Lucas Renglii, netstyle.ch

The audience started by dragging Seaside and the exercise image from the ESUG DVD to their laptops. The exercise was a ToDo list. I paired with Yann Monclair so I could take notes on my machine while his machine did the seaside application.

Seaside is intentionally different from other webapp frameworks; stateful, not using templates or meaningful URLs. Seaside is built out of WAComponent subclasses. These are the views and controllers of the Seaside application. Components keep their model and UI state in instVars.

The image had a ToDo-Model category with suitable simple list code. We created a ToDo-View category, subclassed to ToDoListView.

```
initialize
  self registerApplication: 'todo'
```

We `renderContentOn:` to generate the view. Rendering is a read-only phase that should not change the state of the model (or Seaside get confused). Seaside supports two different rendering classes. The default renderer is the old one, not the new one (VWRendererCanvas) which is the one we will be using in this tutorial, so we need the `rendererClass` method to return VWRendererCanvas.

We now write a method `model` to get at our model (the ToDo list in this case) and start the `renderContentOn:` method by displaying the title.

```
  html render: self model title.
```

(At this point, some did `WAKom startOn: 8999` to get the seaside server started on a port that suited them.) The html var is a rendering canvas, which as its name suggests is like an empty sheet of paper you can paint on.

```
  html render: self model title.
  html break.
  self model items do:
    [:each | html render: each title. html break].
```

As a brief aside, Lucas mentioned how one improves the look of sites. Get a brush via `html div`, configure it, e.g. `html div class: 'stylish'` (or set values individually), and so on. He used a brush for the title, and then for the items (putting the latter's iterator block inside the brush block).

```
html div
  class: 'title';
  with: self model title.
html break.
html div
  class: 'items';
  with: [self model items do:
         [:each | html render: each title. html break]].
```

then refactored to a method `renderItem:on:`.

Lucas reviewed the buttons R (rendered view) and S (source view - the raw html as text) on the displayed page. (At this point, Stephane, on behalf of the audience, begged Lucas to give the audience one minute just to look at the state so far and catch up. Yes, you can build apps fast in Seaside - too fast for us to keep up without a pause. :-)

Now we have our XHTML. We can now either tell the designer to style it or we can click the last top-left button to get the style editor. We can write CSS to style components

```
.title {
  font-size: 20px
  font-weight: bold
}
```

He saved and showed the changed look of the page and also the generated style method that returned the CSS as a string.

We are now generating output from a fixed model. We would like to interact with our webapp. We do this via callbacks. Ask the canvas for an anchor and set a callback for it.

```
html div
  class: 'items';
  with: [html anchor
         callback: [self inform: anItem due]
         with: anItem title]
```

HTML demands that Forms must have form tags rendered around all their content so

```
renderContentOn: ...
  ...
  html form: [... render: item on: ...]

render: anItem on: ...
  ...
  html checkBox
      value: anItem done
      callback: [:value | anItem done: value].
  html anchor
      callback: [self inform: anItem due]
      with: anItem title.
```

Finally we add a submit button at the foot of `renderContentOn:`

```
  ...
  html submitButton.
```

However after saving you can still change the checkboxes, then update and see them revert to their saved values; not ideal from a usability perspective. In the next talk, we will see how AJAX/Scriptaculous solves this problem.

We can select ToDo items but not edit their text. This needs the call architecture: `answer := self call: aComponent`. Now we need another view class: the ToDoItemEditor. As we've already seen view creation, Lucas just grabbed the 10-20 lines of code that set up a complete editor view. If the user selects an item we want to present the editor view.

```
renderItem: item on: ..
  ...
  html checkBox
      value: anItem done
      callback: [:value | anItem done: value].
  html anchor
      callback: [self edit: anItem]
      with: anItem title.

edit: anItem
  self call: (ToDoItemEditor new item: anItem).
```

Now when he clicks on a link he gets (the usual demo hiccough - saw aWAHtmlRender because he forgot to implement `rendererClass` for the new renderer on the ToDoItemEditor - Lucas hopes the default will change to the new class soon - after fixing he gets) the editor view of that item. Now how do we get back? By using `answer:` to return a result of the call.

```
item: anItem
  self answer: ...

edit: anItem
  result := self call:
                  (ToDoItemEditor new item: anItem).
  result ifNil: [^self].
  self model items replace: anItem with: result.
```

Lucas ran out of time for component composition. It is straightforward; just remember to define the `children` method for component classes (see the comment in WAComponent>>children).

In summary, we implemented complex workflow without messing with URLs or serialisation of state.

Silt: lessons learned in a Web Deployment, James Robertson, Cincom
Over the last 4 years, Jim has been developing and blogging on a web server. Get it from www.cincomsmalltalk.com/BottomFeeder or from the Cincom Open Repository (the zipfile download is not always absolutely up-to-date with the public store; take whichever you prefer). This talk is about what he learned while scaling this, updating it during operation, etc.

The architecture started with the classic pattern of one class (BlogSaver:

the entry point for the API) having far too much code; this has now been refactored but still has too much. The StorageManager saves and retrieves entries from and to the BlogSaver. One file per day is serialized to BOSS. Blog posts (and any comments others have written to them) are in a collection in that file. Even a prolific poster will only write 5-10 posts per day so search is pretty simple.

The CacheManager is more complex. At first, it read in every blog post every time someone searched. By year 3, 3 x 365 files being read in every time his greater number of readers searched was too slow; hence the cache. It holds the main page and the last N posts searched for. There is a keyword and a category index into the cache, which is a dictionary of posts.

Initially, BlogSaver was the whole application. It was a singleton: assumed there was only one blog. Then Michael Lucas-Smith asked Jim if he could start a blog. Michael thought Jim was talking to lawyers for three months, not working out how to do it for (moments in those busy) three months (Michael codes fast). It took Jim a long time to convince himself that it was easy to do, which indeed it was: `BlogSaver named: 'Some name'.` A class instance variable holds a dictionary of blog instances.

The next problem was that every time someone requested the blog, each reader read the last few posts repeatedly, so he changed the cache to optimise that.

Jim never took the server down to add these things. He tested on a local copy in his house, then when happy he ftp'ed to the server which watched for new parcels. He similarly added migration code to read and re-serialise when formats changed.

The third problem: category searches scanned every post to see if it was in the requested category. By the second summer he needed a keyword cache and that was when he split out the CacheManager class. There is one per blog, holding a dictionary whose keys are categories and whose values are files containing any posts in that category. The server slowed for a few minutes while he uploaded this and that was all. It speeded up category searches enormously and linear search for the category post(s) within the 5-10 posts in a daily file was fast enough.

Keyword search was the same problem but he could not do a full upfront cache for every word ever used in the blog. The bottleneck was reaching all the posts from disk and that slowed the entire server down. The search was faster and did not block the rest of the server to the same extent. A disk read, once it starts running at the same priority as its rivals, will run till it completes, blocking rivals. Jim found the class Promise, which forks at a lower priority and so avoids blocking. This solved the server-slowdown problem.

```
allResults := self actuallySearchFor...
```

became

```
promise := [self actuallySearchFor... ]
  promiseAt: Processor userBackgroundPriority.
```

```
allResults := promise value.
```

Q. Why not cache in memory? Before Jim had thought about it, he had the idea that caching all this in memory was expensive (the server runs on a low-end Linux machine). A week ago, he found that it was not.

This was still expensive. Many do legitimate searches but there are also many referral spam searches who put keywords into their referral line. So he added a cache for posts keyed to year, updated when something changes. Older posts rarely change.

His fifth problem was spam. Old posts would change if Jim left comments open on old posts. Spammers add comments to your old posts to make Google think their page is hot, chancing that you will not notice. So Jim switches comments off after a period. Trackbacks let you see who refers to your blog but spammers use these to get you to visit their useless or repulsive blog.

The server handles comments and trackbacks the same way. He turns off comments on anything not on the front page. His feed only lists the posts on the front page so he can watch and spot spam on the front page easily, on other pages not so easily. He added a ‘no more than N hrefs’ rule for comments. Most fake trackbacks are for a long list of ‘products’ so this spots them. Lastly he added an IP throttle. (Be aware: trackbacks are a spam garden).

Referral spam: the referral scanner was taking too much time so he unified the scan (not separate for each blog) but it was still too slow so he split it out as a CRON job.The blog instances now look for and cache the referral data every few hours.

Jim only solved these problems as they came up. He never foresaw, and from experience never expected to foresee, where scaling problems would appear. Thus he solved actual problems, not bad guesses, with solutions that worked, not that he thought would work. He tests on the local server, then files out a change set and also parcels. He files in the change set (immediate update of running server) and updates the parcels (if server restarts, parcels load).

Q. Why not reload parcels? Jim has found in BottomFeeder that if you reload parcels often enough, ‘strange things happen’. He cannot define strange things easily and has talked to engineers without resolving this.

Q. Why not load from Store? Jim has been lazy about upgrading all the way from 7.1 to 7.4.1 and at times this might have caused issues. Merge was mentioned and yes if Jim used merge that would work fine but he likes what he is used to.

Q. CAPTCHA is a technique that displays a graphic of partially distorted letters and numbers above an input field. The idea is that real people, but not spam programs, can read the graphic and type it in to prove they are real. Alas, Jim has problems reading these distorted images (i and l

confusion, etc.) and others likewise. As regards other anti-spam approaches, e.g. using a public web-service, well, Typekey goes down fairly often and similarly others. When his server goes down, Jim knows it is his fault and he can fix it.

He pushes out XML to RSSAtom and otherwise does nothing with XML.

Jim has yet to hit a scaling problem that was not due to his own code. The Smalltalk server is fast and robust.

Seaside Web 2.0, Lucas Renglii, netstyle.ch (www.netstyle.ch)
Why upgrade? Wikipedia says Web2.0 improves collaboration, design, liveliness. How upgrade? Can it be done via Seaside?

- The first requirements is XHTML with semantically valid markup. Seaside already does this. It is hard *not* to generate suitable XHTML.
- The second requirement is Cascading Style Sheets. Again, Seaside already does this.

Really Simple Syndication is an XML output file so Seaside can do that easily and if you get it wrong Jim Robertson will soon tell you.

AJAX (he showed a great slide of the washing up liquid :-) stands for Asynchronous Javascript And XML. He opened the counter application in several windows, incremented it and all the browsers saw the same value. This technique is call COMET. It uses HTML streaming pushed to the server. It is a variant of AJAX.

The challenge of upgrading to Web 2.0 is that developers want to concentrate on the web apps, not on browser compatibility issues. So use a Javascript library that hides these issues from users. The prototype library (‘prototype’ is the library’s name, not an adjective) was written by a ruby guy and hides the differences between IE, Firefox, Safari, etc. On top of this, the script.aculo.us library supports drag and drop, controls (sliders, etc.) and widgets.

Seaside provides tight feature-complete up-to-date integration with these tools. Yesterday there was a new script.aculo.us release and today you can download the Seaside code. Above all, you can program in Smalltalk; never code Javascript. All brought to you by Avi Bryant and Lucas Renglii.

Be aware that if you do very complex AJAX things in Seaside, you are producing Javascript and Javascript has limitations that Smalltalk lacks. For the rest, think of it as magic:

```
html effect
  id: ‘jint’;
  shake
```

produces a string of Javascript in the right place but to you it just works. He demoed using the code we produced on Tuesday to shake the title on clicking.

```
html div
```

```
onClick: (html effect shake);
class: 'title';
with: ...
```

AJAX can update the page via updater or periodically or on request or on evaluation. The template is

```
html (updater | periodical | request | evaluator)
[ OPTIONS; ]*
[ HANDLER; ]*
[ TRIGGER; ]*
[ CALLBACK; ]*
```

He enhanced the ToDo list. When you click the links you had to remember to save. Let's make clicking a link automatically click the save button. He made the change and opened a debugger that shows all the requests in a list window. He clicked a link and we saw the save request being sent.

```
onChange: (html updater
       id: 'body';
       triggerForm: 'checkboxes'
...
```

updated chosen parts of page only. `triggerInPlaceEditor:` can also be used to display

```
   html span
     onClick: (html inPlaceEditor
        triggerInPlaceEditor: [: v| anItem title: v];
      ...
...
```

(Usual demo hiccough happened at this point: "If you want to use the inPlaceEditor you should first read the documentation that tells you it is cleverer than you thought"). He could now click a todo item and get an editable field of the item name.

Q. How to style it? Change the style sheets.

Then he showed drag-drop reordering of the list. `renderBodyOn:` sets the order and we want to rearrange that order.

```
html div
   script: (html sortable
      tag: 'div';
      onUpdate: (html request
        triggerSortable: 'items
        callback: [: v| self model item: v]
```

We have lots of tag ids; because we may be doing similar things elsewhere on the page we must have them. There are components that speed writing these. We must assign the items as passengers to the id, i.e.

```
html div
  passenger: anItem
```

so that Seaside can know when generating Javascript that this div needs ids for the items.

Debugging: if the problem is on the Javascript side use Firefox (the only

browser that helps you) with FireBug (a poor man's Smalltalk). Lucas finds these the best of the available tools. See scriptaculous.seaside.st.

Languages, Dialects and VMs

OS and VW integration, Georg Heeg

Georg asked who programmed in OS and got three hands; he was expecting one or two, so three was better than expected.

VW and OS history: OS started as Enfin and became ObjectStudio. Seven years ago (almost exactly: the transfer of VW to Cincom happened on 1st September 1999 and was announced at that year's ESUG in Ghent), OS users started to ask when they would get VW features. VW users sometimes ask when they will get native widgets.

Metamodelling is a key feature of Smalltalk and a key feature of this talk. It is much easier for one company to buy another than for one company's product to achieve synergy with another's. (VW's virtual machine for OS was started but not finished.) But in Smalltalk we can do anything, and in Smalltalk program is modelling, so we can model anything, so we can model any other Smalltalk system.

So let us model ObjectStudio 8 in the VisualWorks metamodel. We use all the reflection capabilities: thisContext ("Who has heard of this?": 1/3 of hands raised; Georg fought not to use it and lost and is now glad he lost), the debugger context and so on. Now OS and VW live in the same image at the same time, sharing the same Smalltalk Kernel.

The first step was Georg having vocal cord surgery (for medical reasons, not because his colleagues demanded it :-), so having two weeks of nothing but work on the basic pattern. At the end of those two weeks, he had OS Smalltalk working (no UI, no primitives). Then Georg hired Jorg Belger who made the CE VM when an intern in VW in 100 days. Jorg did the OS DLL.

The first demo was in Frankfurt at CSUG 2004: customers saw it, the first window opened; this was a proof of concept. Then Jorg implemented at the rate of one primitive a day, but there are 450 MFC primitives and Georg wanted to present at CSUG 2006. So then they took the entire OS C/C++ DLL and made it callable from VW. This began in February 2005. In June it was done to alpha. Then they ran all the components of the system and fixed the many fine-grained incompatibilities they found. Usually, each was easy to understand and fix (e.g. replace BinaryObjectStreams with BOSS) but there were many (more than Georg had estimated).

The aim was to keep it as simple as possible, to have *no* changes to the VW VM, and to preserve OS behaviour. The architecture has the base classes and the ObjectStudio DLL running over the VM. VisualWorks code runs on the base classes. ObjectStudio code runs on the OS compatibility layer, above the DLL or the appropriate base classes.

NameSpaces are about *in*visibility. In Classic OS, all global names live in

GlobalDictionary. In OS8, classes live in NameSpace ObjectStudio.

In OS, there are no PoolDictionaries, just IdentityDictionaries put in Globals. In OS 8, you must set their values manually.

There are classes with the same name but different behaviour such as Float. OS Float is VW Double so they map the classes via

```
ObjectStudio defineSharedVariable: #Float
  ...
  initializer Core.Double.
```

There are source differences: OS .cls files hold one class per file. When parsing OS source, ObjectStudioCompiler calls ObjectStudioParser, which automatically transforms the source from VW .cls files. The subclass OSStudioChunkSourceFileFormat in the VW SourceFileFormat hierarchy reads and writes cls files.

There are syntax differences: OS has some Lisp-like syntax that had to be transformed to what VW is happy with. Assignment to a parameter is allowed in OS. A simple transformation assigns the variable to a temp, then modifies the temp and returns it. When `x` is false, `x ifTrue:` returns nil in VW but False in OS (False, not false). VW's choice is right in this case, but not always: `[:n | ] value: 1` returns 1 in VW and Squeak; it should return nil as it does in OS. You can add to Arrays in OS but not in VW. ProgramNodeEnumerator subclass OSStudioTreePerformers modifies the parsed tree to what VW will accept.

Q. (Stephane): do you have tests for these differences and do you use the ANSI standard tests? They have some tests but do not use the ANSI standard tests.

As of this week, there are many substituted selectors; the slide used a small font to fit them all on. Some were basic ones like size, subclass... and new...

For OS, there is strong integration with C. Many methods are in C and call Smalltalk methods because calling between is as fast as any other message send. You can call C primitives directly, by number, by name, by programmatic creation of interfaces and by writing explicit C datatypes in your method's parameters (ExternalInterface-like). They needed clever hacking to prevent VW garbage-collecting the oop on the OS DLL at the wrong times. OS object pointers (OPTR) never move and OS code relies on this but in VW oops are moved by the GC. So these are keyed in the standard registry and reference counters protect as required. The real _oops live in an array whose indices are referenced by the OPTRs. Georg showed the beginning of the huge C definition of the OPTR class.

Direct primitives they mostly reimplemented in VW, or replaced. The code transformation transforms primitives where required, wrapping the underlying call for errors and mapping. The same approach is good for most OS primitives. External procedures needed no changes. Georg showed an example of a manually-created osprim.

In ObjectStudio 7, you had 2 VMs and 2 incompatible images because you had to decide whether you ran on unicode or not. In ObjectStudio 8 you have one VM and one image. (So e.g. in Czech using cp+1251 instead of western cp_1252 is an easier choice.) Calling Smalltalk from C can by done by PSend or by ASend (whenever I'm idle, do it).

The VisualWorks process model has many processes running at different priorities and processes are objects. Shared Queues move events to the process. It is implemented in C. Every 30ms a thread looks to see whether a windows event is there. It sets a flag if there is and every loop looks at every jump back and every primitive return.

In OS, all execution is called by the Windows Event Loop, doing each one in turn. Only when there are none are all ASends executed from the idle queue, and newly arriving events must wait till it is exhausted. It is implemented in C++

The goal was not to modify the applications. They decided to move the event loop to Smalltalk: Georg saw this as a key enabling decision. His slides showed the code. The code uses class instance variables instead of class variables because the former are faster. The handler just carries on if there is an event it cannot handle; these stay in the queue and get handled later. Thus OS behaviour is provided by VW. The event handler hierarchy lets the Smalltalk talk directly to the OS with no intervening compatibility layers. You must call gcMalloc, peek for the event and so on. (It really looks like C written in Smalltalk but that is to be expected as that is what you are doing.) The code on Georg's slides was easy to read and relate to the process model he had described. Moving things from C to Smalltalk makes things available to understand, to review, to alter, to subclass, etc.

User Interrupts work as in VW, better than old OS.

Class proxies load a class if a message is sent to it. They are shared variables in Root.ObjectStudio.Global and put the classes they load in Root.ObjectStudio.

In classic OS, Smalltalk execution is slow and calling C is fast. In VW, smalltalk execution is fast and calling C is expensive. So they moved many things back to Smalltalk in OS 8 and gained speed. VisualWorks .st file-in is slow: of 1000 seconds, 900 are spent refreshing the Refactoring Browser, 90 secs are spent repeatedly relinking the system, 9 seconds are used to flush the disk for the Changes file and 1 second is compiling. So if you forgot to close the RB before loading, do CTL-Y, close the RB and proceed. You can fix the Undeclared system to avoid the 90 seconds. You can work around the flush. With all these speed-ups in place, you can file-in .st faster than parcels if the parcel is small (larger parcels still beat file-in but it is a similar order of magnitude over a wide range).

OS customers now get Store as an alternative to Smalltalk Archives. The get CodeCritic and other tools. They had their first customer in April and now have several. Porting needs to look at all a customer's base system

modifications. This is of course where most porting compatibilities arise. Immutability usually raises a few issues. The GH Undeclared navigator should look like a few empty panes but usually looks quite full after the first raw port (I recommend the GH Undeclared navigator when keys in the standard Undeclared browser report no references but do not purge.)

So what is the message for an ESUG audience. You can integrate very different Smalltalk systems. Squeak and VW are more similar than OS and VW. VSE and VW?

Q. (Niall) How do you handle duplicate bindings? They do not import Smalltalk.*. Instead they import long lists of chosen specifics such that there are no duplicate bindings.

Q. Business case? Bob answered. They use OS and find the gain of the better tools in VW a huge win. The other business case is wanting facilities that VW offers. (Georg has given this talk many times with a discussion of the business case but for ESUG he had a different message to make.)

Q. Performance? 25 - 30 times faster for trivial fibbonacci calculations and suchlike. In real applications, it can be 20-30% slower to open windows.

Q.(Gorgio) where do you manage class files? The programmer can choose; load and put in Store, retain in Store, or you can continue with .cls files.

Invited Talk: Erlang, Joe Armstrong, Ericsson AB
Joe is a salesman selling mindshare, a business in which it is not easy to measure your volume and profits. Let's start with Smalltalk. It's one of the language's he likes - but it got something wrong.

Erlang started in 1986. At that time, he interpreted his task as "Let's program telephony in as many languages as possible to see how easy it is." He did it in Smalltalk, getting coffee while 1986 Smalltalk GCed. He was the first customer in Sweden (Ericsson) for a tectronics Smalltalk machine but it took 2 months and he spent time writing algebra to explain what he was doing., One day, someone saw it and said, "That's a Prolog program". He found Prolog could run his algebra. So when his expensive Smalltalk machine arrived he was fixated on Prolog and did not unpack it. Eventually he gave it to someone else who prototyped Objectory (which he does not much like; he should have kept it).

Now he had an algebra but it only had one process so he decided to add concurrency to Prolog. Preparing this talk, he looked on the Squeak list and found a post from Alan Kay, "Hi folks, Erlang is worth looking at". So this talk is about why Alan might have said that. Joe's claim is that "Erlang is Smalltalk as Alan Kay would have wanted it to be."

Alan said that eliminating state metaphors from a system was part of the reason for encapsulation. Many programming languages do not support processes well because you cannot create lots of them, the reason being they belong to the operating system, not to the language, with a thin layer

to hide this fact, whereas the language is about creating lots of things in the language and so that's what they understand.

- In Smalltalk, processes belong to the language. Elsewhere, if one OS has preemptive scheduling and another does not, your processes will work differently.
- Suppose someone told you they had a great new OO languages with just a few trivial issues: don't create more than 2000 objects, keep them in a reuse pool, reference count them, etc.; you would be unimpressed. Joe is unimpressed when processes have these constraints.

To make a fault tolerant system you need at least two computers. (He's found this is not always obvious to everyone.) Thus fault-tolerant programming means distributed computing. So let's simplify distributed computing by having

- no sharing
- pure message passing
- no locks

If X shares the data structure that Y needs to recover from an error and X crashes then Y cannot recover, so we must copy everything. This is against the high church of distributed computing who believe in mutexes and critical regions. You can make these work but it is extremely difficult. In physics, two objects in different places cannot share anything. Distributed computing is far simpler if it has the same rule.

He wants to do Concurrency-Oriented Programming. "Concurrency is a program structuring principle" Tony Hoare. You have large numbers of processes, no sharing of data, location transparency and pure message passing. COP design:

- find the concurrent operations: what is the granularity of modelling appropriate to the problem
- find the message channels
- what messages for which channels

Thus you get the protocols and write the code. And if you make it isomorphic to the problem it will be easy(ier) to do.

(Multi-core CPUs then run these programs much faster; God must exist since they turned up just when we needed them. Joe's history slide ended with 2006 Multi-core Erlang.)

The two most significant things in Erlang's history were totally unplanned. In 1995 a big Ericsson project collapsed and they needed a new technology to replace it. Erlang was adopted as a drowning man clutches at a straw. Suddenly, 6-7 Erlangers became 400+ Erlangers. There is a huge different between a prototype one guy does and what a huge organisation does.

In 1998 Ericsson banned Erlang. They had nothing against it but it was not Java or C++. Joe was not very pleased about this. He tried to explain that

telecoms software was not going to be written by applets in Browsers where Java was design but ... . (At much the same time, I was trying to explain much the same thing to the telecoms company I then worked for with an equal lack of success.) But it actually it turned out to be good. It was like trying to stop a plant by smashing it; it reseeds and spreads. Erlang became open source. They formed a company in 1998 (a very good time) and sold it just before the IT crash for $150 million; very good! None of this was planned.

How do you correct hardware errors? By replicating the hardware. How do you fix software failures? Not by running two copies of the same software.

Systems are made of black boxes. Black boxes execute concurrently and communicate; failure in one of them must not crash another. Thus we cannot have synchronous communication. If you want confirmation, put it in the protocol. (FTP has round trip confirm; it should not need it as the lower levels do checksum but people don't trust that.)

Telephone exchanges handle hundreds of thousands of subscribers, each with 6 concurrent processes, in real-time ('soft' real-time; 'hard' real-time means things like plane-control software). Exchanges have very high availability demands and must never shut down ("Sorry, all our phones will be out from 20:00 to 02:00 while we upgrade. If you need the fire brigade, ambulance or police at that time, try shouting loudly." is *not* allowable).

To build telephone systems you need fine-grained massive concurrency. You need error encapsulation: errors must not propagate. Faults must be detected, analysed and so recovered from. You need to do live code upgrade. (Erlang can have different versions running concurrently within one system.) You need stable storage for crash recovery.

Hardware components operate concurrently in isolation and communicate by passing messages. Software can do the same. You would be surprised if an error in your chair affected your computer but you are not surprised when an error in one Windows process affects another. One program must *not* be able to crash another. Processes should share *nothing*. When people say it is not efficient he replies that running on parallel computers is more efficient. Message passing must be the *only* way to share data.

Joe then gave his 11 minute Erlang course (as inspired by the 'Hamlet in 14 minutes' and so on of Tom Stoppard and the Electric Shakespeare company; I could recite 'Doggs Hamlet and Cahouts Macbeth' after hearing it but was not up to doing as much for the 11 minute course - Erlang is not quite as memorable as the bard of Avon :-). Hence see the talk slides in conjunction with my explanatory notes below.

Erlang's arithmetic operator precedence is like everyone else's, i.e. not like Smalltalk's. It has standard curly-brackets, uses ; not . for statements and has round brackets around parameters. Assignment syntax is non-standard, with -> for assignment and ++ for append to.

Concurrency: spawning a function returns a process running that function: `pid -> spawn(fun....` Distribution: spawn a function at a node: `spawn(Fun@Node)`, `alive(Node)`, `not_alive(Node)`.

Fault-tolerance: `catch( ...) of (abnormal_case1, Y) -> ....` It looks like ordinary throw catch but handling errors in Erlang has a different philosophy. In most languages you handle the error near where you made it. But if a computer crashes, another computer will have to handle it. Joe does not want to rewrite code when moving from single to multi-computer systems. So you do not handle an error in the process in which it occurred but in another process. Handlers catch to another process via a link that says who will handle.

```
process_flag(trap_exit, ...
  Pid = spawn_link(fun() -> ... end)
  receive {'EXIT', Pid, Why} -> end
```

Erlang also has bit syntax to help write protocol classes by doing pattern matching over bit fields. See Joe's slides for the full eleven minutes course.

He showed a server that doubled numbers and a client that asked it to. Then he extracted the server semantics to a function and made the client send the function that doubles numbers. Now we have a universal server that is a higher-order function; it takes functions as parameters. We can also send a new function to the old server to change it: dynamic code change. (There are hundreds of lines of code in the generic server to handle all possible horrible error cases.)

Errors can be handled in a supervision hierarchy; if things go wrong, try to do something simpler.

There is an ATM switch implemented in two million lines of Erlang. It runs BT switches and is the most reliable product Ericsson ever released. Alteon (Nortel)'s Secure Shell Accelerator uses Erlang. Teba Bank (South Africa) have a credit card system in Erlang. Other users are T-Mobile (UK), Kreditor (Sweden) and jabber.org.

UBF: contracts are an attempt to formalise the idea of black boxes. There is a protocol checker.

Q. Can you implement black boxes in Smalltalk? Yes. There is a semantic mismatch between data structures in Smalltalk and in Erlang so you define a UBF to map efficiently. People in Erlang have noted that Smalltalk is good for GUIs and suchlike where Erlang is terrible so it would be a good union. Performance needs you to send small messages causing large computations returning small results, not the other way around.

Q. Implementation? A group at Upsalla are using all the compilation techniques known.

(Joe has a blog on which you can read more about all this.)

GemStone/S, Norm Green, GemStone Systems Inc.
Norm is director of engineering at GemStone. He overviewed GemStone and then concentrated on the GemStone64 release. GemStone is a privately owned company: 65% by the investor and 35% by the employees. They have some people who have been with the company for 25 years and more that have been there for 20 years. A slide showed the Logos of GemStone's many customers. A major area is shipping: 20% of the world's containers are now handled in GemStone.

GemStone is three things and one. First and foremost it is an object-oriented database. Secondly, it is a Smalltalk dialect. It has no significant UI component; client Smalltalks usually provide that. GemStone Smalltalk concentrates on collection classes. Applications have been written wholly in GemStone Smalltalk, though that is the less usual way. Thirdly, it connects multiple clients (different Smalltalk dialects, Java and C clients) concurrently.

Some GemStone customers were needing more than the maximum object count (1 billion) and maximum address space of 32 bit. A 2Gb page cache can hold 40% of a 5Gb database so performance is good. Make your database 100Gb and your shared page cache is now 2%; performance is not so good. Thus they went to 64 bit.

In phase 1, GemStone 64 bit handled the address space. It wanted to reduce the persistent garbage and double the object limit just to give affected customers immediate relief. The current release is 1.1.5. It supports very large shared page caches, up to 16 terabits. They do cache warming; some parallel sessions load data into shared cache on startup rather than letting users warm the cache and see slower performance while this happens. The cache can be warmed in 2-3 minutes even for very large databases.

The old versions of GemStone were designed when memory was costly. Large objects (> 8k) would always go to disk whether they were garbage or not. In GemStone64, only referenced objects go to disk. As a result, their largest customer saw a dramatic reduction in the amount of committed garbage that they collected weekly.

Symbol management now has a dedicated session managing AllSymbols. They now say you *cannot* create duplicate symbols. They also have faster symbol lookup (it was a performance bottleneck).

They can do online backups and parallel GCs (up to 255) simultaneous with production use.

They rewrote the object manager. It now has more in common with their Java VM. As in that, you must specify a maximum amount of memory you may consume. He showed some performance data (Georg Heeg asked why factorial 100 was faster than factorial 15? "Good question; maybe I got the rows the wrong way round.")

All this was done in 15 months with a relatively small team.

Users include COSCO (Shanghai, shipping), ICE (Atlanta, energy trading) and Northwater (Toronto, financial). Dutch agricultural-data-analysis user LEI is moving soon. (Adriaan: “It is happening today.”)

In phase 2, they handled the object limit. Their goal was to make GemStone 64-2 the last major database remap for a long time. They improved the VM, following on from increasing the SmallInteger range (from 2^{29} up to 2^{60}). They analysed bytecodes and added new ones for speed. Every time they dispatch a bytecode, they go through the main service loop (bytecode are assembler, no native code) so doing more in each bytecode makes it faster.

They added SmallDouble 5.0e-39 to 6.0e+38. Many emails were exchanged with Eliot to ensure this was compatible with VW, a goal he trusts was achieved.

Indexes had problems so they rewrote it underneath the same semantics and protocol (curly braces and dot-referencing).

GemStone 64-2 was released at the end of March 2006. OOCL have been using it in production since end-July. They have a 360 Gb repository with 1800 concurrent sessions. When they moved from GemStone 64-1 to GemStone 64-2 they saw the VM become faster 30-50% but increased network load due to sending 64 bits, not 32 (network compression keeps the increase to 20%).

VW treats 64 bits as LargeIntegers so the VW client is slower. They are working on treating them as byte arrays but they are not sure they will ever get back to the speed of SmallIntegers in VW. The improved GemBuilder will be out in October.

Q. 64 bit VW? No clients use 64 bit VW today. There is talk about supporting the 64 bit VW client at some point.

In three weeks there will be a new GemStone 64-2 release.

In G64, all objects on the interpreter’s stack must be faulted in, even if only referenced by identity; this is a change, so

```
aLargeArray do:
  [:each | each == someObject ifTrue: [^each]]
```

has different faulting behaviour from before. You must change the code to use `indexOfIdentical:` instead of == to retain the old behaviour (of course if you were doing = instead of == you would have had the slower faulting behaviour anyway.) Other methods to avoid unnecessary object faulting have been added to SequenceableCollection, OrderedCollection and IdentityBag.

Future plans include multi-threaded GC (i.e. multithreaded finding of garbage; the earlier reference was to multithreaded reclaiming of garbage); for a large database, identifying the garbage can take a very long time (days, weeks). They can make indexes on large Rc collections less subject

to conflicts. They want to make the GCI layer thread safe; multiple logins from the same client image looks multi-threaded but the underlying GCI is single-threaded and occasionally this causes problems. They want to implement continuations to support Seaside; some people are looking at it.

Q. Windows server support? We support the clients but the engineers despise Windows so it is hard to make them do it. If there is a clear business case we will do it for next Windows release, not by going back and doing it for XP.

Q. VW and GemStone 64 integration? We are looking to learn from what Eliot has done and there may be some porting if appropriate.

Research Track

UbiquiTalk: Ubiquitous Computing, Noury Bouraqadi and Michael Piel, Mines de Douai

UbiquiTalk is a framework for middleware to help small devices communicate with others. It aims to require zero network configuration or deployment effort. Devices will join and leave the network unpredictably. The network may be of any kind: ethernet, bluetooth, wifi, etc.

A host is a device with the UbiquiTalk runtime and a network interface. A service is an object on a host that can be run remotely. There is a presence notification loop 'host3@ipAddr:ipPort' which discovers hosts automatically and populates the service registry with their services. Deployment can be done on-demand at runtime.

A service offers a GUI and a configuration and usage constraints (e.g. maximum number of simultaneous users permitted) as well as its core function. The service registry keys by white pages (i.e. by name) and by yellow pages (i.e. by description). (Kat mentioned that the owners of the Yellow pages name guard it; Sun had to rename its yellow pages to NIS.) A Service defines the application entry point, whether it is reactive or proactive, the provider part, the client part (usually the service GUI) and the administration part.

The GUI targets a range of devices. The slides show the two layouts for laptop/PC and PDA, each providing all features.

So far they have implemented cross-platform cut and paste, a chat conference, find-geographically-nearest printing (the location property is set from a GPS or by hand). They also have remote administration and remote control services.

Some months ago they started working with a Chemistry consortium (some companies, some academics). Over the last 15 years, each partner in the consortium has developed standalone software implementations that do those chemical analyses that interest them. UbiquiTalk now lets one partner get analyses from another in cases where the remote standalone application is better at a given analysis.

A new project is Robotic Rescue! An indeterminate fleet of robots will cooperate in a hostile environment. The application area is in the third world so it is essential the robots be cheap.

Their current partners are mainly non-Smalltalkers; they would welcome Smalltalkers with project proposals suiting UbiquiTalk, i.e. needing P2P between unanticipated hosts. They want to harden their implementation and add support for automatic deployment. Ideally, they would refactor the system until everything was a service.

Q. PDA UI? They had problems with Morphic, so built a GUI developer in Morphic

Object History, Pluquet Frederic, University Libre Brussels
The aim is to save the state of objects. Typical uses are CVS, undo/redo in text editor, SmallWiki recovery and debuggers (timeless breakpoints). Another idea is to combine with backtracking in SOUL.

They have built a prototype in VW7.4 and Squeak 3.9. They have a trace recorder and a text editor undo/redo-er. Recently they realised that collaboration with ByteSurgeon would construct a really efficient trace recorder.

Q. (Bernard) When do you drop unwanted state? There is protocol for clearing the whole image history or only subparts.

Q. (Bernard and Alexandre) Record exact time for each state? No, we only have the order, so we can roll back or forward but by operation, not by time. We may add time.

RealTalk, Alexandre Bergel, Siobhan Clarke, Distributed System Group, Trinity College Dublin
RealTalk is a programming language for a wireless embedded sensor network. Embedded sensor systems are hard to program. They are meant to run uninterrupted for months or years. C prevents such components being maintained easily. Sensors are sampled asynchronously by emitting and receiving events so the system is not synchronously updated. The CPU requests data from a sensor and gets an answer, during which the usage of the CPU is the critical resource. Between requests, the speed of data passing is critical.

RealTalk aims to make programming such devices easier. Some RealTalk:

```
RTObject subclass: #Counter
  ...
Counter compile: #triggeredMethod
  value := value + 1.
  leds display: value.
```

Problem of associating devices in real time:

```
Counter composeWith:
  { #leds -> Leds. #timer -> Timer }
```

He showed code to emit a sound at each triggeredMethod increment.

```
... self incAndDisplay.
  self isRedOn
    ifTrue: [sounder start]
    ifFalse: [sounder stop]
```

Then the more interesting example of two senors interacting: read two sensors and emit result.

```
  value1 := lightSensor readValue.
  value2 := soundSensor readValue.
  radio emit: (value1 + value2).
```

RealTalk is built on TinyOS and NesC. It runs in devices with 4k of RAM.

Q(Niall) Comparison with Lars OOVM? This is just generating C code; there is no VM. It lets Smalltalkers program devices in the way they prefer.

Spyware-ridden Software Development, Romain Robbes, University of Lugano

Romain has written SpyWare, a research prototype for his PhD. Real-world software must change or become slowly less and less useful. Most of the cost of a program is maintenance. Romain reviewed the process of creating software. As the rate of change increases, there is a tendency to lose reference points for understanding what was done and why.

People may extract data from configuration management versions of the system to see metrics and understand trends. But the version system was not built to do this task and it usually loses information. It also has the wrong information and too much information. The hyper-cautious programmer saves every five minutes. In five minutes, they may extract method, rename method and create accessors for variables. The versioning system would say that many lines have been changed but the programmer would know that three pure refactorings have been done.

SpyWare captures changes, not versions, using the IDE, not the repository. SpyWare is a plugin to the Squeak IDE that captures high-level change data to a changes repository. Compilation triggers SpyWare which resolves the context (new method?, by whom and when?...) and records meaningful change information. SpyWare can monitor multiple projects; he tested monitoring 9 student projects over a week.

He demoed the simple UI. He is working on a standard-browser-like display of changes. A graph of number of methods and number of classes over time has a pop-up giving details of what specific method or class changed when the mouse was moved over a given point of a given line. Other lines showed average method size.

Future work may include tools to focus on when bugs were introduced and to warn of potential merge conflicts faster. Performance and space-usage need to be improved. Meanwhile, he would be glad if people used it.

Q. Capture refactorings rather than just changes? He is working on this.

Q. VW? Squeak only but it could certainly be ported.

Other Discussions

Pollock will have a GUI builder this winter. Pollock will be supported from Jan 2008 (so hardened for use by late 2007). Currently, its individual widgets are robust; multi-threaded interaction of many concurrent panes is being addressed. Jim will release BottomFeeder on Pollock.

http://csl.ensm-douai.fr/noury/SmalltalkJobAndInternshipOffers gets 2 offers each month and has placed two people so far. If you get a place through it please let Noury know. If you want to post an offer on it, email him at bouraqadi at ensm-douai dot com.

Georg and I discussed estimating porting projects. Georg does not have data for OS 8 yet but for porting from older dialects to VW7, Heeg's standard first-estimate formula is to count each system-changing method as equal to 12 application methods when estimating system size. (I asked about shape-changes to system classes. They are too rare to be part of the formula; each would be a one-off addition to the estimate.)

Felix Madrid is now working for eXcept (Claus Gittinger). Clients include:

- ETAS (multinational) use Smalltalk in their Inca system
- Bosh are porting a Smalltalk system for configuring cars from Digitalk to Smalltalk/X

eXcept are also building a product for testing: users define workflow, etc.

Cosmocows are expanding.

IViews have new customers in document publishing and other areas; their dictionary application, whose size once stressed the system, is now a 'small' application compared to their current work.

Conclusions

My eighth ESUG: good to see increasing Smalltalk commercial activity.

- Impara's Squeak products look impressively commercial and professional.
- Erlang looks it has some good distribution ideas but I think I'll stick with Smalltalk for programming.
- Moose' Mondrian programmable visualisations seem quick and flexible.