Contents

Copy Dates

C Vu 17.6: November 1st 2005

C Vu 17.7: January 1st 2006

Contact Information:

Editorial: Paul Johnson
77 Station Road, Haydock,
St Helens,
Merseyside, WA11 0JL
`cvu@accu.org`

Advertising: Thaddaeus Frogley
`ads@accu.org`

Treasurer: Stewart Brodie
29 Campkin Road,
Cambridge, CB4 2NL
`treasurer@accu.org`

ACCU Chair: Ewan Milne
0117 942 7746
`chair@accu.org`

Secretary: Alan Bellingham
01763 248259
`secretary@accu.org`

Membership Secretary: David Hodge
01424 219 807
`membership@accu.org`

Cover Art: **Alan Lenton**
Repro: **Parchment (Oxford) Ltd**
Print: **Parchment (Oxford) Ltd**
Distribution: **Able Types (Oxford) Ltd**

Membership fees and how to join:

Basic (C Vu only): £25

Full (C Vu and Overload): £35

Corporate: £120

Students: half normal rate

ISDF fee (optional) to support Standards work: £21

There are 6 issues of each journal produced every year.

Join on the web at `www.accu.org` with a debit/credit card, T/Polo shirts available.

Want to use cheque and post - email `membership@accu.org` for an application form.

Any questions - just email `membership@accu.org`

Reports & Opinions

Editorial

Paul F. Johnson <cvu@accu.org>

A Disaster Waiting to Happen

It's Sunday night. The compiler is happily running in the background and all seems right in the world. The phone rings and all hell breaks loose. I'm not talking about the latest science fiction blockbuster here either. A friend's computer has gone belly up and the world's most fantastic article has vanished without trace.

Now, if this had happened a couple of weeks earlier, the problem would not be so significant. However, with less than a week until the editorial deadline, it poses a problem. Now, I could just run a previous article or ring a friend to see if they can knock something together, but that would be unfair – I mean, how often does someone phone you up to say "*Hey, can you bash out 2000+ words for the end of the week for me and make it good?*" or I could write something myself – which in itself is problematic as I only have a finite amount of time to do unimportant things like actually living.

I suppose this really is the problem with any form of publication which is reliant on submissions from the readership rather than having an established team of writers. Looking back on the issues I've edited, quite a number of articles are from non-ACCU members. Yep, you read that right. I'm calling in favours from various mailing lists and friends in order to give ACCU members a damned fine read every other month. However, there are only so many favours and times you can ask for submissions and they are starting to dry up – rapidly.

This edition has been the hardest so far in getting quality articles. It may be that you have never written for a magazine before and you find it daunting. Don't worry – I don't bite and offer friendly advice on whipping submitted material into shape. It may be that you don't have anything interesting to say. I find that hard to believe given the diversity of members and the range of experience there is. We have company directors to students to freelance programmers to people who are interested but don't really know that much. What I would love to see are four "strands"

1. C++ for beginners (*this could also be C#, Java, Python or C*)
2 Project management
3. Defensive programming
4. Libraries

Other than (4), I can't recall ever seeing the other three in C Vu. There must be project managers out there, and those who understand defensive programming techniques or even someone who can deliver a basic set of tutorials on a language. In a recent conversation with a couple of prominent academics in UK universities, I asked them what they thought of both C Vu and Overload. I was amazed when they replied with a shrug of the shoulder as they had never heard of either the ACCU or its publications. Luckily, I had a couple of copies as PDF files and they read them over. Other than a few niggles, they did comment that there was a lack of anything they could use in techniques, libraries or for those coming into universities where the student had never used a language before. Sure, they looked good, but it wasn't something they could sell.

As a responsible editor (and someone who wants C Vu and Overload to be as widely read as possible), it is important that these criticisms are addressed – but to do that requires a team effort. If you're happy to keep things the way they are where we are writing for us, then fair enough. If though you want to see the magazines more widely read, then you know what you have to do.

Come on folks – I know you're out there. I can hear you breathe!!!!

Where Did They Go?

Over time, people become disaffected and stop their subscriptions. There is nothing much that can be done about that, but it would be interesting to find out why they decided to leave the fold. If you know anyone who falls into that category and who wouldn't mind answering a couple of questions, then please let me know so I can contact them.

I am interested in why people left. While there is always a small number which can be termed under the unfortunate title of "natural wastage", I would be interested in seeing if there is anything that can be done to entice them back in.

Conference Report

You've probably noticed that over the past couple of editions, I've been giving space to up and coming conferences.

One I didn't mention was the annual DNSCon meeting. As the name suggests, the conference was primarily concerned with network and software security, but also covered other interesting aspects, such as developments in surveillance techniques. From my point of view, the talks on how to use a buffer overrun to compromise a remote machine (demonstrated on a Win2000 laptop running VMWare with both Linux and BSD box all on the same machine!) and the use of fluorescent lights to bug rooms were of the greatest interest.

It was a somewhat strange conference which lasted one day in Blackpool. Due to very poor weather, the "fling an AOL CD" competition and prize sandcastle competition was cancelled as was the "most radioactive sandcastle" competition (the conference was in Blackpool and was won last time by very dubious means!).

One aspect which did puzzle me though. As you are possibly aware, it is possible to read what is on a PC's monitor by differences in magnetic patterns. Nothing new in that. What was good though was that a piece was presented on the problems of monitoring a TFT monitor. Given the amount of electromagnetic radiation of the correct wavelength given off, it is damned near impossible to see what is on one of those screens. It can be done (it was revealed outside of the conference and over a number of pints that a UK security body had demonstrated it live), but how? I thought at 4am of the day of the conference I had it, but though very bleary eyes, spotted the mistake in my maths. Oh well, it only took me 4 hours worth of sleep!

On a sadder note (in one respect), it was announced that this would be the final conference at Blackpool – mainly as it is a pain to get to and that the 2006 conference would be in central Manchester around Christmas. Yay!

Z88DK & Pud Pud

You may remember a few editions back that I was complaining that programming was no longer simple and that for us to get anything now requires large manuals, a compiler and lots of time to debug. I then later just about retracted it in a small piece which answered a number of points from the ACCU general list. Well, I'm resurrecting the subject again. Why? Because of a Z80 cross compiler and an AVI of a BBC programme called "Commercial Breaks" (broadcast in 1984).

The TV programme was a strange piece of TV history as while filming, the Liverpool based software company Imagine went under and the camera crew was there. Other than a historical point of view, it's not that significant. What was important was that there was a 17 year old who had been programming for about a year on a Spectrum and over 7 weeks had developed a multi-level game called Pud Pud. This was a high resolution game, with sound (well, burps), colour an addictive level of play – all of which was in Z80 machine code.

How many people, in the space of a year, can not only learn machine code and then in 7 week create a big selling game? Alright, the processor was an 8 bit Z80 and there was no time frame given for the number of hours spent in front of a TV set learning, but given the age of the chap, I'd guess at 3 hours a night and 6 of a weekend (12 hours in total).

Okay, you can cover a fair amount in 3 hours a night, especially with a simple 8 bit processor. I'm sure though that you'll all accept the achievement was considerable.

What has this got to do with Z88DK?

Z88DK is a Z80 cross compiler which allows you to target just about any Z80 based machine (from a ZX81 to a TI89 calculator to even embedded Z80 based systems). Source code is easy to read and understand – take the following for example:

```
#include <graphics.h>
#include <stdio.h>
#include <stdlib.h>
struct window mine;
/* Window structure */
int main()
{
  int j,i;
    clg();
/* Draw a series of
 * concentric circles in the
 * centre of the screen
 * these go off the screen
 * but don't generate an
 * error - very cool! */
    for (i=90 ; i!=0; i--)
    {
      circle(128,96,i,1);
      if (i < 25 ) i--;
    }
    draw(0,0,255,63);
    /* Draw a diamond - weak,
     * but it demonstrates
     * relative drawing! */
    plot(200,32);
    drawr(10,10);
    drawr(10,-10);
    drawr(-10,-10);
    drawr(-10,10);
}
```

It's almost BASIC! Compilation is a case of running `zcc` followed by a flag for the target machine. Very simple and very effective. Perhaps it's a good way to go to start people learning to program again. Sure, it's not as simple as having a built in interpreter, but it's a start.

Anyway, that's enough for this issue. As always, your views and comments are always welcome. Please feel free to contact me .

Paul F. Johnson

Late News!

I don't normally receive news at 7 minutes past midnight on the day after editorial copy is due, but this one is somewhat useful to those who write software in C# that needs to interface with SQL or PostgreSQL databases. I wouldn't normally give so much room (I do get quite a number of releases about how wonderful such and such is), but this is different as it is a cross platform approach which is offering everyone the opportunity to participate without cost and learn from experts to boot.

ODBMS.ORG Launches Educational Portal on Object Databases

db4objects, providers of the leading object database for Java and .NET, today announced the launch of ODBMS.ORG, the Internet's most up-to-date educational and research portal on object database technology. The initiative was made possible through contributions of a group of high-profile software experts, lead by Prof. Roberto Zicari. It is the first of its kind in nearly two decades, since first-generation object-oriented databases emerged in the early 1990s and subsequently fell dormant.

The open source community has created a new wave of enthusiasm that's now fueling the rapid growth of second-generation, native ODBMSs and demand for appropriate education. The new portal is designed to meet this fast-growing need for educational and research resources focusing on object database technology and the integration of object-oriented programming and databases.

ODBMS Growth Boom

Object databases (ODBMS) have long been recognized as a solution to one of the biggest dilemmas in modern object-oriented programming (OOP): the object-relational (OR) impedance mismatch. Now that OOP languages like Java and .NET are finally becoming mainstream, this problem rests at the heart of information technology.

Thus object databases are increasingly established as a complement to (not a replacement for) relational databases for efficient resolution of the OR mismatch. ODBMSs are flourishing as embeddable persistence solutions in devices, on clients, in packaged software, in real-time control systems, and to power websites.

Expert Resources

The ODBMS.ORG portal features open source software, lecture notes, tutorials, papers and other resources for free download. It is complemented by listings of relevant books and vendors to provide a comprehensive and up-to-date overview of available resources on object database technology.

The portal's editor, Roberto Zicari, is Professor of Database and Information Systems at Frankfurt University and representative of the Object Management Group (OMG) in Europe. His interest in object databases dates back to his work at the IBM Research Center in Almaden, CA, in the mid ?80s, when he helped craft the definition of an extension of the relational data model to accommodate complex data structures. In 1989, he joined the design team of the Gip Altair project in Paris, later to become O2, one of the world's first object database products.

The Expert Section contains exclusive contributions from internationally recognized experts including Scott Ambler, Michael Blaha, William Cook, and Carl Rosenberger.

Scott Ambler is a consultant with Ontario-based Ambysoft and thought-leader of the widely recognized Agile Modeling (AM), Agile Data (AD), and Enterprise Unified Process (EUP) methodologies.

William Cook, professor at the University of Texas, and Carl Rosenberger, chief software architect at db4objects, have contributed their ground-breaking joint paper on Native Queries (NQ), which discusses the use of programming languages like Java or .NET to express database queries that are 100% typesafe, 100% refactorable and 100% object-oriented. Native queries are poised to become the unifying standard for object-oriented queries in the same way that SQL has standardized the query interface for relational databases - replacing earlier, non-native attempts such as ODMG and JDO.

Michael Blaha, co-inventor of UML and co-author of the seminal book *Object-Oriented Modeling and Design with UML* (with James R. Rumbaugh), has contributed a new paper on "The Dilemma of Encapsulation Versus Query Optimization."

View from the Chair

Ewan Milne `<chair@accu.org>`

This edition's View from the Chair is a departure from the usual cut and thrust, but has a more sombre note as it is my unfortunate duty to have to report the untimely passing of Christopher Hill (1955 – 2005), a long-standing ACCU member and book reviews collator, who died suddenly of a stroke in July.

Christopher had many interests in his life, and enjoyed a wide-ranging career. An electrical engineer by training, he worked for Marconi Radar Systems before becoming Computer Manager for the company that organized evangelist Billy Graham's mission to London in 1989. After this he joined a theological college for a time, with the intention of taking Holy Orders. This, however, was not to be and he returned to development work. He joined ACCU in 1997, and from 2000 ran his own consulting company. Two years ago he responded to requests for a volunteer to collate the book reviews for C Vu and the website, and carried out the task with admirable efficiency and conscientiousness. In recent years Christopher was also a dedicated member of the BSI C++ panel, and his contribution to both the panel and the ACCU will be sadly missed.

Ewan Milne

Membership report

David Hodge `<membership@accu.org>`

The peak period for renewals has just gone by. Some members said that they didn't receive the first reminder which was sent out by email in July, and only responded to the second reminder in early September.

If you keep an eye on the journals shipping slip you will find that the expiry date of your membership is always listed. All journals previous to this have had a label printed by me, and then stuck on by our mailing organisation (AbleTypes in Oxford). From this issue I am supplying a file of addresses to AbleTypes who will then print the addresses themselves. If you spot any problems that this new process creates, please let me know.

David Hodge

Secretary's Report

Alan Bellingham
`<secretary@accu.org>`

There is always a long gap between the first and second committee meetings of the year. This is due to the presence of the summer holiday period: it's not hard to see that a meeting in mid August is not going to attract as many of the committee as desired. Hence this year's second meeting took place on the 17th of September, actually after the supposed deadline for this issue of CV u.

This time the venue was in Moseley, Birmingham in the English Midlands. Moseley is the home town of Jez Higgins, and he'd found a meeting room directly next door to a recommendable eating place, and somewhat further from the tornado alley that is where he actually lives. As is the custom, we met beforehand in the eaterie, and discussed some of the topics that would be addressed during the meeting.

At the meeting itself, we as-per-usual checked the minutes of the previous occasion. With the four month gap since that previous meeting, a good number of the actions had been cleared, which is good news. This was followed by the officers' reports which should pretty closely match what those officers have reported in this issue.

Next followed the first non-routine item on the agenda - how we should deal with the treasurer's role. As I mentioned last time, this post has historically been a problematic one, and we may not yet have settled it correctly. Long term attenders at the AGMs will know the perennial attempts of treasurer after treasurer to try to get more up-to-date with the approval of the accounts, and this does indicate that we are possibly approaching the whole issue in the wrong way. Also, despite his reservations, Stewart Brodie has yet to be fully allowed to depart his post.

This is not a satisfactory long term situation, but Jez Higgins has volunteered to try to put in place a new system, to cover this financial year onwards. For the moment, Stewart will remain one of the signatories (changing signatories on the organisation's bank accounts is a very long and drawn out processs), but we hope to reduce his immediate role to that of receiving invoices and sending out payments, while Jez takes over the acting Treasurer's role.

A late addition to the agenda was that of whether the ACCU should offer a special reduced membership rate to the unwaged. We currently have one for students. The constitution prevents us instituting a new membership type, but the committee's conclusion was that yes, the existing discount for students could be made a concessionary rate applicable to students, unwaged and other suitable candidates. In other words, we didn't have to invent a new membership type.

The final, and longest, item on the agenda was that of the website. This is a major project, second only to running the conferences, and it affects the public face of the ACCU. Some of you will have been wondering what has happened to it. What has been happening is that work has been taking place, on a separate devlopment server. What hasn't happened is a big bang - there's just too much to do all at once for that to work. There has been a bit of a lull in the development, which may be due to our taking our eyes off the ball and not riding the developers closely enough. However, we decided to stick with it, and progress should resume. We hope that by the time of the next meeting, the new site will be ready to be unveiled.

And with that, we wound up the meeting. The next one will take place (circumstances permitting) on the 19th November. Deadlines allowing, the report on that meeting should be in the next issue.

Alan Bellingham

Standards Report

Lois Goldthwaite <standards@accu.org>

C Vu has received several letters asking just what it is that 'the standards people' do. In response to popular demand, here are a few words about how things work and how you can become a part of them.

Everyone knows that the C and C++ language standards are written by committees of language experts from many nations. These committees, called Working Groups, meet twice a year for a week each time, to discuss any problems that have been discovered in the current standard and to work on new provisions for inclusion in a future revision of the document. Why?, you might ask. Isn't writing a standard a one-shot effort, after which everyone can pat themselves on the back and go home?

Well, ... no. Even defect-free standards (which these are not) need maintenance, just like software programs need maintenance. Neither one will wear out from use, as engine parts wear out, but the world changes, requirements change, and they need to evolve to address that. Remember Turbo Pascal? For those who don't, I will say only that it was a very nice tool for writing a single file of source code and compiling it into a .COM program to run in less than 64K of memory on a PC running MS DOS. (If that sentence makes no sense to you, find the oldest veteran programmer you know and ask for a translation.) I still hold Turbo Pascal in very high esteem, but I don't use it any more. Now that I need to write more ambitious programs, which not only process large amounts of data but are composed of thousands of lines of source code, I reach for tools which address those requirements. C++ in particular is intended for use in developing large systems.

The C and C++ Working Groups are jointly sponsored by ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission). The language experts are sponsored by their respective national standards body – in the UK it is BSI, in the US it is ANSI, in Germany it is DIN, and so on.

And who are these 'experts' who take part in the Working Groups? For the most part, they are working programmers like you and me. Yes, there are people like Bjarne Stroustrup, who invented C++, and at least half a dozen other people whose books are filling up your must-read shelf. But be aware that 'expert' in this context is just a title – that's what national standards bodies call their delegates to all international meetings. Anyone who volunteers to represent the UK at WG21 can become an officially accredited UK expert on C++. That plus 99p will get you a cup of tea at most company cafeterias.

Another thing to be aware of is that when the UK sends 'experts' to these meetings, they are not expected to carry the Brain of Britain burden all by themselves. They can call on the expertise of the entire BSI standards panel, which is (at latest count) approximately 50 people interested enough in C++ to follow the issues and express their opinion. The C++ panel meets face-to-face six times a year, but attendance is not compulsory and much of the discussion takes place via email reflector. During Working Group meetings, the delegates are in constant email communication with the folks back home, whose comments can then be inserted into the Working Group discussions in real time. (And mouthing these insightful comments makes those of us in attendance look really smart!)

One reason why the UK contributions are valued by WG21 is because our panel does consist of working programmers, whereas many of the committee members from other countries represent compiler vendors or library providers, not programmers in the trenches. Our youngest member is in university, our oldest is retired. There is no entrance exam to get accepted, and no charge to participate. (Unfortunately there is no remuneration either, other than the enjoyment of deeply technical discussions.) If you would like to join the C++ panel – even for a trial period – or just to visit a meeting in London (the final one this year is Nov 14) please do not be shy. Everyone is welcome. And it's a great opportunity to improve your C++ knowledge. Write to `standards@accu.org` for details.

There is also a BSI C panel. Neil Martin has recently been appointed convenor of this group. If you are interested in working on C issues, please write to standards@accu.org and I will put you in touch with Neil.

International standards can come from other sources, too. ECMA (originally the European Computer Manufacturers Association) is one such standards development organisation; various industry consortia, such as IETF, W3C, Oasis, and the Open Group, are some others. Sometimes their standards are adopted as ISO/IEC standards via a 'fast-track' process, in which national standards bodies vote on the final document. (In contrast, standards developed by a Working Group go through several votes on their drafts.)

The BSI standards panels have some input into the UK vote on adopting these standards, and sometimes can take a (limited, unfortunately) hand in their development. We currently are considering revised ECMA standards for C# and CLI (the standardese name for what is commonly known as the .Net infrastructure). And we also are soliciting comments on ECMA's standard for Eiffel in the .Net environment. Please get in touch (see address above) if you are interested in these topics.
Here are some URLs of standards organisations if you want to read further:

```
www.iso.ch
www.iec.ch
www.bsi-global.com
www.ietf.org/
www.w3c.org
www.oasis-open.org
www.opengroup.org/
```

Lisa Goldthwaite

Advertise in C Vu & Overload

80% of Readers Make Purchasing Decisions
or recommend products for their organisation.
Reasonable Rates. Discounts available to corporate members. Contact us for more information.

ads@accu.org

Dialogue

Student Code Critique Competition 36

Set and collated by Roger Orr
Prizes provided by Blackwells Bookshops & Addison-Wesley

Please note that participation in this competition is open to all members. The title reflects the fact that the code used is normally provided by a student as part of their course work.

This item is part of the Dialogue section of C Vu, which is intended to designate it as an item where reader interaction is particularly important. Readers' comments and criticisms of published entries are always welcome, as are possible samples.

Before We Start

Remember that you can get the current problem set in the ACCU website (`http://www.accu.org/journals/`). This is aimed at people living overseas who get the magazine much later than members in the UK and Europe.

Student Code Critique 35 Entries

Here is a C++ header file with a number of potential problems. Please critique the code to help the student identify the problems and to help them to provide some better solutions.
Note: the class `Report` is not shown. It contains a large amount of data, which can be explicitly deleted by a call to `ClearAll`.

```
// Reports : vector of reports
class Reports : public map<Data*, Report>
{
public:
  Reports() : nIndex(0) {}
  void ClearAll()
  {
    for (iterator iter=begin();
         iter != end(); iter++)
      (*iter).second.ClearAll();
  }

  Report& GetReport(int nReport)
  {
    int nSize = size();
    assert(nReport < nSize);
    if (nIndex == 0 || nIndex > nReport ||
                    nIndex >=(nSize-1))
    {
      iter = begin();
      nIndex = 0;
    }

    for (; iter != end(); iter++, nIndex++)

    {
      if (nIndex == nReport)
        return iter->second;
    }
    // keep compiler happy
    return *((Report*)0);
  }

protected:
  int      nIndex;
  iterator iter;
};
```

From Simon Farnsworth <accu@farnz.org.uk>

Firstly, I should apologise for the poor quality of this critique. I've only recently started using the STL, so my understanding is limited. I'm looking forward to seeing and learning from more expert critiques.

My immediate reaction upon seeing this code was confusion: the comment says that Reports is a `vector`, but the code indicates that it is a `map`. While both are containers, `map` is associative (in which you reference elements by a key), while `vector` elements are referenced by position. Given the presence of `GetReport(int nReport)`, I shall assume that the comment is right, and that the code is an attempt to induce `map` to behave like `vector`.

It is possible that the goal is to have a type that is a `map` in some contexts, and a `vector` in others. If this is the case here, the student would perhaps be better advised to write a container which includes a `map` and a `vector` as elements, and provides those operations on `map` and `vector` that are needed.

If this was intended as a vector, not a map, then changing the code to begin: `class Reports : public vector<Report>` removes the need for `GetReport()`; `vector` provides `operator[]` to retrieve items.

`GetReport()` is a mess: it permits `nReport` to be negative, but does not handle this case (`nReport` should probably be `unsigned`, not `int`). It also relies on the fact that `map` is a sorted associative container to ensure that a report's number does not change regularly; since there are no guarantees about the values of the keys used, this could lead to unexpected behaviour. Finally, it uses casts to trick the compiler into returning a bad reference. The comment implies that this was done to silence a compiler warning; in this case, the warning is generated since `GetReport()` can fail to return a report. However, `vector`'s `operator[]` will handle this case for us.

Finally, a couple of style notes: `(*iter).` is better written as `iter->`, and preincrement is generally preferred to postincrement, as the code is slightly simpler (although the optimizer should fix this for you).

Putting this all together results in the following code:

```
// An exception thrown whenever a requested
// report cannot be found

class NoSuchReport {
    private:
    unsigned report_number;

    public:
    NoSuchReport(unsigned nReport) :
      report_number(nReport) {}
    unsigned getReport()
    { return report_number; }
    // Compiler can safely autogenerate the
    // big four.
};

// Reports : vector of reports
class Reports : public vector<Report>
{
    void ClearAll()
    {
        for(iterator iter = begin();
            iter != end(); ++iter)
        {
            iter->second.ClearAll();
        }
    }
};
```

Should `GetReport` be needed for legacy reasons, it is easy enough to implement as follows:

```
Report &GetReport(unsigned nReport)
{
    return (*this)[nReport];
}
```

Commentary

This code looks relatively innocuous on a first reading, but repays further thought as there are many problems lurking within this relatively short piece of code.

To my mind the two most serious problems are (1) the confusion about whether this class contains a `map` or a `vector` and (2) ownership issues.

The first criticism concerns access to the objects of the class. Does `Reports` represent an array or an associative container? It may be both, but there is a fundamental asymmetry about the class as currently coded since all the associative behaviour comes form inheritance and the array like behaviour by an additional method. My inclination is to suspect the inheritance from `map` and I would prefer the class to contain a `map` as member data and provide methods to manipulate it.

There are several ownership issues. Firstly, the default copy constructor and copy assignment operator make it possible to copy the data structure. This is probably not behaviour the writer expected and is likely to be expensive; as the map contains the reports each one will be copied. There are two standard solutions to this particular issue: either provide a private definition of these two methods (but no implementation), or inherit from a non-copyable class such as `boost::noncopyable`.

The second ownership issue concerns the member data `nIndex` and `iter`. These members seem to be designed to optimise iteration over the elements of the class by keeping state between successive calls to `GetReport` but this is unreliable if the collection changes between calls to this function. I am also unhappy that member data like this is made `protected` rather than `private`. If the performance of the iteration actually is an issue (as determined by some performance measurements) then a better solution would be to provide an array-like iterator for the class following the usual STL pattern.

The third question I have about ownership is the onus seeming to be on the user of this class to call `ClearAll()` before the `Reports` object is destroyed. I would prefer this to be linked to the class's destructor for safety.

The final comment I would wish to raise with the writer of the code at this point would over the return of a `null` reference when `GetReport` fails. References never should be `null`, and returning one makes the code non-portable and well as unreliable. It would probably be better to indicate failure by throwing an exception, perhaps `std::out_of_range`, rather than twisting the language simply to remove the compiler warning.

The Winner of SCC 35

The editor's choice is:

Simon Farnsworth

Please email francis@robinton.demon.co.uk to arrange for your prize.

Student Code Critique 36
(Submissions to scc@accu.org by Nov 1st)

Here is a C program generating a couple of prime numbers as part of an exercise on encoding/decoding with public and private keys. There are two bugs with the program: it produces the same output each time it is run with one compiler (MSVC) and it loops forever with another (gcc). Please critique the code to help the student resolve both these problems with the algorithm. Additionally suggest any improvements to the coding style and point out any other issues with the algorithms used. You can also broaden the critique to include a C++ solution if this may assist the student with their original task.

```
#include <stdlib.h>
#include <stdio.h>
int main()
{

//need to generate number, then find out
//whether it is a prime, twice.
//then need to generate e and see if it is a
//factor of n.
  int i1, rem1, i2, rem2, i3, rem3, rem4;
  int p, q;
  int n, phi, e;
  //These are the two prime numbers output
  int m, d;

  i1 = 0;
  i2 = 0;
  i3 = 0;
  while(i1!=1)
  {
    p = 100 + 99*rand()/((double)RAND_MAX+1);
    //p is random number between 100 and 200.

    i1=p-1;
    rem1 = p%i1;

    //find out whether the number is prime
    while(rem1!=0)
    {
      i1--;
      rem1 = p%i1;
    }
  }

  while(i2!=1)
  {
    q = 100 + 99*rand()/((double)RAND_MAX+1);
    i2=q-1;
    rem2 = q%i2;
    while(rem2!=0)
    {
      i2--;
      rem2 = q%i2;
    }
  }

  n = p*q;
  phi = (p-1)*(q-1);
  // phi is the number of primes less than n!

  //e picked such that gcd(e, phi) = 1
  while(i3!=1)
  {
    e = phi*rand()/((double)RAND_MAX+1);
    //e is a random number between 0 and phi.

    i3=e;
    rem3 = phi%i3;
    rem4 = 1;

    //this loop finds the highest value of i3
    //which divides both numbers. It needs to
    // be 1, so they are relatively prime
    while(rem4!=0 )
    {
      i3--;
      rem3 = phi%i3;

      if(rem3==0)
        rem4 = e%i3;
    }
  }

  //the loop will find the value of m such
  //that e*d mod phi = 1.
  m = 0;

  while((e*d)%phi !=1)
  {
    d = (m*phi+1)/e;
    m++;
  };
  printf( "(m,d) is (%i,%i)\n", m,d );
  return 0;
}
```

Francis' Scribbles

by Francis Glassborow <francis@robinton.demon.co.uk>

Multi-threading

Several years ago Gary Lancaster implemented my Playpen library. The implementation worked well and did exactly what I wanted. We fixed a couple of bugs and added a bit to it.

One day I added my own default palette, which I implemented by with a function. The obvious place to call this was from within the constructor for playpen. At that time, everything seemed to continue to work.

Then I noticed that on some machines the screen incorrectly updated when a program first constructed a playpen instance. It was not a big problem and seemed relatively harmless. I mentioned it to Gary but he could not see a cause and assumed that it must be something I had tweaked. At that time, I had forgotten that I had added a function call to the constructor. I left it and simply warned users that there was a harmless bug lurking in the initialisation of a playpen window.

A couple of months ago I was running a quick test and happened to write:

```
int main(){
  playpen paper;
}
```

In other words a bare minimum program using a playpen to which I intended to add some more test code.

To my surprise, the computer seemed to lock up. After about a minute it recovered and displayed an error message about a misbehaving application.

This is no longer a matter that could be ignored. I experimented a bit to discover what I had to do to avoid this lock-up. In the end I discovered that doing almost anything that stopped the program from closing immediately solved the lock-up problem.

I still did not understand the problem but I could see a simple solution, insert a call to a wait function in the constructor of playpen. [It was then that I realised that I had altered Gary's code by adding the call to the function that provided my default palette.]

Now, that fixed the other problem as well, the screen updated correctly when a program created a playpen Window.

I reported my experience and fix to Gary. Now that gave him enough of a clue to really fix the problem, which was in the original code.

You may be wondering what this has to do with your code. My playpen type relies on using multi-threading. Effectively it implements all the special simple GUI functionality as a distinct thread. The underlying problem was that programs were returning from constructing a graphics window too quickly, before Windows had finished.

This only shows up on modern fast hardware, where the CPU speed is high enough to beat some other process.

This is a good example of the problems of multi-threading. You cannot check your code is correct by testing (actually, you never can do that). Your code may work perfectly for years and then start to fail occasionally. You are certain that the code is OK because it has worked for so long and assumes that the problem is somewhere else. However increased hardware performance begins to make a real problem manifest.

By hindsight, I can see how the symptoms should have pointed me to the cause. The added function in the constructor was trying to update the palette definition while the Playpen window as still being constructed. However, had I tested a truly minimal program much earlier I would have got a much better clue, because trying to destroy a window before it has finished being created is sure to provide a recognizable symptom.

Never ignore little irritants, and test code incrementally starting with the simplest program you can write. Even a little extra may hide a real problem.

I would be very happy to hear your comments on this and other aspects of problems revealed by higher performance hardware.

Commentary on Problem 21

Consider:

```
class x;
class xyz {
public:
  xyz();
  ~xyz();
  static int const elements(100);
  // rest of interface
private:
  x * pointers[elements];
};

xyz::xyz(){
  for(int i(0); i != elements; ++i){
    pointers[i] = 0;
  }
}

xyz::~xyz(){
  for(int i(0); i != elements; ++i){
    delete pointers[i];
  }
}
```

Is there a better way to implement the constructor? Of course there is, and I know some of you know it but do you?

Let us have a look at the code before dealing with the problem.

The first point is that it looks as if the design intends that xyz instances own the items pointed to by the elements of pointers. If that is the case, why did I use an array of raw pointers? In addition, why did I use a fixed size array? That seems to make very little sense.

Without some idea as to what xyz is supposed to do (and the name is not exactly helpful), it is hard to see why I did not use something such as `std::vector<x>` or `std::deque<x>`. However consider the case where x is not a copy constructable, nor copy assignable type. If it also lacks a default constructor, we do not have many options left.

While testing the code I got a rude shock. First, this is what I was looking for:

```
xyz::xyz(): pointers() {};
```

While we cannot explicitly initialise an array, we can force default initialisation. In general, that gains us nothing, because that would happen anyway. The default action for the fundamental types is to do nothing; we can force zero initialisation with the above syntax.

Now to the rude shock:

```
static int const elements(100);
```

is ill-formed. The compiler considers it an attempt to declare a static member function and requires that an in class initialisation be written as:

```
static int const elements = 100;
```

Problem 22

Well the problem is that I have run out of a ready supply of little coding surprises and problems. It is time that you got involved. Please send in at least one coding surprise. If you do not have any then I guess you do not actually do much programming.

The surprise can be in any of the programming languages that are used regularly for application programming (C, C++, C#, Java, Python etc.)

Cryptic clues for numbers

Neil Horlock sent in this clue for last issues number (471, which is 20 more than 451 which was the title of a Ray Bradbury novel about a post catastrophe world where books were burned as evil.) The first part of my clue references the issue of C Vu in which it appeared (17.4)

Roundabout a maximum break, a thousand less than the number of the last caller.

I like his idea of using 'roundabout' to clue rotating the digits, however I think the second part of the clue could do with a bit more polish. Anyone like to propose improved wording based on the same idea (for those from elsewhere in the World, 1471 is the number for recovering the number of the last person who called you.)

This issue's clue

One for love too? Sounds like the right day for it!

Provide your clue for the solution to my clue. No prizes, just a warm fuzzy feeling of seeing your name in the next issue.

Francis Glassborow

Francis Glassborow is a freelance computer consultant and long-term member of BSI language panels for C, C++ and more recently Java and C#. He is a regular member of the UK's delegations to WG14 and WG21. He is also the author of 'You Can Do It!' and introduction to programming for novices.

J2SE 5.0 New Features

by Dave Salter <david@dividsalter.co.uk>

J2SE 5.0 has been available for download since the end of 2004. This new release included many changes and enhancements to the Java platform such as speed and stability. Additionally, some changes were made to the Java language itself. These fairly major changes made to the language are:

- Generics
- Enhanced for loop
- Annotations (sometimes called metadata).
- Autoboxing and unboxing
- Typesafe enumerations
- Variable arguments (varargs)
- Static imports

Using these new language features in your applications can have a big effect on your code, so this article aims to provide an overview of these new features so that you can start leveraging them in your code. As a Java developer I make extensive use of these features now and find that they bring the Java language much more upto date.

Getting the JDK

The latest version of J2SE can be downloaded from Sun's website if you are a Windows/Linux developer or from Apple's website if you are an Apple developer. I'm not going to provide details about how to install the JDK as this is primarily an article about the new J2SE 5.0 features. Before you start developing however, its worth checking that you have the correct version of Java installed. Running `java -version` should produce output similar to the following.

```
C:\>java -version
java version "1.5.0_03"
Java(TM) 2 Runtime Environment, Standard
Edition (build 1.5.0_03-b07)
Java HotSpot(TM) Client VM (build 1.5.0_03-b07,
mixed mode, sharing)
```

Generics

What are generics? Generics are akin to templates in C++ in that they allow developers to write type safe code that operates on different types of objects. This is very useful when dealing with collections as it is no longer necessary to cast objects to specific types when extracting them from collections. This therefore eliminates the potential for getting `ClassCastExceptions` and makes code much easier to read.

In this trivial example, consider a collection of Strings (it doesn't have to be strings however, you could have a collection of any type of object). Prior to Java 5, if you wanted to extract the contents of a collection, you would use code such as that below:

```
import java.util.*;

public class Generic {

  public static void main(String[] args) {
  // Create a collection and add some items to
  // it.
    Collection languages = new ArrayList();
    languages.add("Java");
    languages.add("C#");
    // Now get the items from the collection.
    Iterator iterator = languages.iterator();
    while (iterator.hasNext()) {
      System.out.println(
          (String)iterator.next());
    }
  }
}
```

Notice that to extract objects from the collection, we have to explicitly cast the retrieved objects to be Strings (or whatever class the collection consists of), potentially allowing us to get the dreaded `ClassCastException` if we had inadvertently added the wrong type of object to the collection in the first place. Also, its difficult to see from looking at the code what the collection contains. If there was an API method like the following, what exactly would the collection contain?

```
public doStuff(Collection items)
```

Using generics, our simple application can be re-written to get rid of the cast and made type safe.

```
import java.util.*;

public class Generic2 {
  public static void main(String[] args) {
  // Create a collection and add some items to
  // it.
    Collection<String> languages =
       new ArrayList<String>();
    languages.add("Java");
    languages.add("C#");
    // Now get the items from the collection.
    Iterator iterator = languages.iterator();
    while (iterator.hasNext()) {
      System.out.println(iterator.next());
    }
  }
}
```

You can see that the collection has been explicitly declared to contain only `String` objects. Also note, that there is no cast required to extract items from the collection. The VM knows exactly what type of object is in the collection and knows how to return it to the calling application. If we tried to put something into the collection that isn't of the required type (in our small example, anything other than a String), then the code will simply fail to compile.

If we now re-wrote our rather obscure method `doStuff`, it may look something like the following. Here we can see exactly what type of object the collection contains.

```
public doStuff (Collection<String> items)
```

Enhanced for Loop

The enhanced `for` loop construct available in Java 5 allows us to refine this code even further and remove the need for using the Iterator to loop through the collection. This construct allows our simple application to be re-written, yet again, as:

```
import java.util.*;

public class Generic3 {
  public static void main(String[] args) {
    // Create a collection and add some items
    // to it.
    Collection<String> languages =
       new ArrayList<String>();
    languages.add("Java");
    languages.add("C#");
    // Now get the items from the collection.
    for (String language : languages) {
      System.out.println(language);
    }
  }
}
```

This new construct is declaring that the code should loop through each entry in the collection `languages`. Each entry in the collection will be declared as a `String` called `language` which is then printed out to the console.

Hopefully you'll agree that these new features of Generics and the enhanced `for`-loop allow Java developers to write cleaner and safer code.

Annotations

Annotations provide a new feature to the Java language that potentially allow a vast amount of time to be saved when developing applications. They've been available in C# from the beginning and are now available to Java developers.

Annotations allow developers to add "hints" into their code (in a similar fashion to `Xdoclet` or `JavaDoc` tags) that the compiler can interpret at compile time. This allows the build process to do a variety of things such as produce artifacts that would otherwise have to be manually created, or to check that code complies with certain criteria.

There are 3 annotations that are supplied with J2SE 5.0, but developers have the ability to write additional annotations if they desire. These 3 basic annotations are `@Override`, `@Deprecated` and `@Suppress`.

`@Override` is applied to methods and is used by the compiler to check that overridden methods are declared correctly. A simple example of this would be to apply it to the `toString()` method of a class. `@Override` checks that the method to which it is applied correctly overrides its parent object. If the method is overridden incorrectly (e.g. it is spelt incorrectly or has the wrong signature) then a compile time error will be generated.

```
public class Annotation1 {

  @Override
  public String toStrng() {
    return "...";
  }

}
```

In this code fragment above, the method `toString()` has been declared incorrectly, e.g. `public String toStrng()` - note the deliberate spelling mistake. The `@Override` tag causes an error to be issued at compilation time.

```
C:\>javac Annotation1.java
Annotation1.java:3: method does not override a
method from its superclass
        @Override
         ^
1 error
```

`@Deprecated` is applied to methods in a similar fashion to `@Override`. This annotation will cause a compiler warning to be issued if a deprecated method is used.

Finally, `@Supress` is used to tell the compiler to suppress specified warnings.

The use of annotations is particularly of importance in the J2EE arena and is becoming more important in the strive to make J2EE easier. A couple of examples of this are Webservices (JAX-WS 2.0) and Enterprise Java Beans (EJB 3). The J2EE 1.4 way of creating web service and EJBs involves creating several different interfaces and various verbose XML configuration files that describe the webservices and EJBs being developed. With JAX-WS annotations, it will be possible to declare that a class should be exposed as a web service simply by putting a `@WebService` annotation before the class declaration. With EJBs its very similar. To declare a class as a stateless session bean is simply a matter of putting the `@Stateless` annotation before the class declaration. Of course there are more options that can be added into web services and EJBs to control their behaviour, but the fundamental principal is that all these options can be specified *in code* as annotations.

Autoboxing / Unboxing

Autoboxing and unboxing allows Java code to automatically convert between the basic primitive types (`int`, `long`, `double` etc) and their class equivalents (`Integer`, `Long`, `Double` etc.) and vice-versa. Prior to Java 5, if you needed to convert an `Integer` to an `int`, the code required would look something like

```
public class Autoboxing {
  public static void main(String[] args) {
    int value = 10;
    Integer newValue = new Integer(value);
    Autoboxing ab = new Autoboxing();
    ab.doStuff(newValue);
  }
  public void doStuff(Integer value) {
    System.out.println(value);
  }
}
```

In this code, the wrapper class `newValue` has to be directly instantiated from the varible `value` to be passed into the method. With Autoboxing, this is no longer required as the `int` will be converted into an `Integer` for us:

```
public class Autoboxing2 {
  public static void main(String[] args) {
    int value = 10;
    Autoboxing2 ab = new Autoboxing2();
    ab.doStuff(value);
  }
  public void doStuff(Integer value) {
    System.out.println(value);
  }
}
```

Typesafe Enums

Enumerations are now available in Java just like they are in C++ or C#. They are declared in a similar fashion using the `enum` keyword.

```
public class EnumTest {
  private enum Language { Java, CSharp,
     CplusPlus }
  public static void main(String[] args) {
    Language myFavourite = Language.Java;
  }
}
```

Prior to Java 5.0, they recommended way of declaring enumerations like this was to declare an interface which consisted of a set of hardcoded final static integers. Not only did this force classes to implement unrequired interfaces, but it made code fragile in that it wasn't typesafe and you could easily set a variable outside the range of the enumeration. The new enumerations help solve these problems.

Varargs

Varargs allows methods to be written that take a variable number of parameters that is not know at compile time. This is similar to C functions such as `printf()` and `scanf()`. When used in combination with the new `for`-loop syntax it is no longer necessary to pass collections of parameters into methods and then use an iterator to loop through the parameters once inside the method.

The `vararg` parameter(s) on a method are denoted by placing a "..." postfix onto the variable type as shown in the following example:

```
public class Vararg {
  public static void main(String[] args) {
    Vararg vararg = new Vararg();
    vararg.parseData("Java", "C#", "C++");
  }
  public void parseData(String... data) {
    for (String item:data) {
      System.out.println(item);
    }
  }
}
```

Static Import

Static imports are probably one of the least used features available to Java 5. They allows you to specify static variables in another class without having to specify the fully qualified name of the class being imported. Members can be statically imported into a class using the import static construct.

```
import static my.class.static.member;
```

[concluded at foot of next page]

db4objects – Innovating Object Databases with Open Source

by Paul F. Johnson <editor@accu.org>
and Roberto Zicari <roberto@zicari.de>

Many open source products and projects can easily be mapped to closed source products and often match their scope, concept and capabilities. Linux as an operating system, MySQL as a relational database, OpenOffice as an office suite immediately spring to mind as examples of this. Their main competitive advantage often is their license price being zero. As a consequence, it is often claimed that open source is commoditising software, driving down prices and giving buyers a direct alternative to often monopolistic, high priced, closed-source products.

However, away from the limelight of the open source main street, there's also a series of open source products that are not followers but leaders to the way we use and build software (such as Apache, Hibernate, and db4o.)

db4objects?

db4objects is the creator of db4o, the leading open source object database for Java and .NET. Open source since November 2004, it has seen an impressive growth and is poised to change (or add to) the way we think of persistence (a fancy way to say: database) in object-oriented programming environments.

Recently, I had the chance to interview Christof Wittig, CEO of db4objects (www.db4o.com).

Object databases have failed in the 1990s. Why would db4o's open source object database be more successful?

db4o is far more successful than any vendor has ever been to introduce a persistence solution that simply fits object-oriented programming (OOP) languages such as Java and .NET perfectly.
In just 8 month we have had over 250,000 downloads, built a user community of more than 6,000 professional Java and .NET developers and have signed commercial contracts with the likes of BMW, Hertz and Bosch.
The main reason for db4o's successful launch is the fact that we're open source which gives us a powerful tool to enter the market at low cost, build a large user base very fast, and outpace any closed source vendor in setting de-facto standards.

So you just go open source and that makes the difference?

No. There are also differences in the market segment we target and the advent of OOPs in the main stream.
When object-oriented databases came with great fanfare to market in the early 90s, the protagonists saw them as a displacement for relational databases in the data centre. We believe they targeted the wrong segment with the right product.
In contrast to 1st generation object-oriented databases, db4o is the embeddable persistence solution for all client-side Java and .NET applications. As an example, Eastern Data in Hatfield Perevel, Northeast of London, builds its future line of mobile PDA applications for field force automation (FFA), e.g. for van deliveries of milk, with db4o. They enjoy db4o's zero-administration capabilities, reliability and fast performance. Not only is it possible to bring products to market faster, but also to build more object-oriented, better software which is easier to re-factor and reuse in the long run than when using relational database technology.

Christof Wittig, CEO of db4objects

We also benefit from the increased adoption of Java and .NET. Today it is clear that OOP are the future. Hence, the so-called object-relational mismatch, the inherent incompatibility of OOP with relational databases has become a real business problem, not an academic challenge as it was in the 90s.
In addition, we were able to write our product in native Java and .NET which makes the database even less intrusive than old-style databases written in C, more easy to work with and to deploy in large numbers. After adding a single JAR/DLL file to your project, it takes just one line of code to store any object - no matter how complex your object model is!

Relational databases such as those of Oracle, IBM, or MySQL are the market leaders. How do you want to compete with them?

They all also target the server side data storage and do a great job there. But they fall short on clients, in zero-administration environments. The reason for this is the inherent incompatibility of relational data models and the object schema used by Java and .NET developers.
As a result, the big three, Oracle, IBM and Microsoft combined control about 85% of the overall relational DBMS market, while they command only 25.1% of this embedded DBMS market. We believe that these facts prove that customers seek specialized capabilities in various segments of the embedded market, that go beyond what RDBMS can offer.

Let me play devils advocate: There's a lot of data legacy. How do you want to mitigate the immense efforts it takes to switch a database.

That's a great question and surely another reason, why old-style, server side OO databases failed: They weren't able to provide a painless migration path. This is totally different for db4o, the embeddable object database: There's no legacy in devices, on the client side. Every instance of a BMW car, for example, that is shipped with db4o, has a fresh database instance running. No data migration here. Also there's no "legacy in mind" because we have no DBA and his set of tools such as report writers. Usually you don't write ad-hoc reports against the database running in your game box, do you?

But what about object-relational mappers like open source Hibernate from JBoss or closed-source Toplink? Aren't they a solution for the object-relational mismatch problem?

Object-relational mappers are a good solution on the server side where resources are abundant and/or performance not critical. On the client side, i.e. in packaged software, in mobile or gaming devices, and in real-time control systems they are prohibitive. The reason is the lack of zero-administration capabilities through the added complexity and the huge drain on performance.
Open source benchmarks (www.polepos.org) have shown that db4o is up to 44x faster than a MySQL + Hibernate stack, for instance.
However, we're very happy about the immense growth of these platforms, as they validate the extent of the object-relational problem in our industry.

Back to open source. We understand that you get a lot of hype. But how do you make money?

We adopted the dual license model as pioneered by Berkeley DB and MySQL. We provide our software in its entirety under two licenses, the GPL and a

[continued from previous page]

Summary

Java 5 introduced new language features that allow developers to write code that is more robust whilst at the same time reducing the amount of boiler-plate code that needs to be written. If you're new to Java or are still using JDK 1.4, then its worth while investigating all the new features provided in Java 5.

Dave Salter

Resources

Apple Java Website – http://developer.apple.com/java
EJB 3 - http://java.sun.com/products/ejb/docs.html
JAX-WS 2.0 Web Services - https://jax-rpc.dev.java.net/
Sun Java Website – http://java.sun.com
Xdoclet - http://xdoclet.sourceforge.net/xdoclet/index.html

commercial license. The GPL licensed version is available for free download from our website www.db4o.com. People can use it, evaluate it, getting educated about db4o's immense benefits.

However, the GPL has the obligation to open source your derivative work under the GPL, too, if you start to redistribute it. Therefore the GPL is a no-go as an input component to most commercial, product developing companies. For these customers we provide an additional, affordable commercial runtime licenses which frees them from this obligation and enables them to ship closed source products with embedded db4o. To commercial customers, we also provide direct support and vendor relationship, e.g. to discuss product roadmaps, which are important criteria when evaluating a platform as central as a database, in form of a db4o Developer Network (dDN) membership subscription.

However, we do not provide engineering and anything other than product-related support services. We focus on making the product easy to understand rather than artificially building a complicated product that needs expensive training and consulting to be deployed properly.

With db4o you are up and running in 5 minutes. A great interactive tutorial guides you through the few basic APIs that help you to get most out of the product. The number of users and customers asking us for training is basically zero.

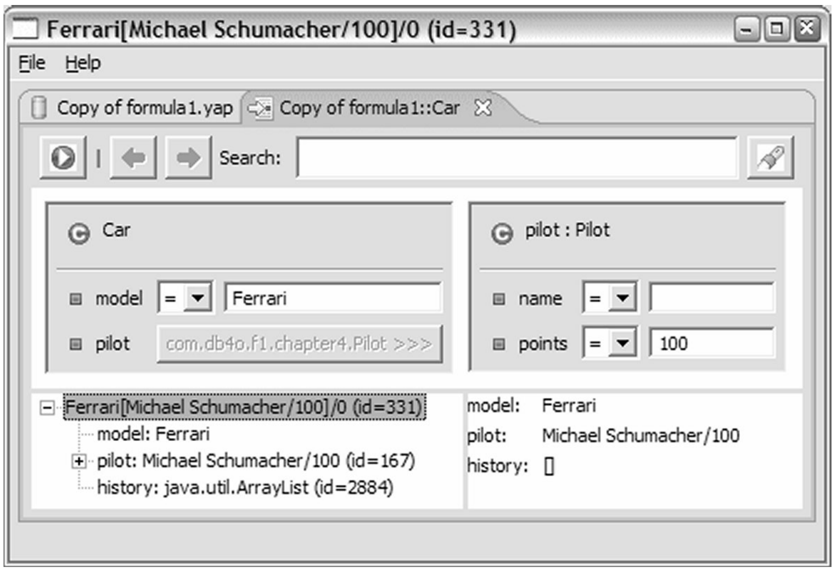
How affordable is your affordable pricing and how can it be sustained?

Affordable means up to 10x less expensive runtime prices than closed-source vendors, such as IBM, Oracle, Sybase, and a long tail of smaller vendors.

We can sustain to charge much less because the open source model basically saves us a lot of spending for product, sales, and marketing.

It is a triple win situation: Commercial customers get better software at lower prices, the community gets a great product for free, and we are able to build a sustainable business. This is only bad news for conventional companies with over bloated sales and marketing departments that will suffer.

You mentioned product development. How does your development model look like?



```
Ferrari[Michael Schumacher/100]/0 (id=331)
File  Help
Copy of formula1.yap | Copy of formula1::Car
Search:
Car                                          pilot : Pilot
  model  =  Ferrari                            name   =
  pilot     com.db4o.f1.chapter4.Pilot >>>     points =  100

Ferrari[Michael Schumacher/100]/0 (id=331)     model:   Ferrari
   model: Ferrari                              pilot:   Michael Schumacher/100
 + pilot: Michael Schumacher/100 (id=167)      history: []
   history: java.util.ArrayList (id=2884)
```

Product development is firmly embedded in the community of many 1,000s of developers that use db4o and a few that are actively writing the core, which are generally paid for by us.

These employees are recruited from within the community. They can be as dislocated as Brazil, Siberia or Germany, but still work very efficiently together.

We use extreme programming blended with open source collaboration techniques and tools. Central communication is through our newsgroups, but we also set up Skype sessions for voice interaction and a Skype/TightVNC combination to run virtual pair programming sessions. Bi-annually we meet face-to-face in destinations as exciting as San Francisco, the Bavarian Alps, or Salvador de Bahia in Brazil to discuss the product roadmap, design proposals, and build team spirit.

All this works with great success for us. We were not only able to commit the smartest guys to our vision, such as Klaus Wuestefeld, author of Prevayler, Rodrigo de Oliveira, author of Boo, and Dave Orme, Eclipse Visual Editor lead, but we also managed to make them work with an efficiency that compares with a 10x over what I know from old-style, collocated software fabs.

What do you look for in a new employee?

I am glad that you mention this. It is very important that each individual software engineer starts to envision how he would fare in this new, dislocated development model. Having experienced its power, I can hardly imagine how siloed software fabs will be able to compete in the long run against it. You can see IBM and others starting to embrace this model already.

We look for individuals with

- *outstanding, relevant accomplishments,*
- *team orientation and good communication skills, and*
- *the ability to self-manage.*

Relevant accomplishments obviously will differ for each area of interest. Someone who has build an open source object-relational mapper "for fun" is certainly of great interest to us. We will look at the quality of the source code and take into account the person has displayed a lot of passion for the subject of our work.

Team orientation is essential. Lonely stars will be rejected by the team and the community. There are great developers out there, but nothing matches the power of a hot core team effectively interacting with a large, diverse community. As a prerequisite, good communication skills, especially via newsgroups and e-mail are required.

Being dislocated and working from your basement also requires a certain discipline as to keep track of priorities and manage and balance life. It is a great benefit for our employees that they can, beside work, take care of kids, travel, live in remote areas, etc. Each of our developers has his reason why he likes the dislocated model - be it the ability to move away from crime-ridden Sao Paolo or to ski in the Alps whenever the weather permits. However, all this requires self-management to bring life in pace with work. We have no line managers but only very broad directives from senior management. Evaluation happens by results not attendance and other behavioural observations. So be sure to find your way to deliver in time.

And here is my advice to people seeking a job: When we hire, we want to see proof of all of the above. We're not training new employees in these skills, we expect them to come with these skill. So think about start building a track record of accomplishment in the open source community that clearly shows proof of your ability to work in the environment I described.

We recently rejected a very qualified candidate with an impressive vita. His job description included extensive posting in newsgroups. However, he didn't have a newsgroup track record at all and the few postings we saw were poor and not very supportive in its nature. Why should we expect he would suddenly change when hired by us? Remember that any code you contribute or any posting you make in a newsgroup is stored by Sourceforge and Google Groups forever. And recruiters will start to look at them (or their absence)!

db4o has recently announced a new way of writing a database query, called Native Queries, using entirely programming languages such as Java and C#. What is the difference with respect to the traditional SQL-way of query a database offered by relational database systems?

Native Queries (NQ) are a new, additional API to db4o which uses the programming language itself - Java or .NET - to query the db4o database. Native Queries are based on Safe Queries as proposed by William Cook, Prof. at University of Texas, at the 27th International Conference on Software Engineering (ICSE).

Over the last 15 years, there was a lot of thoughts and proposals around building a new OO query standard which would be the equivalent of SQL for ODBMS and ORMs -- OQL and JDO are examples for this. None of them have become mainstream and hence fail to be a standard.

We think that an embedded ODBMS doesn't need an additional query language such as SQL. SQL was mainly designed for DBAs that want to query the database directly, e.g. for ad-hoc reporting. Embeddable databases are in zero-admin environments. The only user of the API is the developer who already knows and uses one language: The programming language. So we decided to standardize on the standard he or she already uses: Java or .NET.

As a result, using native queries, you can use a lot of the productivity enhancements provided by your IDE. You get a 100% typesafe code (no

[concluded at foot of next page]

Setting up a Subversion Server for Remote Use

By Craig Ringer <ringerc@scribus.info>

Revision control is a critical part of any significant development project. Having secure full time access to your repository from wherever you are can be important. In some environments, such as open source projects, it's absolutely crucial to the functioning of the development team.

My first exposure to the Subversion[1] revision control system was when I was searching for something to replace CVS for the Scribus project, whose CVS server I administer. There were a number of problems with our use of CVS over ssh, namely server security concerns, cross platform issues, and configuration complexity.

After some research, I decided that Subversion was a good candidate to replace CVS. Subversion is a fairly new revision control system with the stated goal of being a "better CVS than CVS." One particularly attractive feature is the use of HTTPs for a secure, fast, encrypted transport that eliminates the need for an SSH tunnel. Additionally, Subversion is becoming increasingly popular in the open source development community, so more and more useful tools and graphical clients are becoming available.

I'll be covering how to set up a Subversion server for your team, so they can work on your code wherever they are without introducing major security risks. Specifically, I'll be explaining how to set up a Subversion server configured for use with WebDAV over HTTPs, using SSL client certificates for an additional layer of authentication. The goal is revision control that's fast, secure, and easy to use from anywhere. I'll be assuming that you're using a UNIX variant, but it should be quite possible to set all this up on Mac OS X or even Windows NT/2k/2k3 as well.

This article won't try to explain how to use Subversion, why you might want revision control, basic UNIX command line use, or any related topics. We're going to focus specifically on setting up reasonably secure remote access to a subversion repository. As this article also avoids going into extreme depth on the use of subversion and other finer points of its configuration, it is strongly recommended that you examine the excellent Subversion book[2] (freely available on-line) for more detail.

It is important to note that your author is not a security expert. I am an experienced system administrator who has operated Internet-accessible systems hosting public services for some time, but beyond that has no specific security qualifications. This article does not provide some magic recipe for a secure server configuration – but it should help you get started along the way.

Why use Client Certificates?

The use of client certificates provides an extra layer of authentication. A user can't even attempt to authenticate against your svn server until they've provided a client certificate that you can verify is signed by your CA and has not been revoked. For a bit of extra security, you can also store client certificates separately from the client host – on a USB key or potentially even a smart card – and remove them when not in use.

Client certificates are also useful for controlling access to more than your source code repository. They can be used to help secure SSL/TLS-based mail services such as encrypted IMAP, POP3, and SMTP. They are also useful if you wish to offer controlled HTTPs-based remote access to your organization's intranet web server for roaming users. You can also use the same client certificate infrastructure to permit users to encrypt and/or sign email within the organization using S/MIME (though recipients who have not imported your CA certificate won't be able to verify signed mail).

What You'll Need

First, you'll need a server with the Apache 2 web server, the `mod_dav_svn` Apache 2 server module, and the subversion tools installed. On Debian GNU/Linux 3.1, just `apt-get install apache2 libapache2-svn subversion`. For other platforms, if you're unsure how to go about installing Apache 2 or `mod_dav_svn`, the Subversion web site[1] has plenty of information.

OpenSSL[3] will be required to create certificates, so unless you have an existing CA and x.509 PKI scheme you'll need to install that.

If you want to use your repository from anywhere, you'll need an Internet-accessible IP, or a port forwarded through your firewall. You can use a VPN if you prefer to further limit the accessibility of your server.

Certificate Creation and Management

The creation and management of SSL certificates can be a complicated business. Once it's familiar you will find that it's not generally an issue, but the initial process can appear somewhat daunting.

If your organization has existing X.509 based PKI infrastructure, you may

[continued from previous page]

strings!), 100% refactorable code, and 100% object-oriented code, which is easily optimizable.

We believe that this powerful, open concept will find wide industry adoption and become the standard way to query databases in an OO way. db4objects is the first industry player to adopt the standard and puts the power of the open source model behind it. A preview version (V5 milestone 2) of the new API is available for free download from our website. Also I invite to read the free whitepaper by William Cook and Carl Rosenberger, available on www.db4o.com/about/productinformation/whitepapers/#nq which elaborates more on the design concept and goals and discussed advantages and disadvantages.

db4o was originally started in Germany. How did you get to base your business in Silicon Valley? What role does Europe and the UK play?

The product started in Germany, when Carl Rosenberger realized his dream on 1/1/2000, to free OO developers from the OR mismatch.

The corporation started last year in Silicon Valley, the central trading place for ideas and technologies. With private investors such as Mark Leslie, founding CEO of Veritas, and Diane Greene, founding CEO of VMware we had found the right people to put their names and resources behind the idea and launch db4objects, Inc. While we don't produce in Silicon Valley, we see it as the place to do global marketing, sales and finance.

Europe is very strong for us, given our origins and the lead Europe has in mobile applications, for instance. The UK are constantly ranking among the top 5 countries for db4o, together with the US, China, Japan and Germany. On September 29, we will host our first local event in London's Imperial College, where Glasgow University's Professor Jim Paterson will introduce db4o (more information on www.db4o.com/about/productinformation/events/fall05rs*).*

Tell us about how you see the DBMS market evolve? What role does the embedded DBMS play and how does this affect your business plans?

I leave this answer to the leading analysts.

According to IDC's estimates, the embedded DBMS market grew 15% to $1.86 billion in 2004, and is expected to blossom to $3.18 billion in 2009: "Object-oriented DBMSs could well enjoy a second growth period as embedded DBMSs due to the efficient and flexible data management they offer object-oriented applications, and open source DBMSs are also attractive as embedded DBMSs because of the technological control they offer ISVs as well as flexibility in licensing," says Carl Olofson, research director for information and data management software at IDC. "db4objects is in the interesting position of offering the benefits of object-oriented DBMS technology and open source licensing, making its value proposition appealing on two fronts."

Chris Lanfear, director at Venture Development Corporation (VDC), says: "Especially on the client side, such as in stand-alone devices and other zero-administration environments, engineers look for innovative persistence solutions that meet their immediate specifications and help them outrun the competition. As a result, more than 50% of embedded and device software developers still build their own database tools today. With the advent of standardized object-oriented platforms, such as embedded Java and the .NET CompactFramework, we expect object databases to become a universal solution for OO persistence - with db4o's open source offering leading the charge."

I have nothing to add to that!

Christof, thank you for your time in giving this interview.

Paul Johnson
Roberto Zicari

More information on db4objects can be found at:
http://www.db4o.com

well be able to use your existing client certificates and CA certificate to control access to your Subversion server. Should you be so lucky, you can escape the need to deal with OpenSSL. Similarly, if you have an existing server certificate, you can use that rather than creating your own. It doesn't have to be signed by the same CA as your client certificates.

To create your certificate authority, client certificates, and your web server certificate, you can use the OpenSSL tools. There is not enough space to discuss this in the detail it demands, but I can provide some brief coverage of the procedure. Alas, OpenSSL is rather sparsely documented (especially on the broader scale), and does not ship with any suitable references for most tasks. Your author is no expert on SSL in general, or on OpenSSL in particular, being just a lowly system administrator and programmer. Errors are possible, so do be careful.

Some useful additional information can be found in the Apache `mod_ssl` FAQ[4], at `pseudonym.org`[5], and for Windows at `stunnel.org`[6].

I'll be assuming that you have OpenSSL already installed, as per the requirements above.

Creating a Certificate Authority

Before you can create client certificates, or useful server certificates, you need to obtain a signing certificate. It is possible to buy such certificates from SSL certificate vendors, but this is unnecessary if you only intend to use the certificate scheme within your organization and with your own users. You can simply create your own certificate authority with a self-signed certificate. The only significant limitation of such a certificate is that it must be installed in clients before they will recognise the validity of certificates you have signed.

I suggest that you set up your certificate authority on a different host if possible, and in as secure a location as you can arrange. Removable external media may be worth considering.

First you need to create a suitable openssl.cnf, the configuration file that will drive your CA. The OpenSSL distribution ships with a sample file that you can customise. Alas, if you got your copy of OpenSSL through a vendor package library, this could be almost anywhere on your system. If you can't find it, you can download a sample config file from the OpenSSL project's public CVS browser[7].

Now we need to make a directory to store your CA in, and put a copy of openssl.cnf in it. I'm going to refer to this directory as CA hereafter. With that done, openssl.cnf needs to be customised to your site. Start with the section [CA_default], setting dir to the path to your CA directory. This tells OpenSSL where to put (and look for) the various files and directories used when managing your CA. I tend to use an absolute path, but if your CA will be stored on removable media where the path may not be constant, you can use . (the current directory) instead, then always work by changing into the CA directory.

Next, under `[ req_distinguished_name ]`, adjust the `_default` parameters to suit your site. You can add `_default` parameters to options that do not have them, or remove them if you don't want to provide defaults for a parameter.

To finish our preparation, create the structure to store the CA's various information by creating the subdirectories certs, newcerts, crl and private within your CA directory, then create a file called `serial` (no extension) containing only the digits `01`, and create an empty file called `index.txt` . Be sure to set the access permissions on private so that only the user who will be managing the CA can see its contents or modify it.

It's finally time to create the CA certificate that signs all your client certificates and lets you verify them later. Pick a good pass phrase to use on your CA's key, and record it somewhere secure and offline. Now create the self-signed certificate that you can use as your CA, supplying the pass phrase you decided on when prompted:

```
openssl req -new -x509 -keyout private/cakey.pem
    -out cacert.pem -days 365 -config ./openssl.cnf
```

When prompted to enter details for the certificate, you should enter the details you wish to appear if a user queries the CA certificate (eg in a browser). As such, Common Name should generally by set to the organization name, not the name of the creator of the certificate. Note that the certificate was created with an expiry date one year from today. You can extend the expiry date of the CA certificate as it approaches expiry with:

```
mv cacert.pem old_cacert.pem
openssl x509 -in old_cacert.pem -days 365
    -out cacert.pem -signkey private/cakey.pem
```

Note that clients will treat certificates signed by an expired root certificate as invalid, and must import the updated root certificate. As such, you may wish to choose a reasonably long validity period.

You can examine your CA certificate by dumping a human-readable version with:

```
openssl x509 -in cacert.pem -text
```

If that looks alright, you've created your CA. You're now ready to start creating and signing certificates. Make a backup of your CA directory somewhere secure, safe, and off-line, then copy your CA certificate file (but absolutely not the CA key) to somewhere that Apache 2 can access it. Apache needs the CA certificate to verify that client certificates were really signed by your CA.

For security reasons, it is crucial that you do not keep your CA key on the Subversion server. Put it somewhere safe, preferably on encrypted storage that's not connected to the Internet. I favour the use of a small old laptop that's kept in a safe when not in use, but some might accuse me of excessive paranoia. No matter where you store your CA, remember to keep a backup somewhere safe and secure, such as on CD or tape.

Creating a Server Certificate

If you don't have an existing SSL server certificate for the host you want to run your Subversion server on, you need to create one. It is necessary to first create a certificate request, then sign that request with your CA. Be sure to provide the DNS name of your Subversion server in the common name field of the request, otherwise clients will be warned every time they try to connect to your server. You must use the publicly visible DNS name of the server, rather than its internal host name. In the following, replace new with your host name, eg myhostname_req.pem for new_req.pem.

Here I show the creation of a key without a pass phrase. This means that the key can be used by anybody who obtains it. It is possible to use a key with a password on a web server, but with Apache the password must be entered interactively. This interacts rather poorly with log rotation scripts, and means that if your server ever goes down it won't come back up without manual intervention. I dislike the use of an unencrypted private key, but have found no viable alternative for my use. If you can afford the possible issues involved with using an encrypted key, then I encourage you to use one – simply add `-des3` to the first command line below.

Create a private key (append **`-des3`** to encrypt the key):

```
openssl genrsa -out new_key.pem
```

Create a certificate request using your key:

```
openssl req -new -key new_key.pem
    -out new_req.pem -days 360
    -config ./openssl.cnf
```

then sign it with your CA:

```
openssl ca -policy policy_anything
    -out new_cert.pem -config openssl.cnf
    -infiles new_req.pem
```

If all has gone well, you should have a server certificate. Check it with:

```
openssl x509 -in new_cert.pem -text
```

to ensure that it's correct, then copy `new_cert.pem` and `new_key.pem` to somewhere Apache can access them, and save backup copies somewhere secure, safe, and off-line. Be sure to set the permissions on `newkey.pem` so that only the Apache user can read the file, and nobody can modify it. You can now discard `new_req.pem`.

Creating the Client Certificates

With a working CA established, you're equipped to create and sign client certificates for use by your users. While it's possible to get users to make their own certificate requests, I'll assume you'll be doing that for them then supplying them with a pre-made certificate. The first stage of the procedure for making a client certificate request is actually the same as that for the server certificate described above, except that you should provide the user's name and email address for the common name and email fields, respectively. You only need the certificate and key files temporarily, so there is no need to save them anywhere.

Once you have the certificate for the user, created the same was as the server certificate above, you need to convert them to PKCS#12 format, a "packaged" certificate format that most clients understand. You can bundle the CA certificate into this package so that it's automatically imported by most client software when it loads the PKCS#12 certificate. I suggest you save the certificate with a suitable name that makes it easy to identify the owner of the certificate later, such as `firstname.lastname_at_domain.p12`.

To do this, assuming your user's temporary certificate and key files are in `new_key.pem` and `new_cert.pem` respectively, run:

```
openssl pkcs12 -in new_cert.pem
  -inkey new_key.pem -certfile cacert.pem -out
  user_name.p12 -export -name "User's
  Subversion certificate for MyOrganization"
```

If you encrypted the user's key, you will be prompted for the password to decrypt it. You will then be prompted for a password to encrypt their new PKCS#12 file with. This is the password you will need to supply to the user for them to use their new client certificate.

Once the PKCS#12 file has been created you can discard `new_cert.pem`, `new_key.pem`, and `new_req.pem`.

Setting up a Test Repository

For the purposes of this article, it's best if you create a new subversion repository to work with. You should probably work on a dummy repository before going live with your server even if you have an existing one or plan to convert from CVS.

Setting up a new repository is simple. Assuming that you want it to live in `/var/svn`:

```
# mkdir /var/svn
# svnadmin create /var/svn fsfs
```

Now let's add a dummy module for testing. First, create the files to import in some temporary directory:

```
# mkdir testproject
# echo 'It worked!' > testproject/test.txt
```

then import the temporary `testproject` directory into the new repository:

```
# svn import testproject
   file:///var/svn/testproject/trunk
   -m "first import"
```

You can now discard the `testproject` directory.

The Subversion website has detailed documentation on how to create a repository, import sources, or convert a CVS repository to Subversion using `cvs2svn`, so I'm not going to discuss it in any more detail here. You might want to look up the Subversion book[6] if you're unsure how to proceed when it comes time to get your project's live code into svn.

Setting up the Web Server

When using client certificates, it is generally be best to set up your Subversion server in a separate Apache 2 virtual host running on a non-standard port. This is necessary because of limitations in the SSL implementation of common clients.

Apache installations and configuration specifics differ a huge amount across different OSes and even Linux distributions. Consequently I can discuss the general configuration approach to take, but not necessarily all of the specifics of what files to edit and what to put where. You may need to adapt the sample configuration discussed below to suit your system.

In the following configuration I make the assumption that your repository is in `/var/svn`, and you'll use `/var/www/projectname` for things like WebSVN[12]. You can use whatever paths you prefer, so long as you configure Apache accordingly. First, it may be necessary to add or uncomment the configuration directives to load `mod_dav` and `mod_dav_svn`:

```
LoadModule dav_module path/to/mod_dav.so
LoadModule dav_svn_module path/to/mod_dav_svn.so
LoadModule authz_svn_module path/to/mod_authz_svn.so
```

The exact paths to the modules will vary depending on your Apache and `mod_dav_svn` installation. If your Apache is not already configured for SSL, you may also need to uncomment or add a directive such as:

```
LoadModule ssl_module path/to/mod_ssl.so
```

To actually configure the Subversion server you'll need to add something like this to your Apache 2 configuration:

```
# Tell Apache to listen on port 4430 for connections
Listen 4430
# And set up a virtual host to handle connection on
# that port:
NameVirtualHost *:4430
<VirtualHost *:4430>
  # Set up SSL for the virtual host.
  SSLEngine on
  # The CA certificate file that you'll be
  # validating client certificates against:
  SSLCACertificateFile /path/to/your/cacert.pem
# The server certificate the web server will
# identify its self with. If you have an existing
# SSL virtual host, you can use that certificate
# for this virtual host too (just specify
# the same path here). It need not be signed by
# the CA specified above.
SSLCertificateFile /path/to/your/servercert.pem
# Only set this if you have a separate key file
# for your server certificate:
SSLCertificateKeyFile /path/to/your/serverkey
# Require clients to have a certificate signed by
# one of the CA certificates specified earlier:
SSLVerifyClient require
SSLVerifyDepth 1

# Require a valid username and password to access
# any part of this virtual host, and make the
# default access control "deny".
<Directory />
  # Only talk to clients using SSL
  SSLRequireSSL
  # Don't permit the use of .htaccess files to
  # override these settings
  AllowOverride None
  # This tells Apache to use BASIC password
  # authentication. You can rely purely on client
  # certificates if you wish to - look up
  # FakeBasicAuth in the mod_ssl documentation.
  # You can also authenticate against a database,
  # or against LDAP, if you prefer - see the
  # Apache documentation. Here BASIC
  # authentication with a simple password is used.
  AuthType Basic
  AuthName "ProjectName SVN"
  # Your password file. Put this somewhere safe,
  # and outside the DocumentRoot configured below.
  AuthUserFile /path/to/apache2/config/
     projectname_svn_htpasswd
  # Reject access from users who give no / wrong
  # passwords.
  Require valid-user
  # and, for the root directory, reject all
  # access. This is overridden in later
  # subsections.
  Order deny,allow
  deny from all
</Directory>

# Tell Apache to look for files starting in
# /var/www/projectname
DocumentRoot /var/www/projectname
# And permit any authenticated user access to the
# files under it.
<Directory /var/www/projectname>
  Order allow,deny
  Allow from all
</Directory>

# Set up the Subversion server, giving it the
# virtual location "/svn" in URLs.
<Location /svn>
  # Turn on the svn server
  Dav svn
  # Tell it our repository is in /var/svn
  SVNPath /var/svn
  # Uncomment the following line to enable Authz
  # Authentication
  # AuthzSVNAccessFile /etc/apache2/dav_svn.authz
  # Permit access to this location (still requires
  # valid user and client cert as specified
  # earlier).
  Order allow,deny
  Allow from all
</Location>
```

```
    # Enable gzip compression if available
    <ifmodule mod_deflate.c>
    DeflateBufferSize 8096
    DeflateCompressionLevel 9
    SetOutputFilter DEFLATE
    SetInputFilter DEFLATE
    </ifmodule>
  </VirtualHost>
```

You may need to edit the existing virtual host directives to explicitly specify the port they listen on (80 for HTTP, 443 for HTTPs) using the same form as shown above.

If you want to offer anonymous read-only access to your repository (common for Open Source projects) then you can add a section like this to your normal HTTP and/or HTTPs virtual host(s):

```
  # Anonymous read-only access to the repository
  <Location /svn>
    Dav svn
    SVNPath /var/svn
    <LimitExcept GET PROPFIND OPTIONS REPORT>
      Order deny,allow
      Deny from all
    </LimitExcept>
  </Location>
```

You will probably want to enable gzip compression as well, as shown in the main configuration listing. Using gzip compression on the server saves bandwidth and results in faster checkouts, but at the cost of some server CPU time.

You should now be done configuring the web server. Test your configuration's syntax with `apachectl -t` (`apache2ctl -t` on some systems) and restart Apache 2.

Final Server Configuration

Before you can test the newly configured Subversion server, you must ensure that your repository is writeable by the web server. The exact procedure for this depends on if you plan to offer access to your repository using other methods too. Assuming that you'll only be using the repository via the Apache-based Subversion server and you're running under a modern UNIX, `chown -R webserverid /var/svn` (where `webserverid` is the user ID your apache2 is running under) should do the job. If you're not sure what user ID apache is running under, the first column of the output of `ps aux | egrep '(http|apache)'` should show you.

Red Hat Fedora users need to be aware of of SELinux, which can interfere with the operation of your Subversion server. If you find that you are getting "permission denied" or "403 Forbidden" errors that make no sense, the chances are good that SELinux is involved. Reconfiguring SELinux is beyond the scope of this article. For testing purposes only you can disable it using `setenforce Permissive as root`. Consider tweaking the SELinux configuration for Apache 2 rather than permanently disabling SELinux, since SELinux provides additional isolation between the various network services on your system.

Adding Users on the Server

Since you're using HTTP BASIC authentication with a plain password file, you need to add some users to the password file. To set up the password file, use this command:

```
  # htpasswd -c /path/to/htpasswd_file username-to-add
```

where `/path/to/htpasswd_file` is the same as the path you gave in your Apache 2 configuration. Now ensure that the file is readable but not writeable by Apache:

```
  # chgrp apachegroupid /path/to/htpasswd_file
  # chmod 640 /path/to/htpasswd_file
```

and start adding more users with:

```
  # htpasswd /path/to/htpasswd_file
      username-to-add
```

Setting up the Client

If all has gone well, you're now ready to test out the new server by connecting with a subversion client.

First, you need to install your PKCS#12 format client certificate. Exactly how to do this depends on what subversion client you are using. With the command line client on UNIX, I tend to put the certificate file into `$HOME/.subversion`. I then edit `$HOME/.subversion/servers`, adding a line like `mysvn = hostname.of.my.svn.server` to the `[groups]` section, and add a new section for that server:

```
  [mysvn]
  ssl-client-cert-file = /path/to/client/cert/file.p12
  # If you want to have svn remember the password to
  # your cert file, set this. Since you're using BASIC
  # auth as well, this is generally fine. Many GUI svn
  # clients don't seem to be able to prompt for a
  # certificate password, so saving it here also helps
  # to avoid confusing those clients.
  ssl-client-cert-password = yourPasswordToSave
```

At the time of writing the Subversion client did not appear to read the CA certificate out of the PKCS#12 client certificate file. As such, you need to provide cacert.pem to your users if you created your SSL server certificate using your self-signed CA. This can be skipped if you bought a server certificate from one of the major certificate authorities.

To install the CA cert, copy it to:

```
  $HOME/.subversion/myorganization.pem
```

then edit the `[ global ]` section of `$HOME/.subversion/servers` and add a line such as:

```
  ssl-authority-files = /path/to/myorganization.pem
```

Testing

With that configuration done, you're ready to check out the module you created earlier. Using the command line subversion client:

```
  svn checkout https://username@hostname:port/svn/
     testproject/trunk testproject
```

Naturally you'll need to adjust the URL above to use your user name in the Apache password file, svn server host name, and server port.

You should be prompted for your password, then the checkout should complete, leaving you with the same `testproject/test.txt` file you checked in earlier. You can now work within your checkout directory as if it was your usual local source tree, then commit changes back to the repository with `svn commit`. Subversion will remember the repository URL, so you don't need to specify it when working within a checked-out tree.

If you install your client certificate into your web browser (in Firefox: **Edit->Preferences**, **Advanced**, **Manage Certificates**, **import**, then edit the CA cert under **Authorities** and mark it as trusted), you should be able to explore the repository by visiting `https://hostname:port/svn`. This can be useful when troubleshooting configuration problems.

Locking it Down

In the opinion of the author, there's almost no such thing as enough paranoia when securing your source code repository. That's doubly true if you're making it accessible over the Internet. It's well worth looking into hosting the repository on a dedicated server or isolated OS instance, limiting access by IP address, putting the server behind a VPN gateway, storing client certificates on external media, etc. Naturally, you'll also want to keep the server and the clients up-to-the-minute with all security patches. Don't forget to make backups – and to periodically archive the backups.

Remember that your clients are important too. It's easier to clean up from an attacker committing to your repository through a client than it is from a server compromise, but if their changes are not recognised as from an impostor, you still risk shipping malicious code to users.

When it comes to server security, the more you can isolate the Apache 2 / Subversion service from whatever else might be running on the machine the better. An entirely separate Apache 2 instance running under a different user ID just for svn may be worth considering if your server also runs Apache for other public services. A physically separate server or OS a separate instance is even better. Linux users may wish to investigate Xen[8] or VMWare[9] for isolating OS instances on the same hardware; Solaris users may want to consider using zones[10]. The better your svn server and repository are isolated from any other potential sources of attack, the less impact a service compromise is likely to have.

There's no such thing as a secure computer – especially when exposed to the Internet – and this article can't teach you even a fraction of how to truly secure things. Don't just follow what's described here, but try to go out and get help locking your server down as tight as possible.

There's little worse than discovering that the system hosting your source code repository has been compromised, so you should probably do everything you can to make sure that it doesn't happen, and that if it does you're prepared

[concluded at foot of next page]

Pointer Reversal: An Algorithm Design Technique

by Atul Khot <atul.khot@gmail.com>

Why Do We Need Yet Another Algorithm Technique?

I came across the Schorr and Waite algorithm while studying garbage collection in Knuth. (See algorithm E, section 2.3.5 of Knuth Volume I.) One phase in garbage collection typically consists of marking nodes of a data structure not in use. These data structures are Lists. Knuth defines Lists as "a finite sequence of zero or more atoms or Lists". Any forest is a List. Unlike forests, Lists can have cycles. And just the way forests are represented as binary trees, two links are used, each link with a different meaning.
There are three types of Nodes:
i) List Heads
ii) Atom Nodes
iii) List Nodes
An `ATOM` field is maintained, which is `true` for Atom nodes and `false` for List nodes. For a List node, two links, `DLINK` and `RLINK` are maintained. If `ATOM` is `false` and `DLINK` is non-null, it points to a List Head. If `ATOM` is `true`, `DLINK` is irrelevant.

Given a collection of List Heads, you need to visits all nodes reachable from each List Head and mark them. You need to keep track of where you came from and so need to maintain a stack. But there might not be enough memory to keep the stack!!!

To summarize, as memory goes too low, garbage collection could kick in, but it in turn needs memory to maintain a traversal stack … Quite a catch-22.

The Schorr and Waite is an elegant algorithm that solves this problem. It alters the pointer fields of the data structure itself to maintain the traversal stack (Pointer Reuse ☺). Suppose Node A points to Node B and B points to Node C. When the algorithm is executing and marking Node C, it temporarily alters the pointers so that C points to B and B points to A. Hence the name Pointer Reversal. When the algorithm pops the stack, it restores (reverses back) the pointers to their original values.

The ATOM Trick

List Nodes have an `ATOM` bit which is used skip `ATOM` nodes, after they are marked. For a List Node, this bit is always false. The algorithm uses this bit again (in a context where it is known to be false) for remembering state for Linked Nodes!! As pseudo-code

```
if ( ATOM = true )
{
 mark this node and skip rest of the
processing
}
else
{
  if ( we changed DLINK as a stack link )
    ATOM = true
  .....
/*Now here, when we want to reverse the link*/
  if(  ATOM = true )
  {
    we came over a DLINK ( and hence changed
it )
  }
  else
  {
    we came over a RLINK (and hence changed
it)
  }
}
```

The clever trick is manipulation of the `ATOM` bit. While processing a List Node `ATOM` is `false`, so we set `ATOM = true` to remember one pointer and `ATOM = false` (the default) to remember other.

Application to Binary Trees (And The Good Old Inorder Traversal)

However, as it turns out, we don't need this `ATOM` bit for binary trees… We exploit the properties of binary trees (no cycles, all nodes are List nodes etc.) and use the traversal property (inorder traversal) in place of the `ATOM` bit.
The algorithm, in Knuthspeak, looks as below: (See notation below)

K1. Set P <- Root, T <- ^. [Root is root of binary tree, ^ stands for null pointer].

K2. If LLINK(P) = ^ goto K3. Otherwise, set R <- LLINK(P), LLINK(P) <- T, T <- P [pointer reversed], P <- R and repeat this step.

K3. Set MARK(P) <- 1. If RLINK(P) = ^ goto K4. Otherwise, set R <- RLINK(P), RLINK(P) <- T, T <- P, P <- R. Goto K2.

K4. If T = ^, the algorithm terminates. Otherwise, if MARK(T) = 0, Q <- LLINK(T), LLINK(T) = P, P <- T, T <- Q and goto K3. Else, [parent is marked], set Q <- RLINK(T), RLINK(T) <- P, P <- T, T <- Q and repeat this step.

Notation: 1. = is equality operator (= =)
2. T <- P is assignment – T is assigned the value of P
3. LLINK(P) is really P->LLINK in C/C++

Step K2 reverses left pointer.
K3 reverses the right pointer.
K4 pops back (i.e. restores pointers)

Note that we are essentially traversing (marking nodes in) the tree in in-order traversal. It is essentially an in-order traversal without any explicit stack. Instead we need a `MARK` bit (but note that the `MARK` bit really need not be explicitly maintained – you can use other contextual info).

Note that, in K3, when a node is marked, its left subtree (if any) is all marked.

I tried the algorithm with boundary test cases (a binary tree with just the root node, a degenerate tree with only left or only right nodes etc…). It works correctly…

The Master Verifies…

This appears as an exercise in Knuth (2.3.5 – exercise 7). When I come up with a solution, verifying it always teaches me something more… This was a rare case where Don's agreed with mine… Sometime back, I solved exercise 10 and when tallied with the book solution, I came across a very novel use of queues.

Over the years, as I keep reading Knuth I have grown a deep respect and adoration for these volumes. The text is crisp, sharp and cracking the exercises make you ecstatic. Nirvana, anyone?

Atul Khot

[continued from previous page]

and able to quickly recover. "Sorry, we just shipped a rootkit/virus/trojan with our last product" isn't an announcement you ever want to have to make.

Subversion and the Scribus Project

The Scribus project is still using CVS+SSH. Our tests of Subversion were quite successful, but many of the team want graphical merge and history browser tools that they didn't feel to be sufficiently mature. As the KDE project has moved to Subversion, I'm hoping to see some improved svn clients available soon, after which I hope it will be possible to migrate the project to Subversion.

Resources

1 http://subversion.tigris.org/
2 http://svnbook.red-bean.com/
3 http://www.openssl.org/
4 http://www.modssl.org/docs/2.8/ssl_faq.html
5 http://www.pseudonym.org/ssl/ssl_cook.html
6 http://www.stunnel.org/examples/ms-ca-newbie.html
7 http://cvs.openssl.org/getfile/openssl/apps/openssl.cnf?v=1.23.2.4
8 http://www.xensource.com/
9 http://www.vmware.com/
10 http://www.sun.com/bigadmin/content/zones/

Craig Ringer

The Agile Manifesto Explained (and a First Amendment)

by Phran Ryder `<phran@agilenorth.org.uk>`

Last Time

In my last article I set the scene for what I hope will be a series of provocative and informative articles. In the article I described the plight of Pete and his vain efforts to change development processes at his place of work. Peter represented the type of people that will be interested in Agile Software development – professionals who care, perhaps passionately, about software development. As Pete Goodliffe points out: *There is more to being a professional than a trade or a methodology. It is more a state of mind.*

So that's my audience, what of the manifesto. In this article I will be looking deeply into what I think the Agile Manifesto says.

The Manifesto

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- ***Individuals and interactions** over processes and tools*
- ***Working software** over comprehensive documentation*
- ***Customer collaboration** over contract negotiation*
- ***Responding to change** over following a plan*

That is, while there is value in the items on the right, we value the items on the left more.

The Manifesto Analysed

At first impression there is not much to it. 358 characters, 73 words, and 6 paragraphs. Oh and there are 4 bullet points, some indentation, and 10 of the words are in bold.

Lets take the first sentence:

We are uncovering better ways of developing software by doing it and helping others do it.

Now lets take the first two words: *We are*. It does not say *We have*. This says clearly that the work is not complete it is an on going evolving process. Indeed I later I will, bravely, be suggesting a first amendment.

And what about the third word, *uncovering* rather than *developing* or *inventing*. This is because there are a lot good agile software development practices that are currently in use - you may well already being using many of them without realisng how agile you are. This is a very comforting thought. Agile software development is not magic, it is not necessarily a new way, it might just be making better use of practices that we already have and that we know work. Morevoer becoming and being agile is open to all, not just the elite and gifted. Neither is contributing to the uncovering of better ways – we can all contribute. Uncovering the new ways is "by doing it and helping others" the helping of others takes many forms: magazines, consultancy, conferences, courses, books, web sites, mail groups…you can, if you care, contribute to many of these and thus contribute to the uncovering, growth and improvement of software development.

So a simple, short first sentence sets the scene for never ending improvement and shows we can all be involved. Next there is a preamble leading us into the heart of the manifesto.

Through this work we have come to value:

I will draw your attention to the last word value. It is very undersated but, 108 words in, it is crucial as the whole manifesto is a statement of values and it is a word I will return to later.

Next we have the significant formatting - four indented bullets with some of the words in bold. These bullets are four statements of value. Each statement mentions two related things of value with the word over used to indicate that the first value, the one in bold, is of more value.

The bulleting draws the eye to this heart of the manifesto. But just incase you missed the point the final sentence re-states the relationship between each pair of values:

That is, while there is value in the items on the right, we value the items on the left more.

This is very interesting. Why put these items in pairs and why emphasise in three ways that one part of the pair is more important than the other?

Firstly, without the repeated emphasise it might be easy to conclude that the manifesto says that the things on the right aren't important at all. That simply is not true. Secondly, the pairs are not chosen at random there is a relationship between each side of the pair. My intepreation is that each pair provides a morale or a warning - a warning to make sure you get your priorities right.

For reasons I can't explain I am going to call each pair a bi-attitude. In the next sections I explain what I think the warnings are.

Warnings

Individuals and interactions over process and tool

Why are individuals and interactions important?

An organisation, team, or group are made up of individuals. In order for the team, group or organisation to 'work together' and meet common goals they have to interact. The nature of how they interact, and the efficiency of the interaction, can severely affect the performance of that team. Thus, it is worth spending effort making the interaction effective. Now, some would argue that processes and tools can improve the quality of interactions. I have sympathy with such an argument for I think so too.

So why are process and tools less important than individuals and interactions?

In a nutshell "Tools support the process and the process supports the interactions of the individuals". That's why they are there.

In a team of one, or a collocated few, you can get away with out much in the way of a process, and the minimum of tools to build the system. As the number of people involved, the complexity or size of the information, the number of locations where the individuals are, the number of time zones,… grows, defined processes and tools can help to stay on top of the situation.

A danger is that a tool might grow to control the process and/or the process may grow to control the interactions of the individuals. Is that a bad thing? Well it can be if it reduces the value that is provided to the business by the interactions of the individuals. But there are often cases where a tool controls a process and adds significant value. For example, if your configuration management is complex a well chosen tool that drives the process can prevent a myriad of problems. As a second example, Extreme Programming [1] encourages heavy use of tools and process: "You will perform daily builds; You will run all automated tests suites; You will…"

So, defined process and expedient use of tools is not a bad thing. So I'll ask a different question. Why are individuals and interactions more important than process and tools? Sorry that is the same question. How about, how can a process or a tool reduce the value? Or better, how can a process or tool make it harder to add value?

At the heart of the problem is the fact that consultants and software vendors (including those of an Agile persuasion) make their money by selling processes and tools. It is, perhaps, too easy to take a process and/or set of tools without consideration of the benefits that they can supply and without later seeing whether any benefits have been supplied. To me this is the real purpose of this beatitude - think hard before committing to what can be expensive tools and processes. Very often an efficient team can improve the process at less cost.

The moral: Tools support the process and the process supports the interactions of the individuals - and don't you forget it!

Working Software over Comprehensive Documentation

Well obviously – duh! That was my reaction when I read this and I would not be surprised if it was yours - so why include this bi-attitude in the manifesto?

This time I'll start from the back end. Why have a go at comprehensive documentation?

So many processes get carried away creating comprehensive documentation and end up creating too much documentation. The creation of that documentation adds cost to the work and introduces latencies to feedback making it harder to respond to change and harder to collaborate with the customer. The documentation often duplicates information again adding to costs and latency of maintaining the information in two places. The excess of documentation, if not maintained is soon out of date diminishing its value. And oh even worse, out of date documentation can obviously cause problems making its value negative. In short much of the documentation either:

1. Adds only a little value
2. Adds value that is only short term value but is kept and maintained even when its value has diminished.
3. Adds value is but is duplicated in other locations
4. Adds value but is not kept up to date

So there are a few reasons why documentation may not be needed. But what documentation is needed? I'll put that another way. What deliverables provide long term value?

Requirements define what is needed from the software. Software is working if it fulfils the requirements. Tests are used to prove the requirements are fulfilled. In my mind that is what really matters in software development: requirements, software and tests.

So what is all this comprehensive documentation malarkey that the Manifesto refers to? It is High level designs, technical designs, user interface design, standards, use case models, class diagrams, architecture….

Between us we could probably name a hundred different types of documentation. Why do we need any of it? In my mind there are two key purposes: understanding and communication.

When you are faced with a new system you want to be able understand how you are going to solve the problem. You will be analysing and designing a system. This is the forward engineering part of your work.

If you are not working alone you will want to communicate this incite to others. The number and nature of individuals you have to interact in order to make this communication will influence the nature and amount of documentation created – but remember to consider how much will need to be maintained.

In order to understand how you are going to solve the problem you will need to understand what is already there. And you will want to communicate this understanding to anyone who, in the future, wants to evolve the system. This is the reverse engineering part of the work. In summary it is an art of:

1. Understanding and communicating what is needed (forward engineering)
2. Understanding and communicating what is there (reverse engineering)

To conclude, the manifesto is warning us against a costly pitfall in which we strive to provide comprehensive documentation that does not provide value.

Customer Collaboration Over Contract Negotiation

To me this is the hardest bi-attitude to argue in favour of convincingly but I'll try – starting with the supplier consumer model that this alludes to.

The supplier consumer model for software development has a simple but entrenched form. The consumer wants a system that does abc. The supplier says that will cost you £xyz (or $xyz or xyz or…). If the consumer is happy with the cost the supplier, well, supplies. Oh if only it were that simple!

In practice the definition of abc are not really known AND, even if the definition of abc where known, the cost £xyz is not, AND even if the requirements are understand there is a reasonable chance that they will have to change, AND even if the costs are understood there is a reasonable chance that the costs will change AND… It would be easy to rant on. I'll emphasise these perils by putting (some of) them in a list:

1. The requirements (abc) are not understood initially - a risk to the consumer
2. The requirements (abc) are liable to change - a risk to the consumer
3. The costs (xyz) are not fully understood - a risk to the supplier
4. The costs (xyz) may change - a risk to the supplier
5. We can't know if abc is what is needed until it is delivered an used - a risk to the consumer
6. People delay because they are reluctant to sign of the definition of requirements – (a cost to both parties)
7. It costs money to define (or redefine) the contract such that what is to be delivered is understood - a risk to both parties

This last point is interesting. Even though it is difficult to be precise in your definition of requirements, the consumer will not be happy parting with (or committing to part with) money unless they know what they are going to get. Equally the supplier will not be happy committing to deliver until they know what they are supposed to produce. So in order to be sure of what they are going to get/produce they define a contract. And then hope it is fulfilled.
So what is the alternative?
The problem, in my opinion, is that the requirements are part of the contract. The solution is Feedback. This simple notion is so important I will say it again – and louder. Feedback!
Instead of defining requirements fully up front and incorporating them into a contract, the contract does not specify the requirements at all (or at least not very much). Instead it specifies the approach. An approach that gives both parties opportunities to adapt to the perils listed above (and others not listed) while protecting the interests of both parties.

The work is then split into iterations. During each iteration, the consumer and supplier work together to find out how best to give value to the consumer for the money that will be spent during the next iteration. It is a very simple proposition but it seems to work very well.

Short iterations allow BOTH parties to get FEEDBACK and examine whether the requirements have been understood. If the requirements change, the change can be incorporated into the next iteration.

Since the contract defines the approach it doesn't have to change. But it can have a clause in which the consumer can terminate if value is not being provided or if they have enough value so far. This may not appeal to some suppliers but early drop out clauses could be introduced. Forgive me, I am acutely aware that I am not a lawyer and my contract speak is unlikely to be correct, but I hope you get a feel for how it could work.

In summary, the contract defines the approach, the requirements are defined by collaboration between the consumer and supplier, feedback is used to help both parties provide maximum value to the consumer.

And you never know everyone could be happy.

Responding to Change over Following a Plan

A project manager, when presented with this bi-attitude might say. "If you don't follow the plan how will you deliver?" or "If you don't follow the plan your project will deteriorate to anarchy!"

But there is a simple counter to this: "If you follow the plan how do make sure you are delivering what is needed?" Or put another way: "If you don't respond to change, how are you going to deliver what is needed? - You may deliver something but that something may not be needed anymore."

Change is a major obstacle to delivering what is needed – a major obstacle to providing value to the business. Many things can change during the life of a project:

1. Requirements
2. Costs
3. Budget
4. Staff
5. Management
6. Stock market
7. Laws
8. Resources
9. Priorities
10. Understanding of the problem.
11. Sickness

I could easily list loads and loads more, anyone one of which could render the plan invalid.

Let me get one thing straight. Planning is not a bad thing. Planning is a good thing. It is such a good thing that you should do it all the time. The problem is sticking to the plan rather than adapting the plan, re-planning, in the face of change.

The morale:

If you are going to plan, plan often. That way it will be easy to make sure you are providing value to the business. If you are

going to be planning often you need an approach to planning that is a low overhead to the project.

A First Amendment

I have now given my view on what matters in relation to the four bi-attitudes, what they mean and what I think the message is. Having done all that, there is so much more that could be said. For example, I have given no indication on how to provide a system in which you communicate your understanding of the system without creating unnecessary documentation. To do would take a book – fortunately such books exist, for example Fowler[2]

These bi-attitudes do not live in isolation – rather they are closely related and interlinked.

Working Software is for me the most important. But the value of the working software diminishes if it doesn't quite do what the business would like, thus we must respond to change. To respond to change we must collaborate with the customer. And in doing all of this we have individuals interacting while being supported by processes and tools.

Once again I am going to rephrase what I have said. Working Software is for me the most important in the list. For I have a value that both links them and sits above them.

Under Individuals and Interactions I talked about "the value that is provided to the business by the interactions of the individuals"

Under working software I discussed the documentation and deliverables that really add value to the consumer.

In customer collaboration I promoted the idea of "the consumer and supplier working together to find out how best to give value to the consumer for the money that will be spent".

[concluded at foot of next page]

Patterns in C – Part 5: REACTOR

by Adam Petersen `<adampetersen75@yahoo.se>`

This final part of the series will step outside the domain of standard C and investigate a pattern for event-driven applications. The REACTOR pattern decouples different responsibilities and allows applications to demultiplex and dispatch events from potentially many clients.

The Case of Many Clients

In order to simplify maintenance of large systems, the diagnostics of the individual subsystems are gathered in one, central unit. Each subsystem connects to the diagnostics server using TCP/IP. As TCP/IP is a connection-oriented protocol, the clients (the different subsystems) have to request a connection at the server. Once a connection is established, a client may send diagnostics messages at any time.

The brute-force approach is to scan for connection requests and diagnostics messages from the clients one by one as illustrated in the activity diagram in Figure 1.

Even in this strongly simplified example, there are several potential problems. By intertwining the application logic with the networking code and the code for event dispatching, several unrelated responsibilities have been built into one module. Such a design is likely to lead to serious maintenance, testability, and scalability problems by violating a fundamental design principle.

The Single Responsibility Principle

The single responsibility principle states that “a class should have only one reason to change” [1]. The goal of this principle is close to that of the open-closed principle treated below: both strive to protect existing code from modifications. When violating the single responsibility principle, a module gets more reasons to change and modifications to it become more likely. Worse, the different responsibilities absorbed by a single module may become coupled and interact with each other making modifications and testing of the module more complicated.

The single responsibility principle is basically about cohesion. It is useful and valuable on many levels of abstraction, not at least in a procedural context; simply replacing the word “class” with “function” enables us to analyze algorithms like the one above with respect to this principle.

Violation of the Open-Closed Principle

By violating the single responsibility principle, the module in the example above will be hard to maintain; it is code that one never wants to dig into in the future. Unfortunately, on collision course with that wish is the fact that the event loop above violates the open-closed principle [1]; new functionality cannot be added without modifying existing code. Related to our example, a diagnostics server typically allows a technician to connect and query stored information. Introducing that functionality would double the logic in the event loop. Clearly, this design makes the code expensive to modify.

From a Performance Perspective

To make things worse, the solution above fails to scale in terms of performance as well. As all events are scanned serially, even in case timeouts are used, valuable time is wasted doing nothing.

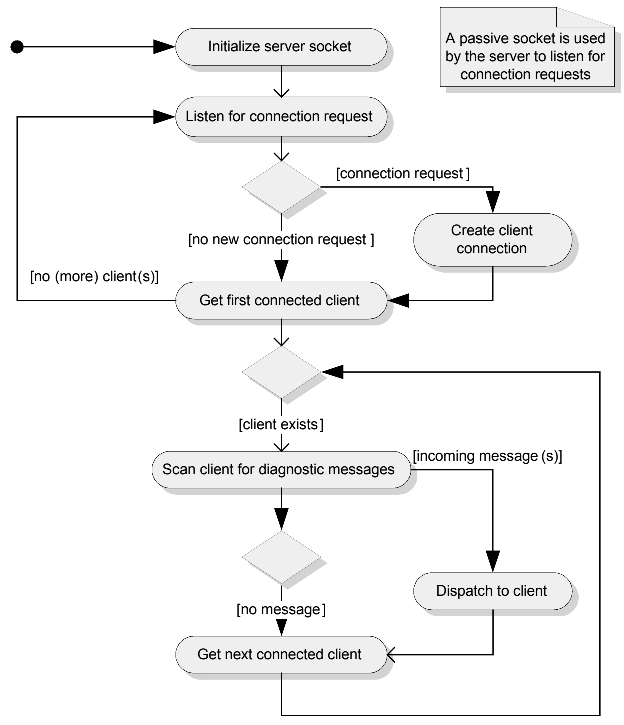

Initialize server socket
A passive socket is used by the server to listen for connection requests
Listen for connection request
[connection request]
[no new connection request]
Create client connection
[no (more) client(s)]
Get first connected client
[client exists]
Scan client for diagnostic messages
[incoming message (s)]
Dispatch to client
[no message]
Get next connected client

Figure 1: Eternal loop to scan for different events

The potential performance problem above may be derived from the failure of taking the concurrent nature of the problem into account. One approach to address this problem is by introducing multiple threads. The diagnostics server keeps its event loop, but the loop is now reduced to scan for connection requests. As soon as a connection is requested, a new thread is allocated exclusively for handling messages on that new connection.

The multithreading approach fails to address the overall design issue as it still violates both the single responsibility principle and the open-closed principle. Although the code for scanning and dispatching diagnostics messages is moved out of the event loop, adding a new server-port still requires modifications to existing code.

From a design perspective threads didn’t improve anything. In fact, even with respect to performance, this solution may due to context switches and synchronization actually perform worse than the initial single-threaded approach.

The sheer complexity inherent in designing and implementing multithreaded applications is a further argument for discarding this solution.

Problem Summary

Summarizing the experience so far, the design fails as it assumes three different responsibilities. This problem is bound to be worse as the design violates the open-closed principle, making modifications to existing code more likely.

`[continued from previous page]`

In responding to change I mentioned “obstacles to providing value to the business”.

There is a common theme. For me the most important is providing (maximum) value to the consumer.

If I want to add this to the manifesto as an amendment I will have to say what I value it more than.

I value it more than “on time on budget”.

Many organisations and process place emphasis on “On time on budget”. But on time on budget is easy to fulfil. Give yourself lots of time and lots of budget, whittle away both writing articles and playing Quidditch and then provide a quality product that might be what the business wants. But equally it might not be what the business wants and it may well not provide maximum value for the money spent. The problem is that the definition on time or on budget can and should be allowed to change.

So I would like to propose a first amendment to the manifest. A fifth bi-attitude:

- ***Providing value to the business*** *over on time on budget*

So what is your favourite?

Phran Ryder

Phran Ryder is Chairman of AgileNorth.org.uk – a non profit organisation for technical and business staff who wish to learn and share experience of becoming and being agile – details at: www.agilenorth.org.uk.

References

1 Beck, Kent with Cynthia Andres, *Extreme Programming Explained: Embrace Change*. Addison Wesley. 2004. ISBN: 0-321-27865-8

2 Fowler, Martin, *Refactoring: Improving the Design of Existing Code*. Addison Wesley. 1999. ISBN: 0-201-48567-2

Summarizing the ideal solution, it should scale well, encapsulate and decouple the different responsibilities, and be able to serve multiple clients simultaneously without introducing the liabilities of multithreading. The REACTOR pattern realizes this solution by encapsulating each service of the application logic in event handlers and separating out the code for event demultiplexing.

The REACTOR Pattern

The intent of the REACTOR pattern is: *The REACTOR architectural pattern allows event-driven applications to demultiplex and dispatch service requests that are delivered to an application from one or more clients* [2].

The roles of the involved participants are:

- `EventHandler`: An `EventHandler` defines an interface to be implemented by modules reacting to events. Each `EventHandler` own its own `Handle`.
- `Handle`: An efficient implementation of the REACTOR pattern requires an OS that supports handles (examples of `Handles` include system resources like files, sockets, and timers).
- `DiagnosticsServer` and `DiagnosticsClient`: These two are concrete event handlers, each one encapsulating one responsibility. In order to be able to receive event notifications, the concrete event handlers have to register themselves at the `Reactor`.
- `Reactor`: The `Reactor` maintains registrations of `EventHandlers` and fetches the associated `Handles`. The `Reactor` waits for events on its set of registered `Handles` and invokes the corresponding `EventHandler` as a `Handle` indicates an event.

Event Detection

In its description of REACTOR, *Pattern-Oriented Software Architecture* [2] defines a further collaborator, the `Synchronous Event Demultiplexer`. The `Synchronous Event Demultiplexer` is called by the `Reactor` in order to wait for events to occur on the registered `Handles`.

A synchronous event demultiplexer is often provided by the operating system. This example will use `poll()` (`select()` and Win32's `WaitForMultipleObjects()` are other functions available on common operating systems), which works with any descriptor.

The code interacting with `poll()` will only be provided as a sketch, because the POSIX specific details are outside the scope of this article. The complete sample code, used in this article, is available from my homepage [6].

Implementation Mechanism

The collaboration between an `EventHandler` and the `Reactor` is similar to the interaction between an observer and its subject in the design pattern OBSERVER [5]. This relationship between these two patterns indicates that the techniques used to realize OBSERVER in C [4] may serve equally well to implement the REACTOR.

In order to decouple the `Reactor` from its event handlers and still enable the `Reactor` to notify them, each concrete event handler must correspond to a unique instance. In our OBSERVER implementation, the FIRST-CLASS ADT pattern [3] was put to work to solve this problem. As all concrete event handlers have to be abstracted as one, general type realizing the `EventHandler` interface, `void*` is chosen as "the general type" to be registered at the `Reactor` (please refer to the previous part in this series - Reference [4] - for the rationale and technical reasons behind the `void*` abstraction). These decisions enable a common interface for all event handlers:

Listing 1 : Interface of the event handlers, `EventHandler.h`

```
/* The type of a handle is system specific -
this example uses UNIX I/O handles, which are
plain integer values. */
typedef int Handle;

/* All interaction from Reactor to an event
handler goes through function pointers with
the following signatures: */
typedef Handle (*getHandleFunc)
   (void* instance);
typedef void (*handleEventFunc)
   (void* instance);

typedef struct
{
  void* instance;
  getHandleFunc getHandle;
  handleEventFunc handleEvent;
} EventHandler;
```

Having this interface in place allows us to declare the registration functions of the Reactor.

Listing 2 : Registration interface of the Reactor, `Reactor.h`

```
#include "EventHandler.h"

void Register(EventHandler* handler);
void Unregister(EventHandler* handler);
```

The application specific services have to implement the EventHandler interface and register themselves using the interface above in order to be able to react to events.

Listing 3 : Implementation of a concrete event handler, `DiagnosticsServer.c`

```
#include "EventHandler.h"

struct DiagnosticsServer
{
   Handle listeningSocket;
   EventHandler eventHandler;

   /* Other attributes here... */
};
```

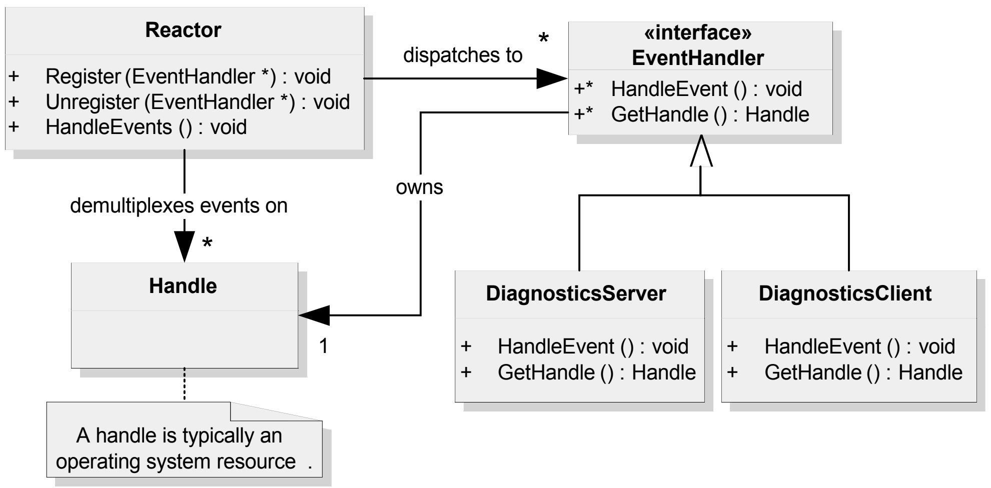


Figure 2: Structure of the REACTOR Pattern

```
/* Implementation of the EventHandler
interface. */
static Handle getServerSocket(void* instance)
{
  const DiagnosticsServerPtr server =
     instance;
  return server->listeningSocket;
}

static void handleConnectRequest(void*
instance)
{
  DiagnosticsServerPtr server = instance;

  /* The server gets notified as a new
connection request arrives. Add code for
accepting the new connection and creating a
client here... */
}

DiagnosticsServerPtr createServer
   (unsigned int tcpPort)
{
  DiagnosticsServerPtr newServer =
     malloc(sizeof *newServer);

  if(NULL != newServer) {
  /* Code for creating the server socket here.
     The real code should look for a failure,
     etc. */
      newServer->listeningSocket =
         createServerSocket(tcpPort);

      /* Successfully created -> register the
         listening socket. */
       newServer->eventHandler.instance =
          newServer;
       newServer->eventHandler.getHandle =
          getServerSocket;
       newServer->eventHandler.handleEvent =
          handleConnectRequest;

      Register(&newServer->eventHandler);
   }
   return newServer;
}

void destroyServer
   (DiagnosticsServerPtr server)
{
   /* Before deleting the server we have to
      unregister at the Reactor. */
   Unregister(&server->eventHandler);

   free(server);
}
```

REACTOR Registration Strategy

When implementing the concrete event handlers as a FIRST-CLASS ADT, the functions for creating and destructing the ADT serves well to encapsulate the registration handling. The advantage is the combination of loose dependencies with information hiding as a client does not even have to know about the usage and interactions with the Reactor.

Another attractive property is that the internals of the server, in our example the handle, is encapsulated within the `getServerSocket` function. Sure, we are giving the `Reactor` a way to fetch it, but the `Reactor` is considered a well-trusted collaborator and we are actively giving it access by registering our event handler. There is no way for any other module to mistakenly fetch the handle and corrupt the associated resource.

REACTOR Implementation

The details of the REACTOR implementation are platform specific as they depend upon the available synchronous event demultiplexers. In case the operating system provides more than one synchronous event demultiplexer (e.g. `select()` and `poll()`), a concrete Reactor may be implemented for each one of them and the linker used to chose either one of them depending on the problem at hand. This technique is referred to as link-time polymorphism.

Each Reactor implementation has to decide upon the number of reactors required by the surrounding application. In the most common case, the application can be structured around one, single Reactor. In this case, the interface in Listing 2 (`Reactor.h`) will serve well. An application requiring more than one Reactor should consider making the Reactor itself a FIRST-CLASS ADT. This second variation complicates the clients slightly as references to the Reactor ADT have to be maintained and passed around in the system.

Independent of the system specific demultiplexing mechanism, a Reactor has to maintain a collection of registered, concrete event handlers. In its simplest form, this collection may simply be an array. This approach serves well in case the maximum number of clients is known in advance.

Listing 4: Implementation of a Reactor using `poll()`, `PollReactor.c`

```
#include "Reactor.h"
#include <poll.h>
/* Other include files omitted... */

/* Bind an event handler to the struct used to
   interface poll(). */
typedef struct
{
   EventHandler handler;
   struct pollfd fd;
} HandlerRegistration;

static HandlerRegistration
registeredHandlers[MAX_NO_OF_HANDLES];

/* Add a copy of the given handler to the
first free position in registeredHandlers. */
static void addToRegistry(EventHandler*
handler);
  /* Identify the event handler in the
registeredHandlers and remove it. */
static void removeFromRegistry(EventHandler*
   handler);

/* Implementation of the Reactor interface
used for registrations.*/
void Register(EventHandler* handler)
{
   assert(NULL != handler);
   addToRegistry(handler);
}

void Unregister(EventHandler* handler)
{
   assert(NULL != handler);
   removeFromRegistry(handler);
}
```

Invoking the Reactor

The reactive event loop is the core of the Reactor and its responsibilities are to control the demultiplexing and dispatch the detected events to the registered, concrete event handlers. The event loop is contained within the `HandleEvents()` function and is typically invoked from the `main()` function.

Listing 5: Client code driving the reactor

```
int main(void){
   const unsigned int serverPort = 0xC001;
   DiagnosticsServerPtr server =
      createServer(serverPort);
   if(NULL == server) {
      error("Failed to create the server");
   }
```

```
    /* Enter the eternal reactive event loop.*/
    for(;;){
       HandleEvents();
    }
 }
```

Before investigating the implementation, which include file should provide the declaration of the `HandleEvents()` function? Unfortunately, adding it to the file in Listing 2 (`Reactor.h`) would clearly make that interface less cohesive; the registration functions models a different responsibility than the event loop. A solution is to create a separate interface for the event loop. This interface is intended solely for the compilation unit invoking the event loop.

Listing 6: Interface to the event loop, `ReactorEventLoop.h`

```
void HandleEvents(void);
```

Despite its simplicity, this separation solves the cohesiveness problem and shields clients from functions they do not use. This technique of providing separate interfaces to separate clients is known as the interface-segregation principle [1].

Implementing the Event Loop

With the interface in place, we can move on and implement the event loop itself. The listing below extends Listing 4.

Listing 7: Example of a reactive event loop, `PollReactor.c`

```
#include "ReactorEventLoop.h"
/* The code from Listing 4 go here
(omitted)... */

/* Add a copy of all registered handlers to
the given array. */
static size_t buildPollArray
   (struct pollfd* fds);

  /* Identify the event handler corresponding
to the given descriptor in the
registeredHandlers. */
static EventHandler* findHandler(int fd);

static void dispatchSignalledHandles(
      const struct pollfd* fds,
      size_t noOfHandles)
{
   /* Loop through all handles. Upon detection
of a handle signalled by poll, its
corresponding event handler is fetched and
invoked. */
   size_t i = 0;
   for(i = 0; i < noOfHandles; ++i) {
      /* Detect all signalled handles and
invoke their corresponding event handlers. */
      if((POLLRDNORM | POLLERR) &
         fds[i].revents) {
         EventHandler* signalledHandler =
         findHandler(fds[i].fd);

         if(NULL != signalledHandler){
            signalledHandler-> handleEvent
            (signalledHandler->instance);
         }
      }
   }
}
/*Implementation of the reactive event loop.*/
void HandleEvents(void)
{
   /* Build the array required to interact
with poll().*/
   struct pollfd fds[MAX_NO_OF_HANDLES] = {0};
   const size_t noOfHandles =
      buildPollArray(fds);
   /*Invoke the synchronous event demultiplexer*/
      if(0 < poll(fds, noOfHandles, INFTIM)){
      /* Identify all signalled handles and
      invoke the event handler associated with
      each one. */
         dispatchSignalledHandles(fds,
             noOfHandles);
      }
      else{
         error("Poll failure");
      }
   }
```

The example above lets each element in the collection maintain a binding between the registered event handler and the structure used to interact with `poll()`. One alternative approach is to keep two separate lists and ensure consistency between them. *Pattern-Oriented Software Architecture* [2] describes another, system specific alternative: in a UNIX implementation using `select()`, the *array is indexed by UNIX I/O handle values, which are unsigned integers ranging from 0 to FD_SETSIZE-1*.

Returning to the example, by grouping the registration and the poll-structure together, the array used to interact with `poll()` has to be built each time the reactive event loop is entered. In case the performance penalty is acceptable, this is probably a better choice as it enables a simpler handling of registrations and unregistrations during the event loop.

Handling New Registrations

In my previous article [4], I discussed strategies for managing changed registrations during the notification of observers. The alternative of forbidding changed registrations is, unlike the OBSERVER pattern, not an option for a REACTOR implementation. In the example used in this article, the server reacts to the notification by creating a new client, which must be registered shall it ever be activated again. This leaves only one option for a REACTOR implementer: ensure that it works.

One solution is to maintain a separate array to interact with the synchronous event demultiplexer as illustrated above. This array is never modified in the event loop. However, this solution has the consequence that handles unregistered during the current event loop may be marked as signalled in the separate array. The code simply has to check for this case and ignore such handles, as illustrated by the function `dispatchSignalledHandles` in Listing 7 above.

The code uses the handle alone as identification. In cases resources are disposed and created during the same event loop, there is, depending on platform, a possibility that the handle ID's are re-used; a signalled handle in the copy may belong to an unregistered event handler, but due to a new registration using the re-cycled handle ID, the new event handler may be erroneously invoked. If this is an issue, the problem may be prevented by introducing further book-keeping data. For example, a second array containing the identities of the handles unregistered during the current event loop makes it possible to identify the case described above and thus avoid it.

More Than One Type of Event

The design in the example above does only allow applications to register for one type of event (read-events). The event type is even hardcoded in the Reactor and it is a simple solution sufficient for applications without any need for further event detection. The REACTOR pattern, however, is not limited to one type of event. The pattern scales well to support different types of events.

Pattern-Oriented Software Architecture [2] describes two general strategies for dispatching event notifications:

- Single-method interface: An event handler is notified about all events through one, single function. The type of event (typically in the form of an `enum`) is passed as a parameter to the function. The disadvantage of this approach is that it sets the stage for conditional logic, which soon gets hard to maintain.
- Multi-method interface: In this case, the event handler declares separate functions for each supported event (e.g. `handleRead`, `handleWrite`, `handleTimeout`). As the Reactor has the knowledge of what event occurred, it invokes the corresponding function immediately, thus avoiding placing the burden on the event handler to re-create the event from a parameter.

Comparision of REACTOR and OBSERVER

Although the mechanisms used to implement them are related, there are differences between these two patterns. The main difference is in the

[concluded at foot of next page]

When Worlds Collide 2 - Circuit Switch Telephony and Packet Switch Networks

Mark Easterbrook `<mark@easterbrook.org.uk>`

Telephones have been around a very long time. Alexander Graham Bell patented the telephone in 1876, so the telephone was over a century old before packet switch networking escaped from the laboratory. Hence the telephony industry is one of the most mature in the world of technology. In comparison, the computer industry is young and immature - if you were to anthropomorphise it you might see a troublesome teenager emerging from a difficult puberty. Asking the two to work closely together is bound to be interesting.

Digital Telephony Primer

Unless you work in the telecommunications industry your knowledge of how it works is likely to be quite limited, therefore it is worth describing the basics.

The analogue signal from your home or work phone is frequency limited to about 3kHz (very little content of speech is above this) and converted to digital using 8 bit samples, 8000 times a second, giving a bit stream of 64000 bits per second (64kbit/s). The telephone network consists of digital circuits capable of carrying 64kbit/s signals, and switches that can connect one circuit to another to form end-to-end paths. The basic building block of the telephony network is therefore the 64kbit/s data stream. An end-to-end circuit provides a dedicated path, it has a fixed latency, a fixed bandwidth, and a fixed quality of service. The resources allocated at every point in the network are fixed for the duration of the call. The number of circuits is also fixed so that once they are all in use all further calls have to be rejected; there is no option to adjust the balance between quality and numbers of calls.

Mobile networks are similar, although advances in processing power in the handset allow more efficient use of bandwidth in the access network so that the voice circuit between the handset and the core network only needs 16kbit/s.

Telephony networks also have a signalling infrastructure to control the calls carried over these 64kbit/s circuits. This is a message-based network designed specifically for call control and is often carried on the same physical media as the voice circuits. Initially this signalling was just restricted to basic call set-up and clear down - passing the called and caller numbers and indicating when the call was answered and finished. As the number of services provided by the telephony networks has increased so has the number of type of messages carried by the signalling network so now it is far more than just call handling. The short message service (SMS) between mobile phones is an example of data carried by this signalling network that is not related to a telephony call.

Circuit Switch telephony networks

Telephony networks have developed slowly over more than a century, and until recently have been built, owned, and controlled by mostly state-owned monopolies. This long heritage has led to the characteristics of the modern telephony network that we are familiar with today:

- **Dumb terminals**. The modern telephone has changed surprisingly little since the 19th Century. It contains a microphone and speaker for the user to communicate with the other end of the call, and a method to signal to the network. The only significant change has been the replacement of the rotary dial with a keypad. You can take a hundred-year-old phone and use it to make a call to the latest model of mobile phone, or vice versa - there are very few other technologies that have remained compatible for such a long time. Even modern devices, such

[continued from previous page]

notification mechanism. As a Subject changes its state in an OBSERVER implementation, all its dependents (observers) are notified. In a REACTOR implementation, this relationship is one to one – a detected event leads the `Reactor` to notify exactly one dependent (`EventHandler`).

One typical liability of the OBSERVER pattern is that the cohesion of the subject is lowered; besides serving its central purpose, a subject also takes on the responsibility of managing and notifying observers. With this respect, a Reactor differs significantly as its whole raison d'être is to dispatch events to its registered handlers.

Consequences

The main consequences of applying the REACTOR pattern are:

1. *The benefits of the single-responsibility principle*. Using the REACTOR pattern, each of the responsibilities above is encapsulated and decoupled from each other. The design results in increased cohesion, which simplifies maintenance and minimizes the risk of feature interaction. As the platform dependent code for event detection is decoupled from the application logic, unit testing is greatly simplified (it is straightforward to simulate events through the `EventHandler` interface).
2. *The benefits of the open-closed principle*. The design now adheres to the open-closed principle. New responsibilities, in the form of new event handlers, may be added without affecting the existing event handlers.
3. *Unified mechanism for event handling*. Even if the REACTOR pattern is centred on handles, it may be extended for other tasks. *Pattern-Oriented Software Architecture* [2] describes different strategies for integrating the demultiplexing of I/O events with timer handling. Extending the Reactor with timer support is an attractive alternative to typical platform specific solutions based upon signals or threads. This extension builds upon the possibility to specify a timeout value when invoking the synchronous event demultiplexer (for example, `poll()` allows a timeout to be specified with a resolution of milliseconds). Although it will possibly not suit a hard real-time system, a Reactor based timer mechanism is easier to implement and use than a signal or thread based solution as it tends to avoid re-entrance problems and race-conditions.
4. *Provides an alternative to multithreading*. Using the REACTOR pattern, blocking operations in the concrete event handlers can typically be avoided and consequently also multithreading. As discussed above, a multithreaded solution does not only add significant complexity; it may also prove to be less efficient in terms of run-time performance. However, as the Reactor implies a non pre-emptive multitasking model, each concrete event handler must ensure that it does not perform operations that may starve out other event handlers.
5. *Trades type-safety for flexibility*. All concrete event handlers are abstracted as `void*`. When converting a void-pointer back to a pointer of a concrete event handler type, the compiler doesn't have any way to detect an erroneous conversion This potential problem was faced in the implementation of the OBSERVER pattern [4] and the solution is the same for the REACTOR: define unique notification functions for each different type of event handler and bind the functions and event handler together using an `EventHandler` structure as described in Listing 1.

Summary

The REACTOR pattern simplifies event-driven applications by decoupling the different responsibilities, encapsulated in separate modules.

There is much more to the REACTOR pattern than described in this article. Particularly several variations that all come with different benefits and trade-offs. For an excellent in-depth treatment of the REACTOR and other patterns in the domain, I recommend the book *Pattern-Oriented Software Architecture, volume 2* [2].

Adam Petersen

References

1. Robert C. Martin: "*Agile Software Development*", Prentice Hall
2. Schmidt, Stal, Rohnert, Buschmann: "*Pattern-Oriented Software Architecture, volume 2*", Wiley
3. Adam Petersen, "Patterns in C, part 1", *C Vu 17.1*
4. Adam Petersen, "Patterns in C, part 4: OBSERVER", *C Vu 17.4*
5. Gamma, E., Helm, R., Johnson, R., and Vlissides, J, "*Design Patterns*", Addison-Wesley
6. The complete REACTOR sample code used in this article, `www.adampetersen.se`

Acknowledgements

Many thanks to Drago Krznaric and André Saitzkoff for their feedback.

as the modem, facsimile, or DECT handset, are based on the same three components, transmit sound, receive sound, and a signalling interface.

- **Intelligent networks**. As services have been developed for the telephony world they have mostly been introduced into the core network. With the intelligence in the network, introduction and upgrade of a service does not require a change to the subscriber's equipment, and the hardware, software, configuration, and security is under control of the network operator. This has been a great benefit to the network operators in the form of increased revenue, to the subscriber in the form of a rich menu of services, and to third parties in that they can sell services to everyone with a phone, no matter how basic.
- **Metered charging**. Calls are usually charged by time from the time of answer to the end of the call. This means that the cost of a call is transparent to the end user. Today's networks support fixed price or unmetered charging, but these are a recent development and are only reluctantly implemented by network operators as they threaten existing revenue streams.
- **Distance charging**. Long distance calls have always cost more than short distance calls. This is based on the idea that the more pieces of equipment, or exchanges, the call passes through, the more expensive it is for the network operator, and that expense is passed to the end user. Additionally, interconnect charges are a valuable source of revenue so operators charge a premium for other operators to access their network, and therefore international and cross-network calls are relatively expensive.
- **Fixed Bandwidth**. The building block of the telecom network is the 64kbit/s channel. This has led to data connections using the circuit switch network reaching their limit at about 56kbit/s for analogue modems and 64kbit/s for end-to-end digital. Exceeding this limit requires more intelligence at the end terminals, for example, 128kbit/s ISDN is achieved by concatenating two 64kbit/s channels (charged by the Telco as two separate calls) and video calls by using 6x64kbit/s channels. Although there is some support in the network to route the concatenated channels via the same path, most of the functionality is implemented at the data terminal.
- **Fixed Latency**. The nature of the dedicated 64kbit/s channel is that every bit takes the same time to transverse the network, and therefore the latency for a connection remains constant. This is particularly important for voice communications as the human ear/brain is reasonably tolerant of delay, but not of jitter. The worst-case latency within a network in a medium sized country (e.g. BT in the UK) can be as little as 10-15ms.
- **Partitioned Signalling**. The circuit switch signalling network is partitioned into core network signalling and access network signalling. In the core network Signalling Scheme Number 7 (SS7 or C7) provides a trusted message transfer and relay mechanism. Access to the messaging in the core network is protected by having protocol conversion from an access network (typically ISDN signalling) that validates message content as well as providing supplementary services.
- **Standards**. The international telephony networks conform to nationally and internationally ratified standards from organisations such as ITU-T, ANSI, and ETSI. Standards are agreed at national or international level and then implemented by manufacturers and operators. The process is hierarchical and participation expensive, leading to domination by a small number of large organisations. There is also a historical global division resulting in a different set of standards in North America and (most of) the rest of the World.

Packet Switch Networks

- **Intelligent Terminals**. Even the first networked computers were substantially more complex than a telephone. Even just providing the network interface required considerably more hardware and software than even the most advanced phones of the day. With intelligence in the terminal equipment, new services and features require modification to that equipment, resulting in costly and time-consuming upgrades. When a new service is rolled out, it is only available to those end users with the equipment that supports it, or those who are willing to upgrade their equipment.
- **Dumb network**. In packet switch networks the network is purely a transport mechanism. Most of the intelligence that seems to be in the network is actually provided by devices attached to the network, and each terminal device needs to know how to access network resources.
- **Per packet or bandwidth charging**. The charging model for access to packet switch networks traces its origins to the government (military) or academic use, which is often perceived as free. In practice, because most packet networks have a high infrastructure cost and low per-use costs, charging has been on a bandwidth basis, except where bandwidth is scarce when per-packet charging encourages efficient use. Where access to a packet switch network such as the Internet is provided by another network, such as dial up telecom access, the access charge is that of the access network.
- **Distance independent charging**. It is rare in the packet switch world to be charged by distance. Users of the Internet are often unaware of the location of the service they are accessing, so charging by distance could not be transparent and therefore would be unacceptable to most users.
- **Flexible bandwidth**. The usual method of sharing the bandwidth over packet switch networks is on a first-come first-served packet-by-packet basis. This means that a heavy user such as a batch file transfer will significantly affect a light traffic interactive user, and that the available bandwidth can vary during the lifetime of a connection.
- **Variable Latency**. Another consequence of the shared bandwidth is the variations in latency depending on the type of traffic sharing the bandwidth. This makes the transport of voice over contemporary packet switch networks a hit and miss affair.
- **Transmission Delays**. The packetising delay and anti-jitter buffering alone is often more than the end-to-end delay of circuit switch networks. There is a trade off between packet size and transmission delay as larger packets make more efficient use of the bandwidth at the cost of delayed transmission. Mixing bandwidth efficient traffic and latency-sensitive traffic on the same transport network requires more intelligence at intermediate nodes.
- **End-to-end signalling**. In most packet switch networks the intermediate nodes only deal with destination routing decisions, leaving all higher-level protocols to the endpoints. With the exception of HTTP, it is rare to validate the communication at any point within the network.
- **Signalling Standards**. The vast majority of packet switching protocols are defined by the IETF (Internet Engineering Task Force) via RFC (Request for comments) documents. This is an "implement first, then document what works" co-operative method of standardisation that allows anyone to partake without significant barriers to entry. Standards are mostly global although they tend to have a North American cultural and language bias.

Collision

The two worlds of circuit switch and packet switch are increasingly meeting and overlapping, and both are highlighting the other's limitations when in the wrong domain:

- In the telecom arena the demands of packet over circuit is only possible by abandoning the current lucrative charging model and therefore disrupting the business model of traditional Telcos. The drive to extract higher transfer rates of data over the telephone access network has reached a limit at 56-64Kbits/s per circuit, and end users are demanding much more. The Telcos have always provided high bandwidth point-to-point circuits for business, but at a price. The charge for a 2Mbit/s E1 (1.5Mbit/s T1 in North America) is charged at thousands of pounds or dollars per month, depending on location and distance. Telcos have the choice of ignoring the demand and watching someone else take their market, or lowering the price and seeing the revenue stream dramatically reduce.
- The computer industry has until very recently always been forced to use the Telcos for anything other than on-campus connections. In many developed countries the stranglehold of the monopolies prevented companies from even linking two of their own buildings if they didn't own the land between, no matter how narrow. This has made wide area networks expensive and time-consuming to install and run, leading to frustration and resentment of the Telco's monopoly position. It is hardly surprising that the computer networking community, and more recently computer users, have taken every opportunity to circumvent or eliminate the Telco's networks. At the extreme this has led to the technically crazy and inefficient use of voice over IP packet over digital voice circuit!

It is not surprising then, that the telecom industry is looking at moving its network infrastructure to packet switch, and the computer industry is increasingly providing services previously the exclusive domain of the Telcos.

For the telecom industry, packet switch allows them to satisfy the demands of IP based equipment connecting over their networks, carry traditional telecom traffic, and introduce new services to open up new sources of revenue. Unfortunately, the characteristics that make telecom networks reliable and predicable are mostly not present in packet networks, and retrofitting them is proving challenging.

Similarly, for the computer industry, retrofitting the reliable and predictable performance that has resulted from a carefully regulated telecom environment to their anarchical culture feels like a paradigm change too far.

Quality of Service (QoS)

The quality of service provided by the fixed line telecom networks in all developed countries is taken by granted by most users to such an extent that customers have become intolerant of dropped calls, lack of dial tone, and expect to get through first time, every time. Compare this with the computer industry where crashes are tolerated, perhaps almost expected, and modems dropping the line, missing web pages, and unavailability are taken for granted. The idea of a telephone exchange running for 25 years without a reboot is so foreign to the computer community that they have difficultly believing such things are possible. There is therefore a gulf between the expectations of telephone users and computer users, even when they are the same people. This gulf could be bridged by improving quality of service in computer networks up to that expected by telecom users, or reducing the expectations of the telecom users. Fortunately for the telecom industry, two technologies have become widespread in the last decade that have done much to lower the expectations of the telephone user: mobile phone networks and IVR (interactive voice response) systems fronting call centres. But can the quality of service in packet switch networks be improved significantly to allow them to match the now reduced expectations of the telephone user?

Voice over IP (VoIP)

VoIP is a much-misused term, often being confused with Internet Telephony. Although the technology has been around for many years, QoS issues are hampering its deployment. In order to provide a QoS comparable with the existing telecom networks, the underlying IP network needs to be carefully designed and managed, particularly with regard to capacity and shared traffic. In practice VoIP networks need to be grossly over-specified and dedicated to VoIP traffic, especially when using IPv4 (IPv6 solves some, but not all, of the QoS issues). It is simply not possible to just piggy-back voice traffic on an existing IP network and expect it to work reliably; either the network needs to be extensively upgraded, or a new VoIP network commissioned, both of which negate any perceived cost advantage of using an IP network. Despite the difficulties, VoIP is slowly being deployed, albeit mainly in two locations, PABXs, and IP islands in core telecom networks, both of which are controlled environments where QoS and other issues can be managed.

VoIP and the PABX

If a company needs to upgrade both its LAN and internal telephone infrastructure, there are significant cost-savings, both up-front and on going, by converging the two. Vendors offering solutions for the integrated corporate market will dominate a Google search for VoIP, and technical news feeds frequently relay press releases of companies making this change. Although this closed environment is a success story for VoIP, it is not the VoIP technology that is driving it, but the cost-savings from avoiding duplication, and often the biggest savings are in the physical wires rather than the protocols carried by them. It is highly likely that VoIP will be the dominant technology for providing voice services in the SME space by the end of the decade.

VoIP islands in core networks

While in most of the VoIP world the publicity and hype precedes the implementation, the major telephone operators are quietly but steadily installing VoIP networks, if not as islands in their core network, at least in their test plant. My interpretation is that they believe VoIP is going to very important in the near future, but there are still significant technological, logistical and financial barriers still to break down, and that giving out too much information about the technology they are using would allow their competitors, especially the new entrants to the market, to piggy-back their research. Similarly, the major telecom equipment manufacturers, once you delve beyond the marketing hype, are vague about the direction they are taking, lest the big IP equipment manufacturers steal their market.

Internet Telephony

Most references to VoIP are actually referring to using the Internet for cheap telephone calls. On the surface this seems an easy way to avoid paying Telco charges, especially for long distance and international calls. In practice it is small niche application beset with problems. The variable, sometimes long, and unpredictable propagation delays of the Internet result in a low quality relegated to those who are willing to trade quality for low price. Unless the person you are calling is in one of the few areas with free Internet to telephone network gateways, you are limited to calling people who are close to their powered-up computer.

Despite these difficulties, a number of companies are providing Internet Telephony products. Noticeably the big players, such as BT with Broadband Voice in the UK, are by-passing existing desktop computers and supplying dedicated boxes plugged into the customer's LAN and providing interfaces to the public telephone network, albeit not at zero cost.

The Catch – Signalling

VoIP is easy. Taking a digital voice stream, putting it into IP packets, and pulling it out at the other end is almost trivial, and the necessary supporting features such as codecs, jitter buffers, and echo cancellation are available both in hardware and as software algorithms. The element that makes VoIP usable but technically complicated is the signalling. Both the traditional telephony and the emerging IP telephony worlds have many copious standards, but whereas the traditional telephony world is in general agreement in which standards and options to apply where, there is no consensus in the IP world and each vendor is pushing its own favourite. Many of the IP telephony signalling protocols were developed before the current explosion in public internet use and the subsequent security and abuse problems, and as a consequence, do not co-exist well with the partitioning of the Internet such firewalls and NAT routers.

Security

In its spring 1999 issue, the hacker magazine "2600" published an article entitled "SS7 explained" in which author Friedo describes in detail how SS7 works. He explains: "the hackability of SS7 does not at first appear possible, unless someone could figure out how to interface directly with the SS7 network". Telecom service providers have been very protective about their internal system since the early 70s when John Draper discovered a toy whistle allowed users to circumvent billing systems for long-distance calls in the US and the resultant development of the so-called "blue boxes" sold to make it easy for end users to phreak the network. Even with the proliferation of mobile telephony networks and the licensing of many small operators, the security of the public telephone network is many orders of magnitude better than the Internet and the other IP networks connected to it. The public telephony network not only provides access to the emergency services but also provides many other critical links such as intruder detection alerts. The reliability and availability of the telephone network really is a life and death matter. It is simply not possible for IP based networks to replace the existing circuit switch telephony networks unless the security of IP is improved by orders of magnitude. This either requires an IP telephony network completely separate from the existing IP data network, which negates much of the advantage of an integrated system, or there needs to be a landmark change in how IP networks are deployed and secured.

The Outlook

There is already enough momentum in the direction that telephony and data networks are moving that by the end of the decade it will be impossible to tell where one network type ends and the other starts, and a time when there is no longer any concept of separate networks for telephony and computers is not far off. This brings great challenges all of us: The telephony world needs to shed the legacy of its monopoly position and gentle pace of technological change, particularly in the way it charges its customers and rations access to new technology. The computer industry needs to take a grown-up attitude to reliability, availability, and security; reboots, denial of service and viruses are simply not acceptable when dealing with universal public services and life and death situations. The most popular cliché at the moment is "wake-up call" - if you are in the telephony or data networks industry, this is yours.

Mark Easterbrook

Tracking Exceptions in Web Services with GUIDs

Matthew Skelton `<matthew.skelton@gmail.com>`

Synopsis

This article demonstrates a technique for tracking exceptions across process boundaries in distributed systems using Globally Unique Identifiers (GUIDs); data from log files, bug reports, and on-screen error messages can then be correlated, allowing specific errors to be pinpointed.

Particular attention is paid to XML Web Services, with additional reference to DCOM, CORBA, Java/RMI and .Net Remoting..

The examples are given in C#, but the technique can be applied easily to Java and other languages.

Introduction

When dealing with modern distributed applications, it is often very useful to track an exception as it crosses process and machine boundaries. A comprehensive picture of the "distributed error" can then be constructed from log files of the different applications and components affected.

Languages such as C# and Java have an inbuilt mechanism to provide detailed error information – the Exception Stack Trace – which helps greatly to identify the source of runtime application errors. However, this information in itself does not always provide the full picture of an error condition.

The approach outlined here uses GUIDs (see Sidebar: GUIDs) effectively to "tag" exceptions before they leave one part of an application and appear in another. This allows exceptions to be tracked irrespective of whether the application is stand-alone or distributed, and irrespective of the transport, protocol or component architecture used.

XML Web Services: A Very Brief Introduction

XML Web Services is the name given to the latest class of middleware technologies designed to provide cross-platform Remote Procedure Calls (RPC). Previous technologies such as DCOM, CORBA and Java/RMI all have strengths, but often suffer from implementation difficulties, and none is really both platform- and language-independent [10], [11].

XML Web Services are described concisely in [16] as follows:

Web services are a new breed of Web application. They are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Web services perform functions, which can be anything from simple requests to complicated business processes... Once a Web service is deployed, other applications (and other Web services) can discover and invoke the deployed service.

The increasing success of XML Web Services, and SOAP [9] in particular, as the "glue" for cross-platform RPC can be put down to several factors, including:

- the technology is "platform-agnostic", relying only on XML as the data exchange format
- there is no explicit transport protocol (most implementations use HTTP, but SMTP or other protocols are also valid [13])
- the use of these standard protocols allows Web Services to be accessed behind a firewall (Port 80 for HTTP will usually be left open, for example)
- Web Services can be discovered and used automatically using UDDI [14], due to Web Services being "self-describing" via WSDL [15].

All these properties make XML Web Services very attractive for building scalable, flexible distributed applications. For a more comprehensive introduction to XML Web Services, see [8].

SOAP

The Simple Object Access Protocol (SOAP) [9] is probably the most common dialect used by XML Web Services. The W3 introduction [17] to SOAP states:

SOAP is fundamentally a stateless, one-way message exchange paradigm, but applications can create more complex interaction patterns (e.g., request/response, request/multiple responses, etc.) by combining such one-way exchanges with features provided by an underlying protocol and/or application-specific information.

Although SOAP does not specify a transport protocol, HTTP is normally used. With the resultant "SOAP-HTTP Binding", XML Web Services using SOAP and HTTP rely on message exchange using a combination of an HTTP Header and SOAP (XML) payload.

GUIDs

A GUID (pronounced like "squid") is a Globally Unique IDentifier, and is normally represented in string form something like this: `DCF70619-01D8-42a9-97DC-6005F205361A`. The GUID (also known as UUID – Universal Unique IDentifier) is an IETF standard, as defined in RFC 4122 [1]. The .Net Framework documentation for `System.Guid` [4] has this to say:

A GUID is a 128-bit integer (16 bytes) that can be used across all computers and networks wherever a unique identifier is required. Such an identifier has a very low probability of being duplicated.

GUIDs are generated using a variety of data, such as the current date & time, the IEEE 802 (MAC) address of the machine's network card, etc.[2]. This is designed to ensure that the likelihood of generating the same GUID twice is very small.

There are UUID/GUID implementations for many different languages and platforms (see Sidebar: GUID Implementations), and even an online GUID generator [3]. In addition, it is easy to identify GUIDs written in the standard string representation (above) using Regular Expressions.

Note: UUIDs/GUIDs should not be confused with GIDs/UIDs on *nix systems!

A SOAP message to retrieve (say) the author and title of a book might look something like this:

```
POST /cgi-bin/book-info.cgi HTTP/1.1
MethodName: GetBookDetails
MessageType: Call
Content-Type: text/xml-SOAP

<GetBookDetails>
  <ISBN>0201615622</ISBN>
</GetBookDetails>
```

Figure 1 - SOAP Request

Notice the standard HTTP POST request (first five lines), and that the SOAP payload (the `GetBookDetails` element) is simply XML.

The response from the Web Service might look like this:

```
200 OK
Content-Type: text/xml
Content-Length: 115

<GetBookDetailsResponse>
  <author>
    Herb Sutter
  </author>
  <title>
    Exceptional C++
  </title>
</GetBookDetailsResponse>
```

Figure 2 - SOAP Response

The XML payload contains a root element `GetBookDetailsResponse` that is the response to the original `GetBookDetails` request. Real-world SOAP messages would be somewhat more complex, but the principle above remains. For a much more comprehensive overview of SOAP, see [10].

SOAP deals with errors using a `Fault` response element. Any errors encountered by the Web Service, either in the request itself, or during the processing of the request, are detailed in the `Fault` element. For example, if the `GetBookDetails` request above failed due to an unknown ISBN, the XML payload of the response might look something like this:

```
<GetBookDetailsResponse>
  <fault>
    <faultcode>700</faultcode>
    <faultstring>Processing Error</faultstring>
  <runcode>1</runcode>
    <details>
      <Message>
        ISBN Not Recognised!
      </Message>
    </details>
  </fault>
</GetBookDetailsResponse>
```

Figure 3 - SOAP Fault

The `/fault/details` node contains a `Message` element, with an explanation of the error. (Note: in practice, XML Namespaces would be used for both the request and the response XML: these have been omitted for clarity.)

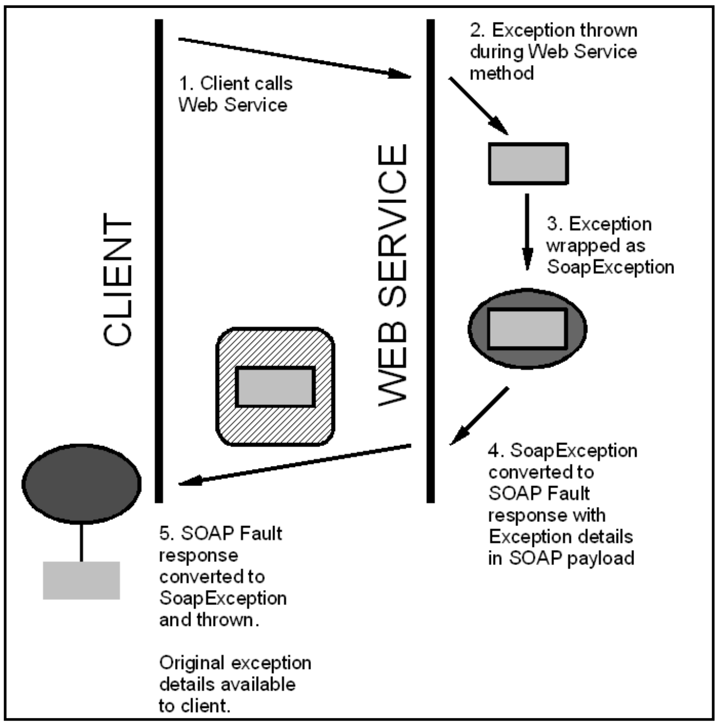
We will return to the SOAP `Fault` shortly.

SoapExceptions in the .Net Framework

The Microsoft .Net Framework simplifies many of the implementation details of SOAP web services by classes in the `System.Web.Services` namespace [20], in particular, the `WebService` class, from which – by default – all other SOAP Web Services in .Net are derived.[1]

One of the most useful aspects of the Framework is that a SOAP Fault response is converted automatically to a `System.Web.Services.Protocols.SoapException` [18], which is thrown in the context of the calling client. Details of the error contained in the SOAP `Fault` element are made available as properties of the `SoapException` instance. [2]

This conversion also works in the opposite direction: a `SoapException` escaping from the Web Service is converted automatically into a SOAP `Fault` response. In fact, the .Net Framework ensures that only exceptions of type `SoapException` escape from a Web Service call: if an uncaught exception raised within a Web Service method is not a `SoapException`, the Framework throws a new `SoapException`, storing the uncaught exception as its `InnerException`. Details of the original exception are extracted into the SOAP `Fault/details` element.



1. Client calls Web Service
2. Exception thrown during Web Service method
CLIENT
WEB SERVICE
3. Exception wrapped as SoapException
4. SoapException converted to SOAP Fault response with Exception details in SOAP payload
5. SOAP Fault response converted to SoapException and thrown.
Original exception details available to client.

Figure 4 - Mapping SoapExceptions to SOAP Faults

Figure 4 shows how information about an error in the Web Service method is transmitted back to the calling client. Thus, if the Web Service were to create a unique identifier for the error, and pass that identifier back to the client, we would be able to track that specific error as it travels across machine/process boundaries. Assuming that the error is logged in both places, we would have a way to link together error information from one application component with that from another.

Implementing Exception Tracking with GUIDs

Figure 5 shows a C# class definition for a simple base class to help with tracking exceptions:

```
using System;

public class TrackedException : Exception
{
  #region .ctor signatures in System.Exception
  public TrackedException() :
this(Guid.NewGuid())
  {}
  public TrackedException(string message) :
     this(message, Guid.NewGuid())
  {}
  // etc...
  #endregion

  #region .ctors providing tracking ability
  using GUIDs
  protected TrackedException(Guid errorID) :
     base()
  {
    this.errorID = errorID;
  }

  protected TrackedException(string message,
     Guid errorID) : base(message)
  {
    this.errorID = errorID;
  }
  // etc...
  #endregion

  #region Tracking
  private Guid errorID;
  /// <summary>
  /// Uniquely identifies this exception
  /// </summary>
  public Guid ErrorID
  {
    get { return errorID; }
  }
  #endregion
}
```

Figure 5 - A basic TrackedException base class

The `TrackedException` class in Figure 5 automatically creates a new GUID in its constructors. Derived classes therefore do not need to concern themselves with GUID creation: in fact, they cannot, as the constructors relating to GUIDs are protected. Other useful base class constructors could be defined (e.g. matching the signatures of `System.Exception`), also priming the `ErrorID` property.

Wherever a `TrackedException` is thrown in code, we know that it will have a GUID-based `ErrorID` property, which we can include in log file data.

Crucially, however, in the context of Web Services, we can also include this GUID in the SOAP `Fault` response, by throwing explicitly a `SoapException` from within a Web Service method if an exception is thrown during processing:

```
[WebMethod] //Attribute needed for Web Service
           //'plumbing'
public string GetBananaPrice(string cityName)
{
  try
  {
    // some processing
  }
  catch (TrackedException te)
  {
    string message = te.GetType().Name + " " +
       te.Message;
    // N.B. Constructor parameters simplified
    // here
    throw new SoapException(
      message,
      ExceptionHelper.WrapDetails(te.ErrorID),
      te);
  }
}
```

Figure 6 - Using a TrackedException in a Web Service

The call to the hypothetical `ExceptionHelper.WrapDetails()` method returns a `System.Xml.XmlNode` object that will be inserted into the SOAP `Fault` response XML payload.

1 Web Services in .Net can also be built 'from scratch' if required.
2 There are similar implementations for Java: see [19] for an example.

The client calling the Web Service in Figure 6 would use code like this:

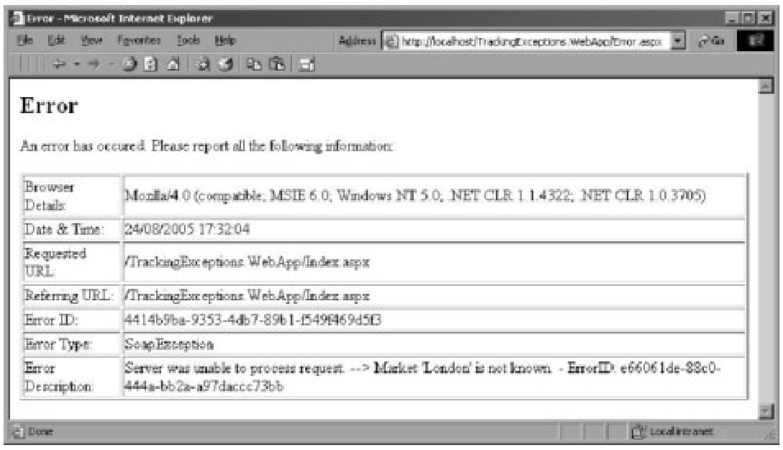
```
// Create local proxy for Web Service
// Connection to remote machine is handled
// automatically
PricesWebService ws = new PricesWebService();
try
{
  string bananaPrice =
     ws.GetBananaPrice("Havana");
}
catch (SoapException se)
{
  // Extract the GUID stored by the Web
  // Service from the XML
  string errorGUID = se.Details.InnerText;
  string message = se.Message;
  // Log the error here at the client...
  Console.Out.WriteLine("Error when calling
     GetBananaPrice(): " + message + " GUID: "
     + errorGUID);
  // TODO:
  // Present the GUID to the user
  // Notify admin using GUID
}
```

Figure 7 - Catching a SoapException at the client and logging the GUID

The `Details` property of the `SoapException` at the client contains the XML from the `XmlNode` that was inserted in the catch handler of the Web Service method: the GUID of the original exception (see Figure 6). Logging this at the client will allow us to tie together the error logs from the two applications; showing the GUID to the user (see Figure 8) would allow her to copy/paste the GUID into a bug report (for example), further correlating the error information.

The same GUID appearing in different log files will refer to the same exception instance; due to the nature of GUIDs (see Sidebar: GUIDs) we can assume that a GUID will never be duplicated.

There is an advantage in presenting only a GUID rather than detailed error information to the user: it may not always be appropriate to divulge the details of an error (for security reasons, for example). The GUID acts as an opaque handle to the already-logged error; a bug report containing just the GUID should be enough to put that report in context.


Error - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/TrackingExceptions.WebApp/Error.aspx
Go
Done
Local intranet

Error

An error has occured. Please report all the following information:

Browser Details:	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET CLR 1.1.4322; .NET CLR 1.0.3705)
Date & Time:	24/08/2005 17:32:04
Requested URL:	/TrackingExceptions.WebApp/Index.aspx
Referring URL:	/TrackingExceptions.WebApp/Index.aspx
Error ID:	4414b9ba-9353-4db7-89b1-f549f469d5f3
Error Type:	SoapException
Error Description:	Server was unable to process request. --> Market 'London' is not known. - ErrorID: e66061de-88c0-444a-bb2a-a97daccc73bb

Figure 8 - Presenting the error GUID to the user

Other Technologies

It is fairly easy to extend the GUID-based error tracking to certain other frameworks and technologies. For Java/RMI (and its .Net analogue, .Net Remoting), it is basically enough to make the `TrackedException` class available to each side of the Remoting channel.[3] The `ErrorID` property of the exception instance will be serialized along with the rest of the object, and therefore be available to the calling client. [Note that a UUID class was introduced only in Java 2 SE 1.5, so earlier versions of Java will have to rely on other UUID implementations – see Sidebar: GUID Implementations, [c]]

CORBA and DCOM differ substantially in their support for exceptions. DCOM does not transmit exception details from server to client, relying instead on (much less useful) "HRESULT" return codes [5] [22]. There is therefore no simple way to extend the GUID-based exception tracking to DCOM.[4]

Unlike DCOM, CORBA does transmit exceptions across the communication channel. If we make `TrackedException` a `CORBA::UserException`, we can define a public property `errorID`, which will contain the string representation of the GUID:

```
// CORBA IDL
#pragma prefix "example.com"
module TrackedExceptionExample
{
  interface FruitPrices
  {
    exception TrackedException
    {
      string errorID;
    };

    string get_banana_price(
       in string cityName)
          raises (TrackedException);
    };
};
```

Figure 9 - TrackedException in CORBA

Some CORBA implementations already provide for a way to associate extra information with the exception – see [23].

However, for situations that do not provide this ability to 'hook' the exception GUID (for example, in 'interop' scenarios [21] [5]), it may be possible to append the GUID to the error message. C# code for a modified version of the TrackedException class to append the GUID to the error message would look like this:

```
using System;
public class TrackedException : Exception
{
  #region .ctor signatures in System.Exception
  public TrackedException() :
     this(Guid.NewGuid())
  {}
   public TrackedException(string message) :
      this(message, Guid.NewGuid())
   {}
   // etc...
   #endregion
   #region .ctors providing tracking ability
      using GUIDs
   protected TrackedException(Guid errorID) :
      base(String.Format(FormatString,
           "TrackedException", errorID))
   {
     this.errorID = errorID;
   }
   protected TrackedException(string message,
      Guid errorID) : base(String.Format
      (FormatString, message, errorID))
   {
     this.errorID = errorID;
   }

  // etc...
  #endregion
  #region Tracking
  public static readonly string FormatString =
     "{0} - ErrorID: {1}";
  private Guid errorID;
  // <summary>
  // Uniquely identifies this exception
  // </summary>
```

3 Technically, it will also be necessary to ensure that the class is serializable. For a C# class with simple (serializable) fields, it is sufficient to mark the class with the `[Serializable]` attribute.

4 The target COM Object could be made to support the `IErrorInfo` interface, which could then be queried for the exception GUID.

5 See also http://udk.openoffice.org/common/man/uno_the_idea.html for an interesting discussion on Java/RMI, CORBA and DCOM.

```
    public Guid ErrorID
    {
      get { return errorID; }
    }
    #endregion
  }
```

Figure 10 - TrackedException modified to store the GUID in the error message

The constructors ensure that a GUID is associated with the exception (as in Figure 5), but also automatically store the string version of the GUID in the `Message` property of the exception, by modifying the data passed to the base class constructor.

This approach sacrifices encapsulation for the ability to track exceptions across process and machine boundaries. Logging the received error message would still allow the correlation of error information from the various parts of the distributed system, because the original exception GUID is stored in the error message.

Future Developments

The `System.Exception` class in C# 2.0 (currently in Beta, and due for release some time in 2005) has a new member: `Data` of type `IDictionary`. This allows the association of arbitrary named data items with a given exception. The `ErrorID` property of `TrackedException` could be re-implemented to store the GUID in the `Exception.Data` dictionary, because this would remove the need for special treatment:

```
  public class TrackedException : Exception
  {
    // Constructors go here...
    // ...
    // for example:
    public TrackedException (Guid errorID) :
       base ()
    {
      this.Data["GUID"] = errorID;
    }
    public Guid ErrorID
    {
      get
      {
        return this.Data["GUID"] as Guid;
      }
    }
  }
```

Figure 11 - ErrorID implementation on C# 2.0

The reason for this becomes clear when logging exceptions, we would just log all information in the Data dictionary (logging the Name, and calling `.ToString()` on the Value for each entry):

```
  try
  {
    // some processing...
  }
  catch (Exception ex)
  {
    // In practice, this code would go in a
    // generic logging method somewhere...
    Exception innerException = ex;
    while (null!=innerException)
    {
      // Log the basic exception details here
      // e.g. StackTrace, Message, etc....
      // Log the Data dictionary entries
      string[] names = ex.Data.Names;
      foreach (string name in names)
      {
        Log.WriteLine(name + " = " +
                      ex.Data[name]);
      }
      innerException =
  innerException.InnerException;
    }
  }
```

Figure 12 - Logging Exception GUIDs in C# 2.0

GUID Implementations

a Python: `http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/163604` and `http://pyro.sourceforge.net/manual/11-implementation.html#util`
b Perl: `http://cpan.uwinnipeg.ca/dist/Data-UUID` and `http://perl.apache.org/docs/2.0/api/APR/UUID.html`
c Java: [in standard Java 1.4 there is no UUID class] `http://platform.jxta.org/nonav/java/impl/net/jxta/impl/id/UUID/UUID.html` (Java 1.4.2) and `http://java.sun.com/j2se/1.5.0/docs/api/java/util/UUID.html` (Java 2 SE 5.0)
d SQL: "UUID() was added in MySQL 4.1.2." - `http://dev.mysql.com/doc/mysql/en/miscellaneous-functions.html` (MySQL) and "Using GUIDs with IDS 9.x" - `http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0401roy/ (DB2)`

Arguably, the need for a separate `TrackedException` class is removed with C# 2.0, because any and every exception can have a tracking GUID associated with it, stored as a named entry in the `IDictionary Data` member, rather than needing a separate property `ErrorID`.

Observations and Notes

The concept of exception tracking presented in this article is similar to the idea presented in [6] and [7], where contextual information associated with an exception is maintained for later analysis. This article extends the idea across process, machine and protocol boundaries, however, relying on offline log file analysis to restore the contextual information.

The overhead associated with generating and transmitting a GUID may be unacceptable in some cases. However, in most situations, the benefits gained from being able to track exceptions using a string of 30-something characters will probably outweigh the slight performance hit.

An example project demonstrating the ideas in this article is available from `www.accu.org`.

Summary

This article demonstrated a simple scheme to track exceptions across Web Services and other distributed systems using GUIDs.

The benefits of using this scheme become apparent when the need arises to correlate information from multiple error logs and bug reports: the details of specific exceptions can be reconstructed at a later stage, and problems diagnosed more thoroughly.

A prototype of this scheme has been in operation for several months for a real-world project using SOAP Web Services (http://www.lamip.org/) and proven to be very useful indeed.

Matthew Skelton

References

1 GUID/UUID Specification - `http://www.ietf.org/rfc/rfc4122.txt`
2 Background to UUIDs/GUIDs (UUIDs in DCE RPC) - `http://www.opengroup.org/onlinepubs/9629399/apdxa.htm`
3 Online GUID Generator: `http://kruithof.xs4all.nl/uuid/uuidgen`
4 System.Guid documentation for the .Net Framework: `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemguidclasstopic.asp`
5 Get Seamless .NET Exception Logging From COM Clients... - `http://msdn.microsoft.com/msdnmag/issues/05/01/ExceptionLogging/`
6 Hughes, Rob, 'Maintaining Context for Exceptions' *CVu 15.4* (August 2003)
7 Nibbs, Andy, 'Maintaining Context for Exceptions (Alternative)', *CVu 15.6* (December 2003)
8 A Web Services Primer - Venu Vasudevan `http://webservices.xml.com/lpt/a/ws/2001/04/04/webservices/index.html`
9 SOAP Specification - `http://www.w3.org/TR/soap/`
10 SOAP (Introduction) - `http://www.microsoft.com/mind/0100/soap/soap.asp`

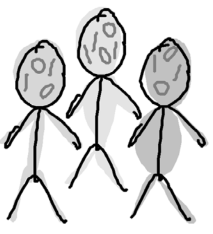
[concluded at foot of next page]

Professionalism in Programming #34

Together We Stand (Part One)

by Pete Goodliffe <pete@cthree.org>

The most important single ingredient in the formula of success is knowing how to get along with people.
Theodore Roosevelt



The gory business of 'paid' professional programming is, depending on how you look at it, either an exhilarating chance to do what you love for a living, or a depressing experience where management incompetence, inept team members, and bad planning conspire to create mediocre software, leaving you no chance to fix the problems. Welcome to life in the software factory. It's a large place, and (however much you'd like to believe it) you don't live there alone.

Being a good software engineer means more than just being a good programmer. You might be able to compute PI to ridiculous accuracy in less than five lines of code. Well done. But there are many other skills required. And one of these is team working

In this series of articles we'll take a long hard look at the Real World scenario of programming in teams. Good teamwork is vital to the survival of a software project. An ineffective team will quickly stifle any software development activity, leaving progress to the heroic efforts of a few dedicated individuals working against huge odds. To work out how write good software we'll examine teamwork as it applies to us, as programmers. We'll look at what constitutes good teamwork, and how we can be more effective in our teams.

In this first instalment we'll investigate what our software writing teams look like, what they do, and how they fit into the software factory. We'll make some observations about what characterizes good and bad software teams and determine what personal skills, tools, and organizational structures will lead to better teamwork.

Our Teams – the Big Picture

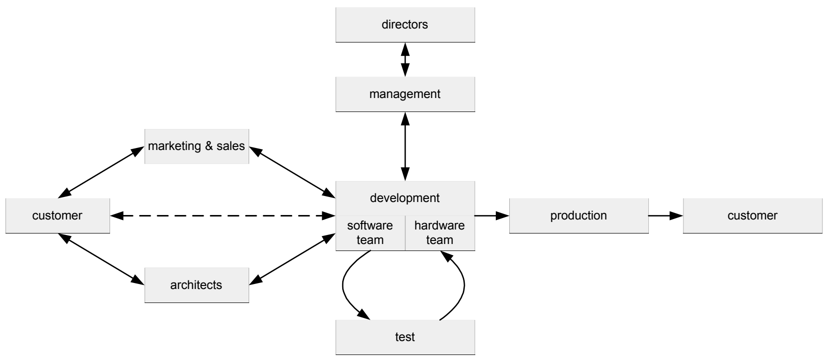
So what kinds of team do we work in? A typical software engineer participates in various levels of teams, each with different dynamics and requiring different levels of contribution. Consider this scenario:

- You're creating a distinct software component which is part of a larger project. You may develop it by yourself, or as part of a team of programmers: team one.
- The component will fit into a wider product. All the people involved with this product (including any hardware designers, software developers, and other non-engineering roles such as marketing) form team two.
- You are also part of a company that may be working on many different products simultaneously. Team three.

In reality there are more levels of team-manship in any reasonably large software-development company. As programmers, we are most involved in the smaller level of team activity – in our day-to-day development teams. We have the most control and influence over this world. This is the level we are responsible for, where we have authority to make design/implementation decisions and to report on team progress. Programmers are less responsible for the effects of higher level teams, but we are affected by teamwork 'in the large' as much as we are by teamwork 'in the small', even if it's not as immediately obvious.

Development teams sit amidst the many other tribes of software factory inhabitants, and must interact with and work alongside them. This dictates the nature of most of our inter-team interactions. The general shape of corporate team structure is shown in the following diagram:

It is vital to cultivate good relations and foster smooth work flow between these different teams. A problem in the wider software factory structure will really scupper your software development. However, it's outside the scope of this article to investigate company culture and process improvements. In this article we'll focus mainly on development teamwork, under the shadow of this organisational context. That's where most of our time and effort is spent, and consequently where our individual improvement will make the greatest difference. It's the teamwork level that most directly influences the quality of our software



[continued from previous page]

11 Java RMI, CORBA or COM? - Prithvi Rao - http://www.usenix.org/publications/java/usingjava13.html
12 Box, Don *A Young Person's Guide to The Simple Object Access Protocol* – http://msdn.microsoft.com/msdnmag/issues/0300/soap/
13 SMTP as a [SOAP] Transport - http://blogs.msdn.com/rdias/archive/2004/06/17/158802.aspx
14 UDDI - http://www.uddi.org/whitepapers.html
15.WSDL Specification - http://www.w3.org/TR/wsdl
16.Web Services - The Web's next revolution - IBM DeveloperWorks - http://www6.software.ibm.com/developerworks/education/wsbasics/wsbasics-ltr.pdf
17.SOAP Version 1.2 - http://www.w3.org/TR/2003/REC-soap12-part0-20030624/
18.SoapException documentation - http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemwebservicesprotocolssoapexceptionclasstopic.asp
19.Using Web Services with J2EE - http://webservices.sys-con.com/read/39434.htm
20.The .Net System.Web.Services namespace - http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemwebservices.asp
21.CORBA / .Net interop: Janeva - http://www.devx.com/interop/Article/19916/1954?pf=true and http://www.borland.com/us/products/janeva/
22.Exceptions in COM - Bob DeRemer - http://msdn.microsoft.com/msdnmag/issues/04/03/ExceptionsinCOM/default.aspx
23.CORBA CORBA::UserException::id() method – http://publib.boulder.ibm.com/infocenter/adiehelp/index.jsp?topic=/com.ibm.wasee.doc/info/ee/corba/concepts/ccor_ipgmce.html

Acknowledgements

Thanks to the Editor, Paul Johnson, for requesting this article.

Thanks also to Kev Watkins, Liz Chapman, Mike Graves, Jason Neylon, Besim Atalay, Gerald Krafft, Denny De La Haye, Esme Tearle, and Rebecca Dyer for help and suggestions.

About the author

Matthew Skelton is a Senior Analyst Programmer for Porism Limited (http://www.porism.com/), an application development company that specialises in bespoke database applications for the web. The company also publishes standards for e-Government, which are available at: http://www.esd.org.uk/standards.

Matthew is also a member of ACCU, and a recent convert to Python.

Development team size dictates the dynamic and nature of shared software construction work as much as the team's place in the organisational food chain. A lone engineer is given responsibility for all software architecture, design, and implementation work. In really small outfits they may also have to work on gathering requirements, and create and run a thorough test plan.

As soon as more developers are added to this mix, the nature of the programming task changes. It's no longer just about coding skill, and requires social interaction, coordination and communication skills. This is where your team working skills will affect the software you build -- for good or ill.

The Good, the Bad, and the Ugly

There are some factors that clearly characterise good and bad software development teamwork. Cataloguing these will help to set the context for our foray into teamwork improvement. Some can be worked on by individual programmers, but most are things that we have to live with, or try to persuade managers above us to get right.

Prerequisites for Good Teamwork

These are influences that will determine the effectiveness of a software team, and the quality of the work produced. For effective teamwork, all of these factors must be in place:

- The correct spread of people, with a range of appropriate technical skills.
- Team members with a range of experience, who are each able to learn from others (a whole team of trainees will clearly be doomed from the start).
- Complimentary team member personality types -- to succeed the team needs encouragers and motivators not people who will drag morale down.
- A clear and realistic goal (even better if it's an 'exciting' project that the team members really want to see completed).
- Motivation (whether financial or emotional).
- Suitable specifications provided as soon as possible, so each member understands what they have to build, and to ensure that the individual pieces of work fit together.
- Good management.
- As small a team as realistically possible, but no smaller. Adding more people makes teamwork harder: there are more lines of communication, more people to coordinate, and more points of failure. We should try not to make things unnecessarily hard.
- A clear and universally understood software engineering process for the team to follow.
- Backing from the company, not hindrance and unnecessary beaurocracy.

Characteristics of Bad Teamwork

In contrast, these are sure indicators of a team that is not able to work effectively. Note that this is a mix of internal and external factors.

- Unrealistic schedules, with deadlines established before the team has scoped their work.
- Unclear project objectives, and a lack of project requirements.
- Communication failure.
- Bad or unqualified team leaders.
- Badly defined individual roles and responsibilities -- who's responsible for doing what?
- Individual bad attitudes and personal agendas.
- Incompetent team members.
- Management not valuing individual engineers, treating them like minions.
- Individual appraisals based on criteria that don't match the team objectives.
- Rapid turnover of team members.
- No change management procedure.
- A lack of training, or of mentoring.

Personal Skills and Characteristics for Good Teamwork

Every team is comprised of individuals. A trite maxim splashed across banal motivational posters says: there is no 'I' in Team. If you can avoid vomiting, you'll realise that your software team doesn't exist to serve you. You serve the software team. To start improving your team's performance you can begin close to home – by addressing your attitudes towards the team and your joint development effort. We're not all managers, so this is really the main area that we have any influence over.

Being a high quality programmer requires you to be a high quality team player, so how do you improve? What skills do you need to be an effective team worker? What does it mean to be a 'good' team member?
Teamwork is a skill acquired over time, no one is born with what it takes to be successful in a winning team. However, it is clear that some people are more adept at it than others; it fits their personality and character better.

A different set of technical skills are required from each different development team role. However, presented below are a list of non-technical skills, characteristics, and attitudes that any effective team member really must have before we can even consider program language dexterity or design capability.

Communication

Teamwork is dead without communication. Individual parts cannot move as a whole without communication. How can the goal and vision be shared without communication? Projects really do fail because of a lack of good communication.

Intra-team communication occurs in several ways: conversations between individual engineers, phone calls, meetings, written specifications, email correspondence, reports, and instant messaging. Sometimes we even communicate in pictures! Each medium has a particular usage dynamic and is most appropriate for a specific kind of discussion.

The most effective communication should involve (or at the very least be visible to) all relevant parties. It should be sufficiently detailed, but not consume too much time or effort. It should be performed in a suitable medium – for example design decisions should be captured in a written specification, not agreed verbally and shared by word of mouth.

Code itself is a form of communication. A programmer must be able to communicate well. This requires both good input and good output:

- to write unambiguous specifications, to describe ideas clearly, and keep things succinct, and also
- to read and comprehend specifications correctly, to listen carefully, and to understand what they are told.

In addition to intra-team communication we must also consider communication between teams. The classic example of bad communication is seen in most companies, between their marketing department and the engineers. If marketing don't ask the engineers what is possible then they will sell products that the company can't make. This kind of problem is cyclical: once it has occurred once and people have been burnt, the two teams are less likely to talk to each other (due to resentment). It will then happen again and again.

Communications Breakdown

There are many communication methods in our highly connected world, and we must learn to use them effectively, to support and facilitate our team interaction. The key to this lies in understanding their particular dynamics, etiquette, and individual merits.

Telephone

Best used for communication that requires an urgent response, a phone call interrupts what you are doing. For this reason it's inconvenient to be called for non-urgent matters – use another method instead. With mobile phones we are far more connected than we used to be; this is a blessing and a curse at the same time.

Being audio only, you can't see the other person's face and their subtle body language cues. It's easy to misinterpret someone on the phone, and draw a different conclusion from the other person.

Too many techies are scared of using the phone. Don't be – for urgent communication it's invaluable.

Email

An asynchronous, out-of-band communication medium. You can specify a level of urgency, but email is never immediate – it's not a real-time conversation. It's a rich medium, allowing you to quickly send attachments, and compose replies when it's convenient for you. It is often used for 'memo'-style broadcasts to many recipients. Your email history provides a reasonably permanent record of communications. Email is an immensely powerful communication mechanism.

You must learn to use email as a tool, and to not become a slave to it. Don't open every new mail as it arrives; your coding will get interrupted far to often, with a hit to your productivity. Designate email reading times, and stick to them.

Instant Messaging

A quick, conversational medium that requires more attention than email, yet one that can be ignored or sidelined easier than the telephone. It is an interesting and useful middle ground.

Written Report

These are less conversational than email communication, and more permanent. Written reports and specifications are formal documents. They take longer to prepare, and are consequently harder to misinterpret. Written reports are generally reviewed and agreed on, so are more binding.

Humility

This is an essential characteristic, and so often lacking in our profession.

The humble programmer wants to make a contribution to serve the team. They don't slack off to let others do all the work. They don't believe that they are the only talented person capable of making a worthwhile contribution.

You can't hoard all the 'good' work for yourself; it's just not possible for one person to do everything. You have to be willing to let another team member contribute – even if it's something you want to do.

You should listen to and value the opinions of other people. Yours is not the only point of view, and not the only solution. You don't necessarily know the only, or the best way to solve every problem. Listen to others, respect them, value their work and learn from them.

Dealing with Conflict

We have to be realistic: some people can't help winding each other up. In this situation we must be mature and responsible in our attitudes, and learn to avoid (or learn to resolve) conflict situations. Conflict and animosity breed more and more bad will, and will severely degrade the performance of a team.

However, harnessed and channelled conflict can be a major success factor in your team work. Team mates who stimulate and provoke each other produce the best designs. Disagreement can act as a refining process, ensuring that ideas are valid. Knowing that your work will be cast under a critical eye keeps you focused.

It's important to keep this kind of conflict constructive; on a strictly professional, not personal, level.

Learning

This doesn't just mean learning new technical skills, but learning to work as a team. It's not a God given gift. A new team has to learn how to work together, how each member reacts, their strengths and weaknesses, and how to capitalise on individual skills to the group's benefit.

Emerson wrote: "every man I meet is in some way my superior". Look at what you can gain from your peers. Learn from what they know, learn from what they're like, and learn how they react. Learn to communicate with them. Seek criticism from them at all levels, from the formal code review to their passing opinion offered in conversation.

Good teamwork builds the project and also builds the team. An important part of 'building the team' is for each member to be learning as they work.

Adaptability

This is tied closely with learning.

If the team has a need that no developer can currently fulfil, and it's not possible to bring in an outside resource, then a solution needs to be found. Adaptable people can learn the new skills quickly to fill the gap and serve the team.

Know Your Limitations

If you are committed to work that you know you can't do, or later find out that you don't have the skills to complete (and you can't realistically learn them in the given time scale) then you should make your manager aware of this as soon as possible.

Otherwise, you will fail to deliver your piece of the project, and the whole team will suffer as a consequence.

Many people believe that admitting they can't do something is a sign of weakness. It's not. It's better to admit your limitations than to be a point of failure in the team. A good manager will provide some extra resource to help you do the work, and along the way you will learn the new skills that you previously lacked.

Teamwork Tools

Having investigated the core personality traits and essential skills of a team-playing software developer, we can now investigate what tools help us to form a functioning software team. There are a few indispensable tools that facilitate collaboration and help to elevate your joint development from chaos to a well-oiled machine. On their own they won't make you a team of commando programmers, but they're the arsenal every crack outfit relies on – the prerequisites for effective software developer interaction.

Source Control

Even if you're the sole developer on a project, you need a source control system – it's a crucial store of the codebase and its history. When there are multiple programmers it becomes even more critical. Source control helps to marshal who is doing what and when, provides the definitive 'latest code' snapshot, and allows you to manage changes, undo mistakes, and make sure that no one misses source code updates.

Without it you'd need to employ someone to integrate all changes and manage the code. And they'd be more likely to make catastrophic mistakes. Joint development without source control is unthinkable.

Faults Database

We've already looked at what this is, and the specific purpose it serves. However, notice how it facilitates interaction between your development team and the test team. A fault tracking system acts as a pivot between test and development. It helps to organise test and repair work, prioritise faults, assign problems to individuals, and track pending fixes in the software.

The tool can be used to make sure that all issues are addressed and that no work is overlooked. It makes clear which faults are currently development's responsibility and which are tests.

Groupware

A team needs effective communications support, especially when geographically split. A centralised calendar, address book, and meeting booking system provides a digital administrative backbone.

You also need a mechanism to share documents, both externally supplied documents and your current works-in-progress. A groupware tool facilitates such collaborative work. Without this facility, an inelegant replacement is a well-defined shared network drive.

There are other tools that facilitate group interaction. Consider using wikis (web-based community documentation tools) and internal newgroups (email discussion boards with more permanent storage).

A Methodology

It's important to establish a defined and universally understood development methodology, or work will be chaotic, performed on an ad-hoc basis. One developer will release their code, when another would refuse to let go of it until they've thoroughly tested and debugged it. One developer will halt all coding until they've produced an intricately detailed specification, whilst another rushes straight into prototyping the code. Holy Wars are made of smaller things than this.

A methodology defines the development process details, who is responsible for what work, and how work is handed on. With one, every developer knows how to work as a part of the team, and what's expected of them. You must pick an appropriate methodology, based on the size of the team, the kind of code you are producing, and the talent, experience, and dynamics of people.

Project Plan

To produce any work in a predictable, timely manner you need some semblance of organisation. This is enshrined the project plan, detailing who is doing what over the course of development. To be of any use, the plan must be based on sound estimates, stuck to by developers, and kept up to date with any changes required.

Team Organisation

The structure of a software development team is shaped by two main factors:

- the management approach, and
- the division of responsibility amongst members.

This will determine the amount of code and the size of the units that you work on. The code produced, in turn, is shaped by the organisation of the team.

[concluded at foot of next page]

Let's Do C# and MySQL – Part 1 - MySQL

Paul F. Johnson <paul@all-the-johnsons.co.uk>

Before I start, this is going to be fun. Understand? Fun. Not dull, but fun. I intend writing this in a banana suit with my feet in a bucket of warm rice pudding[1]. That's how much fun it's going to be!

Objectives

What is the point of the series of articles? I intend it to be a learning progression to take you from the very basics of an application through to implementation. I've chosen to write a small MySQL front end in C# to do this. There will be a progression from setting up the MySQL database, testing using a command line application, moving it over to a `System.Windows.Forms` application and then adding in functionality. I will also look at using db4o. The question is though - *why C#?*

The MySQL API is very well documented and is accessible through many different programming languages with the minimum of fuss. I've chosen C# as I'm yet to find any really good tutorials for using the language. Sure, MSDN contains a great deal of information, but without the linking arguments, the information is almost context less.

1 Your definition of fun may vary from that of the author. The ACCU cannot be held responsible for any discomfort felt that you may experience in the pursuit of what you consider fun. Rice pudding does not mix well with electricity, but is very nice with freshly made blackcurrant and gooseberry jam.

Let's Make a Start

You will need a copy of MySQL installed on your computer to benefit from this series of articles and some form of C# compiler. The code works fine with both Mono and Microsoft C# - I have not tested the C# using gnu.NET. When compiling the code, Microsoft users should use `csc` and Mono users `mcs`. Any additional dlls to be used will be listed as `-r:System.Drawing -r:System.Windows.Forms` (though if you are using Visual Studio, these are normally be hidden).

Setting up MySQL

The first thing we'll need is a working MySQL database. I'm not going to assume that you've set up a database before now, but there are effectively two things you need to decide. What's going into it and how it's laid out. Take the following list of elements:

Element	Atomic Symbol	Description URL	Mass	Atomic Number
Copper	Cu	metals/copper	63.546	29
Chlorine	Cl	gases/chlorine	35.5	17
Nepunium	Np	actinides/neptunium	237	93

I could have one very large database which would contain parameters such as the period number, CAS registry, element block, shell filling, picture URL, if it is radioactive, man made and so on. This would be fine, but very messy and a pain to administer too.

[continued from previous page]

Management Approach

A project may be managed on a peer basis, with no coder considered more important than any other, or under the leadership of an über-programmer/manager. The programming team could be considered part of a software production line: fed designs from a team upstream, they produce code to specification[1]. Enlightened software engineers are given more autonomy and responsibility.

Tasks may be allotted months in advance, on long-range plans (which can rapidly become out-of-date and inaccurate), or 'just-in-time' by assigning each work package when a developer finishes their last one. Programmers might work alone on their individual parts of the system, or work collaboratively, pair programming to spread responsibility and knowledge.

Division of Responsibility

The axis of responsibility determines how each line of development is split amongst programmers:

- With a vertical team organisation you employ a team of generalists, who are skilled in a wide number of roles. They are each given a piece of work and implement it end-to-end, from the architecting and designing, right through implementation and integration, to development testing and documentation.
  The main advantage of this approach is that developers gain a wider range of skills, and become more experienced in the whole software system. With one key developer per feature there is cohesion in its design and implementation. However, generalists are expensive and hard to find. They don't have expertise in all areas, and therefore take longer to solve some problems. There is likely to be less cohesion between separate features, as they are implemented by different developers. The customer has to work with more people, since there's no specific liaison point – each developer needs their input to scope the requirements and validate their design.
  To make this kind of team work you must define common standards and guidelines for development work. You must have good communication to avoid several people reinventing the same wheel. A common architecture must be agreed early on, or a chaotic and haphazard system will ensue.
- In contrast, a horizontally organised team is built from a team of specialists, and every development task is split between them, using their respective talents at the appropriate time. Because each aspect of work (requirements gathering, design, etc.) is done by a specialist it should be of a higher quality.
  This has many opposite characteristics to the vertical arrangement: we build cohesion between separate work packages, but there's a danger that each slice of work holds together less well because more people have worked on it. Interaction outside the team (with customers or other company factions) is made by a small number of specialists. This is easier to manage, for the team itself and the external contacts.
  You must take care to ensure that the specialists are well coordinated, and that they see right through to the end of each work package, or their work will be narrow-sighted. With many people involved in each development procedure, the team is harder to manage; there is more work flow To make this arrangement work requires good communication, defined processes, and smooth handoffs between developers.

1 Here management expect replaceable, commodity -- grunt -- programmers.

There is no 'right' kind of organisation. Which one is most appropriate depends on the team members and the nature of the work produced. The pragmatic arrangement is probably somewhere in the middle.

Organisation and Code Structure

A team's organisation has an inevitable affect on the code it produces. This is enshrined in software folk law as Conway's Law. Simply stated: "if you have four groups working on a compiler, you'll get a four-pass compiler". Your code inevitably takes on the structure and dynamics of your interacting teams. The major software components lie where teams gather, and their communications follow the team interactions. Where groups work closely, component communication is simple and well defined. When teams separate, the code interacts clumsily.

We naturally aim to create opaque interfaces between each different team's work, to facilitate our interaction with that team. We do so even in cases where reaching into some internal part of another component might be valid and better approach. In this way teams can foster arbitrary divisions; despite our good intentions design decisions are forced by team composition.

Of course, there's nothing wrong with encapsulation and abstraction; however they must be designed in for the right reasons. If anything, organise your team around the code you must build, not vice-versa.

Next Time

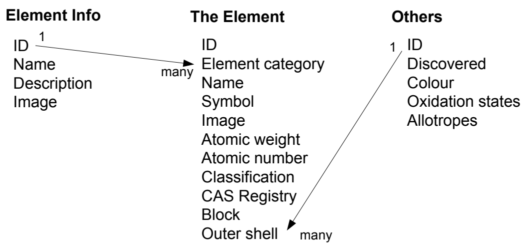
In the next instalment, we'll investigate the death of some failing software teams. Cheery stuff! From this we'll see what we can learn about the characteristics of healthy development teams. Stay tuned.

Pete Goodliffe

A simpler solution is to break the database down and to do that, we need a couple of generic categories.

The list above gives three types of element: metal, gas and man-made. They are good enough and more can be added as and when I need them (such as trans-metal, halide and trans-actinide). But what of the actual data? For that, a second table is created which is linked to the generic first table. In the second table, the specifics of the element are held (such as discovery, relative atomic weight, shell filling, etc.).

Graphically, the database looks like this.



For the *Element Info* column, *ID* represents the classification ID – for example, metal = 1, gas = 2. *Name* is the generic name, *Description* a generic description and *Image* a generic picture. *Element Info* links to *The Element* in what is known as a 1 to many (sometimes known as 1 to infinity) relationship (1 item in the generic table can link to many items in the second table, as long as they match the type of the generic item). *The Element* list is straight forward to understand which again is linked to a third database (again via a 1 to many relationship) for other information.

Okay, what you should now be asking is why the other information should be in its own table?

When you think about it, the answer to that is simple. Say a new element is discovered (or created). While it is entirely possible to enter all of the details into one big table, discovery of elements and the dates are sometimes contested (especially for the trans-actinides where there was originally contested names for elements 104 to 111) – after the dust settles, it may be that the original name has to be changed, the discoverer changed and a few other details to boot. It is far simpler to have all of the "additional" information elsewhere so that the main database is still correct.

When considering a MySQL database, it could also be thought of as spreadsheet in style with columns and rows. A column will always be of a particular type which will be associated with a row. For example

	INT	CHAR	VARCHAR	DOUBLE	TEXT
ID		▨	▨	▨	▨
NAME	▨	▨		▨	▨
DESCRIPTION	▨	▨	▨	▨	
IMAGE	▨	▨		▨	▨

The white blocks indicate the type associated. It's not quite how things are done, but is a convenient way to represent the database. (*In reality, it's actually a two column table with the ID in column 1 and the type in column 2 – my representation is just handy – well, to me at least!*)

MySQL Data Types

If you think of the MySQL server as a form of computer, you can quickly see that objects within the database can be thought of as variables and all variables need to be of a specific type (such as `int`, `char` and `float`). MySQL has specific data types for the components of a database. Effectively,there are three main types for MySQL : **Date & Time**, **Numeric** and **String**.

Date and Time

Field name	**What it does**	**Range**
`DATE`	It's a date. MySQL allows dates to be inputted as either a string or numeric.	1000-01-01 to 9999-12-31. Always in yyyy-mm-dd format.
`DATETIME`	Time and date. As with date, the fields can be entered as strings or numerics. Always displayed as yyyy-mm-dd HH:MM:SS	1000-01-01 00:00:00 to 9999-12-31 23:59:59
`TIME`	Time – has the format HH:MM:SS and can be set using strings or numeric values	-838:59:59 to 838:59:59
`TIMESTAMP`	This allows a timestamp to be added when either update or insert are used. `TIMESTAMP` is returned as a string of the format yyyy-mm-dd HH:MM:SS. TIMESTAMP will automatically add the time and date of the most recent operation if you don't set it.	1970-01-01 00:00:00 to midway through 2037.
`YEAR[(2/4)]`	Gives the year as either 2 or 4 digit values. Default 4 digit.	0000 and between 1901-2155 for 4 digit and 70-69 for 2 digit (represents 1970 to 2069).

String Types

Field name	**What it does & Options**	**Range**
`CHAR(M)`	A character string of size `M`. It is right space padded. Options `BINARY` (also written as `BYTE`), `ASCII` (assigns latin1), `UNICODE` (assigns ucs2). Trailing spaces are removed when retrieved. If `M` > 255, it is automatically converted to `SMALLTEXT` (From version 4.1.0) – this applies to other large values. `CHAR` can be preceded by `NATIONAL`. This tells the database to use the default character set for that nation.	0 - 255
`CHAR`	Synonym for `CHAR(1)`	
`VARCHAR(M)`	A variable length string of maximum size `M`. Can be preceded by `NATIONAL`. Options : `BINARY` (stores the string as a binary set)	M = 0 to 255 (MySQL < 5.0.3) M = 0 to 65535
`BINARY(M)`	Roughly the same as `CHAR(M)`	
`VARBINARY(M)`	Roughly the same as `VARCHAR(M)` except is stored as binary byte string instead of non-binary character bytes.	
`TINYBLOB`	A small blob column.	255 bytes
`TINYTEXT`	A small text column	255 bytes
`BLOB[(M)]`	A blob column. The server will create the smallest possible blob column for the size. If `M` < 255 then a `TINYBLOB` is used.	65535 bytes
`TEXT[(M)]`	A text column. The server will create the smallest possible blob column for the size. If `M` is < 255, then a `TINYTEXT` is used.	65535 bytes
`MEDIUMBLOB`	A medium sized blob column	16,777,215 bytes
`MEDIUMTEXT`	A medium sized text column	16,777,215 bytes
`LONGBLOB`	A somewhat large blob column	4,294,967,295 bytes (4 Gb)
`LONGTEXT` bytes (4 Gb)	A large text column	4,294,967,295
`ENUM(c1,c2,..)`	An enumeration column. It is very similar to C enumerations with the strings being held internally as integer values.	Max. 65535 enumerations
`SET(c1, c2,...)`	A string which can have more than one value (`0` to `n`). The strings (`c1`, `c2` to `cn`) are held internally as integers.	Max. 64 strings.

Numeric Values

Field name	What it is, options and aliases	Range
`BIT [(M)]`	A bit field type. If M is not specified, it is taken as 1.	1 - 64
`TINYINT[(M)]`	A very small integer. Can be signed (default) or `UNSIGN`ed. Can also be set with `ZEROFILL`.	0 – 255 (unsigned) -127 to 128 (signed).
`BOOL, BOOLEAN`	This is a synonym for `TINYINT(1)`.	0 = false, non-zero = true.
`SMALLINT[(M)]`	A small integer range. Can be signed (default) or `UNSIGN`ed. Can also be set with `ZEROFILL`.	-32767 – 32768 0 - 65535
`MEDIUMINT[(M)]`	A medium range integer. Can be signed (default) or `UNSIGN`ed. Can also be set with `ZEROFILL`.	-8388608 – 8388607 0 - 16777215
`INT[(M)]`	A normal sized integer. Can be signed (default) or `UNSIGN`ed. Can also be set with `ZEROFILL`. Can also be written as `INTEGER`.	-2147483648– 2147483647 0 - 4294967295
`BIGINT[(M)]`	A big integer. Can be signed (default) or `UNSIGN`ed. Can also be set with `ZEROFILL`. Use with care (see the notes after this table).	-9223372036854775808 - 92233720368547758 0 - 18446744073709551615
`FLOAT[(p)]`	A floating point value. Can be signed (default) or `UNSIGN`ed. Can also be set with `ZEROFILL`.	p = 0 – 24, single precision p = 25 – 53, double precision
`FLOAT[(M,D)]`	A floating point figure. Can be signed (default) or `UNSIGN`ed. Can also be set with `ZEROFILL`. `M` is the display width, `D` is the number of decimal places. If no values are set or `FLOAT(p = 25 to 53)` is not used (`M` can be blank), the value is single precision. Can also be written as `REAL`.	-3.402823466E+38 to -1.175494351E-38, 0 and 1.175494351E-38 to 3.402823466E+38
`DOUBLE[(M,D)]`	A double precision value. Can be signed (default) or `UNSIGN`ed. Can also be set with `ZEROFILL`. `M` is the display width, `D` is the number of decimal places. Also can be written as `DOUBLE PRECISION`.	-1.7976931348623157E+308 to -2.2250738585072014E-308, 0, and 2.2250738585072014E-308 to 1.7976931348623157E+308
`DECIMAL[(M[,D])` (MySQL 5.03)	A packed exact fixed point figure of length `M` and with `D` digits. If `D` is omitted, `0` is used (there can be no decimals or fractions with this value). If `M` is omitted, `10` is used. Can also be `UNSIGN`ed and `ZEROFILL`. Also written as `DEC[(M[,D])`, `FIXED[M[,D])` and `NUMERIC[(M[,D])`.	Max D = 30 Max M = 64
`DECIMAL[(M[,D])` (pre MySQL 5.03)	An unpacked exact fixed point figure of length `M` and with `D` digits (it acts more like a `CHAR` column). All parameters and synonyms for the above apply.	Max D = 30 Max M = 64

Notes for BLOB

A `BLOB` is a Binary Large OBject – it can be just about any piece of binary data (picture, video or audio are examples). A `BLOB` can be useful in some circumstances (saving of avatars or to replace a directory of binary objects) but can also be a hindrance (increases the size of the database and may take longer to serve up the data).

Notes for BIGINT

All arithmetic is done using signed `BIGINT` or `DOUBLE` values, so you shouldn't use unsigned big integers larger than `9223372036854775807` (63 bits) except with bit functions! If you do that, some of the last digits in the result may be wrong because of rounding errors when converting a `BIGINT` value to a `DOUBLE`.

MySQL 4.0 can handle `BIGINT` in the following cases:

- When using integers to store big unsigned values in a `BIGINT` column.
- In `MIN(col_name)` or `MAX(col_name)`, where `col_name` refers to a `BIGINT` column.
- When using operators (+, -, *, and so on) where both operands are integers.
- You can always store an exact integer value in a `BIGINT` column by storing it using a string. In this case, MySQL performs a string-to-number conversion that involves no intermediate double-precision representation.
- The -, +, and * operators use `BIGINT` arithmetic when both operands are integer values. This means that if you multiply two big integers (or results from functions that return integers), you may get unexpected results when the result is larger than 9223372036854775807.

Setting Up the Database for Next Time

MySQL can be set up from the command line or by using something like `mysqlcc` [1] or `phpMyAdmin` [2]. Being a bit of a traditionalist, I'll go with the terminal windoUw (mainly as it also means that you don't need to download anything else!).

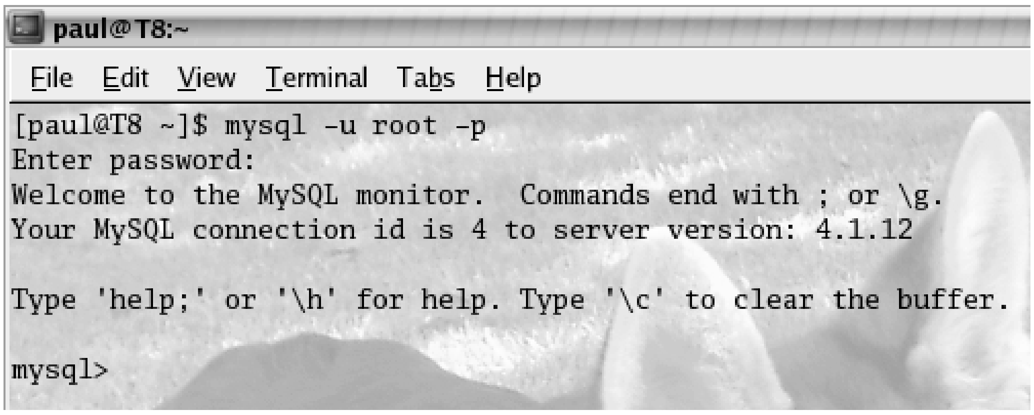
To call up MySQL, open a terminal window and type `mysql -u root -p`. This tells MySQL that you want to start up with the user "root" and with a password. The actual nuts and bolts of setting up MySQL isn't that hard.

If you've never used MySQL before, then the default password is nothing – just hit the return key. This is a hideous security hole and one which needs to be rectified as soon as you log in.

Once in, you'll see something similar to the image on the right.

If you've not set a password, then enter the following:

```
UPDATE user set password =
PASSWORD("newpassword") where user = 'root';
FLUSH PRIVILEGES; exit;
```



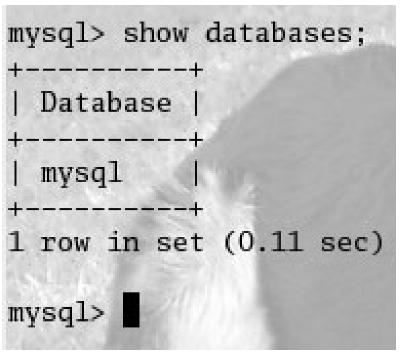
```
[paul@T8 ~]$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4 to server version: 4.1.12

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

(In this instance, `newpassword` is a password of your choice)

You now have a new password for the root user on your MySQL server. `FLUSH PRIVILEGES` clears all privileges on the server and exit does what it says on the tin – it exits the MySQL terminal.

That done, login to the server again. This time, let's examine what tables come by default.

```
mysql> show databases;
```



```
mysql> show databases;
+----------+
| Database |
+----------+
| mysql    |
+----------+
1 row in set (0.11 sec)

mysql> █
```

This will produce a table containing all of the databases the MySQL server is currently serving (right). You can see there is only 1 database on the MySQL server this series is being composed on. `mysql` is the database which contains details about users, passwords and the such.

When `show databases;` didn't show up the name of the database to be created, that gave the green light to say the database can be created.

Before creating the database (and as is common with creating anything on a computer), the database, linking, names and types should be written on paper first.

I have already specified earlier the names which I will use for the columns, but what about the types?

`ID` (here) is an `INT` and is also given the special type of `PRIMARY_KEY` when defining. The tables below show all of the types for all of the names.

Category Info	Type	Size	Special
ID	INT		PRIMARY_KEY (1)
NAME	VARCHAR	30	
DESCRIPTION	VARCHAR	30	
IMAGE	VARCHAR	50	

Element Info	Type	Size	Special
ID	LONGINT		
Element category	INT		PRIMARY_KEY (2)
Name	VARCHAR	50	
Symbol	VARCHAR	5	
Image	VARCHAR	50	
Atomic weight	FLOAT(1)		
Atomic number	INT		
Classification	VARCHAR	50	
CAS registry	LONGINT		
Block	VARCHAR	5	NON_ZERO
Outer shell	VARCHAR	50	

Other	Type	Size	Special
ID	INT		PRIMARY_KEY (3)
Discovered	INT		
Colour	VARCHAR	20	
Oxidation states	VARCHAR	50	
Allotropes	VARCHAR	200	

The `PRIMARY_KEY`s all link to each other. `KEY(1)` is the "root" key which links to `KEY(2)`. `KEY(3)` is linked from `Others ID` to the `Catalogue` table.

At the end of the second table, there is an "others" id. This is the anchor from which the `ID` in the `Others` table is attached. It would be pointless to try and add further to these tables as they are pretty much in their clearest format.

`VARCHAR` has been used for the images. While I could have used `BLOB`, using `VARCHAR` allows me to say, "okay, the URL for the images is going to be fixed, this is just the filename".

Finally, I have "Last ordered" set as a `DATE` which is `ZEROFILL`ed. This means I can set the date myself which is more useful than reliance on a `TIMESTAMP` (useful as it means I can say that I had the stock before the database was set up!)

Creating the Database

Simple enough.

```
mysql> create database theelements;
mysql> use theelements;
mysql> create table catagoryinfo (
  `id` smallint(6) NOT NULL auto_increment,
  `name` varchar(30) NOT NULL default '',
  `description` varchar(30) NOT NULL default
'',
  `image` varchar(50) NOT NULL default '',
  PRIMARY KEY  (`id`)
);

mysql> create table elementinfo (
  `id` longint NOT NULL auto_increment,
  `elementcatagory` int NOT NULL,
  `name` varchar(50) NOT NULL default '',
  `symbol` varchar(5) NOT NULL default '',
  `image` varchar(50) NOT NULL default '',
  `atweight` float(1) NOT NULL,
  `atnumber` int NOT NULL,
  `classification` varchar(50) NOT NULL
default '',
  `casreg` longint NOT NULL,
  `block` varchar(5) NOT NULL,
  `outershell` varchar(5) NOT NULL default '',
  PRIMARY KEY  (`id`)
);

mysql> create table other(
  `id` smallint(6) NOT NULL auto_increment,
  `discovered` int NOT NULL,
  `colour` varchar(20) NOT NULL default '',
  `oxstates` varchar(50) NOT NULL default '',
  `allotropes` varchar(200) NOT NULL default
'',
  PRIMARY KEY  (`id`)
);
```

That's the tables set. Next stage is to get the data in. As the datasets are somewhat large for this article, I've placed them on my website[atj] - you will need to grab the file `article1.zip` and de-archive it; I would suggest saving it to your CSD[1]. There are three files: `catinfo.csv`, `elemitems.csv` and `other.csv`. These will need importing into your database and it is very easy to do.

```
mysql> LOAD DATA LOCAL infile 'catinfo.csv'
INTO TABLE catagoryinfo FIELDS TERMINATED BY
',' LINES TERMINATED BY '\n' (id, name,
description, image);
```

Repeat for the other files. The only difference being to change the filename and the contents of the `()` to reflect the names given to the table rows. Once imported, you should get the following message back from the MySQL server

```
Query OK, 10 rows affected, 0 warnings (0.51 secs)
Records: 10 Deleted: 0 Skipped: 0 Warnings: 0
```

This means the data has been read in happily. A quick test to show that the data is indeed in the table should confirm this

```
mysql> select name,description from catagoryinfo;
```

[concluded at foot of next page]

2 On my Linux machines, that would be /home/paul. I would suggest for those using Windows using C:\ rather than anywhere else.

Silas's Corner

Silas S. Brown <ssb22@cam.ac.uk>

Recycling throwaway hardware

I was recently given the challenge of setting up a system suitable for an individual to use for selecting and playing educational sound recordings. What made this particularly interesting was that the individual was an adult with very little schooling (so no complicated user interfaces allowed) and no knowledge of the English language. His accommodation was too small to set up a full PC with monitor, and there was absolutely no budget.

I was able to obtain an old PC that was being given away for free. It had 16M of RAM, a 2G hard disk, a slow CD-ROM drive and a soundcard. The system unit was not exactly small, but I planned to set it up in such a way that nothing else would be needed apart from the system unit and keyboard (and a pair of headphones). Of course the Windows 98 on it ran extremely slowly, was full of viruses and was probably unlicensed, not to mention being unsuitable for "blind" use, so I decided to erase it and install Linux.

First I used Knoppix (`www.knoppix.org`) to detect the sound hardware. Knoppix isn't very good at dealing with a low amount of RAM, so I had to go into "expert" mode and override some of the bootup behaviour. It helped to repartition the hard disk as soon as possible and create a swap partition. I typed lsmod and noted which kernelmodules Knoppix had loaded; this told me which modules would be needed for the soundcard.

The Linux installation to the hard disk had to be as smallas possible so as to make room for the sound recordings and also given the limited amount of RAM. Graphics was out ofthe question (but not needed for this application anyway). Knoppix was too big: I didn't want to do the drudgery required to trim it down, and anyway its installation script is broken in recent releases, and older versions of Knoppix didn't detect the soundcard at all; the unusual type of soundcard installed was supported only by the latest (2.6) version of the Linux kernel. So I needed a distribution that is up-to-date, that can give you a near-minimal installation without too much hassle, and that contains players for MP3, OGG and SPX sound files.

The obvious answer was Debian. Its most recent release (3.1 or "sarge") has a much improved installer; it probably wouldn't detect the old soundcard, but thanks to Knoppix Inow knew which module to load. I burnt myself a copy of the first CD of Debian (you don't need all the CDs) and installed a minimal system. Then I installed the 2.6 kernel, and installed the sound packages by copying the necessary package files onto a floppy disk (they weren't popular enough to be on the first CD) and finally trimmed out any remaining unnecessary packages (such as the old 2.4 kernel). Then I customised the startup scripts into /etc/init.d so as to load the correct module with modprobe, set the volume with aumix, clean out old logfiles (the disk would be very nearly full once the recordings were on it) and run my program. I also commented out potentially time-consuming startup scripts such as the filesystem check (since nowadays it uses a journalling filesystem, the check is less necessary and it doesn't matter if the user prematurely switches off the power, especially given that this application does not require saving anything).

The main part of the program was written in Python. Basically I wanted a system that would read out (in the user's native language) a list of recordings and invite him to press a number to choose one to play. Of course, I couldn't have the complication of asking him to press Enter after the number, so I used the Python curses library which interfaces with the terminal at a lower level:

```
import curses
curses.initscr()
curses.cbreak()
```

`initscr()` is required to initialise the curses library, and `cbreak()` tells it to accept one character at a time (but still allow sequences such as control-break to interrupt the program, which may be useful for debugging). Then to read a character, one would do something like:

```
import sys
response = sys.stdin.read(1)
```

That wasn't too difficult to find in the Python library documentation, and didn't require any extra libraries to be set up (it's all included with Python). We can build it up into a function that repeatedly plays a prompt while waiting for an answer like this:

```
def get_selection(play_prompt_command):
  pid = os.spawnlp(os.P_NOWAIT,"/bin/bash",
        "/bin/bash","-c",
        "while true; do %s; done"
        % play_prompt_command)
        curses.flushinp()
        response = sys.stdin.read(1)
    os.kill(pid,signal.SIGKILL)
    os.system("killall -9 ogg123;
    killall -9 mpg123;killall -9 speexdec")
    return response
```

The `killall` command is hacky but the alternative would be to write a function that goes through all the processes and checks each one to see if it is in the same process group as the current process but is not the same as the current process (we cannot include the current process in an uncatchable signal, and if any other signal is used then some sound-playing commands do not stop immediately). Also we use curses.`flushinp()` to flush the keyboard buffer (typeahead would probably be too confusing in this application).

As for what to do when something was selected (say, option 2), I decided to check for `2.mp3`, `2.ogg`, `2.spx` or adirectory called `2` (and option `0` always goes up one level).

Here's the rest of the script:

```
def get_play_command(filename):
  for extension,format in [
    (".mp3", "mpg123 %s.mp3"),
    (".mp2", "mpg123 %s.mp2"),
    (".ogg", "ogg123 %s.ogg"),
    (".spx", "speexdec %s.spx")]:
   try:
    if open(filename+extension):
       return format % filename
   except IOError: pass
  # by default returns None
```

[concluded at foot of next page]

[continued from previous page]

This will display all of the category names and their descriptions from the catagoryinfo table. Similar tests can be performed for the other tables.

I'll leave this article at this point. Next time, I'll cover SQL manual insertion, manual alteration and manual deletion and introduce the C# command line interface which I will subsequently be developing.

I must thank the following people for helping me out on this project:

- Paul Grenyer and Patrick De Ridder – both of whom have tested the compiled binaries under the .NET framework to ensure they have worked.
- Novell – for the production and development of Mono [4].
- Steve Hopley – my MySQL tutor at St. Helens College [5] and a good friend who helped me design the database these articles are based on.
- Dr. Alex Woods - my former Physical Chemistry lecturer at Liverpool John Moores University, who inspired this due to his slightly "forgetful" nature.
- Pippa Hennessy & Laurence Murphy – proof reading.

Disclaimer

No furry animals were hurt during the production of this article. I did step on something while chasing some information, but it didn't squeak, so I'm not going to count that.

Paul Johnson

Webliography

[1]http://www.mysql.com/products/mysqlcc/
[2]http://www.phpmyadmin.net/home_page/
[3]http://www.all-the-johnsons.co.uk/accu/articles
[4]Novell. The Mono Project. http://www.mono-project.com
[5]St Helens College. http://www.sthelens.ac.uk

Reviews

USUAL PREAMBLE

The editor writes

The book review section of C Vu is one of the most important parts of our output, but we are in need of more reviewers. While it is good that we have a strong cohort willing to do the reviews, fresh blood will bring another perspective to the book reviews.

For a mere £5 per book (this is for postage – not the book itself), you have a list of some of the most up to date books around (some of which aren't even released in the UK at the time of being offered to the ACCU). All we ask is that you review the books. You even get to keep it!

Calling all those who have not returned their reviews:

The only condition when you receive a book from the ACCU is that you actually review it! Both Francis and I have a list of books with names next to them and no reviews to show. Could I please ask that you write your review(s) in a timely fashion?

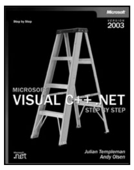
All prices given are from the Blackwells website where they have not been provided by the reviewer. It may be possible to purchase them at a reduced cost elsewhere.

Paul

C and C++

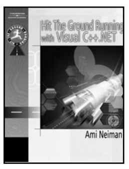
1001 Visual C++ Programming Tips by Charles Wright Prima Tech ISBN: 0761527613 £51.99 Pages: 500, plus 2 CDs: Microsoft's VC++6.0 Introductory edition and sample code Reviewer: Frances Buontempo

This book claims to be aimed at beginners and experts. Some sections give concise information clearly, but most are written in a chatty, misleading style. The author readily admits the aim is to present code that "works." Unfortunately this sometimes means a lack of rigour. For example, `new` is encouraged over `malloc` since `sizeof` is not needed. Though rectified elsewhere, it leaves the reader misinformed.The tips are a mixture of C and C++, showing how to use VC++ to write applications.Most are variations on text editors, with pointers on using the GDI to draw points and lines. Occasional reference is made to FORTRAN and BASIC for comparative purposes. Many complicated ideas are introduced before a whole sample that will compile is presented. This is an MDI MFC app, which may terrify a beginner. Furthermore, when `CArray`, `CList` and `CMap` are introduced, templates have not been explained. Overall, this book therefore is unsuitable for a C/C++ beginner.On a more positive note, it introduces some methods through API code, before doing the same thing in MFC, illustrating what the MFC is up to and why. However, the tips are often about several different issues, so require skipping about to get the whole story, which is difficult with no page numbers. Also, being so vast, it is difficult to read from cover to cover. For someone who knows C/C++, there are some good tips, but they are buried in the chatty style, are few and far between and frequently are not explained clearly. It reads more like a stream of consciousness than a reference book. On balance, I would not be happy paying the full cover price for this.



Microsoft Visual C++ .NET Step by Step by Julian Templeman and Andy Olsen Microsoft Press ISBN 0-7356-1907-7 £28.49 Reviewer: Mike Spence

Visual C++ .NET Step by Step is part of the Microsoft Press library of "Step by Step" titles designed to "Teach yourself Visual C++ - and begin developing for Microsoft .NET - one step at a time." I (slightly mis)quote from the back of the book. The book is split into 27 individual chapters, each of which should require no more than a single night of study to cover completely, and probably a lot less if you have any previous experience in the matter under discussion. The choice of subject matter for some of these chapters I felt rather basic at times, attempting to cover the fundamentals of OOP and C++ (to be fair the C++ .NET version was covered in the few placed where this differs from standard C++). Then later chapters covering .NET specific subject seemed slightly rushed in comparison. Maybe trying to teach C++, OOP and C++ .NET in a single book is expecting a little much in slightly over 550 pages. The chapters take the form of descriptive text, maybe a diagram and an example that is a list of step by step (hence the series title I guess) instructions, which when carried out demonstrate the topic being discussed. The descriptions themselves appeared well written and I had no problem understanding the subject matter. The examples were again well written and to me appeared to contain no obvious technical errors. However the Step byStep style of the examples I personally didn't like. I'm not convinced that following a number of simple steps teaches anything other than what to do with the mouse and keyboard. However in the book's defence the individual steps do try and convey the reason behind the each step. My big problem with the book is that like so many "Teach Yourself" books before it, Visual C++ .NET Step by Step falls somewhat short of the level of detailed needed to truly teach anything but an overview of the subject matter. It's a good starting point for a novice who doesn't mind the step by step format of the examples, but for others it's probably best to look elsewhere.



Hit The Ground Running with Visual C++ .NET by Ami Neiman Thomson/Delmar ISBN: 1-4018-7880-6) 689pp @ £52.91 Reviewer: Alan Lenton

If you hit the ground running with this book, you will break both legs. I got it because I needed to use Visual Studio .NET and needed some help to get it running and learn about managed code. (Bring back proper written documentation with software!) To be fair I did find pages 6 through to 13 useful. That, unfortunately, was about as far as it went. This style of walking you through various tasks has limited utility at the best of times, since it provides little understanding of the concepts involved. Without that understanding you can't apply what you've learned to anything that's similar, but different. The author falls into the familiar trap of trying to cater for both 'basic and advanced' programming skills. The result is a mish-mash of material, much of it too trivial to be useful to an experienced programmer, but without the explanation a beginner would need to make sense of it. And last, but by no means least, the code has typographical errors. This is an absolute no-no. It's difficult enough learning something new without being fed incorrect code. Oh, and incidentally, the author has the

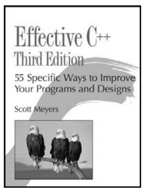
[continued from previous page]

```
  def menu():
    play_prompt_command =
  get_play_command("index")
    if play_prompt_command:
      while True:
        response =
  get_selection(play_prompt_command)
        if response=="0": return
        play_cmd = get_play_command(response)
        if play_cmd: os.system(play_cmd)
        else:
          # it might be a directory
          try:
            os.chdir(response)
            except OSError: continue
            menu()
            os.chdir("..")
      else:
        print "\a Warning: index not found!"
        # but continue running (up one level)

  def main():
    try:
      while True: menu()
    except KeyboardInterrupt: curses.endwin()if
  __name__ == "__main__": main()
```

I threw in a feature that checks for an executable script on any CD-ROM that is inserted (for future expansion) and then delivered the box. It's a shame that throwaway PCs don'tcome any smaller.

Silas Brown

most execrable taste in colours I've ever come across - how could any self-respecting author tell their readers to colour the window background lavender? Avoid like the plague.



Effective C++ Third Edition by Scott Meyers, Addison Wesley ISBN: 0-321-33487-6, 297 pages £31.99
Reviewer: Pete Goodliffe

Highly Recommended. First, the punchline: this book is highly recommended. There, I said it. Now you can skip the rest of this review. Scott Meyers is one of those few industry gurus with the rare combination of exceptional technical talent and a keen ability to explain the intricacies of C++ (a powerful but - at times painfully - complex language). He does this with clarity and entertainment. This is a new edition of the first part of his "effective" book series (including "*More Effective C++*" and "*Effective STL*"). The previous editions of *Effective C++* book are legendary, so what has he done to this third edition, and is it an improvement? The answer is: an awful lot, and yes. This edition has been greatly revised and is thoroughly up-to-date. Meyers has totally reworked, restructured and improved the text. The contents are considerably different from the second edition, and more appropriate to an age where templates, generic programming, and exceptions are the staple fare of C++ work. Meyers weaves in many references the new tr1 additions to the standard library, and evangelises the popular Boost library. Effective C++ contains a series of 55 "items" (short self-contained articles on a specific subject) grouped into nine chapters. The material ranges from a foundational understanding of what C++ is, through basic mechanical maxims, to design considerations, OO concerns, and generic programming fundamentals. Meyers covers the essential precepts of C++ programming. You must absorb every item, and integrate them all into your coding DNA, to be a truly useful C++ programmer. I couldn't find anything to argue with technically. The book is lucid, accurate , and thoughtfully produced, with well-chosen topics. It is undoubtedly better than previous the editions. The presentation make occasional, and clever, use of colour to highlight specific parts of code examples. A nice touch. It's well-geared to the target readership (be they newbie C++ programmers, or people coming from a background in other languages). There are some downsides. A few pages suffered colour bleed from the red ink. Hopefully this is just my review copy, but check your book before taking it to the till. The new/delete chapter could have been replaced with something less specific and more generally useful (this material is more "advanced" and should have been pushed off elsewhere). So do you need to buy this book?

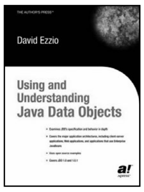
If you're a new C++ programmer: Yes, definitely.

If you are an experienced C++ programmer: Check it out - I'd highly recommended it.

If you own a previous edition: The book is significantly different from previous versions. A different take on some of the material is available in Sutter's books and also in Vandevoorde and Josuttis' C++ Templates (but they go into far greater detail). Consider this edition if you don't have those books.

The punchline once again: this is a great book. Essential reading for C++ programmers.

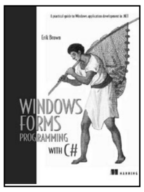
C# and Java



Using and Understanding Java Data Objects by David Ezzio PressISBN: 1-59059-043-0Pages: 426 £35.20
Reviewer: Michel Greve

From the back cover: *This book has two missions: The first mission is to give you a tour of Java Data Objects. The second mission is to give you a tour of the open source JDO Learning Tools.*. From the introduction (a snippet): *This book is intended for Java programmers and application architects. It assumes that you know how to program in Java, and it assumes that you want to use JDO and understand how it works. This book emphasizes what you need to know to use JDO effectively.* We shall see of this holds. Basically this book consists of three parts: JDO itself, using it and an appendix. It has a nice layout and it reads well. The paragraphs are short, with the chapters at about 40 pages. The chapters have some example code, tables and diagrams in it, so it isn't a massive piece of text. David describes in his book all the important parts of the JDO standard (persistence managers, queries, transactions etc). He also shows you the traps and the not so well defined parts of the JDO standard. He writes how to use JDO in three different applications (swing, a web application and enterprise Java beans). In the appendices there are some large UML diagrams and I really mean large which are, to me, a waste of space. The glossary is a nice thing to have and the index works but could be better (searching for Fields gives me four sub indexes. Two of them ("application data class" and "cached persistent"), should be left out because on the referring pages there is no real information about them. They are only mentioned there.

The problem with this book is that once you have read it, you will never read it again. The information is there, but it is hidden in the text. Nowhere there is an exclamation mark in the margin with the important information besides it, or a grey area with an important remark (although there are some notes in the book, but with less useful information). What is the verdict? The mission statements are fulfilled. The rather bold introduction puts you on the wrong foot. This book will not help you to learn JDO, but it will help you to deepen your knowledge. If David changed his book to emphasize the traps and the not so well defined parts of the JDO standard, he would get a highly recommended. Unfortunately, I only can give it a recommended.



Windows Forms Programming with C# by Erik Brown Manning ISBN 1-930110-28-6 715 pages £49.48 Reviewer: Max Palmer

Windows Forms Programming with C# is aimed at people who wish to develop desktop applications for .NET using the classes that are part of the Windows Forms namespace. Although it is not a general book on .NET or C#, some more general material is provided in the first part and the appendices to get people up to speed in these areas. The book itself is divided into three parts. The first provides an overview of Windows forms and using Visual Studio, the second part deals with basic Windows form programming and the final part deals with advanced controls, such as list views, data binding and tree views. All code, as the title suggests, is in C# and the approach that is adopted is very much 'hands on'. As is common with books of this type, the author introduces concepts through the development of an application, or suite of applications, in this case based on organising and browsing photos. Often this kind of approach can seem contrived, and can at times be both difficult or tedious to follow. However, in most cases the author manages to introduce new topics without seeming to add unnecessary features to the application(s) being developed and the style adopted for explaining tasks is very clear and readable. It is certainly one of the best tutorial approaches I have come across. The book also attempts to engender good general programming and Windows UI development practices such as the use of application versioning, setting the tab order on dialogs and support for keyboard shortcuts - important areas which can easily be overlooked. However, I would have liked to have seen localisation covered in more detail.The material presented is well explained and most of the common windows controls are covered. It is also useful to see sections on multiple document interfaces (MDIs) as well as single document interfaces (SDIs) and dialogs. However, on occasion I felt that the use of a control within an application may have slightly limited the breadth of the discussion. The other criticism I have is that the topics discussed in the final chapter, including printing and support for drag and drop operations, felt as if they had been tagged on to the end of the book. They should really be explained in more detail or not at all. In summary, the book is both clear and well written. I would recommend it to someone who has limited knowledge of programming Windows form applications and who enjoys a 'hands on' approach to learning.

Software Development

An OO Approach to Programming Logic & Design by Joyce Farrell Thompson ISBN 0-619-21563-1
Reviewer: Paul Thomas

I really like just over half of this book. It is, on balance, an extremely well written introduction to programming as a craft. I would recommend it to anyone who wanted to start programming in modern language like Java or C# and wanted to make a career of it rather than a hobby. My problem is that it is not truly OO. I have heard good things about the author's other books on programming and I have the sneaking suspicion that this is just a marketing-led update on the original but with enough objects in it to get OO into the title. The introductory chapters contain enough to get anybody started without the need for much background knowledge. I particularly like the style of writing as I feel it has the right balance between a gentle introduction and a deep explanation. There is enough structure to make revision from the text simple as well as plenty of diagrams and code examples in the right places.

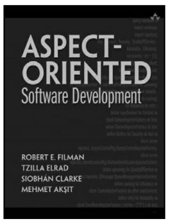
Once the book moves on to programming proper, we meet the all-important flow chart and structures of code flow are discussed before technical details of ifs and whiles. This way round, new programmers are taught why certain structures are preferred instead of being given the means to write sloppy code followed by the restrictions.

In the advanced topics, there is an excellent chapter devoted to event-driven programming for GUIs that recommends story-boarding as a design technique and introduces threads where lesser books might gloss over them. The UML introduction is useful from a programmers point of view but there are a few mistakes in the diagram notations. The book rounds off with a chapter on exception handling.

The book is promoted as language independent and it almost is. It would be more true to say that it covers a large array of languages. There are enough examples to give the reader a feel for specific languages like Java without tying them to it. The text focuses on teaching you how to program like a professional. No tricks or hacks, just well proven techniques. It does not represent the state of the art or “High Church” but is practical and well presented. Each chapter has a thorough review and exercise section with a case study running throughout.

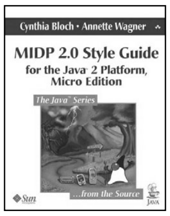
The only let down is, again, the claim to an Object Oriented approach. Objects are introduced early in the book, but only as a means of programming with a modern language. By chapter 3 we are introduced to getter/setter methods and from then on, all objects are either pure data or pure code where all methods are static. Inheritance is dealt with near the end as little more than an extension mechanism. Even in the GUI chapter, the main example ends up as a procedure call with no object assigned responsibility. This is not a book on design, so it might not seem an issue, but any new programmer would be left with the impression that OO is little more than a technique to break up software into smaller blocks. This is fine if the student just wants to write their own code, but is at odds with the rest of the book which is thoroughly professional.

I would recommend this book to anyone new to programming that is likely to use VB, Java or C# for their own small programs. Anyone serious about learning OO as a paradigm should steer clear.



Aspect Oriented Software Development by Robert E Filman, Tzilla Elrad, Siobhan Clarke and Mehmet Aksit Addison-Wesley 755pp @ $59.99 (£50.58 – Blackwells) ISBN: 0-321-21976-7
Reviewer: Alan Lenton

I picked up this book to review because I wanted to learn what Aspect Oriented Programming (AOP) was all about. I chose the correct book, and found out more than I ever wanted to know about the subject! Anyone who has written an object oriented program of even moderate size or complexity will have discovered that no matter how carefully you try to resolve the problem into loosely coupled classes, there are always some issues that spread themselves across all your object boundaries. Debugging information, logging and security are examples that spring immediately to mind.In AOP these issues are known as ‘crosscutting’ issues or concerns, and AOP is an effort to deal with these problems. To simplify greatly (this is not the place for a major discussion of AOP) AOP handles this by concentrating the crosscutting code in its own classes and then arranging for a separate compiler, or pre-compiler, to weave the crosscutting code into the appropriate places in the other classes. As you can imagine figuring out the appropriate places is not a trivial task!The book itself will give any competent programmer a thorough understanding of AOP, and also some excellent ideas about design and analysis using Aspect Oriented methods. The book is cutting edge stuff, and requires hard intellectual work, but it is, in my opinion well worth the effort. The editors have brought together some thirty papers on the topic, and they provide a useful assessment of the state of the art.One caution though. Don’t expect to be able to read this book and then leap out and start programming in an AOP language, such as AspectJ. Much work remains to be done before Aspect Oriented Programming is ready to move out of the academic world and into the rough and tumble of production work.It’s not a cheap book, or an easy read, and it won’t immediately impact your work. Nonetheless, I thought it was well worth the time I put into reading it, and it did give me some ideas about how to improve my non-AOP programming.



MIDP Style Guide by Cynthia Bloch and Annette Wagner Addison-Wesley iSBN 0-321-19801-8 £30.99
Reviewer: Paul Thomas

There is not a great deal I can say about this book. If you are implementing applications or platforms for MIDP 2.0 then you quite simply need this book. Without it, you will likely create something that users find impossible to use; it might have the tightest coding imaginable and use every trick known to man, but put it on a platform it wasn’t developed for and it’s a chocolate teapot.

Interoperability is the diplomacy of the technology world and requires some finesse. Focus exclusively on trade relations with one partner and the fickle nature of the market can leave you in legacy-land watching the world leave you behind. To survive requires an enormous effort to keep up with events at court, or an etiquette guide. The style guide provides invaluable advice on maintaining good relations between platform vendors, application developers and users.

There’s no code in the book. Implementation details are not covered, but it is assumed that the reader is at least familiar with the capabilities of the MIDP platform. Some sections do delve into the specific behaviour of form items and abstract commands, but the book as a whole can be read by anyone involved.

Advice is split into categories by severity. The “strongly recommended” flavour is something that other standards groups might have mandated in specifications. As an example, it is strongly recommended that platforms display at least 4 characters in a soft button label. This kind of advice gives developers a platform with more predictable behaviour than the raw technical specs alone. From the other side, application authors are advised on just about every aspect of usability. While this might not suit the bedroom hacker, it’s equivalent to thousands in consultancy fees and could make all the difference to an authoring house.

Highly recommended

Due to lack of space, not all book reviews could be printed in this issue.
Reviews of the following books can be found on the website (www.accu.org) and will be printed in the next issue if space permits.

Software Development

Visual Studio Hacks by James Avery O’Reilly 0-596-00847-3 £17.50/$24.95
Reviewer: Derek M. Jones

Perl

Computer Science and Perl Programming by Jon Orwant (ed) O’Reilly ISBN: 0-596-00310-2 £28.50
Reviewer: Joe Mc Cool

Perl Graphics Programming by Shawn Wallace O’Reilly ISBN: 0-596-00219-X £28.50.
Reviewer: Joe Mc Cool

Extending and Embedding Perl by Tim Jenness & Simon Cozens Manning ISBN 1930110 82 0 £40.50
Reviewer: Joe Mc Cool

Linux

The definitive guide to SAMBA by Roderick W Smith APress ISBN 1-59059-277-8. 625 pages £31.50
Reviewer: Mark Easterbrook.

MySQL Cookbook by Paul DuBois O’Reilly BN 0-596-00145-2 £24.85
Reviewer: Joe Mc Cool

Building Secure Servers with Linux by Michael D. Bauer O’Reilly ISBN 0-596-00217-3 £31.95
Reviewer: Joe Mc Cool

Running Linux by Matt Welsh, et alO’Reilly ISBN: 0596002726 £31.95
Reviewer: Joe Mc Cool