Volume 6, Number 10 October 1994 $2.50

MUUG Lines

Manitoba UNIX® User Group

Newsletter of the Manitoba UNIX® User Group

ATM: The Next Big Thing in Communications

By Peter Dyson

Every business has its Holy Grail. Physicists root through the cosmic strings seeking a Grand Unified Theory of the Universe; empire builders salivate over the perfect hostile takeover. And telecom wizards hunt for the ideal data-transfer protocol: suitable for all data types and all wiring schemes; robust in recovering errors, yet light on overhead; backward compatible, yet a solid foundation for future growth, and so on.

To hear the evangelists of the Atm Forum tell it, at least this Grail has been found. Asynchronous Transfer Mode (ATM) is everything you ever wanted for data transfer, and more. It is scalable over a wide range of speeds and network sizes; it carries voice, motion video and computer files with equal ease. And it's coming to a telco near you real soon —in some areas, sooner than isdn.

ATM characteristics

Atm is a group of standards for moving packets of data over a network. It is characterized by several features: switched-circuit topology (think of the telephone system); small packets of data (53 bytes apiece); virtual circuits; and negotiable, guaranteed bandwidth.

Atm uses a connection-oriented approach. Each device on the network is connected to a central switch, which sets up connections between devices; a connection stays in place until the switch is told to disconnect it. While a connection is in place, a workstation can count on a known amount of bandwidth being available, regardless of the other connections that may be going through the switch. Atm is similar in spirit to the telephone system: local switches connect close neighbors, central exchanges interconnect the switches and fiber backbones span long distances.

This is in marked contrast to computer networking schemes such as Ethernet, Token Ring or LocalTalk. These are all broadcast-oriented approaches; every packet, no matter which station it is addressed to, gets sent to many stations. As you add more stations, the available bandwidth is spread ever-thinner.

The advantages of a switched-circuit network are flexibility and scalability. Each device on an atm net can run at its own speed; an interface card at the switch handles the clocking and buffering issues. (In contrast, every device on an Ethernet must go at the same speed or the whole net fails.) An atm network can grow as big as needed without cutting performance simply by adding more switches.

Another advantage is that switches can be connected by multiple paths. If a circuit fails or a fiber is bulldozed, the traffic can be rerouted around the damage. When everything is working, the network can allocate bandwidth to match demand by adjusting the routes.

All packets are created equal. The universal atm packet (also called a cell) is 53 bytes long — 5 bytes of header and 48 bytes of message — no matter whether it contains voice, video or files. Because all packets are the same size, the network and the switches don't have to understand each separate kind of data, but can blindly forward packets to their destinations.

Although 48 sounds like a small number, it was deliberately chosen. Audio and video quality suffers if there are long time delays in transmission. (For voice conversations, the threshold of annoyance is about 50 milliseconds of lag time. Anyone who has made a satellite phone call knows how disconcerting longer delays can be.) It is thus important to minimize the time to assemble and forward packets. The shorter the packet, the less time to process it — the packet cannot be sent until the last byte has been digitized.

This month's MUUG meeting features an excellent video presentation on ATM technology. If you are like most of us, you have heard of ATM and may even know a little about it — but don't really know how it will affect you. Hope to see you at the October meeting.

This Month's Meeting

Meeting Location:
Our next meeting is scheduled for Tuesday, October 11, at 7:30 PM. Once again, the meeting will be held in the auditorium of the St-Boniface Hospital Research Centre, just south of the hospital itself, at 351 Taché. You don't have to sign in at the security desk — just say you're attending the meeting of the Manitoba UNIX User Group. The auditorium is on the main floor, and is easily found from the entrance.

Meeting Agenda: See inside for details.

Inside This Issue

One Year Later...

By Andrew Trauzzi

Well here I am, finishing up my tenth issue of MUUG Lines. From your comments over the past year, you have generally appreciated the content of our newsletter. I hope that you will continue to do so. Looking back, I seemed to have more member-written material in earlier issues. I think that articles written by members are important as they give other members a look at other viewpoints and interests.

GNU Where are You?

One notable column that has been absent so far is the ever-popular GNU Review by Peter Graham. Maybe we will see another couple of columns this year (how about it Peter?). If anyone else would like to review some GNU product that they installed or liked, please send it in! I'm sure Peter would appreciate the help.

Ch Ch Ch Ch Changes...

Last year I said that I would change the look of the newsletter somewhat. So far, it hasn't changed at all. The reason for this is twofold. First, Gilbert did such an excellent job on the current look that it's difficult to decide where to make changes. I hate change for the sake of change — it should always be for the better. Second, I haven't had the time to make any alterations. I don't have a PostScript-capable laser printer at home, so it's a little difficult to see those PostScript effects! In any case, I will either get a new laser printer, and/or make some time to alter the look.

Thanks for the Fish!

I would like to personally thank a member of the board who will not be running for re-election this year — Rick Horocholyn. Rick has been MUUG's treasurer for a number of years (he never seemed to be able to get someone else to do that job!), and has done an excellent job. Whoever fills your shoes will have a big job to do. I mean that the person will have to re-calculate all of the books! Seriously, enjoy the extra time you have, and I hope to see you at least a few meetings.

Coming Up

Here are a few things to look for in upcoming issues of MUUG Lines:

- A review of the MKS Toolkit for DOS;
- CD Rom and Imaging Technology for UNIX;
- Audio and Digital Recording in UNIX.

Until then, keep those keyboards busy!

The 1993-1994 Executive

President:	Bary Finch	(W) 934-2723
Vice-President:	Ramon Ayre	(W) 947-2669
Treasurer:	Rick Horocholyn	(W) 474-4533
Secretary:	Brad West	(W) 983-0336
Membership Sec.:	Greg Moeller	(H) 786-6132
Mailing List:	Roland Schneider	1-482-5173
Meeting Coordinator:	Roland Schneider	1-482-5173
Newsletter editor:	Andrew Trauzzi	(W) 986-3898
Publicity Director	Rory Macleod	(W) 488-5168
Past President	Susan Zuk	(W) 989-3530
Information:	Bary Finch	(W) 934-2723
		(FAX) 934-2620
(or)	Andrew Trauzzi	(W) 986-3898
		(FAX) 986-5966

Copyright Policy and Disclaimer

This newsletter is ©opyrighted by the Manitoba UNIX User Group. Articles may be reprinted without permission, for non-profit use, as long as the article is reprinted in its entirety and both the original author and the Manitoba UNIX User Group are given credit.

The Manitoba UNIX User Group, the editor, and contributors of this newsletter do not assume any liability for any damages that may occur as a result of information published in this newsletter.

Advertising Rates

Quarter page	$50
Half page	$75
Full page	$100
Insert (1-4 pages)	$100

Above prices are per issue. The first ad is charged at the full price; each successive month is 1/2 price.

Ad copy must be submitted by the final copy deadline for an issue (usually 3 weeks prior to the monthly meeting) in a format acceptable to the editor. (Please make arrangements with editor beforehand.)

Internet E-mail: editor@muug.mb.ca

Group Information

The Manitoba UNIX User Group meets at 7:30 PM the second Tuesday of every month, except July and August. Meeting locations vary. The newsletter is mailed to all paid-up members one week prior to the meeting. Membership dues are $25 annually and are due as indicated by the renewal date on your newsletter's mailing label. Membership dues are accepted at any meeting, or by mail.

Manitoba UNIX User Group
P.O. Box 130, Saint-Boniface
Winnipeg, Manitoba R2H 3B4

Internet E-mail: membership@muug.mb.ca

PRESIDENT'S CORNER

October Overview

By Bary Finch

This month we will be holding our Annual General Meeting (AGM), as the first part of our regular meeting. This is when we will introduce the members of the new board for the 1994 / 1995 year. There will be four of us continuing on from last year, with our new additions rounding out the board. Having several new people added will give an always helpful change in perspective that keeps the board dynamic.

There may be changes in the roles of the existing board members that remain, and the new members will take on their desired positions. However, this will all be transparent to our membership. The board will strive, as always, to make the operations of MUUG unobtrusive to the general membership.

The second half of our meeting will be a video presentation on Asynchronous Transfer Mode (ATM). ATM is becoming more noticeable in the trade press all the time, and hopefully this video will give you a good introduction as to what ATM means to you.

I would like to take this opportunity to thank Susan Zuk, our Past President, for hosting the September meeting. I was away for an extended period of time, and appreciated Susan taking the time to organize the meeting.

From what I heard of our September meeting, we had a great presentation on Object Oriented Technologies from Online Business Systems. As one of our Corporate Sponsors, Online Business Systems helps us even when they are not last few weeks.

Our main plan while travelling had been to use ATMs (Automated Teller Machines, not Asynchronous Transfer Mode) to get cash. This worked well in London and Austria, due many of their banks using the PLUS network, and not just the Cirrus network. It still seems rather amazing to be standing in Salzburg giving a presentation at one of our meetings.

We will be looking forward to more presentations from other Corporate Sponsors as the year progresses. This is one of MUUG's best resources for getting current industry information to you, our members.

Well that's about it for MUUG business for now. I thought some people may be interested to here a few words about the state of technology that I saw while I was travelling thru Europe over the last few weeks.

Our main plan while travelling had been to use ATMs (Automated Teller Machines, not Asynchronous Transfer Mode) to get cash. This worked well in London and Austria, due many of their banks using the PLUS network, and not just the Cirrus network. It still seems rather amazing to be standing in Salzburg and getting Austrian Shillings from my bank account in Winnipeg.

The key is the bank's choice of PLUS versus Cirrus, the large USA bank network. Unfortunately France and Italy didn't choose to use the PLUS network, but anyone that uses Cirrus would likely have no problem as many ATMs connect to the Cirrus network.

The ideal backup plan to getting cash via the PLUS network is to use your VISA card for cash advances from the ATMs that don't have PLUS. Almost all the ATMs we saw had VISA access.

You may think this is an expensive way to get cash, but when you see the bad exchange rate that many Change offices offer, added to their high commission rates per traveller's cheque, VISA isn't so bad.

The quick view I had of PC technology came from visiting large department stores: Harrod's in London, and Galleries Lafayette and Printemps in Paris. Both had extensive PC departments, with seemingly as many models as one would find here, and with pricing very comparable to here as well, or maybe slightly higher.

I also walked by a number of smaller computer stores that looked to offer a variety typical to a store here. So I don't think Europe is missing out on much of what we get here. As to whether the average person can afford much of the technology is another question, as I heard salaries were typically less than what we would get here.

Well that's my quick overview of technology in Europe. Now I'm looking forward to introducing our new board to you at the October meeting. See you then!

CORPORATE SPONSORS

The Manitoba UNIX User Group

gratefully acknowledges the generous support of the following

Corporate Sponsors

IBM

digital™

HEWLETT PACKARD

XEROX The Document Company

UM

Great-West Life Assurance Company

ONLINE BUSINESS SYSTEMS

TANDEM COMPUTERS

C++ Q&A

By Marshall P. Cline

This month's column finishes inheritance and looks at virtual member functions — thought by some to be the key to understanding C++. The complete C++ FAQ is now available in book format — Addison-Wesley Publishers 0-201-58958-3 $32.25.

Question 51: Does array-of-Derived is-NOT-a-kind-of array-of-Base mean arrays are bad?
Yes, 'arrays are evil' (just kidding :-).
There's a very subtle problem with using raw built-in arrays. Consider this:

```
    void f(Base* array_of_Base) {
      array_of_Base[3].memberfn();
    }
    main() {
      Derived array_of_Derived[10];
      f(array_of_Derived);
    }
```

This is perfectly type-safe, since a `D*` is-a `B*`, but it is horrendously evil, since `Derived` might be larger than `Base`, so the array index in `f()` not only isn't type safe, it's not even going to be pointing at a real object! In general it'll be pointing somewhere into the innards of some poor `D`.
The fundamental problem here is that C++ cannot distinguish a `ptr-to-a-thing` from a `ptr-to-an-array-of-things` (witness the required '`[]`' in '`delete[]`' when deleting an array as another example of how these different kinds of ptrs are actually different). Naturally C++ 'inherited' this feature from C.

This underscores the advantage of using an array-like *class* instead of using a raw array (the above problem would have been properly trapped as an error if we had used a '`Vec<T>`' rather than a '`T[]`'; ex: you cannot pass a `Vec<Derived>` to '`f(Vec<Base>& v)`').

Inheritance — Virtual Functions

Question 52: What is a 'virtual member function'?
A virtual member function is a member fn preceded by the keyword '`virtual`'. It has the effect of allowing derived classes to replace the implementation of the fn. Furthermore the replacement is always called whenever the object in question is actually of the derived class. The impact is that algorithms in the base class can be replaced in the derived class without affecting the operation of the base class. The replacement can be either full or partial, since the derived class operation can invoke the base class version if desired. This is discussed further below.

Question53: What is dynamic dispatch? Static dispatch?
In the following discussion, '`ptr`' means either a pointer or a reference. When you have a ptr to an object, there are two distinct types in question: the static type of the ptr, and the dynamic type of the pointed-to object (the object may actually be of a class that is derived from the class of the ptr).

The 'legality' of the call is checked based on the static type of the ptr, which gives us static type safety (if the type of the ptr can handle the member fn, certainly the pointed-to object can handle it as well, since the pointed-to object is of a class that is derived from the ptr's class).

Suppose ptr's type is 'List' and the pointed-to object's type is 'FastList'. Suppose the fn 'len()' is provided in 'List' and overridden in 'FastList'. The question is: which function should actually be invoked: the function attached to the pointer's type ('List::len()') or the function attached to the object itself ('FastList::len()')?

If 'len()' is a virtual function, as it would be in the above case, the fn attached to the object is invoked. This is called 'dynamic binding', since the actual code being called is determined dynamically (at run time).

On the other hand, if 'len()' were non-virtual, the dispatch would be resolved statically to the fn attached to the ptr's class.

Question 54: Can I override a non-virtual fn?
Yes but you shouldn't. The only time you should do this is to get around the 'hiding rule' (see below, and ARM sect.13.1), and the overridden definition should be textually identical to the base class' version.

The above advice will keep you out of trouble, but it is a bit too strong. Experienced C++ programmers will sometimes override a non-virtual fn for efficiency, and will provide an alternate implementation which makes better use of the derived class' resources. However the client-visible effects must be *identical*, since non-virtual fns are dispatched based on the static type of the ptr/ref rather than the dynamic type of the pointed-to/referenced object.

Question 55: Why do I get the warning "`Derived::foo(int) hides Base::foo(double)`"?
A member function in a derived class will *hide* all member functions of the same name in the base class, *not* overload them, even if `Base::foo(double)` is virtual (see ARM 13.1). This is done because it was felt that programmers would, for example, call `a_derived.foo(1)` and expect `Derived::foo(double)` to be called. If you define any member function with the name '`foo`' in a derived class, you must redefine in class Derived all other `Base::foo()`'s that you wish to allow access from a Derived object (which generally means all of them; you should [generally] *not* try to hide inherited public member functions since it breaks the 'conformance' of the derived class with respect to the base class).

```
      class Base { public: void foo(int); };
      class Derived : public Base {
            public: void foo(double);
            void foo(int i) { Base::foo(i); }
            // ^ override it with itself
      };
```

Dr. Marshall P. Cline is the founder and President of Paradigm Shift, Inc., a firm that specializes in on-site training for C++, OOD, OOA, consulting, and reusable/extensible C++ class libraries. For more information, send e-mail to "info@parashift.com".

UNIX Q&A

Originally Compiled by Ted Timar

Submitted by Andrew Trauzzi

This month's UNIX Q&A looks at an interesting way to pass shell variables to awk, and getting rid of those pesky zombies.

Question 1: Is it possible to pass shell variable settings into an awk program?

There are two different ways to do this. The first involves simply expanding the variable where it is needed in the program. For example, to get a list of all ttys you're using:

```
who | awk '/^'"$USER"'/ { print $2 }'
```

Single quotes are usually used to enclose awk programs because the character '$' is often used in them, and '$' will be interpreted by the shell if enclosed inside double quotes, but not if enclosed inside single quotes. In this case, we *want* the '$' in "$USER" to be interpreted by the shell, so we close the single quotes and then put the "$USER" inside double quotes. Note that there are no spaces in any of that, so the shell will see it all as one argument. Note, further, that the double quotes probably aren't necessary in this particular case (i.e. we could have done:)

```
who | awk '/^'$USER'/ { print $2 }'
```

but they should be included nevertheless because they are necessary when the shell variable in question contains special characters or spaces.

The second way to pass variable settings into awk is to use an often undocumented feature of awk which allows variable settings to be specified as "fake file names" on the command line. For example:

```
who | awk '$1 == user{ print $2 }'user="$USER" -
```

Variable settings take effect when they are encountered on the command line, so, for example, you could instruct awk on how to behave for different files using this technique. For example:

```
awk '{ program that depends on s }' s=1 file1 s=0 file2
```

Note that some versions of awk will cause variable settings encountered before any real filenames to take effect before the `BEGIN` block is executed, but some won't so neither way should be relied upon.

Note, further, that when you specify a variable setting, awk won't automatically read from stdin if no real files are specified, so you need to add a "`-`" argument to the end of your command, as I did at above.

Question 2: How do I get rid of zombie processes that persevere?

Unfortunately, it's impossible to generalize how the death of child processes should behave, because the exact mechanism varies over the various flavors of Unix.

First of all, by default, you have to do a `wait()` for child processes under ALL flavors of Unix. That is, there is no flavor of Unix that I know of that will automatically flush child processes that exit, even if you don't do anything to tell it to do so.

Second, under some SysV-derived systems, if you do "`signal(SIGCHLD, SIG_IGN)`" (well, actually, it may be SIGCLD instead of SIGCHLD, but most of the newer SysV systems have "`#define SIGCHLD SIGCLD`" in the header files), then child processes will be cleaned up automatically, with no further effort in your part. The best way to find out if it works at your site is to try it, although if you are trying to write portable code, it's a bad idea to rely on this in any case. Unfortunately, POSIX doesn't allow you to do this; the behavior of setting the SIGCHLD to SIG_IGN under POSIX is undefined, so you can't do it if your program is supposed to be POSIX-compliant.

If you can't use SIG_IGN to force automatic clean-up, then you've got to write a signal handler to do it. It isn't easy at all to write a signal handler that does things right on all flavors of Unix, because of the following inconsistencies:

On some flavors of Unix, the SIGCHLD signal handler is called if one *or more* children have died. This means that if your signal handler only does one `wait()` call, then it won't clean up all of the children. Fortunately, I believe that all Unix flavors for which this is the case have available to the programmer the `wait3()` call, which allows the WNOHANG option to check whether or not there are any children waiting to be cleaned up. Therefore, on any system that has `wait3()`, your signal handler should call `wait3()` over and over again with the WNOHANG option until there are no children left to clean up.

On SysV-derived systems, SIGCHLD signals are regenerated if there are child processes still waiting to be cleaned up after you exit the SIGCHLD signal handler. Therefore, it's safe on most SysV systems to assume when the signal handler gets called that you only have to clean up one signal, and assume that the handler will get called again if there are more to clean up after it exits.

On older systems, signal handlers are automatically reset to SIG_DFL when the signal handler gets called. On such systems, you have to put "`signal(SIGCHILD, catcher_func)`" (where "`catcher_func`" is the name of the handler function) as the first thing in the signal handler, so that it gets reset. Unfortunately, there is a race condition which may cause you to get a SIGCHLD signal and have it ignored between the time your handler gets called and the time you reset the signal. Fortunately, newer implementations of `signal()` don't reset the handler to SIG_DFL when the handler function is called. To get around this problem, on systems that do not have `wait3()` but do have SIGCLD, you need to reset the signal handler with a call to `signal()` after doing at least one `wait()` within the handler, each time it is called.

The summary of all this is that on systems that have `wait3()`, you should use that and your signal handler should loop, and on systems that don't, you should have one call to `wait()` per invocation of the signal handler.

One more thing — if you don't want to go through all of this trouble, there is a portable way to avoid this problem, although it is somewhat less efficient. Your parent process should fork, and then wait right there and then for the child process to terminate. The child process then forks again, giving you a child and a grandchild. The child exits immediately (and hence the parent waiting for it notices its death and continues to work), and the grandchild does whatever the child was originally supposed to. Since its parent died, it is inherited by init, which will do whatever waiting is needed. This method is inefficient because it requires an extra fork, but is pretty much completely portable.

Question 3: How do I get lines from a pipe as they are written instead of only in larger blocks?

The `stdio` library does buffering differently depending on whether it thinks it's running on a tty. If it thinks it's on a tty, it does buffering on a per-line basis; if not, it uses a larger buffer than one line.

If you have the source code to the client whose buffering you want to disable, you can use setbuf() or setvbuf() to change the buffering.

If not, the best you can do is try to convince the program that it's running on a tty by running it under a pty.

Meet the MUUG Board Nominees for ’94/95

Since the preliminary list of nominees was published last month, we’ve had one additional nomination — Dan Little. This leaves us with a slate of seven nominees, which is one less than the number to be elected to the board. The current nominees will be elected by acclamation. An eighth member (if any) will be assigned to a position at a later date. This will be made official with a vote on a motion to accept the new board, most likely by a show of hands at the October MUUG meeting. Here are the seven nominees (in no particular order).

Bary Finch

Bary Finch has been with IBM for five years as a Systems Engineering Representative. He has specialized in AIX and the RISC System/6000 since its announcement. Previously he has worked as a Systems Analyst in the Engineering Department of Versatile Farm Equipment, and as a Programmer Analyst at both Westfair Foods and Canadian Indemnity. Bary holds a B.Sc. in Computer Science from the University of Manitoba. Last year, Bary was the President of MUUG serving his second term on the board. Previously, he worked with Susan Zuk to develop the Corporate Sponsorship program, and helped start the first Special Interest Group (SIG) for MUUG. Bary looks forward to another year of serving on the board, and helping MUUG grow even more to meet the needs of the membership.

Rory Macleod

Rory Macleod is an Integrated Systems Consultant for the Midwest District of Xerox Canada Ltd. Rory has been with MUUG for over a year. In that time, Rory has gratiously helped with the production of the newsletter. Xerox has agreed to photocopy the newsletter as their form of corporate sponsorship of MUUG.

Andrew Trauzzi

Andrew Trauzzi has been with the City of Winnipeg for 6 years as a programmer/analyst. He has been working with UNIX since 1986, and currently works with UNIX machines both at home and at work. His current project could be described with a large number of buzzwords, but can be summed up as a tax certificate request system using C++ and Sybase on PC and UNIX platforms. Previously, Andrew obtained his B.C.Sc. at the U of M, and has been an active member of the MUUG for three years.

Doug Mclean

Doug Mclean has 16 years experience in the information systems management industry. He is a graduate of the University of Manitoba, with a BSc in Computer Science. Doug spent 3 years with Wawanesa Insurance as both an application and systems programmer before joining Manitoba Data Services (MDS). AT MDS, Doug was a systems programmer specializing in wide area networking. Later in his career at Information Systems Management Corp (ISM) he provided sales and marketing support for systems outsourcing and distributed computing environments. Doug is currently with SHLSystemhouse where he is a Project Manager for Network and Systems Management solutions. He is also involved in the marketing and implementation of UNIX based solutions. For over 2 years Doug has been working with Unix based solutions for the IBM, HP and SUN platforms.

Roland Schneider

Roland Schneider recently completed his PhD in Electrical Engineering. He started working with UNIX in 1984 when the EE department got its first workstation and administered the department’s network of Sun workstations. He and his brother started a software consulting company specializing in engineering applications.

Doug Shewfelt

Doug Shewfelt has a B.C.Sc, M.Sc. and MBA, all from the University of Manitoba. He is a senior analyst/programmer for the City of Winnipeg Computer Services Department, supporting the Fire and Ambulance Departments. Last year he served on the speaker selection committee for the MUUG/CIPS fall seminar.

Dan Little

An autobiography for Dan was unavailable at press time. ✒

The Fortune File

What if Operating Systems were Airlines?

Unix Airline: Everyone brings one piece of the plane with them when they come to the airport. They all go out on the runway and put the plane together piece by piece, arguing constantly about what kind of plane they’re building. When another plane lands nearby the passengers begin breaking up into meetings and trading pieces of the plane.

Fly NT: Everyone marches out on the runway, says the password in unison, and forms the outline of an airplane. Then they all sit down and make a whooshing sound like they’re flying. The flight attendant then announces that soon all flying will be just like this.

DOS Airline: Everybody pushes the airplane until it glides, then they jump on and let the plane coast until it hits the ground again, then push again, jump on again, and so on. For relaxation Bill’s autobiographies are tucked in the back of every seat.

MAC Airways: All the flight attendants, captains, baggage handlers, and ticket agents look the same, act the same. Every time you ask questions about details, you are told you don’t need to know, don’t want to know, and everything will be done for you without you having to know, so just shut up. Cute little signs are everywhere and outside every window there are flying toasters.

Windows Airline: The airport terminal is nice and colorful with friendly flight attendants, easy access to the plane, an uneventful takeoff...then the plane blows up without any warning whatsoever. In mid-air you are informed that the rest of your flight will be continued on DOS airline. (See DOS Airline)

Newton Airline: After buying your ticket 18 months in advance, you finally get to board the plane. Upon boarding the plane you are asked your name. After 46 times, the flight attendant recognizes your name and then you are allowed to take your seat. As you are getting ready to take your seat, the flight attendant announces that you have to repeat the boarding process because they are out of room and need to re-count to make sure they can take more passengers.

NeXT Airline: Everyone gets a paint-brush and paints the airport terminal. Right before takeoff they are told that the plane will have only one engine and that the air-phones can only be used to call other NeXT Airline planes. On take-off you are told that its peanuts all the way — Every time you ask for food/beverages you are told that other food/beverage items are not necessary and that these peanuts have no fat and come in a wonderful black bag, with Steve’s picture and personal greetings!

VMS Airline: Everyone has difficulties getting into the plane and once they are in it starts to make a *lot* of noise and takes off very slow. The plane only blows up if you think you are safe. Occasionally the captain mumbles, “off to the left of the aircraft, you can see the Unix desert, and don’t worry — we won’t be going over there.”

Get the Visual Edge

By Andrew Trauzzi

With the explosion of graphical-based applications (caused by you-know-who), GUI builders and code generators seem to be more popular than now then ever before. As usual, we find a disproportionate number of vendors slugging it out in the PC arena, while the UNIX world seems relatively quiet. Sometimes keeping quiet is good — it shows that you are actually thinking instead of figuring out what your competition is doing.

My recent experiences with PC-based GUI builders and 4GL's has left a bad taste in my mouth, so I gave them up for a while. Eventually, it became necessary for me to research UNIX-based GUI builders. UNIX users are a picky lot — we are very demanding and quickly abandon products that are not up to snuff. Maybe that's why I was so impressed with the quality and features I found in UNIX GUI builders. Of all the code-generators and screen painters out there (and there are many!), one product in particular stood out — UIM/X.

I had a personal mental 'wish-list' of GUI builder features. I thought most should be standard features, some were a little unrealistic, and a few were downright crazy! UIM/X met them all, and even had a few that I never even considered. The rest of this article consists of a product overview that I requested from Visual Edge — the Montréal company that developed UIM/X. If you have any questions or comments, you can either contact me at `<editor@muug.mb.ca>` or:

Visual Edge Software Ltd.
3870 Cote Vertu
St-Laurent, Quebec
H4R 1V4
(514) 332-6430
(514) 332-5914 (Fax).

Standards

UIM/X 2.5 fully supports standards so GUI software development investment is protected. It generates K&R C, ANSI C or C++, and pure Motif code (as an option) with no dependence on convenience libraries. In addition, UIM/X 2.5 generates code which conforms to the Object Management Group's CORBA C API specification; it also reads and writes UIL.

UIM/X 2.5 supports 100% of the OSF/Motif Toolkit release 1.2—all shells, widgets, gadgets, convenience dialogs and resources. UIM/X 2.5 handles even difficult resources such as XmNcolormap (for graphics applications) and XmNinsertPosition (for building dynamically growing menus). UIM/X 2.5 enables developers to build production quality, style guide-compliant interfaces without the need to modify generated C code and set resources or create widgets that the GUI builder doesn't handle.

Layout

UIM/X 2.5 comes with a preconstructed, style guide-compliant application framework to minimize the time and effort necessary to ensure standards compliance. The framework also serves as an excellent introduction to a wide variety of OSF/Motif techniques. In addition, UIM/X customers can build and enforce corporate or project-wide GUI style guidelines.

UIM/X 2.5 was designed to move the programmer quickly up the learning curve. It provides everything one has come to expect from an easy-to-use WYSIWYG drawing editor... and more. A developer can create widgets (or widget hierarchies) from user-customizable palettes, pulldown menus, or short cut pop-up menus. Those widgets can then be selected and moved or resized as a group. And, with UIM/X 2.5 it is possible to use various operations — like cut, copy, paste, and align — to speed the development task. Moreover, the tool also comes with a wide variety of specialized editors — all smoothly integrated through the consistent use of a drag-and-drop metaphor.

The UIM/X 2.5 Widget Property Editor lets developers edit the properties of multiple widgets and components at the same time, so it is easy to work quickly. UIM/X 2.5 comes complete with color, font, pixmap, and callback selectors/editors—or developers can add their own, if desired. This editor also gives lists of options to choose from, so it is easy to set resource values.

The UIM/X 2.5 Widget Tree Browser lets programmers see and edit the entire widget hierarchy of an interface in either tree or outline form. They get real time feedback as they select, drag and drop widgets to rearrange or reorder the widget hierarchy. This greatly simplifies the task of understanding and editing complex interfaces.

The UIM/X 2.5 Menu Editor lets developers quickly and easily create pop-up, pull-down, and option menus (all of which have a very complex structure in OSF/Motif). They can set labels, accelerators, mnemonics, callbacks, etc.; it has everything developers need to build and edit menus in a fraction of the time it would take to construct them widget by widget.

The UIM/X 2.5 Main Window Editor lets the developer view and set—with a graphical editor—the variety of options that OSF/Motif main windows provide.

UIM/X 2.5 lets developers build and edit palettes which hold customized templates of widgets, widget hierarchies, and/or components. From palettes, they can either create duplicates of their templates one at a time, or create arrays of duplicates. UIM/X palettes save time not only on the current project, but also on subsequent projects.

Behavior

UIM/X 2.5 lets developers type their C code callback directly into a specialized Callback editor. UIM/X 2.5 keeps the callback code as part of the definition of the project, so there's no need to modify generated C code, or merge in their changes each time the GUI is modified. They have access to all callback arguments, and can set XmNclientData. They can use any X, Xm, or Xt function calls—they are all linked in to UIM/X 2.5. UIM/X 2.5 provides developers with the full flexibility of hand coding, without the problems of keeping callback code consistent with their GUI.

UIM/X 2.5 also has editors to help developers specify Application Window behavior (the area where they would draw in a drawing program, for example). They graphically choose events, and tie them to the C code actions they have created. These "Translation Tables" can be used to customize the behavior of existing widgets. UIM/X 2.5 handles the full toolkit capability of translation tables. It supports the entire job of building a GUI, so there's no need to resort to modifying generated C code.

UIM/X 2.5 lets programmers enter a C expression for the value of any property, so they can determine initial values at runtime—without hand-modifying files of generated code. This feature is essential for building X/OPEN-compliant international applications.

UIM/X 2.5 has a library of convenience functions developers can use to handle the programming of behavior. The UIM/X 2.5

convenience library significantly reduces complexity and code size compared to programming in straight OSF/Motif code (which they are free to do in UIM/X 2.5, if they wish). The UIM/X 2.5 convenience library is available on a wide variety of platforms (source is also available), so programmers can port their code to any platform, independent of GUI tool vendor.

Testing

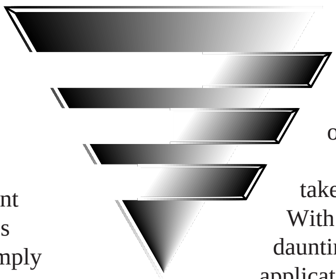
UIM/X 2.5 has an integrated C interpreter, so developers can test the interface with their underlying application connected and running. The interpreter can mix compiled and interpreted code, so they can call their compiled functions, or incrementally compile their interfaces for performance.

The UIM/X 2.5 C interpreter is tightly integrated so there is no need to generate code, recompile, relink, and restart execution every time a developer makes a change to the interface. In addition, the UIM/X 2.5 C interpreter is specifically designed to leverage the development of the GUI. For example, when there is a C code error, it tells the developer specifically which widget and resource to look at. With UIM/X 2.5 there's no need to "work backwards" from line numbers to find errors.

UIM/X 2.5 enables developers to put custom C code virtually anywhere in their interface. With UIM/X 2.5, they don't have to manually keep generated C code files consistent with their interfaces. Hence, their turnaround cycles are shorter and their life-cycle costs are lower.

Reusability

UIM/X 2.5 lets programmers construct their interfaces out of reusable GUI object classes called "Components." Components feature encapsulation, inheritance and polymorphism — which means that UIM/X 2.5 provides all the benefits of Object Oriented Programming (OOP) to GUI developers.



Components comply with the Object Management Group's CORBA C API specification, so a company's investment in OOP is 100% protected. Developers simply use drag-and-drop to create Components from widgets, or widget hierarchies. This natural visual metaphor works the way people do — they build something, then they generalize and reuse it — so it is easy for developers to work quickly.

Once a Component has been built, it can be placed in the palette for reuse. Gaining the benefits of reusability has never been so interactive, so easy, or so intuitive. And, if a developer changes the original Component class definition, all instances of it are immediately updated. This change propagation provides developers with tremendous productivity improvements.

Components provide full encapsulation, so ongoing software development and maintenance costs are minimized. Components can be given both public and private instance properties. These instance properties (which may or may not be Motif-related) are specified by the developer.

When instances of a Component are placed in the property editor, only their public properties appear. To select values for these public properties, developers can use all the features they have come to expect in UIM/X — option menus, built-in property editors like color, font, and pixmap editors — or they can add their own editors.

Developers treat Component classes identically to widgets; there is no additional learning curve involved.

UIM/X 2.5 was designed so that the details of a Component's implementation are hidden from developers. Components can be given methods using the UIM/X 2.5 Method Editor, so encapsulation is complete. Because of this encapsulation, "Component developers" can configure UIM/X for other developers so as to simplify the task of building interfaces and/or following corporate standards.

When code for the project is generated, the implementation of Components is kept separately from the GUIs where they are used. The generated code for the interfaces where they are used contains only calls to the Component's public interface. Consequently, even if a Component's implementation changes, the GUIs where that Component is used are not affected (resulting in reduced software maintenance costs).

The encapsulation, inheritance and polymorphism provided by UIM/X 2.5 Components mean that companies gain the benefits of OOP for their GUI development from Day One.

Building components also makes it easy to develop — and maintain — corporate standards. Developers simply relink their application to take full advantage of any revisions to the standard.

UIM/X 2.5 components can also be used to cost-effectively build applications which are readily ported across platforms. Developers construct their interfaces entirely out of custom-built portable GUI component classes which they create using drag-and-drop. The resulting interface code is portable and separate from their GUI component implementation. With UIM/X 2.5, developers are finally able to pick the level of abstraction for GUI classes that is just right for their interfaces, instead of having to rely on a "lowest common denominator" toolkit approach.

UIM/X 2.5 also automatically generates the Motif code for developers' portable GUI components. This means that developers gain vastly superior portable interfaces for a fraction of the cost it would take to manually create their own portability layer.

UIM/X 2.5 can even be used interactively to take existing GUIs and "layer" them for portability. With UIM/X 2.5, developers no longer face the daunting task of a total GUI rewrite to move their applications across platforms.

Code Generation

UIM/X 2.5 generates either K&R C, ANSI C, or C++ code, and either pure OSF/Motif code or UIM/X 2.5 convenience library code. This assures that programmers' software development investment is protected, and their work can be ported.

UIM/X 2.5 takes advantage of the object oriented features of C++. All Components developers have created are generated as true C++ classes. Public and private properties of Components become public and private data members and methods become member functions. Further, the generated code for any interface (or class) that contains a Component creates and manipulates the Component using its C++ Application Programming Interface (API). The associated benefit is that developers gain productivity leverage in two ways: first, through the generation of reusable code; and second, through its use.

Because UIM/X 2.5 is compliant with the CORBA C API, all of the object-oriented C code that is written for Components still works when the Components are generated as C++ classes. As a result, developers have a smooth migration path from C to C++; they gain most of the benefits of object oriented programming today using C and, when they are ready to make the switch to C++, they simply change a code generation option. All of their object-oriented C code becomes well-structured C++ code, and their initial investment in object oriented development remains 100% protected.

UIM/X 2.5 has a Declarations Editor to let developers customize the template for code generation on an interface-by-interface basis. They can declare global variables, #defines, and include files. They can modify the declaration of generated functions, even add arguments to them. Moreover, these arguments can be used in the resource slots of widgets to create parametric interfaces. The flexibility UIM/X 2.5 provides in code generation means that developers won't have to abandon the tool after their first round of prototyping and code manually from that point on; UIM/X 2.5 will take them from prototype stage all the way to production software.

Large Project Support

UIM/X 2.5 is designed to work with large development projects in a corporate environment. The tool can ensure compliance with project- or corporate-wide standards, and it can be used to create and integrate reusable tools. In other words, UIM/X 2.5 leverages the entire GUI development process.

UIM/X 2.5 enables corporations or project groups to create and enforce style guidelines. They can build style guide prototypes. They can control on a widget-by-widget basis what resources may be changed, and designate the allowable values for those resources. In addition, they can control on a widget-by-widget basis what operations (e.g. moving, resizing, deleting, reordering, etc.) developers can perform on a widget. This ensures that the project's or corporation's style guide prototypes remain intact. No other tool

provides this level of completeness in creating and ensuring strict adherence to project or corporate GUI style guidelines.

UIM/X 2.5 can be integrated with project- or corporate-wide CASE tools and development environments. It supports integration with both data-driven CASE tools such as repositories (or, more simply, SCCS and RCS), and message-driven frameworks such as Hewlett Packard's Softbench. UIM/X 2.5 makes it easy for programmers to follow a software development methodology, and for companies to ensure that their developers follow established standards and guidelines.

The seamless integration of UIM/X 2.5 and CASE tools such as Softbench means that from within UIM/X 2.5, developers can: 1) check files in and out of configuration control; 2) use vi, emacs, or their favorite editor to edit files of text; and 3) use their favorite build tool to create the final executable. By providing a single, integrated development environment for the entire application, UIM/X 2.5 reduces project turnaround time even further.

UIM/X 2.5 assists developers in documenting their interfaces by automatically generating documentation of the GUI structure and hierarchy. This helps organizations reduce downstream maintenance costs for their GUIs ... an issue that will become of increasing importance over time.

UIM/X 2.5 puts the developer in control of the entire GUI development process. It can load, save, or write code for an entire project. And, developers can incrementally compile interfaces and integrate them into their development environment. UIM/X 2.5 generates a main program and a makefile for a project. Programmers can edit these within UIM/X 2.5 to add their own initialization code, makefile rules and libraries. There is no need to modify the generated code or re-integrate changes to a customized makefile each time a new interface is added to the project.

Extensibility and Programmability

UIM/X 2.5 is extensible. Developers can fully integrate their own widgets into UIM/X 2.5. As a result, they can use UIM/X 2.5 to leverage past investments they have made in building custom widgets.

UIM/X 2.5 is programmable. Developers can add operations and editors to UIM/X 2.5 using internal UIM/X 2.5 functions. In fact, by using UIM/X 2.5, they can create their own custom GUI builder.

Market- and Application-Specific GUI Builders

UIM/X 2.5 consists of a core GUI Builder Engine, surrounded by a wide variety of configurable editing tools and capabilities. Given this architecture, UIM/X can be configured rapidly to create market- or application-specific GUI builders. In fact, with UIM/X 2.5, a custom production-quality GUI builder can be created in a mere fraction of the time it would take to build one from scratch.

Currently, a variety of software developers, resellers and systems integrators are using the UIM/X Builder Engine to create either market- or application-specific builders for resale/redistribution. By utilizing UIM/X 2.5, they can focus on their added value ... rather than diverting resources to GUI builder technology.

UIM/X 2.5 can also be simplified for use by non-OSF/Motif programmers. In fact, UIM/X can be made to look like a simple drawing program, if desired. Developers simply remove menu entries and editor functionality. They can hide complex resources, change resource names and provide new resource editors. They can also combine individual widgets into "Compound Widgets" and treat them as a single entity. Compound widgets can be selected, moved, resized, etc. as a single unit, and they can have their own custom editors (e.g. radio buttons with a radio button editor). Developers can also reduce testing capability and hide the C interpreter, or make UIM/X 2.5 into a layout-only tool.

With UIM/X 2.5, developers can cost-effectively create custom GUI builders tailored to users who are not OSF/Motif literate (e.g. - user interface specialists, retrained COBOL programmers, DBMS application experts, CAD programmers and the like).

UIM/X 2.5 can also be tailored to support a 4GL. Programmers can replace the UIM/X 2.5 code generation module with their own 4GL or "easy-to-use language" code generator. Moreover, they can incorporate an additional preprocessor to work with languages embedded in C. Or, they can add new resources and editors to the widgets (or compound widgets) that their code generators might require. In this way, customers who aren't OSF/Motif literate can work with languages familiar to them.

UIM/X 2.5 provides full access to its X/OPEN-compliant message catalog. Developers can change labels, messages, resource names, etc., and they can add their own messages. With UIM/X, they can make a custom builder look like their own, and then expand their market opportunities by localizing the builder for international use.

End-User Customization

UIM/X 2.5 can be used to provide end-user customization for applications — within the bounds of corporate standards or application requirements. Particular UIM/X 2.5 editing capabilities can be integrated into the application in one of two ways: 1) UIM/X can be a separate (tailored) application that generates customization files; or 2) UIM/X can execute as a run-time module, and load the application from a database. In either case, programmers can significantly reduce their development costs by leveraging UIM/X 2.5 GUI technology.

Further addition from Visual Edge for UIM/X:

Cross-Platform Toolset (CPT) — The toolset embodies a cross-platform methodology, and includes a collection of pre-built GUI classes. Together, they allow developers to build portable GUIs on Motif, and deploy them on a variety of target platforms. This initial release offers a single development, multiple deployment strategy — OSF/Motif and Microsoft Windows 3.1 are the target platforms supported (with Windows NT support planned for Q2).

SIG Sideline

By Brad West, SIG Coordinator

The new season has started and we had a good turnout for our Sept. 20 meeting. The meeting started with the round table format. The topics dealt mostly around Linux with discussions from setting up X-windows, how stable it is, network capabilities, what CD-ROM support is available, where is a good FTP site for Linux software (`sunsite.unc.edu` recommended) and what's new in Linux. Greg Moeller graciously brought everyone up to date on what's new in Linux. Some of the highlights were: Slackware is now at Version 2.0 with the stable Linux kernel at 1.0.8, and it now has token-ring support. Also, Linux is now being adopted by GNU. On a more Important note :) DOOM is now available on Linux. If anyone is interested in being a guest speaker at a SIG meeting or you have a specific topic of interest, let me know. I can be reached by email <`bwest@muug.mb.ca`> or my work phone is 983-0336. The presentation for next month's meeting will be announced. The next meeting is scheduled for Tuesday, October 11, at 7:30 PM. This meeting will again be held at ISM, 400 Ellice Avenue, behind Portage Place. Our host is Wolfgang von Thuelen. He will be waiting in the lobby as of 7:15 PM to let everyone in. Hope to see you all at the meeting.

Accent Server News

Free Big Dummy Guide to the Internet Available

The Electronic Frontier Foundation publishes the Big Dummy's Guide to the Internet. Although the author, Adam Gaffin, admits that it doesn't include everything about the Internet, it does include much useful information about getting started. We think it's a pretty good introduction, especially because you can get monthly updates to stay current with the latest developments.

To obtain a copy of the entire Big Dummy's Guide to the Internet, use anonymous ftp to connect to `ftp.eff.org` and look in the `/pub/Net_info/Big_Dummy` directory, or use gopher to connect to `gopher.eff.org` and then select Net Info and then Big Dummy. You can also get one by sending "`62.27`" in the subject of your message to `flashadm @sun.com`. (Try "`65.01`" for updated information.)
The Big Dummy Update is available by several routes. It is posted monthly on USENET in the `comp.org.eff.talk`, `alt.internet.services` and `news.newusers.questions` newsgroups. To receive the newsletter by e-mail, send a message to `big-dummy-update-request@eff.org`. As the message, write: `add big-dummy-update` (don't include your name).

For general information on the Electronic Frontier Foundation, send an e-mail message to `info@eff.org`.

Agenda

for

Tuesday, October 13, 1994, 7:30 PM
Samuel N. Cohen Auditorium
St-Boniface Hospital Research Centre
Main Floor, 351 Taché

1.	President's Welcome	7:30
3.	Business Meeting a) Old Business b) New Business	7:35
5.	Presented Topic As this month is our Annual General Meeting (AGM), we will be holding elections for the 1994-1995 board. For a complete list of board nominees, please refer to page 6. Following the elections, a 45-minute video on Asynchronous Transfer Mode (ATM) will be shown. Sorry, no popcorn is allowed in the lecture theatre!	7:45
4.	Coffee Break and Informal Discussion	9:00

Note: Please try to arrive at the meeting between 7:15 and 7:30, to avoid disrupting the meeting in progress.

Coming Up

Meeting:
Next month's meeting is scheduled for Tuesday, November 8, at 7:30 PM. Meeting location will be the St-Boniface Research Centre, as usual. The March meeting topic is security. Stay tuned for details.

Got any ideas for meeting topics? Any particular speaker, company, or product you'd like to see at one of our meetings? Just let our new meeting coordinator, Roland Schneider, know. You can e-mail him at <`rsch@muug.mb.ca`>.

Newsletter:
If you are interested in a particular topic, let me know. I'm sure I could coerce you into writing an article! I could use a few articles — especially shorter ones — half a page to one page (400 to 1000 words) would be fine.

Monsieur Ex has also let me know that his mail-box has room for more of your wonderful queries again – please submit your questions to the old guy via e-mail to <`m-ex@muug.mb.ca`>. He may be old, but he's not ready for retirement yet!