FIRST CLASS

Information Technology Solutions, Inc.

Object-Based Computing is published often by

Object - Based Computing

THE OBJECTWARE PROFESSIONALS NETWORK

Eric Wespestad, Chicago
Information Technology Solutions

OPN was founded shortly before NeXTWORLD Expo 1993 under the name of Open Protocols for NeXTSTEP - but later changed its name, as it's focus expanded, to reflect the many facets of the ObjectWare equation. Since that time individuals from about 80 organizations have participated in discussions like the following:

- brokering of objects (finding them),
- documenting objects (figuring out how to use commercial objects),
- licensing objects (how to distribute and pay for third-party objects),
- normalization of protocols by which objects interroperate (assembling applications from the parts of other applications),
- quality (trusting objects you didn't create, standards for testing), and
- storage of objects (persistency and archiving)

OPN maintains an archive of its mailing list that is available via ftp for those interested in the topics mentioned above.

"OPN's mission is to evangelize component software which can be woven by users into customized solutions. OPN will provide the technical and political infrastructure for a vibrant component software market."—Marcos Javier Polanco

"NeXT has for years offered the best technical foundation for this new age, but it did not come about. Why not? Selling kits to developers will not cause a paradigm shift in the industry. Selling components to users will, and that means asking them "what will it take" for them to prefer components to Lotus SmartSuite."—Marcos Javier Polanco

Robust interobject communication is a necessary precondition for component software. Among other methods to accomplish this end, NEXTSTEP is graced with Portable Distributed Objects, a nearly transparent method of interapplication messaging, even across heterogeneous networks.

Component software also needs standard object programming interfaces to be exported by the various components. OPN is serving as a forum where the NEXTSTEP community can discuss and agree on what these standards must be. Once users obtain these various objects they will want to weave them together into customized applications, using either the Objective-C++ compiler or systemwide scripting languages. ⊗

See the related article "A Scenario for the ObjectWare Marketplace" on page 5 .

■

INFORMATION TECHNOLOGY SOLUTIONS, INC
500 WEST MADISON, SUITE 2210, CHICAGO, IL 60661

these in the future. In any case, be sure to BACKUP THE ORIGINAL MAKEFILES FIRST!

You might also consider modifying the basic ProjectBuilder templates (located inside of its ".app" folder) in order to have your libraries (say, "libSuperDuperKit.a") as a default library (much like "libSys_s.a" and "libNeXT_s.a") for every new project that you start. Once again, the warning is to BACKUP THE ORIGINAL FILES FIRST!

Some other issues which I haven't addressed here (but may in a future article) include such things as: Kit/Library documentation; an automatic kit/library makefile generator; and, a kit/library hierarchy viewer

In order to best facilitate the code re-use that we advocate and that should be generally encouraged, comprehensive documentation is a requirement. Programmers must be able to look at, not only, the header files, but also, fully written class description files (such as those provided by NeXT for its classes). While access to the source code is okay, this should generally be discouraged, since one of the basic tenets of good object-oriented design is that each object is a "black box" and the programmer should not design around how an object does its work, only what the results are. But...reality is reality, and typically, programmers "must" look at the source code. To work-around this, we could use an "automatic document generator" that would create fully formatted ".rtf" class description files (much like NeXT's).

Secondly, while our basic Makefile is easily created/modified for each new kit, we would be more inclined to create more such kits, if the process of putting the kit together (i.e. generating the Makefile) were as simple as ProjectBuilder makes it for "normal" application projects. Therefore, some kind of "automated makefile generator" could be used here.

Finally, it's always nice to be able to "see" your kit. By this I mean programmers often want to view the relationship between objects (in the case of an OO environment, the class hierarchy). Some tool for being able to quickly and easily look at your kit would be helpful. Diagram! from Lighthouse Design is delightful, but manual. Who wants to draw out class hierarchies by hand?

FUTURE DIRECTIONS

There are future directions that code re-use might go in on the NEXTSTEP operating system. We might talk about these in future articles. Specifically, other code re-use strategies might include:

- Using the ".subproj" as the basic unit of re-usability. This accommodates the inclusion of ".nib" files as well as other resources such as sounds and images (which our current system, sadly, does not accommodate.)
- Using the ".bundle" as the basic unit of re-usability. This accomplishes the same as above, but also opens the possibility of creating a "/LocalLibrary/Bundles" folder which would contain bundles of re-usable code that can be dynamically loaded. This issue has been (and probably will continue to be) debated regarding performance particularly.
- Shared libraries. This is what NeXT does with NEXTSTEP. This is useful (like the bundles approach described above) for having common, shared code, which is dynamically loaded. Additionally (this benefit is also present in the bundle idea) this common code could be changed, and all other applications that use it would have the new and (hopefully) improved code automatically! (Much like NeXT's famous claim about faxing in the Print Panel.)
- Object repositories and class version management. This addresses the need to maintain a history of revisions of code, and to restore previous versions as necessary. This is an item that must be addressed regardless of the choice of strategies suggested above. Since irrespective of how you share code, ultimately there will be the need to track and manage revisions of code.

CONCLUSION

- In conclusion, you can see how easy it is to get started with a simple code re-use strategy right away. This will help to address many of your needs to leverage off of previous work. As we continue with this (hopefully) series of articles, and experiment (as is always required) with new tactics, we will cover new and better ways to re-use your older code and continue to improve your programming leverage! ⊗

You can put your generic ".c" files (such as functions) or ".m" files (such as categories) on the "CFILES =" or "MFILES = " lines, as follows:

```
CFILES = ReallyWickedFunctions.c
    EvenMoreFunctions.c
MFILES = ZippyCategory.m
```

As with the class files above, the header files don't need to be explicitly listed.

Next, for any headers such as protocol declarations, or files that simply define macros or, enums, or new types, you must list the header on the "HEADERS = " line:

```
HEADERS = $(CLASSES:.m=.h)
    $(MFILES:.m=.h) $(CFILES:.c=.h)
    SuperNeatProtocol.h
    ReallySpecialMacros.h
    UnbelievableTypedefs.h
```

Finally, you have to specify the name of the directory in the "Headers" folder, that all of the headers (including a pre-compiled header) for this kit will be put in. In this case:

```
HEADER_DIR = SuperDuperKit
```

When you build this library, a complete library archive file (called "libSuperDuperKit.a" will be built in:

```
/LocalDeveloper/Libraries/Source/
    SuperDuperKit"
```

and copied into:

```
/LocalDeveloper/Libraries
```

Headers (including a pre-compiled header) will be copied into:

```
/LocalDeveloper/Headers/
    SuperDuperKit
```

Now all you have to do is open a Terminal window, and "cd" into your kit's source folder:

```
localhost> cd /LocalDeveloper/
    Libraries/Source/SuperDuperKit
```

and build the library by typing "make install", as follows:

```
localhost> make install
```

This will build and copy everything for you.

Now...you want use your new library in your latest project ("MyProject"). To do this, go to your ProjectBuilder "PB.project" file, and select the "Libraries" category at the bottom of the first column in the ProjectBuilder browser. From the Files menu, select the "Add..." command to add your new library. Find your library ("/LocalDeveloper/Libraries/lib SuperDuperKit.a") in the "Add" panel browser, and click "OK".

Now, one final addition and you're ready to fly. If already have a "Makefile.preamble" file in your project, then add the following line:

```
OTHER_CFLAGS = -I/LocalDeveloper/
    Headers -L/LocalDeveloper/
    Libraries
```

If you don't have a Makefile.preamble already, create one with Edit, and add this line. This enables your project Makefile to find the new library or libraries that you have specified, as well as any headers, of the form:

```
#import <SuperDuperKit/
    MyCoolView.h>
```

Now your ready to go, use anything that's in that library.

Some additional points about this approach. You might notice that I haven't put the libraries into:

```
/usr/local/lib
```

and the headers into:

```
/usr/local/include
```

The reason for this is that we prefer to keep everything centralized so that we can (easily) pick everything up and move it. The "Makefile.preamble", is simple enough, and helps us add to the standard search path for these resources. (I believe that "/LocalDeveloper" is supported as a standard search path under NEXTSTEP Release 3.1 now, but I have confirmed that.) You can use "/usr/local/lib" and "/usr/local/include" if you so desire, then you will not require the "Makefile.preamble" line as specified above.

Secondly, the Makefile outlined above works fine for NEXTSTEP Release 3.0 and 3.1. But, inevitably, you'll want to build "fat" or "Multi-Architecture Binary" libraries for NEXTSTEP Release 3.1 and on. This requires only two minor changes:

```
CFLAGS = -Wall -g
```

should be changed to:

```
CFLAGS = -Wall -g -arch m68k -arch
    i386
```

in order to specify the various (in this case two) architectures you want to build for.

And this:

```
$(LIB):$(OFILE_DIR) $(OFILES)
ar rc $(LIB) $(OFILES)
```

should be changed to:

```
$(LIB):$(OFILE_DIR) $(OFILES)
libtool -o $(LIB) -s - $(OFILES)
```

Using NeXT's new "libtool" program to properly build your libraries. There are certain bugs with "ar" as used previously, when applied to "fat" libraries. What I might suggest, for the time being, if you are transitioning from 3.0 to 3.1, is creating a "Makefile_3.0" and making the following changes:

```
LIB = libSuperDuperKit_3.0.a
OFILE_DIR = obj_3.0
```

This will enable you to build a 3.0 compatible library, if you need to, by simply typing:

```
localhost> make -f Makefile_3.0
    install
```

OTHER POINTS

There are several possible additions to this scheme that you (and we) might make to provide a more seamless environment. First, if your centralized developer folder (be it "/LocalDeveloper", or, as in our case, "/ITSDeveloper") is not in the standard search path for libraries and header files, and you must add a "Makefile.preamble" to your projects, you might consider actually modifying the default/standard NeXT makefiles (located in "/NextDeveloper/Makefiles"). Though, the warning here is that this is DANGEROUS territory since NeXT may (and probably will) change

To pull this off, there are three simple things you must do. First, create a folder called "/LocalDeveloper/Headers" and a folder called "/LocalDeveloper/Libraries". Inside the "Libraries" folder, create a folder called "Source". This is where you will organize the actual source code for your "kits".

Once you have done this, collect all of the necessary class, category, function and protocol source files, into the "Source" folder. Although you can put all of your code to be re-used into a single "monster" library, it is probably better to break this code into separate "kits" which represent distinct functionality. This is what NeXT has done, for example, with the DatabaseKit, the 3DKit, and the SoundKit.

Let's call the first "kit" the "SuperDuperKit". Create a folder called "SuperDuperKit" inside your "Source" folder, and move your code into it. Now...here is what we use as a Makefile for building libraries:

```
LIB = lib?Kit.a
INSTALLDIR = /LocalDeveloper
MODE = 0644
CLASSES =
CFILES =
MFILES =
CFLAGS = -Wall -g -arch m68k -arch
    i386
HEADERS = $(CLASSES:.m=.h)
    $(MFILES:.m=.h) $(CFILES:.c=.h)
HEADER_DIR = ?Kit
PRECOMP = $(INSTALLDIR)/Headers/
    $(HEADER_DIR)/$(HEADER_DIR)
OFILES = $(CLASSES:.m=.o)
    $(MFILES:.m=.o) $(CFILES:.c=.o)
OFILE_DIR = obj
OTHER_GARBAGE =
RMFLAGS = -rf
VPATH = $(OFILE_DIR)

.c.o:
$(CC) $(CFLAGS) -c $*.c -o
    $(OFILE_DIR)/$*.o

.m.o:
$(CC) $(CFLAGS) -c $*.m -o
    $(OFILE_DIR)/$*.o

$(LIB):$(OFILE_DIR) $(OFILES)
ar rc $(LIB) $(OFILES)

headers:
@echo >$(PRECOMP).h
@for i in $(HEADERS); do echo "#
    import \"$$i\" >>
    $(PRECOMP).h; done
@$(CC) -precomp $(CFLAGS)
    $(PRECOMP).h -o $(PRECOMP).p
install:$(LIB)
@echo Installing library...
@install -r -m $(MODE) $(LIB)
    $(INSTALLDIR)/Libraries
@echo Installing headers...
@mkdirs $(INSTALLDIR)/Headers/
    $(HEADER_DIR)
@install -c -m $(MODE) $(HEADERS)
    $(INSTALLDIR)/Headers/
    $(HEADER_DIR)
@echo Making precompiled headers...
@make headers
clean::
$(RM) $(RMFLAGS) $(OFILE_DIR)
    $(LIB) $(OTHER_GARBAGE)
$(OFILE_DIR):
@mkdirs $(OFILE_DIR)
```

With this Makefile, all we have to modify is the following lines:

```
LIB = lib?Kit.a
INSTALLDIR = /LocalDeveloper
CLASSES =
CFILES =
MFILES =
HEADERS = $(CLASSES:.m=.h)
    $(MFILES:.m=.h) $(CFILES:.c=.h)
HEADER_DIR = ?Kit
```

In this case, we'll fill in "libSuperDuperKit.a" for the "LIB =" definition, as follows:

```
LIB = libSuperDuperKit.a
```

You may change the "INSTALLDIR =" definition, if your "Headers" and "Libraries" folders are located somewhere besides "/LocalDeveloper".

The "CLASSES =" line is where you put the names of your class ".m" files, so put "MyCoolView.m" and "ReallyNeatObject.m" here.

```
CLASSES = MyCoolView.m
    ReallyNeatObject.m
```

You won't have to worry about the header files for these, the Makefile takes care of this.

continued on page12

spell checker is being distributed with Mesa, Pages, CustomApp, etc. Is controlling distribution important?

What a software component does when there are no more licenses is its business. The Kala Persistent Data Server starts metering use if no more licenses are available. Other component vendors may decide to assume honesty on the part of their customers and continue functioning, even without paid licenses. A variety of vendor policies must be supported.

Separating the customer (developers) from the bill-payer (users) may be problematic; developers still have an interest in keeping component prices low; Pages has great incentives to find some other component vendor if SuperSpell becomes too expensive for users.

Software costs often pale in comparison to the costs of software distribution, installation, and administration. Building the licensing domain management tools could put us well on our way to automatic software distribution & asset management tools.

IN CLOSING:

This business model for ObjectWare vendors provides strong incentives for customers to purchase component software, provides clear pathways for ObjectWare vendors to be compensated for their efforts, and fosters greater diversity in the applications market, as the vast supply of ObjectWare is reconfigured and repackaged into application suites, custom applications, "Works" integrated applications, etc. Most importantly, it accurately prices skills in the application value-added chain, allowing for an efficient market.

This model preserves the greatest virtue of Brad Cox's superdistribution (ease of replication) while teasing it away from the (uncomfortable) concept of meterware. It also covers the three cases posed by Mark Thomsen

1. ObjectWare→ISV→Users
2. ObjectWare→Integrator→Users
3. ObjectWare→Users

as to ObjectWare distribution. It provides a vast user market for ObjectWare, and counts on robust systems tools to manage the added complexity. Customers are happy with this greater complexity since this reduces software costs.

As technology moves forward customers must make compromises. Faced with the choice between the unruly jungle of client/server computing and the warm, fuzzy assurance of IBM's hand-holding customers performed a quick cost/benefit calculation and chose to jump off the cliff. An incremental improvement over IBM's service and price would not have done the trick...the benefits must be tangible and overwhelming. Does this ObjectWare world present a comparable level of rewards for customers?

EDITORS POSTABLE: Marcos is the founder of the ObjectWare Professionals Network. To be added to the OPN mailing list or to contact Marcos, please send electronic mail to shiva@vega.stanford.edu – for more about OPN see page 14. ⊗

NEWS FROM ITS

ITS recently signed an agreement with Hewlett-Packard Company to develop a NEXTSTEP interface to HP OpenMail—which is recognized as the market leading enterprise messaging backbone. The HP OpenMail client will be delivered to several very large NEXTSTEP sites in Q2/94, a commercial offering will follow shortly afterwards.

Strategies for Re-use under NEXTSTEP

Chris Cuilla, ITS

The NEXTSTEP environment is touted with great fanfare among developers because of its object-oriented development environment and design features, which presents stronger opportunities than any previous operating system, for the re-usability of previously written code (in this case "objects" or "classes").

However, when it comes to implementing effective strategies for class, category, protocol & function re-use, many developers are at a loss, and as a result they seldom, if at all, re-use old code. This article will cover some basic steps that you can take, either as an individual developer, or programming team leader, to implement a simple code re-use strategy.

The first question is, "What tools does NeXT provide to facilitate this code re-use?" Unfortunately, NeXT provides very little in the way of tools to assist developers in executing effective code re-use strategies. Once again, UNIX comes to the rescue providing a selection of tools (i.e. make, ar) that developers can use to execute these strategies.

BUILDING A LIBRARY

In this example, we will demonstrate how to execute a simple library based code re-use system. What we do, currently, is to group a set of similarly related classes, categories, protocols, and functions into a library or "kit" (in NEXTSTEP parlance). This kit is compiled and built into a centralized/shared "Libraries" folder, with headers organized into a shared "Headers" folder.

■

Information Technology Solutions

400 West Erie, Suite 301
Chicago, IL 60610
(312) 587-2000
info@its.com

PagerKit • StringSurgeon • Yahv.app (Header Viewer)

Insight Software

(503) 222-2425
info@insight.com

ImageView • ScannerKit

Itasca Systems, Inc.

(612) 851-3155
info@itasca.com

ITASCA NEXTSTEP Client • ITASCA ODBMS

Joe Barello Consulting, Inc.

(212) 580-8366
(212) 580-1857 FAX
info@jbc.com

Tab Palette

Kapiti Limited

(071) 587-0033
(071) 735-3765 FAX

FIST: Complete Dealing Room System

KCW Consulting

(703) 938-4152
curt@kcwc.com

PhoneTones

Lamb Software Design

41-22 735.96.03
lamb@lsd.ch

LSDDistMatrix

Liveware Corporation

(303) 484-7607
info@liveware.com

LockOut Object Set

Look Glass Design, Inc.

(604) 739-3131
(604) 739-3008 FAX

LDGCreditCardAuthorization • LGDModem • LDGSerialPort

Metaresearch, Inc

(503) 238-5728
(503) 232-6323 FAX
info@metaresearch.com

Color Digital Eye Objects • SoundWorks Objects

Mouthing Flowers.

(206) 325-7870

slugg@mouthers.wa.com

Nightshade Software

(403) 492-9343
nightshade@niagara.ucs.ualberta.ca

FilteredFields • GraphMe • NiftyButton

Objective Technologies, Inc.

Suite 1502
7 Dey Street
New York, NY10007
(800) 3-OBJECT
(212) 227-6767
(212) 227-3567 FAX
info@object.com

ChooserPalette • GraphPalette • MathPalette • OTDBKit • OTI Extended Text Object • OTI Tabular Text Example Objects • OTString Kit • SmartFieldPalette

RDR, Inc.

Suite 350
10600 Arrowhead Dr.
Fairfax VA 22030
(703) 591-8713
(703) 273-8170 FAX
info@rdr.com

RDRGadgets • RDRImageView • RDRSelector • RDRSound • RDRSwitchView

The Stepstone Corporation

(203) 426-1875
hotline@stepstone.com

ICpak 101

Stream Technologies, Inc.

+358 0 4357 7340
info@sti.fi

Object Store

Target Development

(800) 444-5435
(717) 898-9190
objects@target.com

Link View Palette• Retriever Palette

Trillium Sound Research, Inc.

(800) L-ORATOR
(403) 284-9278
manzara@cpsc.ucalgary.ca

Text-to-Speech Kit

Uptime Object Factory, Inc.

(+41) 55 12 42 29
(+41) 1 932 4923 FAX
info@uptime.ch

FilterKit • Generic Search Facility for DBKit • Nikita: Advanced NetInfo Kit

Versant Object Technology

(415) 329-7542
geoff@osc.com

Versant ODBMS

VNPSoftware

(617) 661-4292
(617) 864-6768 FAX
info@vnp.com

AccessKit • UIBinder Palette

Workstation AG

+41 91 505094
rgi@wag.chation

DBDragger • FuncEdit • Knob

ZGDV Darmstadt

+49 06151 293863
essmann@igd.fhg.de

Realtime Voice and Video Communication

ZippyTech

(412) 421-9588
ztech@well.sf.ca.us
PO Box 322
Homestead PA 15120

InetObjects Base Collection & Protocol Collection

We would be happy to list your company -- please let us know how to list your company and what objects are available from your company. Filling out the object submission form on the next page will also help us to provide a complete list of resources for the object based NeXT developer.

In future issues an important part of this publication will be a descriptive list of tools and objects that are available to NeXT programmers. We will be trying to review two or three objects or programming tools each month. If you would like to have your object or tool reviewed send a copy, with all associated literature and documentation to:

Ted Shelton, Publisher
Object-Based Computing
c/o I T Solutions
500 W. Madison, Suite 2210
Chicago, IL 60661

We apologize in advance that submissions for review cannot be returned.

An ObjectWare Scenario continued…

Pedro decides to purchase a used computer from Ana Franco, a student who had been running OnDuty under grace period; her licensing domain is "tainted"; Ana's transgression will deprive Pedro of a grace period for OnDuty for his whole licensing domain. Solution: For a while I thought the licensing domain's the 'Scarlet Letter' should follow Ana, that it would not be handed to Pedro. But this gets ugly quickly. We may say Pedro *has* to accept responsibility for Ana's abuse; then the market value of Ana's computer is adversely affected. Resolving this issue may also provide for a way to deal with economical licensing for notebook computers, which are detached from the network periodically.

OPI experiences a one-time spike in the usage of SuperSpell, resulting in twenty grace-period licenses to be issued. After the grace period ends, users find that no more grace-period licenses are available, at least until the twenty from the previous spike are paid for. Solution: Pedro may set a limit on how large the usage spike may be; this limits how badly his licensing domain is tainted.

OTHER THOUGHTS:

This approach to ObjectWare licensing teeters on the commonality of the ObjectWare underlying applications. Will there be much commonality? Is there a way of measuring what it will be? With the supply/demand problem for ObjectWare solved (!), we'd do well to adopt standard documentation & distribution format, such that ObjectWare repositories can easily be browsed; third-party ObjectWare should be as well integrated into the NEXTSTEP environment as NeXT's own: NeXT should help craft standards which will encourage a combinatorial explosion of multi-vendor ObjectWare.

ObjectWare vendors must decide to what extent they want to control the distribution of their product. Under the scenario described here, it will be only after the fact that ObjectWorks will realize their

continued on page10

Object Catalog

ObjectWare for NEXTSTEP

Objects, palettes, and other tools for NeXT Developers:

ABComputers

(401) 521-2829
(401) 521-2829 FAX
rca@cs.brown.edu

ProFuse Rule

Anderson Financial Systems, Inc.

(800) 237-8723
(215) 653-0911
(215) 653-0711 FAX
kits@afs.com

AFSApplication • AFSButton • AFSEventManager • AFSFindPanel • AFSForm • AFSHelpPanel • AFSLookupsPanel • AFSMatrix • AFSMouseCalcPanel • AFSReportPanel • AFSScanPanel • AFSTextField • AFS3DButton/Graph • AFSWindow,AFSPanel • TradeKit

Archetype, Inc.

(617) 890-7544
(617) 890-3661 FAX
info@architype.prospect.com

Document Engine

BenaTong

(614) 276-7859
(614) 276-7859 FAX
benatong@count0.uucp or chuck@kiwi.swhs.ohio-state.edu

Serial Solutions

Black Market Technologies, Inc.

(718) 522-5090
(718) 522-5090 FAX
info@bmt.gun.com

GridPalette • Multicell

Conextions, Inc.

(508) 689-3570
(508) 689-2450 FAX
info@conextions.com

3270Builder • 3270Palette • 3270Toolkit • 5250Palette • 5250Toolkit

Dept. of Radiology Ohio State University Hospital

(614) 447-9194
mitroo@magnus.acs.ohio-state.edu

ImageScrollView/MiniView

Digital Composition Systems, Inc.

(415) 673-5322
gary@dreyfuss.portal.com

SpreadSheetVue

Digital Tool Works

(617) 742-4057
lexcube!equation@bu.edu

Equation

Doberman Systems

(801) 944-4329
doberman!mike@esunix.sim.es.com

Simulation Kit

Elysia, Inc.

(+33) 1 47 49 61 96
(+33) 1 47 14 99 09 FAX
info@elysia.fdn.org

ImageView Palette • ScanKit

Eye Research Institute

(416) 369-6478
(416) 369-526 FAX
info@eric.on.ca

Calera Network OCR Toolkit

Harvard Toolworks, Inc.

(508) 772-4420
(508) 772-4603 FAX
info@magdalen.dmc.com

DBCalendar & DBText Adaptor

Hot Software

(617) 252-0088
(616) 876-8901 FAX
info@hot.com

BarCodeKit • SerialPortKit

Hutchison Ave. Software Corp.

(514) 499-2067
(514) 845-5236 FAX
darcy@solutions.ca

NewsKit • QuoteKit

PROBLEM SITUATIONS:

Arsenio Rodriguez, a mere end-user roaming the Internet, finds and downloads a copy of Pencil Me In (PMI). He runs it, thus triggering a Sarrus-specified grace period for software use. Arsenio then tosses PMI. Unaware of this occurrence, Pedro tries to take PMI for a test drive a month later, and finds the software refuses to run. Solutions:

1. Rather than refusing to run at all, PMI runs in demo mode. Arsenio could have also run it in demo mode from the start, preventing the license server from being informed.

2. Refuse to run full-blown software without authorization from the administrator of the licensing domain.

A massive application utilizes hundreds of components, resulting in a flurry of requests to the license server. Solution: This is unlikely to happen, unless all the components are loaded at the same time. But components could have "license delegates"; SuperSpell could delegate acquiring a license to the application, which would then go license shopping for all its components.

A customer finds he must cut twenty checks resulting from the purchase of just one application! Solution: as suggested by Ben Bernhard, one-stop shopping is where the action is. It is easy to imagine that licensing clearing houses will emerge such that customers can write a check to one of them; the clearing house then distributes payment to all involved parties. (1-800-OPN-SALE? Naw, a private concern can probably do a good job of this.) We may even do without clearing houses altogether if payment mechanism are built into the license servers.

Appsoft Write's text engine captures critical mass; other text engines are at a decided disadvantage, as Write's text engine has (for all practical purposes) become system software. Solution: This is where the Open Protocols Project comes in and tries to hammer out @protocol(TextEngine), such that the market stays as accessible as possible.

A new version of SuperSpell is shipped (not directly to Pedro, who is *not* an ObjectWorks customer, but maybe in the next release of Mesa). Should we automatically upgrade the SuperSpells used with Pages? (This is just like upgrading system software.) Solution: Maybe, maybe not.

Arsenio decides to use the color separator in Pages, thus triggering a license grace period for that component...Pedro gets to see the bill -> unpredictable expenditures! Solution: the grace period provides a buffer for triggering licenses you did not really want. Also, ISV's should inform their customers what the top price for their product will be, assuming all the components are licensed. Alternatively, customers could pay for all the components up-front, and get it over with.

ObjectWorks decides to unleash a SuperSpell virus; the software spreads and Celia Cruz, the administrator at Mission Critical Inc., sees she owes money to ObjectWorks. Solution: Besides being a terrible PR move, the grace period is key here. A related case is the licensing of invisible components, say a DBKit adaptor. (Notice the ISV can bundle every known adaptor with their software; only the ones used by the customer need to be licensed.) Components are not free entities; they are useful only within some context. Maybe Celia can query the network for which applications utilize SuperSpell; she can then match the bills demanded by ObjectWorks with the usage patterns for the applications which invoked it. This is a tough problem.

McCaw Cellular talks to Pages about an enterprise-wide license. However, SuperSpell is licensed per-workstation. Solution: Component manufacturers must be *very* flexible in the way their software is distributed; the components should have some built-in circuitry to account for the various ways in which they are being licensed by customers, which affects the bill the license manager will present to the customer.

continued on page 8

manager finds that there are five unpaid licenses which are being granted under a grace period. This is a clear signal for Pedro to send more cash to Pages.

Pedro is happy to get fully functional software to try out, and that the application was trivially easy to acquire. Furthermore, he can tune purchases to the real, actual software usage patterns of his users.

One of the components shipped with every copy of Pages is SuperSpell, the multilingual spell checker from ObjectWorks. SuperSpell is, for the first time, invoked by one of OPI's users. We enter the grace period for SuperSpell's first license. Pedro follows the same procedure as before, sending a check to ObjectWorks at some point, and gradually upping the number of licenses he purchases.

Pedro is a bit annoyed at having to pay individually for these various components. However, he may actually be happier, since he pays only for components that actually get used; if his users never touch the table editor, he never has to pay for it. Pages is very happy, since it was able to distribute SuperSpell at no cost to itself. Pages charged Pedro for their VALUE ADDED, not for any components which may have been shipped with Pages. ObjectWorks is overjoyed, since it got SuperSpell in more users' hands.

The plot thickens. Pedro acquires Athena Design's Mesa, and deploys to his users, purchasing thirty licenses. Mesa *also* comes with a bundled copy of SuperSpell. But OPI does *not* need to purchase any more SuperSpell licenses. Pedro can leverage the components he has already purchased in making his new acquisition. He only needs to pay Athena Design for their value added by their application. When he purchases a custom application which uses Mesa as a backbone, he can again leverage the licenses he already paid for.

Pedro is overjoyed that the more he buys the more he saves! This situation also points to why ObjectWare should run even without licenses. While it may be relatively easy to determine how many Mesa and Pages users there will be, estimating simultaneous use of component software is virtually impossible, particularly if the component is invisible to users.

Let's look at this scenario from the perspective of the various players:

Application Developers - will ship more capable, integrated, and *larger* applications. It costs the ISV very little to ship copies of SuperSpell, for example, with their software. If SuperSpell is a user-visible component, then ObjectWorks should be the one supporting it; that is unless SuperSpell has been deeply integrated with Mesa/Pages, in which case Athena Design/Pages has "internalized" SuperSpell's functionality, making for tighter integration than the arms-length communication normal for applications from multiple vendors. The ISV also find it easier to price their applications: they charge users for their value added in putting components together. The ability to price this value added accurately leads to more specialized firms and efficient markets. Interestingly, as sales go through the roof, Pages is not as tempted to develop their own replacement to SuperSpell: they pay nothing for it anyway. Time-to-market is enhanced.

Incidentally, this market encourages application vendors to componentize their applications to a greater level. Suppose that in order to communicate with Pages' color separator I have to invoke the whole of Pages. Well, other application vendors probably want to ship just the color separator with their custom applications; Pages will be at a disadvantage if it cannot deliver such a component on its own.

ObjectWare vendors - are in heaven. For one, pricing ObjectWare becomes much, much simpler: they can estimate their market penetration and amortize development costs over the huge user market. (This is similar to the pricing rationale behind Renderman, PostScript, and PhotoCD licenses, although system software can guarantee 100% market share.) There is, of course, the dead calm when the ObjectWare vendor must support application developers before they even start seeing end-user checks in the mail. Smart vendors will eat this support cost rather than charge an up-front fee for ObjectWare: their job is to get their ObjectWare into as many customer installations as possible; once a particular piece of ObjectWare achieves wide coverage it becomes systems software from the application developer's standpoint. ISV's can also get into this game: Appsoft would do well to ship Write's text engine as a basic word processor for very cheap, as a loss-leader. This causes other application vendors to utilize Write's text engine (which customers have already bought) versus PasteUp's (which customers would have to pay for again). Appsoft makes money selling extensions to their text engine.

With a vast user market becoming established, ObjectWare will proliferate. ObjectWare's customers are other developers, although the bills are ultimately paid by users. So in order to convince other developers to ship with their components ObjectWare must attain unparalleled quality and its vendors must provide security to the application vendors. OPN may also get into the game of setting quality standards and certification procedures.

Customers - will see lower costs and better applications. They will *demand* componentized software. When they open the appWrapper in Pages, for example, they see eight or nine components which they will be able to leverage to lower the cost of future software purchases. Users may also utilize scripting languages or Objective-C to craft their own applications with these components. With so many APIs exposed, deeply integrating even multi-vendor applications will become routine; monolithic applications will be at a decided disadvantage in this world.

Customers will also see more applications, as the available ObjectWare is reconfigured and repackaged into a variety of specialized applications.

THE INTEGRATED DESKTOP AS A BASELINE

From a user perspective, NEXTSTEP's integrated desktop has always offered a powerful form of leverage. The seamless, consistent integration of the Workspace Manager, Mail, and 3rd-party applications brought users a new baseline of interapplication cooperation that other platforms are only now catching up with. However, the really unique thing about NEXTSTEP is the synergy between custom and shrink-wrap software that extends the user environment into something new and compelling.

EXTENSIBLE SHRINK-WRAP EXTENDS LEVERAGE

The phenomenon of ObjectWare on NEXTSTEP is no longer limited to individual software components. It now includes shrink-wrap applications on NEXTSTEP that offer a new level of extensibility that has yet to be matched on any other platform.

That's why I think that much of the attention being paid to component software on NEXTSTEP is focusing at too low a level. Most componentized software is pure code, useful mainly to other engineers. Typical components by themselves are just too small to give custom application developers any serious leverage on a large-scale development effort. There is much more to be gained by merging custom applications with extensible, open-ended, shrink-wrap applications.

Why do extensible applications offer more leverage than componentized objects? Because 3rd-party application developers invest the one to two years that it takes to develop and refine a high quality user interface. Why should corporate developers spend 18 months reinventing a spreadsheet, word processor, or scheduling application when they can harness an existing application and simply incorporate it into their own work?

The most compelling promise of NEXTSTEP's object orientation is a wide variety of shrink-wrap applications with open architectures and clean, reusable APIs. The current examples of these open-ended applications are already surpassing the more isolated shrink-wrap software that is currently available on other platforms. And that brings me to my last point

LEGACY VERSUS LEVERAGE

There is a lot of talk these days about how the availability of SoftPC on NEXTSTEP 3.2 will make shrink-wrap applications on NEXTSTEP obsolete, but I don't see things playing out in quite that way. In fact, SoftPC's isolation from the rest of the integrated desktop will most likely reinforce customer commitment to NEXTSTEP's native applications.

Customers who develop and deploy mission-critical custom applications on NEXTSTEP are using this technology to gain competitive advantage today. They are using object technology to develop and deploy better solutions faster to their users. They are harnessing extensible shrink-wrap applications and integrating this functionality into their own mission-critical software.

In short, successful NEXTSTEP customers are focusing their efforts on looking forward, not backward.

So if, after experiencing the benefits of Object Technology, the AppKit, and NEXTSTEP's integrated desktop complete with extensible shrink-wrap applications, a customer still wants to relegate their productivity suite to a dark, isolated corner of their Workspace using SoftPC, they can go right ahead and do that. However, while this company is reliving the past, its competitors are looking to the future by building an advantage with native NEXTSTEP solutions. The bottom line is, if they're not leveraging their desktops, they might as well not be using NEXTSTEP. ⊗

A Scenario for the ObjectWare Marketplace

Marcos Javier Polanco

EDITORS NOTE: This article is from a posting Marcos made to the OPN (Objectware Professionals Network) mailing list in late July 1993 - I have left the name of real products and companies intact—their use is necessary. I had originally thought to write a piece describing OPN and the topics it has addressed. However, upon re-reading the ObjectWare Scenario, I decided to include it verbatim, and then add a few comments myself.

Pedro Knight manages a medium-sized NEXTSTEP installation at Operational Productivity International. He has just decided to get copies of Pages by Pages. He acquires a full-blown copy of the software from anywhere (Paget Press' Electronic AppWrapper, the guy next door, the FTP archives, Pages' BBS, anywhere. As soon as Pedro activates Pages, the software registers with the license server, standard with NEXTSTEP, to see how many Pages licenses OPI has purchased. Finding zero, Pages informs Pedro that it will run without limits for a vendor-specified grace period. Pedro deploys Pages to his users. At some point before the grace period ends Pedro sends payment to Pages for the twenty licenses Pedro expects to use simultaneously, and gets keys to these licenses. Over the next few months, more users try Pages, and the license

a particular Document. I was even able to use this scheme to implement many-to-many relationships using correlation objects.

LIMITATIONS OF THE KIT

One must be aware that the IndexingKit is not as mature as some of the other kits and hence has some limitations. The kit is quite vast and complex. There are some areas that have not been fully implemented and others that have not been adequately tested. If you work with the core functionality, the IndexingKit will work well for you. If you venture out into some of the outer reaches of the kit, you might encounter some problems.

When the kit was first released with NEXTSTEP 3.0, it had some major flaws. The kit was vastly improved with 3.1, and 3.2 promises to be even more robust. In addition, the kit has not received the high level of support from NeXT as, say, the DBKit even though it is just as complex and perhaps more so.

For the way I have used the kit, it has been quite robust. Classes that I've used with success include IXBTree, IXBTreeCursor, IXPostingCursor, IXPostingSet, IXRecordManager and IXPostingList. IXFileFinder has also worked well for me although I don't take full advantage of many of it's features. The IXAttributeQuery and the query language is not fully implemented so if you try to do anything other than straightforward queries, you will have problems.

CONCLUSION

Despite its limitations, the IndexingKit as it now exists is a very powerful tool for developers. It can add a lot of value to your applications if used properly, and frees your application from depending on an expensive relational database. Were it not for the IndexingKit, PDL would not be as functional as it now is without a tremendous amount of additional expense and engineering overhead. If you are working on a data-oriented application, it would be well worth the effort to look into the IndexingKit. ⊗

Leveraging Shrink-Wrap on NEXTSTEP

Andrew K. Turk, President
Sarrus Software, Inc.

NEXTSTEP is all about leverage. With NEXTSTEP's object-oriented development environment, MIS departments are using smaller teams of engineers to deliver more solutions in less time. Internal customer satisfaction is on the upswing, and businesses are working smarter.

This is the leverage of NEXTSTEP's object-oriented programming environment. But there is another kind of leverage to be gained on NEXTSTEP – a kind of leverage that also grows out of NEXTSTEP's object model. I'd like to talk here about how customers are also leveraging shrink-wrap software on NEXTSTEP and examine the competitive advantage that this new leverage provides.

The Record Manager Layer

This is where the foundation of the lower layers gets put to good use. This layer, in which the IXRecordManager is the main class, implements a persistent object store of Objective-C objects. With some limitations, you can literally pass it the id of an object and it will tuck it away and return you a unique handle which you can use as a reference. For example, suppose you were writing an address book application. You might define an object for each entry that looks something like this:

```
@interface AddressEntry : Object
    <IXRecordTranscription>
{
char *name;
char *mailAddress;
char *phone;
char *fax;
char *emailAddress;
char *otherInfo;
}
- (const char *)name;
.
.
@end
Adding the object to the record
    manager is simple:
id myObj = [[AddressEntry alloc]
    init];
unsigned int handle;

handle = [recordManager
    addObject:myObj];
```

By conforming to the IXRecordTranscription protocol, the object can be "serialized" by the IXRecordManager into the data store (an IXBTree). Serializing is a much more efficient way of moving the object into and out of the store. The alternative is to archive the object into a typed stream using the NXWriteRootObject function which has a lot of overhead. However, certain data types cannot be serialized (e.g. void *, union, struct) and id's are not followed, but for simple objects like the one above, serializing is the way to go.

The other important feature of IXRecordManager is that it can automatically build inverted B-Trees on certain attributes (instance variables) of an object. An inverted B-Tree uses the values of the attribute as keys and the handle of the object as the associated values. This way we can quickly search for an object based on the value of one of its instance variables. In our example we probably want to be able to find an AddressEntry based on the name attribute. We would set up the attribute inversion as follows:

```
[recordManager
    addAttributeNamed:"entry name"
forSelector:@selector(name)];
[recordManager
    setComparator:&IXCompareMonocas
    eStrings
andContext:NULL
   forAttributeNamed:"entry name"];
[recordManager
    setTargetClass:[AddressEntry
    class]
forAttributeNamed:"entry name"];
```

The first statement creates the attribute inversion. The second statement defines a function that will compare the values of the attribute so that it can be inserted properly into the B-Tree. Whenever subsequent objects are added to the store, the IXRecordManager will send the name message to the object to determine the value of the attribute. The returned value will be the key used to define the location in the inverted B-Tree, and the handle of the object will become the value associated with that key. The third statement associates the AddressEntry class with the attribute. Note that an IXRecordManager can manage objects of various classes in the same store.

With attribute inversions we can quickly find an object that has a certain value for one of its attributes by searching the inverted B-Tree based on the value we are given, then reading the handle (or handles) associated with that value.

The File Indexing Layer

This top layer of the IndexingKit is a sophisticated system for indexing files in a UNIX file-system. It contains classes that manage conversion of files into an acceptable format, parse the file for words, build up indices of those words, and allow queries to be performed based on the words. The main class, IXFileFinder, can be given a directory path containing files to index. It will recursively scan the directory and create an index of the textual content of the files it encounters. The Digital Librarian application is basically a front-end to the IXFileFinder.

HOW PDL USES INDEXING KIT

The Perennial Document Librarian (PDL) is an example of how the power of the IndexingKit can be leveraged effectively. PDL is a system for managing and sharing libraries of documents among users on a network of NEXTSTEP computers. PDL includes a server that manages a secure library where the documents are stored. It features access control, check-in/check-out, revision maintenance, searching, notification and compression.

PDL uses the IndexingKit in two ways. First it uses it at the IXRecordManager level to store records of the documents in the library as well as other information. It also uses the IXFileFinder to build indices of the textual content of documents in the library.

With the IXRecordManager, I was able to build a quasi-relational/object-oriented database. I built a model (schema) of the database which showed all the objects (entities) and the relationships between them. This model was then implemented by using Objective-C objects which are stored in the database. Objects contain references to other objects not via the id but via the unique handle assigned to each object in the database. I then built inversions on these references so that I could easily find related objects. For example, I have a Document class and a Revision class. There is a one-to-many relationship between these classes such that a Document can have many Revisions, and each Revision is a version of a single Document. This relationship is formalized by including in the Revision class a reference to the instance of Document it is a revision of (i.e. an instance variable that contains the handle of the related Document object). By building an inversion on this reference, I can easily find all the Revision objects that are associated with

Object - Based Computing

A forum for developments in object-based computing in the NeXT community.

An ITS Publication®

Editor: Eric Wespestad

Circulation: Annalea Sommerville

Publisher: Ted Shelton

EMail: obc@its.com

Information Technology Solutions, Inc.
500 West Madison, Suite 2210
Chicago, IL 60661

Subscription Information

12 Issue subscriptions are $28. Please specify hard copy version, or electronic version (available only to subscribers with NeXT Mail). We regret that subscriptions are available only on a pre-payment basis. Please send checks or money orders to: Information Technology Solutions, Inc. 500 W. Madison Suite 2210, Chicago IL 60661. Please include your name, company, address, fax and telephone numbers, and an EMail address if available (specify NeXT Mail). Object-Based Computing is available on-line to subscribers.

Advertising Information

Center pages 2 (7.5×9.5) $1100 • Inside front or back pages 2 (7.5×9.5) $800 • Full page (7.5×9.5) $350 • Half page (7.5×4.5) $200 • Third page (7.5×3) or Full column (2×9.5) $120 • Business card (3.5×2) $50.

Article Submissions

Thank you for letting us know what kind of articles you'd like to read. We are also looking for articles to print... so please consider sharing some of your experiences with the NeXT community. Please send submissions or questions about submission deadlines to the editor.

ITS INFORMATION TECHNOLOGY SOLUTIONS

IndexingKit continued…

work of data management. The IndexingKit provides objects that allow creation and management of the database itself. The kit does not implement any particular type of database such as relational or object-oriented, rather it provides fundamental building blocks that allows programmers to develop their own custom database.

The core of the kit provides reliable transaction based data storage and structures for efficient data organization. At higher levels of the kit are classes that implement a persistent object store, allowing Objective-C objects to be directly inserted into the database. This is especially nice for NEXTSTEP programmers. At still higher levels of the kit are objects that can parse files and build indices of their textual content, and it is from this functionality that the IndexingKit gets its name.

Besides data storage and management, the IndexingKit provides objects for fast searching and retrieval of data. Objects can be written to and read from the database directly without having to be archived into a typed stream. Also instance variables of objects in the database can be read directly without having to read the entire object out of the database. This makes data access fast and efficient.

Any application that manages information either at a personal or a group level can benefit from the functionality of the IndexingKit. Often, it is not practical to rely on a commercial database for your application. Commercial databases often add cost as well as overhead. For example, a commercial database would be overkill for an address book application, yet the IndexingKit can easily be used to store and search address book entries without any additional cost (as long as it is bundled with NEXTSTEP).

Bulletin board systems, news feed management, customer service tracking, flat-file databases, calendering, address books, scheduling, and client tracking systems are all examples of applications that can benefit from the IndexingKit.

THE INDEXINGKIT ARCHITECTURE

The IndexingKit consists of several layers of functionality. Each layer, consisting of several classes, is built atop another. The application programmer can use the kit at any level depending on the requirements. In my opinion, the IXBTree class is really the foundation of the kit. Lower levels of the kit provide a backing store for the B-Trees and higher levels use the B-Trees to contain data, record attribute inversions and indices.

I will discuss briefly the four layers of the kit and the key class of each layer.

The Storage Management Layer

This lowest layer provides a backing store for small to large sets of data. It allocates and manages blocks of storage for a client application. IXStore is the key class of this layer. It manages memory-based storage consisting of elements called blocks. These blocks are relocatable within the store and thus the store can be compacted to optimize memory usage. IXStore also implements transaction management so that several operations made to your database may be committed to the store at once. This also allows you to undo the changes before committing them. A sub-class of IXStore, the IXStoreFile, keeps its storage in a file.

The B-Tree Layer

This next layer up implements B-trees, which define an organization that allows for fast searching of data elements and efficient paging. The main class here is IXBTree. It is similar to the HashTable class in that it stores ordered associations of un-typed data using key/value pairs, therefore it can be used as an alternative data structure to the HashTable and List classes.

IXBTrees are created in an IXStore (or IXStoreFile) and thus gain the benefit of transaction management.

Object - Based Computing

Published often by

INFORMATION TECHNOLOGY SOLUTIONS INC.

A forum for developments in object-based computing in the NeXT community.

VOLUME 1, NUMBER 5, December 1993

editorsDesk

A lot seems to be happening on the object-based computing front recently—from the activities of objectware companies (those that sell objects, possibly in addition to applications and consulting services), to rumors of large new "object providing" companies being created. Hopefully we are beginning to see signs of a vibrant new marketplace? Will NeXT's latest operating system release (NEXTSTEP 3.2) and the forthcoming Portable Distributed Objects play a greater role? We'll have to wait and see.

While preparing some notes on the ObjectWare Scenario presented in this issue, I came across the following in one of my mailboxes – lucky for us a group called OPN is taking this seriously.

> "If you are a vendor, are you ready to componentize your software?"

"No, because the developer market is much smaller than the user market and most of the user market knows absolutely nothing about objects. First, we have to build the market by educating the marketplace before we try to sell something."

One quick (but important) announcement: WE HAVE MOVED. Information Technology Solutions, Inc. relocated to larger (and I must say, fancy) space in Chicago's CityBank Tower. See the back page and page footers for the new address. During this move I (I'm not Ted by the way :-) have adopted the editors role of OBC.

IndexingKit: The Unsung Object Library

Phil Zakhour, San Franciso Information Technology Solutions

One of the most neglected object libraries that NeXT has to offer is the IndexingKit. I believe this is so because it is useful at the "back-end" of an application and therefore lacks the glamour and sex appeal of GUI related object libraries such as the AppKit and the 3DKit. Also, unlike the AppKit which is used by most NEXTSTEP applications to build the user interface, the utility of the IndexingKit is more limited in scope to applications that deal with information management. It is important not to let these drawbacks hide the fact that for information management applications, the IndexingKit is one of the most useful and powerful kits that NeXT has to offer.

In this article I will give an overview of the IndexingKit, why it is useful, and a quick rundown of its key objects. I will also give a practical example of how an application that I developed, the Perennial Document Librarian, was able to use the IndexingKit to its advantage.

WHAT IS THE INDEXINGKIT?

The IndexingKit is the only kit NeXT offers that is data oriented. The DBKit is not a data oriented kit because all it does is provide a layer between the user and a commercial database which does all the

continued on page 2

Submissions Encouraged!

Thank you for letting us know what kind of articles you'd like to read. We are also looking for articles to print... so please consider sharing some of your experiences with the NeXT community.

thisIssue

In this issue the aspects of a component based object marketplace are explored from several different view points: from a tour of the IndexingKit that is *bundled* with NEXTSTEP - and how it was used in building a shrink-wrap application, to the mechanisms for *licensing* of (or paying for) third-party objects that are used *within* third-party applications, to the treatment of third-party *applications as reusable objects* (though APIs) in and of themselves, and, rounding this off is some advice on tailoring and operating a development environment for producing your own objectware for NEXTSTEP.

contents

features