

# Ein zweiter Frühling für die Workplace Shell

ADRIAN GSCHWEND, GRÜNDER UND WEBMASTER VON NETLABS.ORG

*OS/2 und eComStation besitzen eine sehr treue Anhängerschaft. Um zukunftssicher zu sein, müssen der Kernel und auch die Oberfläche neu aufgebaut werden. Das Projekt »Voyager« für den Nachfolger der Workplace Shell soll deren Vorzüge auch neuen Hardwaregenerationen erschließen.*



eComStation

**O**ktober 2005 auf der Systems in München. Auf dem Open-Source-Stand, der durch die freeX organisiert wurde, vertrat der Autor dieses Beitrag netlabs.org und somit die OS/2- und eComStation-Community. Nebst dem üblichen »Ach, das gibt's noch?« und einigen ungläubigen Blicken hörte man vor allem Sätze wie »Die Oberfläche war genial«, »Nie war ein Fileserver so einfach zu administrieren« oder »Wieso hat IBM das nie Open Source gemacht?« Zur letzten Frage kann man ohne Probleme ein Buch schreiben, es gibt unzählige Gerüchte und Geschichten, warum die IBM OS/2 vor Jahren absägte und schließlich zum 1. Januar 2006 den Support komplett einstellte. Die einzige Möglichkeit, heute noch an ein OS/2 und an Support zu kommen, besteht im Kauf einer eComStation. Das ist der lizenzierte Nachfolger von OS/2, der unterdessen in der Version 1.2 verfügbar ist. Die Version 2.0 befindet sich momentan in der Betaphase. Die Tendenz in der eComStation ist klar: Möglichst viele ursprüngliche Komponenten werden durch offene Software ersetzt, sei es ein XUL-basierter Installer, OpenLDAP zur

Benutzerverwaltung, Portierung und Integration von Industriestandards wie OpenOffice.org, Firefox, Thunderbird und eine Integration von möglichst vielen Treibern. Projekte wie der in der freeX 2'2006 vorgestellte GenMac-Wrapper und das ACPI-Projekt tragen ihren Teil dazu bei.

## Spurensuche

Ein grundsätzliches Problem wird damit aber nicht gelöst: Längerfristig wird es sehr schwierig werden, die eComStation auf neuer Hardware zu betreiben. Eine 64-Bit-Version des Kerns selbst ist technisch nicht durchführbar, und die IBM wird kaum etwas dazu beisteuern. In der OS/2-Community hörte man diesbezüglich oft den Vorschlag, daß man das System als Open-Source-Software von Grund auf neu programmieren solle. Dieser Vorschlag ist aber aus mehreren Gründen nicht machbar. Erstens gibt es genug offene Kernels, zweitens gibt es genug APIs und es ist kaum anzunehmen, daß neue Programmierer das offene OS/2-API übernehmen werden, und drittens ist die Community für so ein Unterfangen ganz einfach zu klein.

Um an das Ziel zu gelangen, muß die Frage anders gestellt werden: Warum setzen immer noch erstaunlich viele Anwender OS/2 und eComStation ein? Warum weigert sich diese Gruppe so hartnäckig, auf offene Software wie beispielsweise Linux mit Gnome oder KDE umzusteigen? Der Autor gehört natürlich auch zu dieser Gruppe. Grund genug also, einige dieser Punkte aufzuzeigen:

- Der Kernel von OS/2 ist zwar relativ alt, allerdings wurde er klar strukturiert entworfen und sauber implementiert. Für Applikationen und Treiber gibt es ein stabiles(!) API/ABI, das heißt, Applikationen und Treiber von 1992 laufen auch auf der aktuellen eComStation 1.2 ohne Probleme.
- Der eigentliche Desktop, auf OS/2 *Workplace Shell* genannt, ist komplett objektorientiert. Statt »everything is a file« gilt hier »everything is an object«. Das ist das, was vom größten Teil der OS/2-User als das »OS/2 Feeling« beschrieben wird, und wer mit der WPS ernsthaft gearbeitet hat, weiß wovon die Rede ist. Entgegen allen Behauptungen kommen weder Gnome

noch KDE auch nur annähernd an die WPS heran. Ihr Ansatz ist nach wie vor einmalig.

- Durch die WPS können viele Dinge wesentlich »logischer« implementiert werden. Als Beispiel sei hier der LAN-Manager von IBM erwähnt, der es auf eine sehr intuitive Art erlaubt, SMB-User und -Shares zu verwalten, dies voll integriert in die Arbeitsoberfläche.
- Die meisten Applikationen halten sich an den Common-User-Access-Standard von IBM, das heißt, die Dialoge und Menüs sind klar strukturiert, durchdacht und verwenden dieselben Tastenkombinationen. Für den Anwender ist dies ein nicht zu unterschätzender Vorteil.
- Das Multimedia Subsystem bietet schon seit 1994 eine Abstraktion von Codecs und Transports. Das heißt, daß ein zehn Jahre alter Media Player Ogg- und Flac-Dateien spielen kann, wenn das System den Ogg/Flac-Codec implementiert hat. Die Applikation muß sich nicht um das Format kümmern, sondern gibt diesen Teil an das Subsystem weiter. Das gleiche gilt für den Ursprung der Datei, sei dies das lokale Dateisystem oder ein http-Stream, auch das wird durch das System abstrahiert.
- OS/2 und eComStation sind in vielen Sprachen verfügbar, bei der Übersetzung wird auf gute, lesbare Sätze geachtet.
- Für Entwickler gibt es ebenfalls eine sehr gute Dokumentation, das API ist klar aufgebaut und gut dokumentiert. Das API war auch während der Blüte von OS/2 stabil und wurde nicht alle paar Monate geändert.
- OS/2 und eComStation kommen mit vergleichsweise wenig Hard-

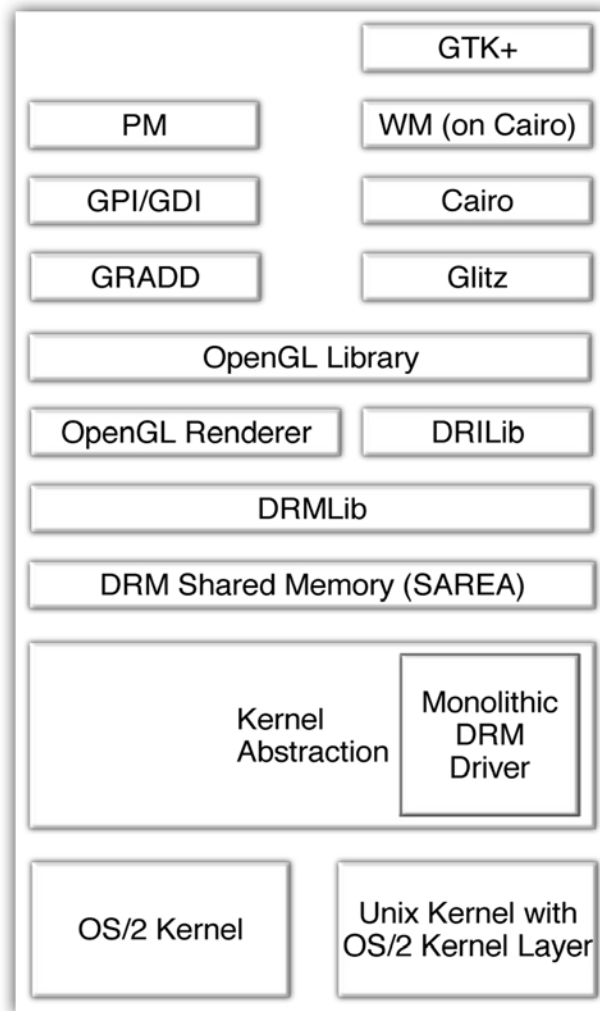


Bild 1: Eine mögliche Variante des Zusammenspiels der Komponenten

ware-Ressourcen zurecht, die Interaktion mit dem Desktop wird als sehr »schnell« bezeichnet. Ein Klick auf dem Desktop führt meistens direkt zu einer Aktion, was leider bei vielen anderen Systemen heute nur mit einer merkbareren Verzögerung der Fall ist.

### Das Voyager-Projekt

netlabs.org bietet nun seit über acht Jahren freie Software für OS/2 und eComStation an. In dieser Zeit wurde oft überlegt, wie man diese Ideen in der Zukunft erhalten und weiter entwickeln kann und trotzdem auf dem Boden bleibt, was die Zeitdauer eines solchen Projekts betrifft.

Es ist klar, daß man sich auf das absolut Erforderliche an Arbeit konzentrieren muß. Alle Projekte, die zuerst mit dem Kernel beginnen,

haben folglich den falschen Ansatz. Ein solches Projekt kann nur Erfolg haben, wenn es möglichst schnell funktionierende Komponenten bereitstellt und somit für Programmierer und Benutzer auf möglichst vielen Plattformen attraktiv wird, sei dies eComStation, BSD, Linux oder sogar ein Windows-Kernel.

Für den Kern des Projekts hat sich das Core-Team aus diesen Überlegungen auf folgende Punkte geeinigt:

- Implementierung eines Objektmodells, das einen WPS-Nachbau ermöglicht. Dieses Modell wird sich stark an SOM (System Object Model) orientieren (Codename *Voyager Object Model*).
  - Als Toolkit für den Workplace wird GTK+ verwendet.
  - Als 2D-Library dient Cairo. Die Xlib ist tabu.
  - Das Projekt soll so portabel wie möglich sein. Was plattformabhängig ist, muß über Bibliotheken abstrahiert werden.
  - Das Projekt soll von Anfang an sauber dokumentiert werden, neue Programmierer sollen sich schnell zurechtfinden und mithelfen können.
  - Die Lizenz des Projekts soll offen sein, durch das binärkompatible Objektmodell macht aber die GPL keinen Sinn, weil sie zu restriktiv ist.
- Dazu kommen noch weitere Projekte, die nicht zwingend und somit nicht kritisch für das Projekt sind, aber trotzdem Sinn machen und implementiert werden sollten:
- Implementierung eines neuen Window-Managers auf Basis von Cairo, ohne Xlib-Abhängigkeiten (Codename *Neptune*).
  - Integration des Xorg-OpenGL-Treiber-Backends mit dem Ziel, Glitz als Cairo-Backend zu verwenden. Das ermöglicht Hardwarebeschleu-

nigung für den Workplace, im besten Fall kann längerfristig die Xlib komplett entfallen.

- Implementierung eines neuen Multimedia-Subsystems, das Codecs und Transports komplett abstrahiert (Codename *Triton*).
- Implementierung eines OS/2-ABI-Layers, der es ermöglicht, bestehende OS/2- und eComStation-Applikationen sanft zu migrieren.

Diese Ideen werden momentan als Ganzes unter dem Codename »The Voyager Project« entwickelt. Bild 1 verdeutlicht das Zusammenspiel der verschiedenen Komponenten in einer möglichen Designvariante mit einer OS/2-ABI-Layer oder ohne diesen Teil direkt auf einem unix-like System mit Xorg-Grafik-Backend.

Ein zentraler Bestandteil von Voyager ist das Objektmodell. Es ermöglicht den Bau eines objektorientierten Desktops. Eine Datei ist nicht wie bei einem taskbasierten Desktop eine Datei im eigentlichen Sinn, sondern wird vom Desktop ebenfalls als Objekt repräsentiert. Objekte werden als Klasse implementiert und erben somit auch die Eigenschaften von den Eltern-Klassen und deren Methoden. Klassen können zu jedem Zeitpunkt registriert oder abgemeldet werden, somit kann man den Desktop jederzeit um neue Funktionen erweitern. Das geht soweit, daß man ganze Klassen durch eine neue Version ersetzen kann, die zum Beispiel das Verhalten von Methoden ändert oder auch ganz neue Methoden ergänzt. Abgeleitete Klassen funktionieren nach wie vor, profitieren aber von den erweiterten Möglichkeiten der Ersatzklasse.

## Das Voyager Object Model

Im Gegensatz zu anderen Projekten wie Gnome bietet SOM ein binär-kompatibles Interface an. Ein Entwickler kann also binäre Objekte bereitstellen, die auch innerhalb von verschiedenen Versionen des Desktops kompatibel sind und verwendet werden können.

Im Gegensatz dazu kann bei einem

Desktop-Konzept wie Gnome schon ein Wechsel von Version 2.10 auf 2.12 ein Widget für das Gnome-Panel unbrauchbar machen, weil ein Interface geändert wurde. Der Entwickler muß sein Widget an das neue API anpassen und es danach neu kompilieren, während man unter Voyager das binäre Objekt ohne Änderung weiterverwenden kann. Diese Eigenschaft macht es möglich, den Desktop zu erweitern, ohne den Quelltext bestimmter Klassen oder Applikationen zur Verfügung zu haben. Oben wurde schon das Beispiel des Multimedia-Subsystems erwähnt, wo eine Applikation um Ogg/Flac-Fähigkeiten erweitert wird, ohne an der Applikation selber etwas zu ändern.

Entsprechend wird auch beim Desktop verfahren, wie Bild 2 verdeutlicht. Die Klasse *MMAudio* stellt Methoden für die Wiedergabe von Audio-daten zur Verfügung. Ein Programmierer kann nun die Klasse *MMMP3* erstellen, die sich nahtlos integriert und die Abspielfunktion von der Elternklasse erbt. Zusätzlich implementiert diese neue Klasse zum Beispiel die Verwaltung von ID3-Tags. *WPDataFileExt* ist ein Beispiel, wie eine Basisklasse (*WPDataFile*) durch eine neue Klasse mit neuen Funktionen ersetzt werden kann. Die Klasse *MMAudio*, die ursprünglich von *WPDataFile* abgeleitet war, bemerkt gar nicht, daß sich die Elternklasse geändert hat, und arbeitet wie zuvor. All das geschieht völlig transparent für den Benutzer und ein Neuübersetzen der Desktop-Applikation ist nicht erforderlich.

Ein wichtiger Punkt eines Desktops sind die Applikationen. Sowohl Gnome wie KDE investieren einen großen Teil ihrer Arbeit in Applikationen, die im Desktop integriert sind. Dazu werden mit wenigen Ausnahmen komplett neue Applikationen geschrieben, ein Ansatz, der zu sehr viel Mehraufwand führt. Das Voyager-Projekt versucht auch hier, möglichst viel zu übernehmen. Statt einen eigenen Webbrowser zu schreiben, sollte die Zeit darin investiert werden, Firefox in den Desktop zu integrieren und die Gren-

zen zwischen Browser und Desktop immer mehr zu verschmelzen, Stichwort Web 2.0. Dasselbe gilt für Applikationen wie OpenOffice.org und Eclipse. Man kann praktisch jede Applikation mit einigen Erweiterungen in einen Desktop integrieren und Funktionen wie Drag&Drop und Objektvorlagen hinzufügen. Der Netscape Browser (jetzt Mozilla) bekam diese Funktionen zum Beispiel erst eingebaut, nachdem IBM den Netscape Browser nach OS/2 portierte, entsprechende Erweiterungen vornahm und diesen Code wieder zurück an Netscape gab.

## Triton

Mit OS/2 Warp 3.0 hat IBM 1994 zum ersten Mal ein Multimedia-Subsystem eingeführt, das die Idee von SOM auf Audio- und Video-Codecs erweiterte und somit die Formate gegenüber den Applikationen abstrahierte. Das Konzept funktioniert nach wie vor gut für Audio-Codecs und die Entwickler arbeiteten auch damit. Bei Video sieht die Situation etwas schlechter aus, nicht zuletzt wegen der Komplexität der einzelnen Codecs.

Triton übernimmt im Voyager-Projekt diesen Teil, es ist eine Implementierung einer Playback-Engine, die an die Anforderungen moderner Video-Codecs optimiert ist. Codecs können von Grund auf in Triton implementiert werden, der Vorteil liegt darin, daß so möglichst wenig Speicheroperationen ausgeführt werden und der Codec somit sehr effizient implementiert werden kann. Es soll aber auch möglich sein, bestehende Bibliotheken wie FFmpeg zu integrieren und ohne großen Aufwand viele Codecs zu unterstützen.

Triton abstrahiert auch den Transport des Mediums, sei dies eine lokale Datei auf der Festplatte oder ein http-Stream aus dem Internet, der Applikationsentwickler muß sich darum nicht kümmern.

Mit Triton kann man auch Bildformate im System abstrahieren, da ein Bild nichts anderes ist als ein Film mit einem einzigen Frame.

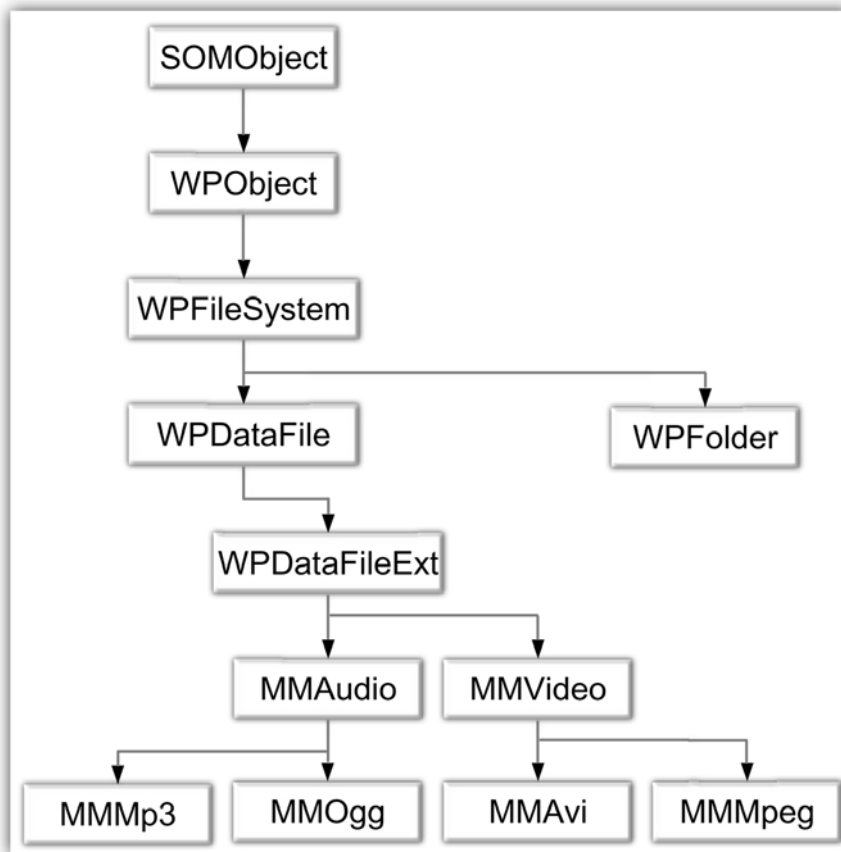


Bild 2: Objektorientierung am Beispiel; die WPDataFileExt Klasse ersetzt WPDataFile als Basisklasse für MMAudio und MMVideo

Triton ist momentan als Prototyp verfügbar, bei der ersten Vorführung auf dem Developers Workshop 2006 konnte man MP3-Dateien von der lokalen Festplatte abspielen, bereits integriert im ganzen Triton-Framework. Details zur Implementation von Triton kann man der Präsentation *IO Proc Replacement* entnehmen, die als PDF auf der Webseite verfügbar ist.

### Neptune

Voyager soll längerfristig auf unix-artigen Systemen ohne die Xlib auskommen können. Man hört zwar regelmäßig die Aussage, daß ein großer Vorteil der Xlib die Netzwerk-Transparenz sei, allerdings ist das immer weniger der Fall. Applikationen, die mit hardwarebeschleunigtem OpenGL arbeiten, rendern bereits an der Xlib vorbei, anders wäre die hohe Performance nicht erreichbar. Dazu wird typischerweise das Direct-Rendering-Interface (DRI) von X.org

verwendet.

Um einen Desktop ganz ohne Xlib aufbauen zu können, müssen einige Dinge ersetzt werden, unter anderem der Window-Manager. Neptune übernimmt diesen Teil, indem das Projekt einen Window-Manager bereitstellt, der statt in Xlib in Cairo rendert.

Jeder Window-Manager muß sich typischerweise um folgende Teile kümmern:

- Die offenen Client Fenster,
- die Stacking Order der Fenster (welches Fenster liegt über wem),
- die Zuweisung der Clients in die virtuellen Desktops,
- die Session,
- das Handling mehrerer Bildschirme,
- Client-Dekorationen (Themes/Skinning).

Ein großes Handicap der X-basierten Desktops stellen heute oftmals Dualhead-Systeme oder das Betreiben eines Beamers an einem Laptop dar. Der Konfigurationsaufwand ist

sehr groß und die momentanen Window-Manager sind nicht für solche Setups ausgelegt. Neptune versucht das von Anfang an zu berücksichtigen, indem mehrere physikalische Bildschirme ins Konzept aufgenommen werden. Bild 3 verdeutlicht das.

### Virtuelle Screens

In der untersten (ersten) Schicht hat der Benutzer in diesem Beispiel vier physikalische Monitore. Die Auflösungen müssen nicht bei jedem Monitor zwingend gleich sein. Die nächste Schicht stellt einen virtuellen Screen dar, der alle physikalischen Monitore überspannt, und somit die maximale Fläche darstellt, die gleichzeitig angezeigt werden kann. Dieser Teil wird als Cairo Surface implementiert. Die dritte Ebene zeigt die Unterteilung des virtuellen Screens der zweiten Layer in beliebige Subparts. In der vierten Schicht ist wieder jeder Subpart aus dem Gitter der dritten Lage ein neuer virtueller Screen, der unterteilt werden kann. Die oberste (fünfte) Schicht zeigt dann wieder eine mögliche Unterteilung eines der virtuellen Bildschirme aus dem vierten Layer.

Der Vorteil dieser Unterteilung liegt auf der Hand: Auf jeden virtuellen Screen kann ein virtueller Desktop gemappt werden. Ein Beispiel dafür ist das typische Beamer-Setup. Während der Beamer meistens hinter dem Sprecher steht und somit nur durch eine Drehung einsehbar ist, könnte das Bild nebst dem Beamer auch noch in der Hälfte des Laptopmonitors gemappt werden. In der anderen Hälfte kann der Sprecher weitere Applikationen geöffnet haben, zum Beispiel Notizen zu der Präsentation. Trotzdem kann er aber die Präsentation selber sehen und auf die nächste Seite umschalten, indem der Fokus kurz dem richtigen virtuellen Screen gegeben wird. Natürlich ermöglicht dieses Konzept viele weitere Möglichkeiten, man könnte sich zum Beispiel vorstellen, einen 3D-Shooter im mittleren von drei physikalischen Monitoren im

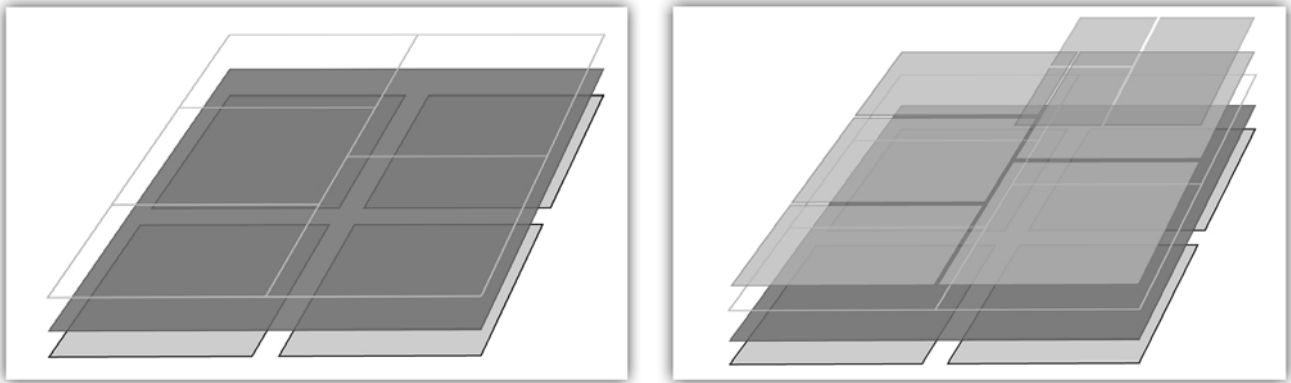


Bild 3: Virtuelle Desktops in Perfektion

Vollbildmodus zu betreiben, währenddem die Karte des Spiels im Monitor links davon dargestellt wird und der Monitor rechts für Nachrichten zwischen den Spielern eingesetzt wird.

Die technische Implementation dieses Konzepts wird über sogenannte *Rendering-Pipelines* implementiert, der momentane Prototyp hat diese Ideen bereits umgesetzt. Als nächster Schritt werden die anderen Komponenten des Window-Managers angegangen. Details dazu sind der Präsentation *A WM based on Cairo* zu entnehmen, die als PDF zum Download zur Verfügung steht.

Da es noch nicht möglich ist, die Xlib komplett wegzulassen, unterstützt Neptune auch bestehende Systeme mit Xlib und wird momentan auch unter einem normalen X-System entwickelt. Dazu werden auch freedesktop.org-Standards wie *ewmh* implementiert.

## Ausblick

Die in diesem Beitrag erwähnten Ideen wurden im April 2006 auf dem netlabs.org Developers Workshop in der Schweiz einem breiteren Publikum vorgestellt, darunter waren auch einige Entwickler, die OS/2 und die WPS gar nicht kannten. Die Reaktion der Besucher kann als sehr positiv beschrieben werden, wer das Konzept der WPS nicht kannte, war erstaunt über die vielen Möglichkeiten, die eine solche Oberfläche sowohl dem Entwickler wie dem Anwender bietet.

Es wurde oft gefragt, ob ein Desktop

überhaupt performant arbeiten kann, wenn alles als ein Objekt repräsentiert wird. Das Voyager Team verweist hier gerne auf die WPS als »Prototyp« dieser Idee, die seit OS/2 2.0 (1992) produktiv eingesetzt wird und beweist, daß man mit diesem Konzept auch auf älterer Hardware sehr flüssig arbeiten kann.



Mit der WPS auf OS/2 konnten auch sehr viel wertvolle Erfahrung gesammelt werden. Die Core-Entwickler wissen, welche Konzepte gut funktionieren und wo Probleme auftreten. Dies ermöglicht es dem Team, eine neue Implementierung zu schreiben und die erkannten Probleme von Anfang an zu vermeiden, was dem längerfristigen Ziel klar helfen wird und eine gewisse Stabilität auf API- und ABI-Level sicherstellen wird. Momentan wird an diversen Kom-

ponenten bereits gearbeitet, Quelltext ist bereits im Subversion-Repository auf [netlabs.org](http://netlabs.org) verfügbar und sobald die Frage der Lizenz definitiv entschieden ist, wird der Sourcecode öffentlich zugänglich werden. Die momentane Entwicklung läuft mit Ausnahme des Window-Managers in erster Linie auf OS/2 und eCom-Station ab, rein vom Code her sollte es aber mit etwas Aufwand möglich sein, ihn auf unixbasierten Plattformen zu kompilieren. Mittelfristig wird das Team Build-Umgebungen für möglichst viele Systeme anbieten, es ist ausdrücklich erwünscht, daß Entwickler auf anderen Systemen mithelfen, dies zu ermöglichen. Voyager ist noch kein fertiger Desktop und wird es noch eine zeitlang nicht sein. Das Team hat lange genug Software geschrieben um zu wissen, wie ambitioniert so ein Projekt ist. Aber wer nicht wagt, kann nicht gewinnen, und folglich ist das Team der Meinung, daß mit Voyager ein schönes Stück Software geschrieben wird. Mithilfe am Projekt ist natürlich in jeder Beziehung sehr willkommen, an Arbeit mangelt es nicht! ◆

### Links:

Die Seite zum Developers Workshop 2006. Allen Präsentationen sind als PDF- und Ogg-Dateien verfügbar:

[http://wiki.netlabs.org/index.php/Developers\\_Workshop\\_2006](http://wiki.netlabs.org/index.php/Developers_Workshop_2006).

Voyager Frequently Asked Questions. Ein guter Start für alle, die mithelfen möchten: [http://wiki.netlabs.org/index.php/Voyager\\_FAQ](http://wiki.netlabs.org/index.php/Voyager_FAQ).

netlabs.org: <http://www.netlabs.org>

OS/2 Warp V4 in Team, PDF-Download des vergriffenen C&L-Buchs, enthält einen guten Überblick über OS/2 und dessen Konzepte:

<http://www.cul.de/download.html>.