JOHANNES KEPLER
UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Anforderungen an den Kernel einer objektorientierten, web-basierten e-Learning Plattform im Vergleich zu herkömmlichen Betriebssystemen

Diplomarbeit zur Erlangung des akademischen Grades

Diplom-Ingenieur

in der Studienrichtung *Informatik*

Angefertigt am Institut für

Informationsverarbeitung und Mikroprozessortechnik

Betreuung:

o. Univ.-Prof. Dr. Jörg R. Mühlbacher

Von:

Oliver Kronawittleithner

Linz, *Jänner 2003*

meinem bruder

INHALT

Danksagung

Mein besonderer Dank gilt Herrn o. Univ. Prof. Dr. Jörg R. Mühlbacher für die Unterstützung dieser Diplomarbeit und die Begleitung während des gesamten Projekts. Mit Erfahrung und Umsicht hat er so manche Fehleinschätzungen berichtigt, machen Irrweg aufgezeigt und mir Ziele immer vor Augen geführt. Durch seine Art und Weise, das Denken mit dem Handeln zu verbinden, habe ich weit über diese Arbeit hinaus positiv profitieren können.

Darüber hinaus danke ich allen Projektmitarbeitern am FIM für ihre Unterstützung. Dr. Susanne Reisinger, DI Alexadros Paramythis und Dr. Michael Sonntag standen mir immer mit Rat zur Seite.

Auch möchte ich meinen Eltern danken, die trotz aller zusätzlichen Aktivitäten während der Studienzeit nie das Vertrauen in mich verloren haben und mich maßgeblich auf diesem Lebensabschnitt unterstützt und gefördert haben.

Kurzfassung

Nicht nur vom Ministerium für Bildung, Wissenschaft und Kunst forciert, verändern sich die Paradigmen des Lernens. Weg vom „Frontalvortrag“ hin zum selbstgesteuerten Lernen, zeit- und ortsunabhängig, führt dieser Weg zu einer neuen Art der Wissensvermittlung, dem e-Learning. Über Internet (oder andere meist asynchrone Kommunikationsmechanismen) wird Wissen zur Verfügung gestellt, das jederzeit aber eben selbst gesteuert abgerufen werden kann.

In dieser Arbeit wird nun die technische Seite einer solchen e-Learning Plattform, nämlich das am Institut für Informationsverarbeitung und Mikroprozessortechnik (FIM) entwickelte WeLearn System verglichen mit „herkömmlichen“ Betriebsystemen wie etwa Windows 2000/XP. Zunächst werden die verschiedenen Systeme vorgestellt und im Überblick beschrieben. Im Speziellen wird danach auf die Funktionsweise und den Aufbau der jeweiligen Kernel näher eingegangen und die sich aus der Webbasierung von WeLearn ergebenden besonderen Anforderungen an den WeLearn Kernel in Bezug auf die Erweiterbarkeit des Gesamtsystems beleuchtet.

Auch werden mögliche (noch nicht realisierte) Ausbaustufen wie etwa Schnittstellen zu Agentensystemen oder eventgesteuerte, autonome Komponenten innerhalb des Systems angedacht und verschiedene Varianten aufgezeigt.

Die Konsequenzen der Javaimplementierung des WeLearn Kernels, dessen Zusammenspiel mit der Java Virtual Machine und den diese beiden Komponenten umgebenden Webserver werden genauso betrachtet wie nötige Kompromisse in der Modellierung der Gesamtarchitektur.

Im „Ausblick“ werden schließlich Schwachstellen in der Architektur, Lösungsvorschläge und mögliche Weiterentwicklungen aufgezeigt.

Abstract

Not only because the ministry of science pushes e-learning the paradigms of learning change. In the past the only well known kind of teaching was lecturing but in the future the self-driven, time and place independent learning will get on top und lead to e-learning. The internet will get the prime medium for communication and will provide people with the necessary information asynchronously.

This thesis is about the technical realisation of such an e-learning platform called "WeLearn" that has been developed at the "Institute for Information Processing and Microprocessor Technology (FIM)" at the University of Linz. It is compared to traditional operating systems like Windows 2000/XP. The core part of this paper deals with the differences in functionality of the WeLearn kernel and especially considers the requirements that arise from the web-based architecture as far as extensibility of the kernel is concerned.

More over, possible (but not yet implemented) enhancements of the system (interfaces to agent systems for instance) are discussed in the second part.

Consequences coming from the implementation in java are discusses as well. Necessary trade-offs as a result of the interaction of the WeLearn kernel with the java virtual machine and the web server are described in detail.

The chapter called "Ausblick" summarizes the weaknesses of the WeLearn architecture and shows possible ways out respectively possible further developments.

1 Konzept des Mikrokernel

1.1. Einleitung

Einer der Hauptaufgaben eines Betriebssystems besteht darin, die zur Verfügung stehenden Ressourcen wie Speicher, Geräte, etc. möglichst effizient und transparent für den Benutzer zu verwalten. Wünschenswert ist weiters ein reibungsfreies und nahtloses Zusammenspiel aller Komponenten. Je nach Anforderungsprofil ist strengeres oder weniger strenges Management dieser Ressourcen von Nöten. Echtzeitsysteme (wie etwa Flugzeugsteuerungen) verlangen naturgemäß mehr Zuverlässigkeit im Sinne von Einhaltung vorgegebener Zeitschranken als Textverarbeitungsprogramme, um ein praktisches Beispiel zu nennen. Demnach ist es einsichtlich, dass auch verschiedene Architekturen des Betriebssystems und im Besonderen deren „Herzstücke“, deren Kernel, nötig geworden sind. Während bei weniger zeitkritischen Applikationen meist monolithische Kernel zum Einsatz kommen, wo möglichst viel Funktionalität implementiert ist, wird bei Echtzeitsystemen zunehmend auf die „schlanken“ Mikrokernel vertraut. Die Besonderheit solcher Mikrokernel besteht darin, dass sie lediglich einige wenige Grundfunktionen umfassen, die mit besonderen Rechten bzw. in einem besonderem Modus, dem so genannten Kernel-Mode, ausgeführt werden und als einzige auf alle Systemresourcen uneingeschränkten Zugriff besitzen. Alle auf höheren Ebenen liegenden Abstraktionen bedienen sich lediglich dieser mehr oder weniger als „atomar“ zu betrachtenden Grundfunktionen im so genannten „User-Mode“.

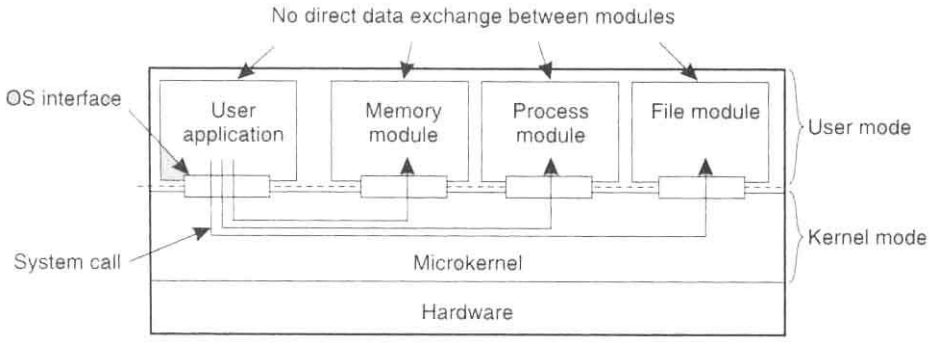

Abbildung 1 Die Mikrokernel Architektur im Allgemeinen, Quelle [Tan02]

In diesem Kapitel soll einerseits die Entwicklung des Mikrokernel näher betrachtet, und andereseits ein Überblick über seine Grundfunktionalitäten an Hand des QNX Kernels gegeben werden.

1.2. Historische Entwicklung

Einer der ersten Mikrokernel wurde Anfang der 80er Jahre an der Carnegie Mellon University entwickelt. Damals war „Mach“ einer der ersten seiner Art, da er auf völlig neue Weise das Virtual-Memory Management bzw. die IPC, die InterProcessCommunication implementierte. Wesentliches Designziel war, einen möglichst kleinen, aber leicht erweiterbaren Kernel zu schaffen, der klar definierte Schnittstellen nach außen besitzt. Die Grundidee kam ursprünglich aus dem UNIX Bereich, denn auch dort galt (zumindest zu Beginn der Entwicklung), dass die Kommunikation zwischen Kernel und Objekten über eine kleine Zahl von einfachen Operationen stattfinden sollte. Im Laufe der Zeit wurde dieser Ansatz allerdings immer öfter zu Gunsten von Performance, aber auch anderen Gründen aufgeweicht, sodass bei heutige Betriebssystemen kaum noch von „echten“ Mikrokernel gesprochen werden kann.

Die bereits angedeutete Modularität des Mach – Kernel erleichtert die Adaptierung des Systems an unterschiedlichste Hardwareanforderungen einerseits und andererseits bietet sie die Möglichkeit auf unkomplizierte Art und Weise komplexere Dienste sozusagen „zusammenzubauen“.

1.2.1. Tasks, Threads, Ports and Messages

Der Mach Kernel kennt vier Abstraktionsniveaus: Tasks, Threads, Ports und Messages, die zwar unmittelbar miteinander verbunden sind, aber unterschiedliche Aufgaben erfüllen.

Der Task versteht sich als Laufzeitumgebung und stellt den Virtuellen Adressraum zur Verfügung. Darüber hinaus kann der Task auch geschützten Zugriff auf andere Systemressourcen wie Prozessoren, etc. anbieten. Ein Thread hingegen wird verstanden als Grundeinheit für die CPU Auslastung. Jeder Thread gehört genau einem Task und hat Zugriff auf all seine Ressourcen. Die Frage warum Threads dann überhaupt existieren, wenn sie ganz offensichtlich ja die selben Möglichkeiten wie ihr „Mutter-Task“ haben, ist einfach damit zu beantworten, dass das switching, also das Umschalten der CPU zwischen Threads, wesentlich weniger Overhead erzeugt als das switching von ganzen Tasks.

Wesentlicher Bestand der Inter Process Communication stellen die Ports dar, die bei Mach *„a kernel-managed finite-length message queue“ [Scheu02]* repräsentieren. Als Input Warteschlange ausgelegt, hören sie auf beliebig große Collections von

Datenobjekten. Geschützt werden die Ports dabei durch die so genannten Capabilities, die Objekten das Recht geben können, auf gewissen Ports zu schreiben oder eben nicht. Die IPC kann dabei synchron oder asynchron stattfinden. Bei der asynchronen Kommunikation werden Tasks durch Signals benachrichtigt, dass Messages auf sie warten, während bei der synchronen Variante die Nachrichten direkt weitergeleitet werden. Bei sehr großen Messages wird darüber hinaus aus Performance-Gründen der Speicherbereich des Senders, indem die Nachricht enthalten ist, in den virtuellen Speicherraum des Empfängers abgebildet und kann so direkt abgerufen werden.

1.2.2. Die Mikrokernel Konzepte des Mach

Obwohl eigentlich bei der Entwicklung des Mach auf die großen Vorteile eines Mikrokernel nicht so recht geachtet wurde, sind dennoch wesentliche Konzepte davon implementiert: zum Beispiel die Trennung zwischen Kernel und User Speicherbereich. So werden page-faults an einen eigenen „User-Space Paging Server“ weitergeleitet, der für die Seitenverwaltung verantwortlich zeigt. Auch netzwerktransparente IPC oder die Umleitung von Systemaufrufen zu „User-Mode-Tasks“ ist so in dieser frühen Phase der Mikrokernel bereits realisiert gewesen.

1.2.3. Mikrokernel der zweiten Generation

Auch wenn die ursprüngliche Idee des Mikrokernel im Laufe der Zeit sehr stark verwässert oder von Anfang an im Design nicht zu 100 Prozent berücksichtigt wurde, so ist sie dennoch nie untergegangen. Auch wurden manche Problembereiche erst durch die Erfahrungen im praktischen Einsatz deutlich.

Jochen Liedke *[Lied96]* beschreibt den Einsatz von Mikrokernel sogar als zwingend, da seiner Meinung nach eine solche Architektur grundlegende Vorteile bietet: Ein System, das diesem Konzept zur Gänze untergeordnet ist, ist stark modular, was höchste Flexibilität in Bezug auf Hard- aber auch auf Software und Erweiterbarkeit im Sinne eines „Baukastensystems“ garantiert. Nach Liedtke geht das sogar soweit, dass Fehlfunktionen von wesentlichen Teilen des Betriebssystems dadurch nicht notwendigerweise das ganze System zum Absturz bringen müssen. Er denkt auch anforderungsorientierte Adaptierungen des OS an, die zum Beispiel in unterschiedlichen Speicherverwaltungsmodulen, die in verschiedenen Servern parallel laufen, resultieren könnten. Darüber hinaus besteht kein Zweifel, dass ein kleiner, klar

strukturierter Kernel leichter wartbar ist und im Normalfall auch weniger Bugs enthält und somit größtmögliche Stabilität garantiert.
Die Mikrokernel der ersten Generation, von denen Mach die erfolgreichste gewesen ist, konnten eben beschriebene Vorteile freilich noch nicht lukrieren. Ein wichtiger Schritt allerdings war die Handhabung von Hardwareinterrupts durch IPC Messages, die vom Kernel erzeugt, aber vom user-level Gerätetreiber bearbeitet werden. Pferdefuß der frühen Mikrokernel war dennoch die eingeschränkte Performance bzw. die Kompromisse, die in der Architektur eingegangen wurden, um diesem Problem Herr zu werden. So entscheidend die IPC für das neue Konzept war, so dramatisch waren auch die leistungsmindernden Auswirkungen: Overhead im Ausmaß von etwa dem 10-fachen eines Standard UNIX-System-Calls. Ein zweites Problem stellte die zentralistische Speicherverwaltung sowohl des Kernel, als auch des User-Speicherraums durch den Kernel dar, die für Anwendungen zu wenig Flexibilität zuließ. Liedtke *[Lied96]* gibt als Beispiele dafür real-time Applikationen oder Multimedia File Server an, die die volle Kontrolle über ihren Speicherraum benötigen. Neuere Entwicklungen wie etwa der hardware-abhängige Exokernel entwickelt vom MIT oder der GMD L4 Kernel haben erfolgreich versucht durch tatsächliche Minimierung des Funktionsumfangs und einfache Architektur diese Probleme in den Griff zubekommen.

1.2.4. Ausblick

Generell ist zu sagen, dass das Konzept des Mikrokernel viele Vorteile gegenüber herkömmlichen monolithischen Kernel in Bezug auf Modularität und die damit verbundene flexible Erweiterbarkeit bzw. Anpassungsfähigkeit an unterschiedlichste Rahmenbedingungen (von Serverfarmen bis PDAs) bietet. Zur Zeit wird diese Entwicklung lediglich von Macintosh (Mac OS bedient sich des Mach 3.0 Kernel) und Linux ernsthaft aufgegriffen. Bei letzterem wird auch versucht unterschiedliche Kernel anzubieten, allerdings, wie in Linux üblich, nur nach Neukompilierung.
Nach Scheuermann *[Scheu02]* werden sich reine Mikrokernel Systeme dennoch hauptsächlich im Bereich der Embedded Systems durchsetzen können, da dort die idealen Vorraussetzungen für einen effizienten Einsatz gegeben sind: kleiner Speicher und begrenzte Prozessorleistung. Dazu kommt, dass solche Architekturen meistens von „unangenehmen“ Rückwärtskompatibiltätsanforderungen verschont sind und somit genau auf ihre Aufgaben zugeschnitten werden können.

2 Der QNX Mikrokernel

2.1. Einführung

Die Erfolgsgeschichte von QNX *[QNX02]* begann Anfang der 80er Jahre als ein kleines Team von IT-Entwicklern mit der Implementierung eines neuen Betriebssystems begann, das insbesondere für Anforderungen einer Echtzeitumgebung geschaffen werden sollte. Heute ist QNX einer der weltweit führenden Anbieter von Echtzeit-Mikrokernel-Betriebssystemen mit Kunden wie Cisco, Delphi oder Siemens.

QNX besteht dabei aus einem kleinen, effizienten Kernel mit einer Gruppe kooperierender Prozesse. Die Kommunikation zwischen den einzelnen Komponenten wird dabei durch einfaches MessagePassing realisiert.

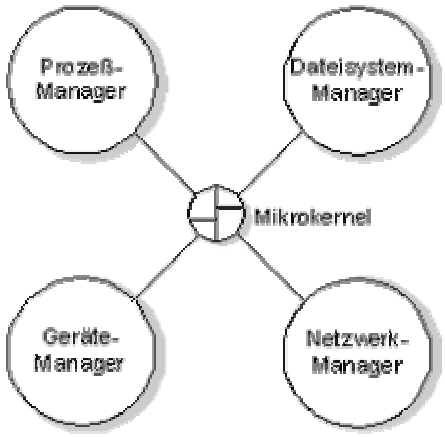

Abbildung 2 Der QNX Mikrokernel koordiniert die System Manager, Quelle: [QNX02]

Demnach ergeben sich zwei wesentliche Aufgaben dieses Kernels, *[QNX02]*:

- **Message Passing**: Routing aller Nachrichten im gesamten System
- **Scheduling**: Der Scheduler wird immer dann aufgerufen, wenn ein Prozess in Folge einer Nachricht oder eines Interrupts bestimmte Zustand ändert

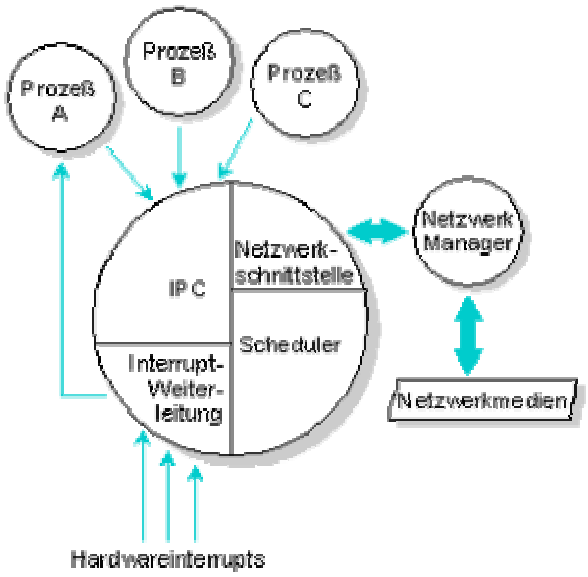

Abbildung 3 Inside the QNX Mikrokernel, Quelle: [QNX02]

Flexible Skalierbarkeit wird dem System durch Äquivalenz von System- und Anwenderprozessen verliehen, d.h. es existieren auf Seiten der Systemprozesse keinerlei Schnittstellen, die nicht auch für User-Prozesse zur Verfügung stehen. Erweiterungen der angebotenen Dienste sind somit als (selbstimplementierte) Anwenderprozesse zu verstehen, die im Wesentlichen das MessagePassing System des QNX Kernels verwenden, d.h. Anfragen (= Nachrichten) akzeptieren und darauf entsprechend reagieren können.

Als Beispiel wird in *[QNX02]* ein Datenbankserver angeführt, der ähnlich einem Dateisystem requests in Form von messages akzeptiert und Ergebnisse auf demselben Weg wieder zurückleitet. Dabei ist irrelevant ob dieser Prozess als Systemprozess oder Anwenderprozess deklariert ist und es ist weiters nicht nötig, Standardkomponenten zu modifizieren, um das System an spezifische oder sich ändernde Anforderungen anzupassen.

2.2. QNX Interprozesskommunikation (IPC)

Aufgrund der Architektur eines moderene OS und insbesondere bei Echtzeitbetriebssystemen kommt der Interprozesskommunikation besondere Bedeutung zu. Prozesse, ob System- oder Anwenderprozesse, laufen parallel, haben verschiedene oder sich überschneidende Aufgaben, nutzen aber jedenfalls ein gemeinsames Set an Ressourcen. Wie bereits angesprochen kommt zwar dem Kernel, der alle Informationen aller Prozesse hält, die Aufgabe zu, diese Ressourcen zu verwalten und den Prozessen zuzuordnen, aber bestehen Anwendungen meist aus mehreren Prozessen, die miteinander kommunizieren müssen. Dies ist bei QNX durch das genau so einfache wie leistungsstarke Message Passing System realisiert, das im wesentlichen Nachrichten, Proxies oder Signale weiterleiten kann und darüber hinaus zur Synchronisation einzelner Prozesse zur Verfügung steht.

2.2.1. IPC über Nachrichten

Nachrichten sind die fundamentale Form des IPC bei QNX und bieten synchrone Kommunikation zwischen Prozessen. Eine solche Message ist als Bytepaket zu verstehen, welches von einem Prozess zum anderen übertragen wird. Der sendende Prozess erwartet dabei obligatorisch die „Empfangsbestätigung“ und optional eine Antwortnachricht des empfangenden Prozesses. Wesentlich dabei ist, dass der Kernel keinerlei inhaltliche Prüfung der Message vornimmt, sondern lediglich für die korrekte Weiterleitung verantwortlich zeichnet. Es genügen drei Funktionen, die den Nachrichtenaustausch realisieren:

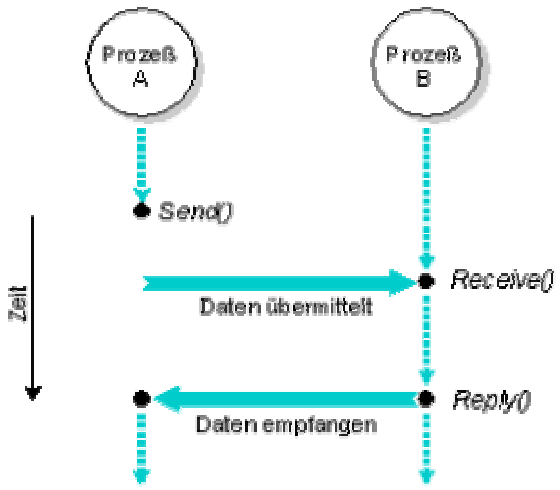

Abbildung 4 Das einfache Modell des Message Passing bei QNX, Quelle: [QNX02]

In Abbildung 3 bleibt Prozess A solange blockiert bis von Prozess B eine Antwortnachricht eingelangt ist. Reply-messages sind wie angesprochen optional. Sollte A keine Antwort erwarten, bleibt er dennoch blockiert bis B die „Recieve“ message abgesetzt hat. Dieser Ansatz erlaubt es Anwendungen, ihre Prozessabläufe auf einfache Weise zu synchronisieren, auch ohne Daten dabei auszutauschen, da auch Null-Byte Nachrichten erlaubt sind.

Im Allgemeinen ergeben sich dadurch folgende unterschiedliche Zustände für QNX Prozesse:

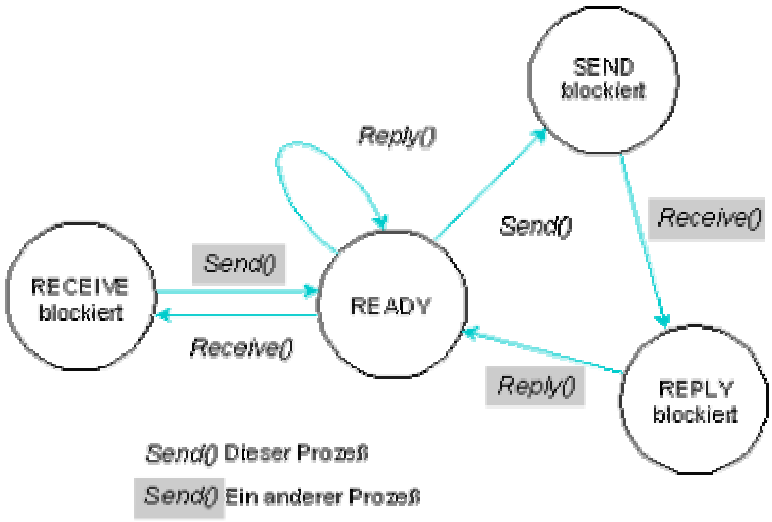

Abbildung 5 Die QNX Prozesszustände, Quelle: [QNX02]

Die häufigste Art der Interaktion wird als „sendegetriebene Kommuniktion“ bezeichnet. Sie geht davon aus, dass der sendende Prozess die Aktion initiiert und auf Antwort wartet, zB eine Clientanfrage an einen Server.

Seltener eingesetzt, aber dennoch möglich, ist die „antwortgetriebene Kommunikation“, wo beispielsweise ein Client dem Server seine Dienste anbietet und der Server bei Bedarf diese in Anspruch nimmt. Der Client ist dabei solange blockiert, bis der Arbeitsauftrag an ihn ergeht.

Im Allgemeinen ist wesentlich zu bemerken, dass im Kernel keine Kopien der Nachrichten gehalten werden, sondern diese beim Sender bleiben bis sie in einer Art „atomaren“ Aktion dem Empfänger übermittelt werden. Gleiches gilt für Antwortnachrichten. Der Vorteil dieses Systems liegt darin, dass der sendende Prozess keine Informationen über den Zustand des Empfängers haben muss, sondern lediglich solange blockiert wird, bis die Zustellung möglich ist. Prozesse haben darüber hinaus die Möglichkeit bei Anstehen mehrerer Nachrichten nicht nur nach dem First-Come-First-Server Prinzip die Nachrichten zu empfangen, sondern auch in Reihenfolge, die

den Prioritäten der Absender-Prozesse entspricht, was insbesondere für ein Echtzeit-Betriebssystem von entscheidender Bedeutung ist.

Darüber hinaus wird ein „bedingter“ Nachrichtenempfang unterstützt, der nichts anderes bedeutet als die Möglichkeit, für Prozesse auf die für sie wartenden Messages zu prüfen, ohne dabei selbst blockiert zu werden, bis die erste solche Nachricht tatsächlich eintrifft. Als Beispiel werden in *[QNX02]* Prozesse angeführt, die mit Geräten zu tun haben, die keine Interrupts absetzen können und somit durch eine Art polling auf Serviceanforderungen abgefragt werden müssen.

2.2.2. IPC über Proxies

Eine weitere Form der Kommunikation bei QNX bietet das Modell der Proxies, wobei diese mit nicht-blockierenden Nachrichten vergleichbar sind, die zwischen Prozessen ausgetauscht werden können. Diese Funktion dient dazu vorab definierte Nachrichten an einen Prozess weiterzuleiten ohne eine Antwort darauf zu erwarten oder entgegenzunehmen. In der Praxis wird eine solche Kommunikation hauptsächlich dann eingesetzt, wenn bestimmte Ereignisse eintreten oder Prozesse Daten versenden, für die sie keine Bestätigung erwarten, wie beispielsweise allgemeine Statusinformationen.

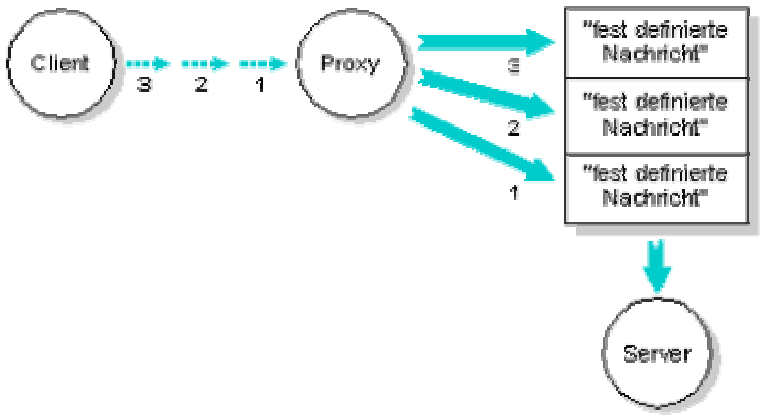

Abbildung 6 Ein Clientprozess löst hier eine Proxy 3 mal aus, Quelle: [QNX02]

2.2.3. IPC über Signale

Die Interprozesskommunikation über Signals ist an sich die herkömmliche Methode, die seit vielen Jahren in Betriebssystemen zum Einsatz kommt. Sie beruht darauf, dass Prozesse einfache Informationen oder „Befehle“ (aber keine Daten) untereinander austauschen können. Jeder Prozess kann dabei für sich entscheiden, ob er an ihn

gesandte Signale (mit Ausnahme einiger weniger, die zur Beendigung des Prozesses führen) ignoriert oder beispielsweise durch einen eigens installierten Signalhandler abfängt und entsprechend darauf reagiert. Grundsätzlich ist zu sagen, dass solche Botschaften erst an den Empfänger ausgeliefert werden, wenn dieser vom Prozess-Scheduler aktiv gesetzt worden ist. Darüber hinaus besteht keine Garantie über die Reihefolge bei möglichen mehreren anstehenden Signalen. Auch bestehen wesentliche Abhängigkeiten bzw. Interaktionen zwischen Signalen und Nachrichten.

Ein Beispiel: Angenommen Prozess A, der über einen Signalhandler verfügt, ist auf Grund eines Kommunikationsvorgangs durch Nachrichten mit Prozess B SEND- oder REPLY-blockiert und gleichzeitig wird ein Signal für ihn generiert. Als Reaktion auf dieses Signal wird sofort die Prozessblockierung aufgehoben, die Signalbearbeitung beginnt und die Methode Send() oder Recieve() gibt einen Fehler zurück. Wie aus Abbildung 3 zu erkennen ist, ergibt sich bei einer vorliegenden SEND-Blockierung kein Problem, da zu diesem Zeitpunkt Prozess B die Nachricht noch nicht empfangen hat. Sollte Prozess A allerdings REPLY-blockiert sein, also auf die Bestätigung der Verarbeitung der Message durch Prozess B warten, kann zum Zeitpunkt des Abbruchs A nicht verifizieren ob die gesendete Nachricht ordnungsgemäß behandelt wurde oder ob ein neuerliches Senden nötig ist. Die Lösung für dieses Dilemma liegt bei QNX in der Möglichkeit, dass ein Prozess, der sich wie ein Server verhält um Benachrichtigung bittet, wenn an einen REPLY-blockierten „Client“-Prozess Signale versandt werden. Technisch wird das durch eine SIGNAL-Blockierung dieses Prozesses realisiert, wobei der „Server“-Prozess zwei alternative Reaktionsmöglichkeiten hat: entweder beendet er die Originalanfrage bevor das Signal ausgewertet wird oder der Sender erhält eine explizite Fehlermeldung, dass seine Anfrage nicht bearbeitet wurde.

2.3. Prozess – Scheduling

Eine weitere Kernaufgabe des Mikrokernels von QNX besteht im Prozess-Scheduling. Der Scheduler entscheidet über die Reihenfolge, in der die Prozesse abgearbeitet werden. Im Grunde wird dazu ein einfaches Prioritätenverfahren eingesetzt, wobei jeder Prozess eine Priorität besitzt, die im Standard vom „Vaterprozess“ geerbt wird. Der Prozess mit der höchsten Priorität bekommt die nächste Zeitscheibe. Zusätzlich hat der Scheduler jederzeit die Möglichkeit, einen laufenden Prozess zu stoppen und einem anderen die nächste CPU-Zeitscheibe zuzuteilen. Dies geschieht nach *[QNX02]* immer dann, wenn

- eine Prozessblockierung aufgehoben wird
- die Zeitscheibe für einen laufenden Prozess abläuft
- ein höher-priorisierter Prozess in den READY Zustand kommt (vgl. Abb. 4)

Je nach Anforderung der Applikation wird im QNX Scheduler FIFO-, Round-Robin oder adaptives Scheduling der konkurrierenden Prozesse angewandt. Welches Verfahren tatsächlich verwendet wird entscheidet der Prozess selbst. Auch hier gilt: sollte nichts anderes festgelegt sein, erbt er auch dieses Attribut vom Vaterprozess.
Die Schedulingverfahren im Vergleich sind:

- **FIFO (mit Prioritäten)**: ein Prozess setzt seine Ausführung solange fort bis er entweder die Kontrolle freiwillig abgibt oder von einem Prozess höherer Priorität verdrängt wird
- **Round-Robin**: im Grunde gleich wie das FIFO Verfahren, allerdings gibt ein Prozess, der das Round-Robin Scheduling nutzt auch dann die Kontrolle ab, wenn seine Zeitscheibe abgelaufen ist
- **Adaptives Scheduling**: um möglichst große Ausgewogenheit im Scheduling zu erreichen wird beim adaptiven Scheduling die Priorität eines Prozesses, dessen Zeitscheibe abgelaufen ist, um 1 erniedrigt. Dies wird in *[QNX02]* als *„priority decay (Prioritätsverfall)“* bezeichnet. Entscheidend ist dabei, dass dieser priority decay maximal eine Prioritätsstufe betragen kann und der Prozess immer dann seine ursprüngliche Priorität zurückbekommt, wenn er blockiert. Adaptives Scheduling wird hauptsächlich eingesetzt in Systemen, wo gleichzeitig rechenzeitintensive Background-Jobs und interaktive

Benutzerapplikationen laufen, und ist somit der Standardalgorithmus für die meisten Anwendungen.

3 Windows 2000/XP

3.1. Einführung

Ende der der 80er Jahre entschied sich Microsoft für einen neuen Weg. Bis zu dieser Zeit nämlich kooperierte der Weltkonzern mit IBM bei der Entwicklung des OS/2 Betriebssystems, das in Assembler für Single-Prozessor-Maschinen geschrieben wurde. Einer der Gründe für diesen Neustart war der Wunsch, ein portables Betriebssystem zu schaffen, das nicht nur das OS/2 API unterstützt, sondern auch das POSIX Interface. Als schließlich 1988 Dave Cutler bei MS verpflichtet wurde, war dies die Geburtsstunde der „new Technology", Windows NT. Obwohl zu Beginn der Entwicklungen noch das OS/2 API als nativ-API Verwendung fand wechselte man auf Grund des Erfolgs von Windows 3.0 auf das Win32 API, das bis heute den Kern der MS Betriebssystem darstellt.

3.2. Designziele

Bei der Entwicklung von Windows XP bzw. Windows 2000 ging Microsoft beim Design auf verschiedensten Ebenen völlig neue Wege. Ziel war es ein neues, stabiles und vor allem performantes Betriebssystem zu schaffen, das beispielsweise Portabilität nicht mehr auf mehrere, unterschiedlichste Architekturen bezieht: sondern man spezialisierte sich auf Intel, was solchen System den Beinamen „Wintel" einbrachte. Im Allgemeinen sind nach *[Sil02]* sechs unterschiedliche Designziele zu unterscheiden: Erweiterbarkeit, Portabilität im bereits angesprochenen Sinn, Verlässlichkeit, Kompatibilität, Performanz und als wichtiges Kriterium, um weltweit erfolgreich zu sein, „international support".

3.2.1. Erweiterbarkeit

Erweiterbarkeit ist wahrscheinlich eines der Schlüsselkriterien in Zeiten einer immer kürzeren Halbwerts- bzw. Einsatzzeit von modernen Betriebssystemen. Im Wesentlichen kann sie als das Potential eines OS zur Anpassung an neue Technologien oder Marktsituationen betrachtet werden. Bei Windows 2000/XP wurde dieser Anforderung durch ein Schichtenmodell in der Architektur gerecht. So wird die Ausführungsschicht im so genannten Kernel- oder protected-Mode ausgeführt, die ihrerseits den darüber liegenden Schichten gewisse Grundfunktionalitäten zur

Verfügung stellt. Alle „höheren“ Server-Subsysteme laufen im „User-Mode“. Aus Gründen der Abwärtskompatibilität gehören zu diesen auch wie es Silberschatz *[Sil02]* nennt: *„environmental subsystms“* für MS-DOS, Windows 3.x und POSIX. Es kann schon erahnt werden, dass auf Basis dieser modularen Struktur von WinXP es relativ leicht fällt, beliebige Erweiterungen der Subsysteme vorzunehmen, ohne dabei die wesentlichen Teile des Gesamt-OS antasten zu müssen. Einen weiteren Fortschritt bieten die so genannten „ladbaren-Treiber“, die die Eigenschaft besitzen, während des laufenden Betriebs, sozusagen „on-the-fly“, hinzugefügt werden zu können. Dieser Meilenstein hat ganz besonders auf die Benutzerfreundlichkeit sehr positive Auswirkungen, da nun Änderungen der Hardware- bzw. Netzwerkkonfiguration nicht notwendigerweise zum Neustart des Systems führen (vgl. auch: plug and play).

3.2.2. Portabilität

Im Allgemeinen ist die Portabilität eines Betriebssystems zu verstehen als die Eigenschaft, es von einer Architektur auf eine andere mit möglichst geringen Änderungsnotwendigkeiten transportieren zu können. Wie auch in UNIX ist bei Windows XP der Großteil in C bzw. C++ implementiert und bei letzterem ist sämtlicher architekturabhängiger Code in der „Hardware Abstraction layer (HAL)“ zusammengefasst, technisch implementiert durch eine DLL (Dynamic Link Library). Durch diesen Indirektionsschritt zwischen Hardware und Applikationen ist es nun möglich, unabhängig von der darunter liegenden Hardware Anwendungen zu entwickeln, die sich lediglich einem standardisierten Interface, bereitgestellt durch die HAL, bedienen. Bei Portierungen des OS auf andere Architekturen muss demnach nur die HAL-DLL adaptiert werden. Die strikte Einhaltung dieses Designprinzips führt in letzter Konsequenz allerdings dazu, dass völlige Abwärtskompatibilität zu älteren Betriebsystemen nicht mehr besteht, denn MS-DOS zum Beispiel hatte die Möglichkeit direkt auf Hardware zuzugreifen, was unter WinXP nicht mehr möglich ist. Als work-around bleibt dann nur die Emulation als virtuelle Maschine.

3.2.3. Kompatibilität

Windows XP bietet wie Windows 2000 source-level Kompatibilität zu Applikationen, die dem IEEE 1003.1 Standard entsprechen (POSIX), was nichts anderes bedeutet, als dass Anwendungen, die dieser Vorgabe folgen, lediglich kompiliert werden müssen, um unter WinXP zu laufen, aber keine zusätzlichen Änderungen notwendig sind. Wie

bereits oben angeführt ist es kompatibel zu einer Reihe von Programmen, die für Intel X86 Systeme implementiert wurden, wie zB MS-DOS, OS/2, oder auch LAN Manager bzw. eine Vielzahl von unterschiedlichen Dateisystemen (DOS FAT, OS/2 HPFS, etc.).

3.2.4. Performanz

Als eines der wesentlichen Designprinzipen ist sicherlich eine weitere Steigerung der Performanz zu sehen. Was bei Windows NT damit begonnen hatte, die Benutzerschnittstellen und den Code für die Grafikausgabe in den Kernel-Mode zu transferieren, um damit (allerdings auf Kosten der Verlässlichkeit) Geschwindigkeitsgewinne zu erzielen, wird bei WinXP noch weiter gesteigert. Mit Hilfe einer schlanken und äußerst schnellen local-procedure-call routine, durch die ein effizientes Message Passing System die einzelnen Subsysteme miteinander verbindet, werden die Aufgaben der Ausführungsschicht intelligent bewältigt. So können (mit Ausnahme des Kernels) alle Threads durch höher priorisierte unterbrochen werden, was dem System die Möglichkeit gibt, auf externe Ereignisse (zB I/O Interrupts) rasch zu reagieren. Dazu kommt, dass alle signierten Gerätetreiber ebenfalls im Kernel-Mode laufen und dadurch ebenfalls zu Performanzsteigerung beitragen.

3.2.5. National Language Support (NLS)

Stärker als je zu vor ist Windows XP auf den Einsatz in unterschiedlichsten Regionen dieser Erde vorbereitet. Durch das NLS API werden spezielle Routinen angeboten, um regional unterschiedlichen Anforderungen gerecht zu werden. So können Datum, Zeit, und Währungssymbole, um nur einige Beispiele zu nennen, einfach und unkompliziert adaptiert werden. Wesentlich für Programmierer ist auch die Möglichkeit eines einfachen Stringvergleichs zwischen verschiedenen character-sets, die intern in UNICODE konvertiert und dann erst verglichen werden.

3.3. Systemkomponenten

Wie bereits angesprochen ist Windows 2000/XP als Schichtenarchitektur implementiert. Im Wesentlichen unterscheidet man zwischen der HAL (Hardwar Abstraction Layer), der Ausführungsschicht und dem eigentlichen Kernel, die alle im „procteded-Mode“ arbeiten, also in geschützten Speicherbereichen, sowie einer Reihe von Subsystemen, die im „User-Mode“ laufen, d.h. die den ihnen zugeteilten Speicher selbst verwalten.
Klar getrennt sind Subsysteme, die andere Betriebsysteme emulieren (zB das POSIX oder MS-DOS Subsystem) und solche, die Security Aufgaben übernehmen (zB der Logon process).

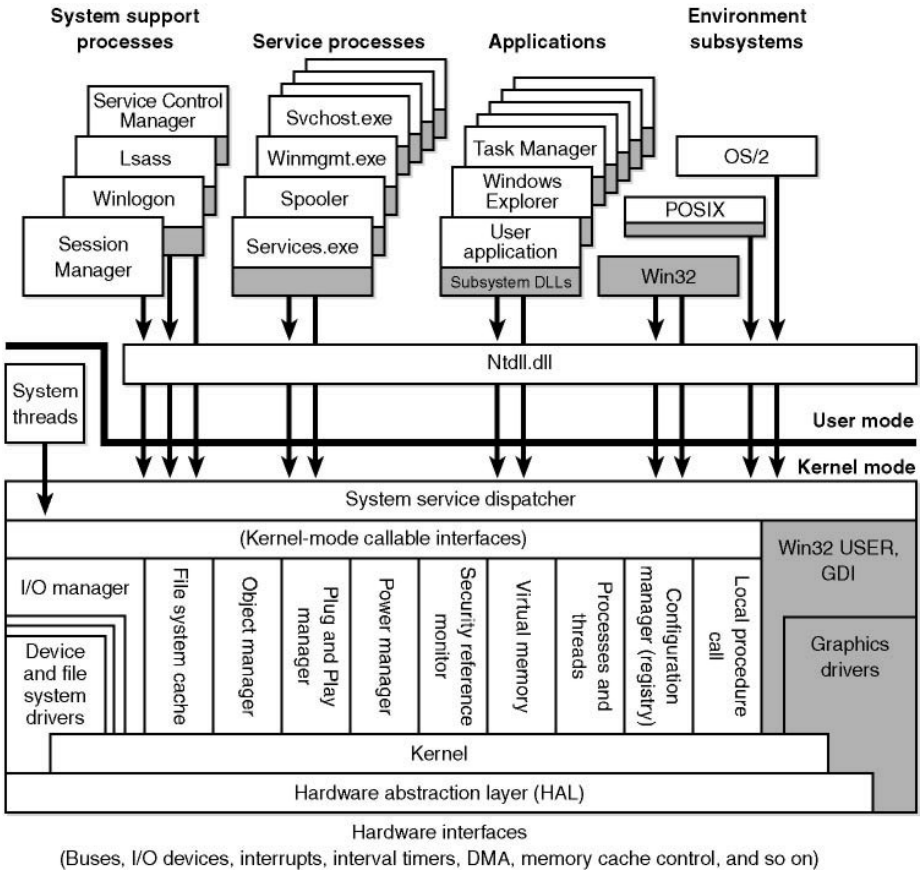

Abbildung 7 Die Windows 2000 Architektur, Quelle: [Sol00]

Der Große Vorteil dieser Architektur liegt ganz klar in der sich ergebenden, relativ einfachen Kommunikation zwischen den einzelnen Komponenten. Im Folgenden Kapitel wird genauer auf den eigentlichen Kernel und die darüber liegende Ausführungsschicht eingegangen.

3.3.1. Der Kernel

Der Kernel in Win2000/XP kann als zentrale, immer verfügbare Kommunikationsschnittstelle für die einzelnen Module der Ausführungsschicht verstanden werden, dessen Hauptaufgaben nach *[Sol00]* in den Bereichen Thread Scheduling, Interrupt und Exceptionhandling, low-level Prozessor Synchronisation und Recovery nach einem Systemausfall zu finden sind. Als zentrale Anlaufstelle für sämtliche Anfragen ist der Kernel der einzige Teil des Betriebssystems, der nie aus dem Speicher ausgelagert und dessen Ausführung nie unterbrochen wird.

Um seine Aufgaben wahrnehmen zu können bedient sich der Windows2000/XP Kernel einiger spezieller Kernel Objekte, deren Attribute die nötigen Daten und deren Methoden die Funktionalität darstellen. Diese Bemerkung lässt bereits darauf schließen, dass der Kernel objektorientiert implementiert. Ein Objekt in Windows 2000/XP entspricht daher einer konkreten Ausprägung eines speziellen Objekttyps, der wiederum mit Attributen und Methoden ausgestattet ist. Der Kernel verwendet im Wesentlichen zwei Mengen von solchen Objekten: Dispatcher- und Controlobjects.

3.3.1.1. Dispatcher und Control Objects

Die Dispatcher Objekte kontrollieren und steuern das Dispatching, also die Verteilung von Nachrichten zu den entsprechenden Objekten, und sind zuständig für die gesamte Synchronisation innerhalb des Systems. Als Beispiele für solche Objekte sind in *[Sil02]* events, mutants, mutexes, semaphoren, threads oder auch timer genannt. Event Objekte registrieren events und synchronisieren diese wenn nötig. Mutant Objekte werden herangezogen um mutual exclusion sowohl im Kernel-Mode als auch im user-mode zu garantieren. Einen Schritt weiter gehen die mutex objects, die zwar nur im kernel-mode angeboten werden, aber zusätzlich zur mutual exclusion auch noch Deadlockfreiheit versichern. Im Unterschied dazu agieren die Semaphore-Objekte lediglich als „Zählstelle“ für Threads, die auf eine bestimmte Ressource zugreifen wollen. Thread-Objekte selbst sind immer assoziiert mit einem Prozess und werden vom Kernel koordiniert (siehe folgender Abschnitt). Timer Objekte nehmen in einem preemtive-multitasking OS wie Windows2000/XP eine besondere Stellung ein, da sie verbrauchte CPU Zeit monitoren und wenn nötig auf TimeOuts oder abgelaufene Zeitscheiben hinweisen und damit eigentlich erst diese Form eines Betriebssystems ermöglichen.

3.3.1.2.Threads und Scheduling

Wie viele andere moderne Betriebssysteme auch basiert Windows2000/XP auf einer Prozess/Thread Architektur, genauer gesagt einem *„priority-driven, preemptive scheduling system“ [Sol00]*, wobei jedem Prozess ein oder mehrere Threads, ein virtueller Adressraum und zusätzliche Informationen, wie etwa bei Multiprozessorsystemen der zugehörige Prozessor, zugeordnet sind. Aufgabe des Kernels ist es nun, der Gesamtheit der Threads aller unterschiedlichen Prozesse, nach einem bestimmten Prioritätssystem, CPU Zeit zuzuordnen.

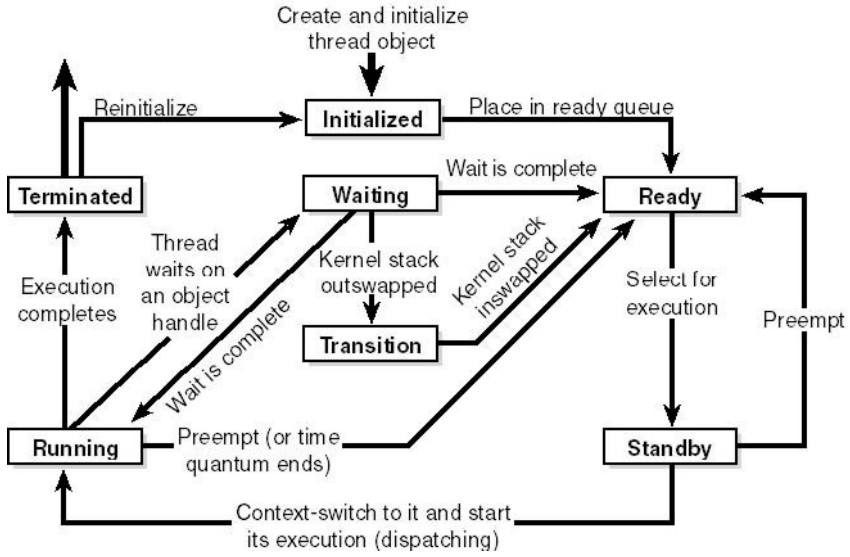

Abbildung 8 Die verschiedenen Thread-States, Quelle: [Sol00]

Dabei können Threads folgende Zustände annehmen:

- **Initialized**: Dieser Zustand wird lediglich intern verwendet während ein Thread erzeugt wird
- **Ready**: Alle in diesem Zustand befindlichen Threads werden vom Dispatcher herangezogen, um den nächsten auszuführenden Thread zu bestimmen
- **Standby**: Ist schließlich ein Thread ausgewählt, kommt dieser in den Standby-Zustand bis alle Bedingungen erfüllt sind, um nach dem Context-Switch den Thread auszuführen. Bei Mehrprozessorsystemen kann es pro Prozessor nur einen Thread in diesem Zustand geben.
- **Running**: In diesem Stadium läuft der Thread und führt seine Aktionen durch bis entweder seine Zeitscheibe abgelaufen ist oder der Kernel diesen unterbricht

(Grund dafür kann beispielsweise eine höher priorisierter Thread sein, der in den Ready-Zustand gelangt).

- **Waiting**: Threads können aus verschiedenen Gründen von Running in den Wait-Zustand versetzt werden: zum Beispiel dann, wenn ein Thread auf die Ergebnisse einer I/O Anfrage wartet oder wenn Synchronisation mit anderen Objekten erforderlich ist oder aber auch ist es Subsystemen direkt möglich Threads zum „Waiting“ aufzufordern. Je nach Priorität des Threads kann dieser anschließend wieder „Running“ werden oder zurück in den Ready Status versetzt werden.
- **Transition**: ist dann der Fall, wenn während ein Thread „Waiting“ ist, seine Speicherseiten aus dem Kernel-Speicher ausgelagert werden. Beim „Erwachen“ werden die entsprechenden Pages wieder eingelagert und der Thread wechselt in den „Ready“ Status.
- **Terminated**: ist der Zustand nach Beendigung der Ausführung eines Threads. Das Thread Objekt selbst wird danach (entsprechend der Policy des Object Managers) aus dem Speicher entfernt, außer es wird durch die Anwendung reinitialisiert.

Beim Scheduling der einzelnen Threads kommen dabei 32 Prioritätsstufen zur Anwendung, wobei die Stufen 1-15 der so genannten „variable class“ und 16-32 der „real-time class“ zugeordnet sind. Der Dispatcher besitzt für jede Stufe eine eigene Queue, in die die entsprechenden Threads eingehängt werden und entscheidet, welcher Thread gestartet wird. Sollte ein Thread auf eine bestimmten, zur Zeit nicht verfügbaren, Prozessor warten wird dieser einfach ignoriert bzw. sollte überhaupt kein Thread „Ready“ sein wird der „Idle-Thread“ ausgeführt.

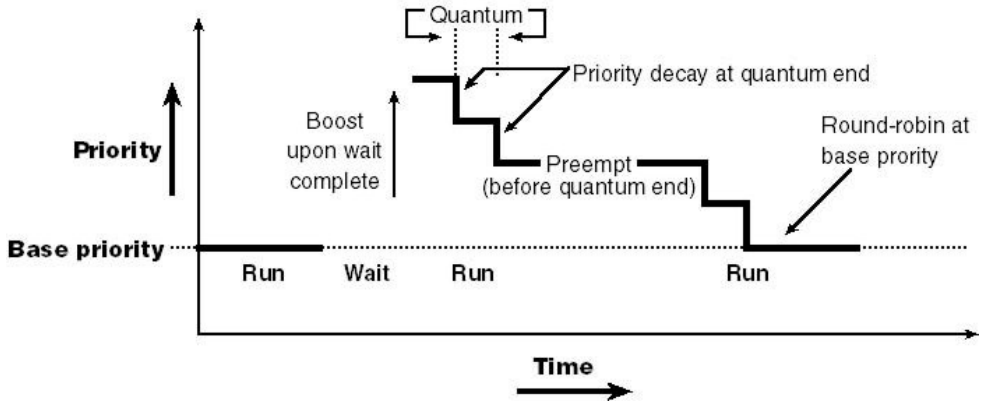

Abbildung 9 Priority boost und decay in Windows2000/XP, Quelle: [Sol00]

Die „variable class“ bezieht ihren Name aus der Tatsache, dass hier die Prioritäten, die Threads zugeordnet sind, variieren. Um einerseits Starvation einzelner Threads zu vermeiden, aber auch andererseits exzellente Antwortzeiten zu erzielen, weicht die Prioritätsstufe von der zu Beginn festgelegten Basisstufe wie folgt ab: verlässt ein Thread den „Waiting“-Zustand wird seine Priorität gesteigert, je nach Wichtigkeit dessen auf das gewartet wird. Das Spektrum der Anhebung reicht hier von minimal bis exorbitant. Insgesamt unterscheidet *[Sol00]* fünf Fälle, in denen solche „priority boost“ auftreten:

- Nach einer I/O Operation (1 {Disk} bis 8 {Sound} Stufen)
- Nachdem auf Semaphoren oder events gewartet wurde (1 Stufe)
- Nach Beendigung eines „Waits“ von Vordergrundprozessen (0 bis 2 Stufen)
- GUI Threads bei Aktivität (2 Stufen)
- Wenn Starvation zu verhindern ist (für 2 Zeitscheiben auf Stufe 15)

Ein Thread, der beispielsweise auf eine Tastatureingabe wartet, wird einen entsprechend starken Prioritätszuwachs erlangen, hingegen wird ein CD-ROM Zugriff niedriger bewertet. Diese Strategie neigt im Allgemeinen dazu Anwortzeiten von interaktiven Threads zu verkürzen, während „Backgroundjobs“ tatsächlich im Hintergrund laufen. Das entspricht somit dem Bild eines modernen, benutzerorientierten Betriebssystems, das die Interaktion mit dem User in den Vordergrund stellt. Nun können selbstverständlich Prioritäten nicht nur erhöht werden, sondern müssen auch sinken, was immer bei Ablauf einer Zeitscheibe für einen Thread geschieht. Zu diesem

Zeitpunkt wird seine Priorität um eine Stufe erniedrigt bis die Basispriorität erreicht ist, allerdings nicht tiefer.

Threads, die in die Gruppe der „real-time class“ fallen haben selbstverständlich höchste Priorität und kommen sofort nachdem sie den „Ready“ Status erreicht haben zur Ausführung, indem jeglicher laufender Thread unverzüglich unterbrochen wird. Auch wenn durch alle diese Maßnahmen eine optimale CPU Auslastung und möglichst komfortable Antwortzeiten erzielt werden, kann bei Windows2000/XP nicht von einem Echtzeitbetriebssystem gesprochen werden, da nicht garantiert werden kann (und dies ist Kern der Definition eines Echtzeit-OS), dass ein Echtzeitthread innerhalb fester Zeitschranken ausgeführt und beendet wird.

3.3.1.3. Trap Dispatching - Exceptions und Interrupts

Exceptions und Interrupts sind Ereignisse in einem Betriebssystem, die den Prozessor veranlassen, vom “normalen” Betrieb abzuweichen und durch fix vorgegebene Verfahren diesen Systemzustand zu behandeln, vergleichbar mit dem Eintreten eines unvorhergesehen Ereignisses, zB ein unvorhergesehener Kurswechsel oder ein Triebwerksausfall im Flugzeugcockpit, worauf die Piloten mit standardisierten Checklisten oder Vorgehensweisen reagieren. Sowohl durch Hard- als auch Software ausgelöst, werden solche Traps durch trap handler bearbeitet (vgl. Abbildung 9), die in *[Sol00]* beschrieben werden als *„…a processor’s mechanism for capturing an executing thread when an exception or an interrupt occurs and transfering control to a fixed location in the operating system“*.

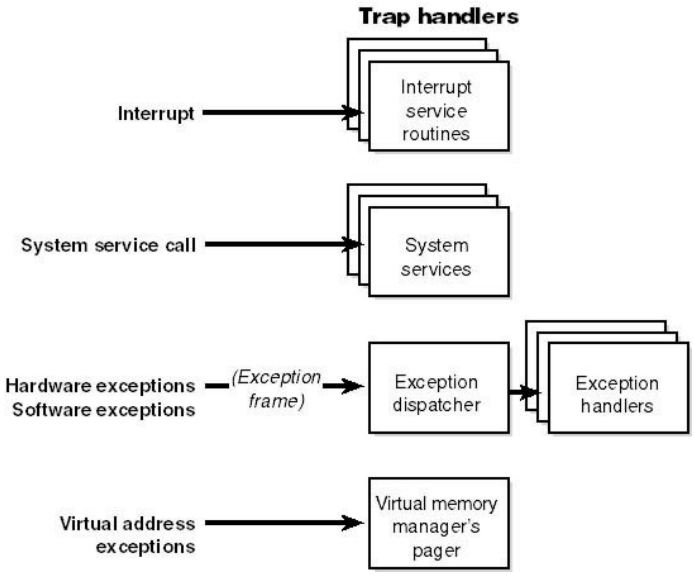

Abbildung 10 Windows2000/XP Trap handler, Quelle: [Sol00]

Je nach Art des Traps werden also unterschiedliche trap handler mit der Aufgabe betraut, diese Ausnahmesituation zu bewältigen. Der Kernel an sich unterscheidet zwischen Interrupts und Exceptions. Interrupts stellen einen asynchronen Event dar, der zu jeder Zeit auftreten kann und in keiner Weise mit dem soeben ausgeführten Code in Verbindung stehen muss (Triebwerksausfall). Als Beispiele seien hier hauptsächlich I/O Interrupts genannt. Im Gegensatz dazu sind Exceptions zu sehen, die in direktem Zusammenhang mit der Ausführung eines Threads bzw. einer einzelnen Anweisung stehen und damit mehr oder weniger als synchron bezeichnet werden können (unvorhergesehener Kurswechsel). Ein weiterer Unterschied besteht darin, dass Exceptions zwar von Hard- und Software hervorgerufen werden können, aber reproduzierbar sind, zB Bus Fehler (Hardware) oder divide-by-zero Fehler (Software).

Exception Dispatching

Obwohl einfache Exceptions von trap handlern direkt bearbeitet werden, ist es doch die Mehrzahl, die zum so genannten Exception Dispatcher weitergereicht wird. Der trap handler generiert und sichert den „trap frame“, der alle wesentlichen Informationen des aktuellen Threads enthält (um zu einem späteren Zeitpunkt die Ausführung an dieser Stelle weiterzuführen) und reicht den „exception record“ mit Ursache und Ort der Ausnahme an den Exception Dispatcher weiter. Seine Aufgabe liegt nun darin einen Handler zu finden, der mit Ausnahmen umgehen kann. Sollte die Exception im Kernel-Mode aufgetreten sein, wird lediglich eine Routine zur Suche des entsprechenden Exception handlers aufgerufen, die im Regelfall auch erfolgreich abgeschlossen wird. Wenn dem allerdings nicht so ist, was eine nicht-abgefangene Ausnahme im Kernel-Mode bedeutet, dann wird das System sofort angehalten und es kommt zum berüchtigten „Blue-Screen“.

Wesentlich komplizierter stellt sich das Exception Dispatching im User-Mode dar. Es existiert daher eine ausgefeilte Strategie zur Behandlung solcher Situationen.

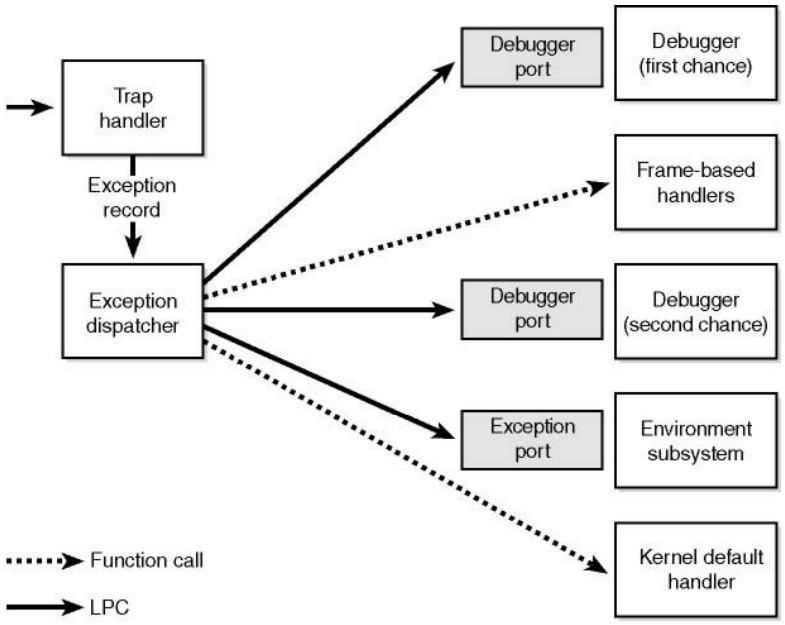

Abbildung 11 Exception Dispatching im User Mode, Quelle: [Sol00]

Die Schwierigkeit ergibt sich daraus, dass es den environmental subsystems (wie POSIX oder WIN32) möglich ist, bei ihren Prozessen einen Debugger Port bzw. eine Exception Port zu registrieren, die in der Folge einer Exception mit Nachrichten vom Kernel versorgt werden: sollte also ein debugger port existieren, wird die exception zunächst an diesen weitergeleitet und versucht eine Ausnahmefallbehandlung durchzuführen. Schlägt dieser Versuch fehl, ist der zweite Schritt, in den User-Mode zu wechseln und dort einen Exception Handler zu finden. Wenn auch das scheitert bekommt der Debugger Port – zurück im Kernel Mode – die Ausnahme erneut, um sie dem Debugger zu übergeben. Sollte jener, aus welchen Gründen auch immer, nicht laufen, wird die Exception an den Exception Port gesandt, was lediglich dem Zweck dient, dem entsprechenden Subsystem die Möglichkeit zu bieten, die Ausnahmemeldung zu übersetzen (zB POSIX: Windows Nachrichten werden in POSIX Signale übersetzt bevor sie dem Thread weitergeleitet werden, der die Ausnahme verursacht hat). Sollte alle Versuche fehlschlagen, die exception zu behandeln, wird der Default Handler des Kernels aufgerufen, der den entsprechenden Prozess einfach terminiert.

Interrupt Dispatcher

Interrupts, die zu meist von Hardware generiert werden, setzen den Prozessor davon in Kenntnis, dass sie Dienste in Anspruch nehmen wollen (zB I/O Geräte wie Maus,

Tastatur, etc) und werden im Kernel vom Interrupt Dispatcher bearbeitet. Dafür stehen zwei Möglichkeiten zur Verfügung: entweder existiert für den generierten Interrupt eine eigene ISR (Interrupt Service Routine wie beispielsweise ein Gerätetreiber) oder eine interne Kernelroutine übernimmt die Aufgabe. Der Interrupt wird dabei durch ein eigenes interrupt object repräsentiert, das jegliche Information zur korrekten Behandlung beinhaltet. Vorteil dieser Methode ist, dass damit unabhängig von der Hardware, die den Interrupt ausgelöst hat, Interrupts den entsprechenden ISRs zugeordnet werden können. Weil unterschiedliche Prozessorarchtikturen aber verschiedene Typen von Interrupts kennen, werden diese in Windows2000/XP auf eine fixe Zahl von 32 Stufen abgebildet, den so genannten „interrups levels“ (IRQLs). Wie auch beim Exception Dispatcher sind diese priorisiert, die durch die höhere Nummer repräsentiert werden. Der Kernel selbst behält sich 8 davon für interne Zwecke vor. Die restlichen 24 stehen explizit für Hardware Interrupts zur Verfügung (siehe Abbildung 11).

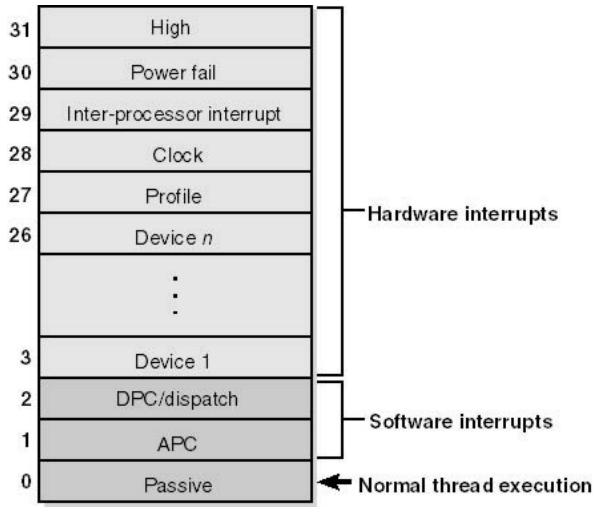

Abbildung 12 Die Interrupt Stufen in Win2000/XP, Quelle: [Sol00]

Die Kernel Interrupts im Detail:

- **High**: Wird nur beim Anhalten des Systems verwendet, um alle anderen Interrupts zu maskieren
- **Power fail**: Ist Spezifiziert in WinNT, wird aber nie verwendet
- **Inter-processor**: In Multiprozessorumgebungen wird dieser Interrupts verwendet, um einen anderen Thread um eine Aktion zu bitten (zB einen bestimmten Thread ausführen)

- **Clock**: wird für die Systemzeit und für die Zeitscheibenvergabe benutzt
- **Profile**: steht für Performanzmessungen zur Verfügung
- **DPC** (Deferred Procedure Call)/dispatch und **APC** (Asynchron Procedure Call): Software Interrupts die vom Kernel zum Context-Switching zwischen Threads verwendet werden bzw. von Gerätetreibern erzeugt werden
- **Passiv**: ist eigentlich kein Interrupt, sondern der Default für die laufende Thread-Execution

DPC/dispatch

Wie bereits angedeutet wird der Dispatch Interrupt zur Kontrolle des Thread Context Switching benutzt. Ein Beispiel: Aufgrund eines Hardware Interrupts (Annahme: Tastatur) wird der IRQL des Prozessors auf eine Stufe über dem dispatch-level angehoben. Gleichzeitig erkennt der Kernel, dass ein Thread dispatch notwendig ist und generiert deshalb einen dispatch-interrupt. Dieser ist allerdings auf Grund der momentanen Stufe des IRQL blockiert und wird erst nach Abarbeitung des Tastatur-interrupts behandelt, dh der Dispatcher wählt einen neuen Thread zur Bearbeitung durch die CPU aus.

Derselbe Mechanismus findet bei den Deferred Procedure Calls immer dann Anwendung, wenn der Kernel entscheidet, eine zeitunkritische Funktion müsse angestoßen werden. In diesem Fall wird ein DPC Objekt mit der Adresse der auszuführenden Funktion erzeugt und eine DPC-interrupt ausgelöst. Diese sind von der Priorität zwar höher als User-Threads (und deshalb von der Struktur her einfach gehalten), dennoch kommen sie erst dann zur Bearbeitung, wenn alle Hardware Interrupts beendet sind.

4 Des WeLearn System und Java VM

4.1. Einleitung

Ziel dieser Arbeit ist es, die Anforderungen an einen Kernel einer webbasierten, objektorientierten e-Learning Plattform im Vergleich zu herkömmlichen Betriebssystemen darzustellen. Im vorherigen Abschnitt wurden Aufgabe und Funktionsweise des Windows2000/XP Kernels erläutert. Im Folgenden sollen an Hand der am Institut für Informationsverarbeitung und Mikroprozessortechnik (FIM) der Universität Linz entwickelten e-Learning Plattform WeLearn die angesprochenen Unterschiede deutlich gemacht werden.

Dazu ist es aus meiner Sicht nötig, zu Beginn einen kurzen Überblick über das WeLearn-Framework aus einer High-Level Perspektive zu geben, die als Orientierung dienen soll bevor Details betrachtet werden.

4.1.1. WeLearn ist…

In *[Div02]* wird WeLearn beschrieben als *„…eine universell einsetzbare Plattform, die für diverse Kurse und Kursanbieter angepasst werden kann, so dass Kurse vollständig oder größtenteils online abgehalten werden können… Mit dem WeLearn System werden sowohl die Administration der Kurse, des Kursmaterials und der Kursteilnehmer als auch die eigentliche Kursabwicklung durchgeführt. Dazu wird dem Lernenden eine intuitiv benutzbare Lernumgebung geboten werden, die ihm den Zugriff auf alle relevanten Kursdaten... ermöglicht und ihm Kommunikationsmittel (wie Diskussionsforen, Nachrichten, Up- / Download, ...) zur Verfügung stellt.“ [Div02]*.

Wie hier bereits erkennbar ist, handelt es sich bei WeLearn um die technische Unterstützung für die Realisierung eines neuen Lern-Paradigmas, das Lernende am Weg zum selbstgesteuerten Lernen begleiten und motivieren soll. Darüber hinaus unterstützt die Plattform für Kursmaterialien den CPS (Content Packaging Standard) des IMS Global Consortium und bietet daher die Möglichkeit der Einbindung bestehender, standardisierter Unterlagen ohne auf die spezielle Implementierung des Systems Rücksicht nehmen zu müssen. Dazu kommt, dass auf Grund tiefgreifender technischer Weiterentwicklungen in WeLearn Release 2, die Performanz der Darstellung und Navigation durch solche Kurse erheblich gesteigert werden konnte. Es geht hier also um eine Orientierung an neuen Anforderungen im Bildungssektor, die orts- und

zeitunabhängig einen starken Fokus auf den Lernenden und dessen Bedürfnisse richtet und gleichzeitig die Wissensvermittlung als eine Holschuld darstellt.
Kurz gefasst ist WeLearn:

- **Benutzerfreundlich** durch intuitive Handhabung
- Für beliebige Lernszenarien **adaptierbar**
- **Flexibel und skalierbar** als Ergebnis extensiver Benutzerorientierung
- **Plattformunabhängig** auf Grund der Java Technologie

Der bereits angesprochene CPS Standard gibt die Struktur von Lehrmaterialien in Form des so genannten „Manifests“ vor. In einem XML File werden hier die einzelnen Kursunterlagen (Ressourcen) in Beziehung gesetzt zu einer Art Inhaltsverzeichnis (Organization), das der Dezimalklassifikation folgen kann und sich damit auf einfache aber einheitliche Art und Weise Lehrveranstaltungen abbilden lassen.

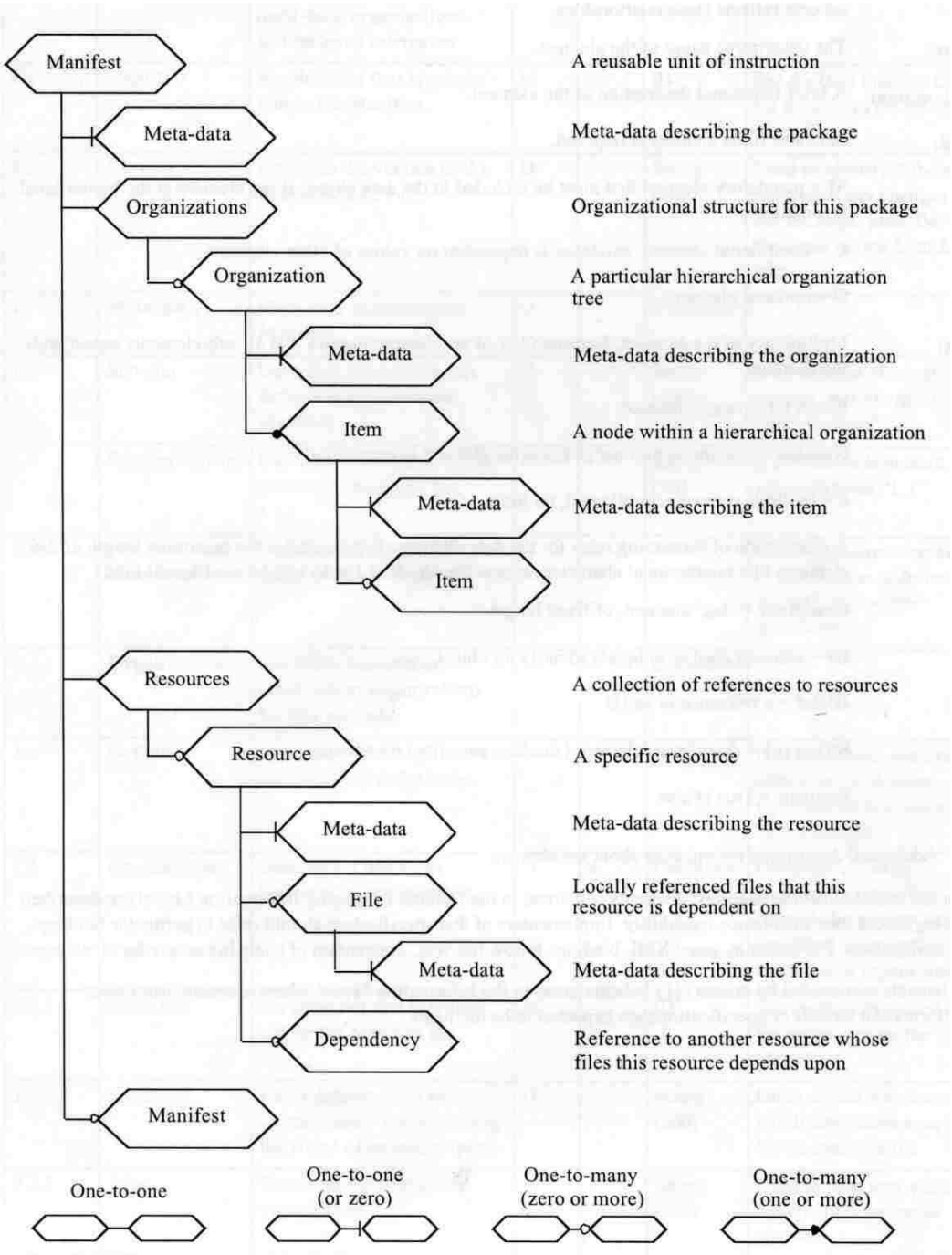

Abbildung 13 Die Struktur eines IMS Manfests, Quelle: [IMS01]

4.2. Die Java Virtual Machine (JVM)

4.2.1. Die Architektur von Java

Von SUN Microsystems entwickelt war Java ursprünglich als Basis für unterschiedlichste, meist kleinere, elektronische Geräte wie etwa PDAs gedacht. Ziel der Entwicklung war es daher, eine plattformunabhängige Umgebung zu schaffen, in der ein und dieselbe Applikation auf verschiedenen Geräten oder Architekturen ohne Modifikationen laufen kann. Um diese Vorgabe zu erreichen wurde schnell klar, dass dies nur durch die Einführung einer Zwischenschicht, die die eigentliche Hardware von der Software trennt, möglich sein wird. Diese Zwischenschicht, die Java Virtual Machine, setzt zwar auf dem bestehenden Betriebssystem auf und muss daher für jede Architektur eigens entwickelt werden, verhält sich der Applikation gegenüber allerdings immer identisch. Implementiert ist diese wie ein eigener Prozessor, sie übernimmt also Aufgaben wie Memory Management (inklusive Garbage Collection), Scheduling oder Security Belange für die eigentliche Anwendung.

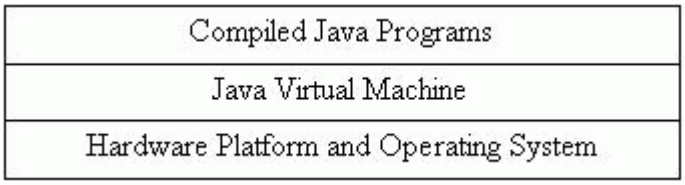

Abbildung 14 Die Java Virtual Machine bietet die Schnittstelle zwischen Hardware und Applikation, Quelle: [Ven96]

Auch wenn unter „Java“, etwas unpräzise, meist nur die Programmiersprache gemeint ist, so gliedert sich nach *[Ven00]* Java eigentlich in vier wesentliche Bereiche:

- Die Java Programmiersprache
- Das Java Class-File Format
- Das Java Application Programming Interface (API)
- Die Java Virtual Machine

Bei der Entwicklung einer Java-Applikation kommt der Programmierer mit allen Teilen in Kontakt. Nach der Erstellung des Programms wird dieses in Form von Bytecode in Class-Files kompiliert und von der Java Virtual Machine ausgeführt, in dem über das

Java API zB Hardware Ressourcen wie I/O Geräte anfordert werden wie in Abbildung 13 dargestellt.

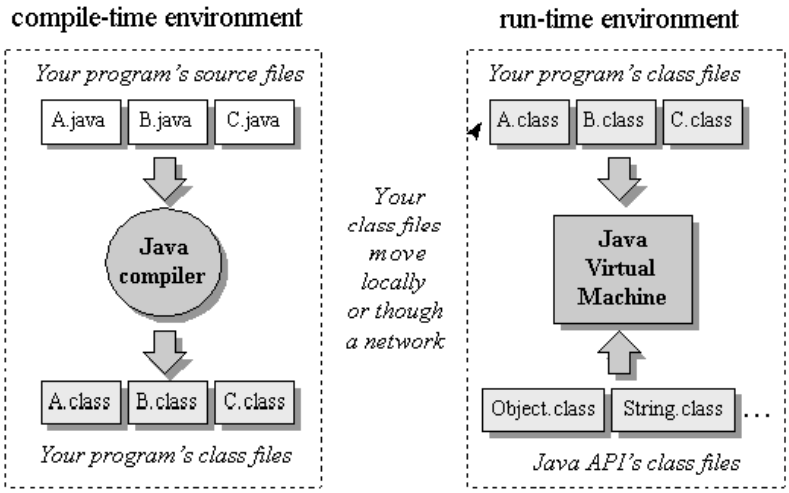

Abbildung 15 Die Java Programmierumgebung, Quelle: [Ven00]

Hier wird die klare Trennung zwischen Anwendungsentwicklung und Run-time Umgebung deutlich. Die kompilierten Java Class Files können in beliebigen Laufzeitumgebungen abgearbeitet werden, ohne Änderungen vornehmen zu müssen. Die JVM übernimmt, alle nötigen Adaptierungen an die darunterliegende Hard- & Software Architektur transparent für die Applikation.

4.2.2. Die Architektur der JVM

Die Java Virtual Machine wird in der Literatur fast immer beschrieben als „abstract computer“, der auf eine bestehende Hardware Plattform und ein bestimmtes Betriebssystem aufsetzt und den Java Applikationen einen immer gleichen Computer im Computer „vorgaukelt“. In Wirklichkeit ist genau dieses Konzept die Basis und der Erfolg von Java, denn es garantiert grundsätzlich gänzliche Portabilität. Performance Engpässe, die durch die „Interpretation“ des Bytecodes der Class-Files in früheren Java Versionen beträchtlichen Einfluss auf die Akzeptanz und Verbreitung hatten, wird heute durch massive Weiterentwicklungen versucht, diese in den Griff zu bekommen. Als Beispiel sei hier nur die Just-in-Time Compilation genannt. Ein Wesen von Java ist nämlich auch, dass die Spezifikation viel Spielraum in der Implementierung der Features lässt und somit den Entwicklern von VMs großer Freiraum für plattformabhängige Optimierungen bleibt. Darüber hinaus wird für jede Applikation eine eigene JVM gestartet, was eine klare und einfach Trennung (und Schutz) zB des jeweils zugeordneten Speicherbereichs zulässt.

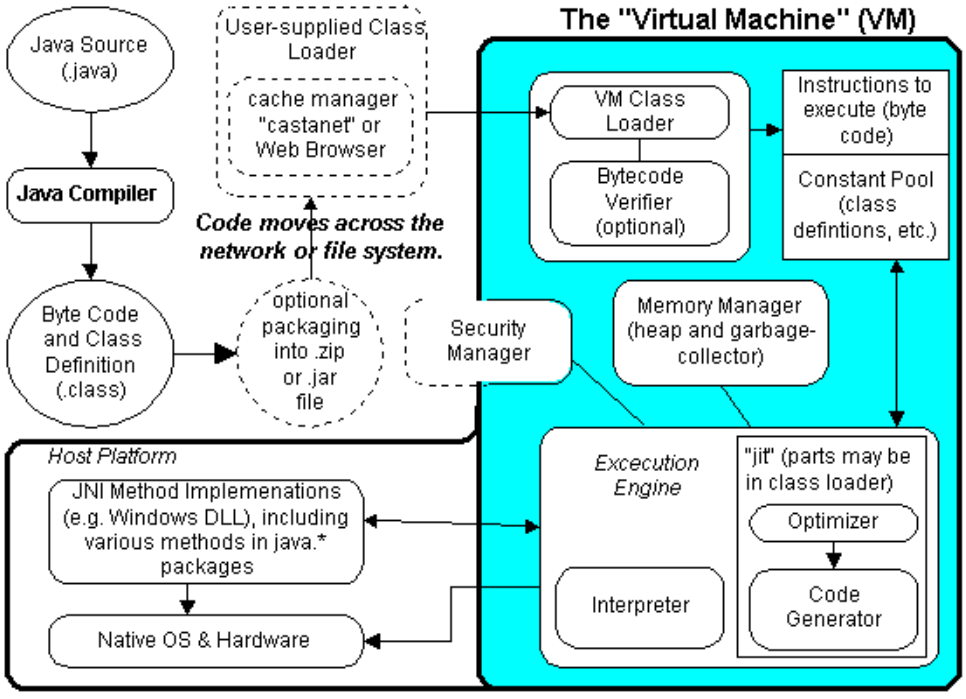

Abbildung 16 Die Architektur der JVM, Quelle: [Hol01]

Im Wesentlichen besteht die JVM aus zwei Teilen, die von zum Teil optionalen „Hilfsmodulen“ wie dem Memory oder Security Manager oder auch dem Bytecode Verifiyer oder Just-in-Time Compilern unterstützt werden: dem Class Loader und der

Execution Engine. Hauptaufgabe ist demnach das Laden der Class-Files und deren Ausführung auf der darunterliegenden Hardware. Dabei werden vom Class-Loader nicht nur User-Class-Files geladen, sondern auch, Java API Class-Files soweit diese benötigt werden.

Je nach Möglichkeit, die die Hardware bietet, bzw. Anforderungen, kann die Implementierung der Execution Engine stark variieren: vom einfachen Interpretieren des Bytecodes bis hin zur hoch optimierten aber Ressource vergeudenden, Just-In-Time Kompilierung erstreckt sich hier das Feld. Die Königsklasse ist in diesem Zusammenhang sicherlich eine Execution Engine als adaptiver Optimierer, bei dem zunächst der Bytecode interpretiert, aber gleichzeitig erkannt wird, welche Bereiche des Codes häufig benutzt werden, um diese stark frequentierten Stellen schließlich on-the-fly in Native-Code zu compilieren. Nach *[Ven00]* kann damit erreicht werden, dass die JVM 80 – 90 % der Zeit mit der Ausführung von solch höchst-optimiertem Native-Code beschäftigt ist und damit beträchtliche Performanzsteigerungen erzielt werden können.

4.2.2.1.Run-time Data Areas

Die formale Spezifaktion der Java Virtual Machine sieht neben Subsystemen, Datentypen und Instruktionen, auf die später näher eingegangen wird, auch verschiedene Laufzeit Speicherbereiche vor, die mehr eine abstrakte Vorgabe des Verhaltens als eine konkrete Implementierungsvorschrift darstellen. Wenn die JVM ein Programm abarbeitet, sind dabei vielerlei Dinge im Speicher abzulegen: Objekte, Variablen, Zwischenergebnisse, Rückgabewerte oder auch der Bytecode selbst. Die JVM organisiert diese Dinge den Run-time Data Areas, dargestellt in Abbildung 15.

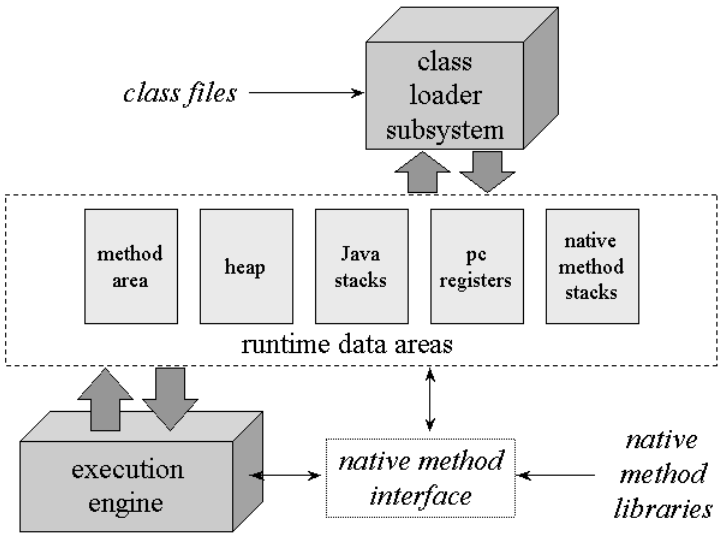

Abbildung 17 Die Run-time Data Areas der JVM, Quelle: [Ven00]

Obwohl diese Datenbereiche in der einen oder anderen Form in jeder Implementierung einer Virtual Machine existieren, ist ihre konkrete Ausformung dennoch unterschiedlich. Oft muss auf Gegebenheiten oder Einschränkungen der Hardware Rücksicht genommen werden: Handelt es sich um PDAs mit eingeschränktem Speicher oder um Workstations, mit ausreichend Memory bzw. Virtual Memory; je nach Möglichkeiten werden Execution Engine und Run-time Data Areas angepasst werden müssen, was auf Grund der sehr abstrakten Spezifikation der JVM kein Problem darstellt. Fest steht jedenfalls, dass manche Datenbereiche innerhalb einer VM lediglich ein einziges mal existieren und daher von allen Threads gemeinsam genützt werden müssen, wie die method area oder der heap und andere pro Thread zur Verfügung stehen wie die Java Stacks oder PC (program counter) Register.

Method Area, Heap, Java Stacks und PC Register

Beim Laden eines Class Files werden die Typinformationen der Klasse gesammelt und als Class Data in der gemeinsamen methods area abgelegt. Dabei ist wesentlich, dass diese Sychronisationsmechanismen implementiert hat, die den Umgang mit Threads regelt. Zum Beispiel muss festgelegt sein, was passiert wenn zwei Threads gleichzeitig eine noch nicht existierende Klasse laden wollen. Dabei muss die Frage geklärt werden, wer sie laden darf und wer warten muss.

Zur Laufzeit angelegte Objekte kommen hingegen auf den Heap, wobei hier die Synchronisation der Zugriffe auf Objekte bereits im Programmcode vorgesehen sein muss. Da in der Java Programmiersprache kein Konstrukt zur Freigabe von erzeugten Objekten existiert, kümmert sich der integrierte Garbage Collector darum, nicht mehr referenzierte Objekte vom Heap wieder zu entfernen.

Sobald ein Thread erst einmal erzeugt ist, wird ihm von der JVM ein eigener Java Stack und ein PC Register zugewiesen, der die Position der nächsten Instruktion einer Java-Methode anzeigt. Im Java Stack wird der Status aller eingebunden reinen Java Methoden gespeichert, dh lokale Variable, übergebene Parameter, Rückgabewerte bzw. Zwischenergebnisse. Jeder Methode wird dabei durch einen eigenen Stack Frame dargestellt, der nach Beendigung derselben wieder entfernt wird. Im Unterschied dazu werden dieselben Dinge von Native-Methods in eigenen, architekturabhängigen native-method-stacks abgelegt wie Abbildung 16 illustriert. Während der Ausführung einer solchen Funktion ist das PC Register aus trivialen Gründen undefiniert und daher grau gezeichnet.

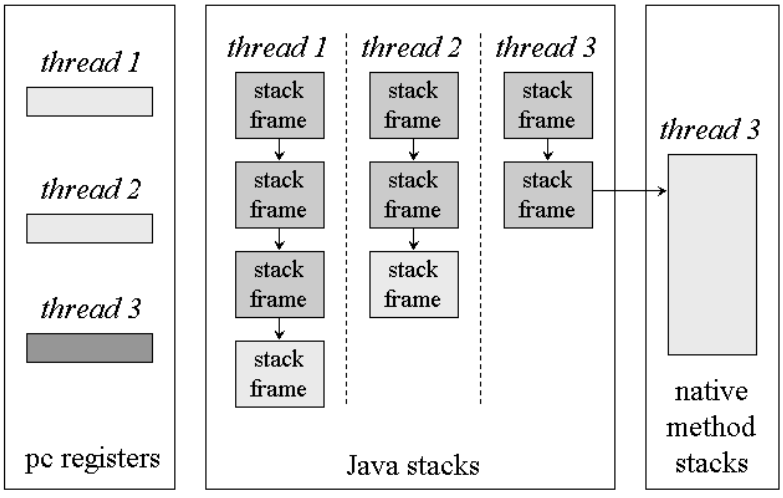

Abbildung 18 PC Register und Java Stacks existieren pro Thread, Quelle: [Ven00]

4.2.2.2. JVM Daten Typen

Im Gegensatz zu vielen bereits beschriebenen Freiheiten, die die JVM Spezifikation der konkreten Implementierung zugesteht, ist sie sehr detailliert und strikt in Bezug auf Daten und deren Datentypen. Im Wesentlichen werden dabei zwei Arten unterschieden:

- **Primitve Typen**: wie float, int, boolean, etc. Sie enthalten selbst die tatsächliche Größe
- **Referenztypen**: Sie verweisen lediglich auf bestehende Objekte enthalten selbst aber keinen Wert wie zB Arrays

4.2.2.3. Das Class-Loader Subsystem

Bisher, wie etwa in Abbildung 14 oder 15, ist davon ausgegangen worden, dass für das Laden der diversen Klassen ein einzelner Class Loader existiert. Tatsächlich besteht aber die Möglichkeit, mehrere so genannte „User-defined“ Class Loader selbst zu implementieren, die dann nach Bedarf vom Class Loader Subsystem der Virtual Machine eingebunden werden. Lediglich ein einziger, der Bootstrap Loader, ist in der JVM enthalten und sofern nichts anderes bestimmt wird, ist jener auch zuständig, die benötigten Klassen (im Default auf der Lokalen Festplatte) zu suchen und für die Ausführung zu laden. Zur Run-time können aber beliebige andere solcher benutzerdefinierten Class Loader erzeugt werden. Diese werden als „normale“ Objekte am Heap abgelegt und dazu verwendet, um zum Beispiel Klassen über Netzwerkgrenzen hinweg einbinden zu können.

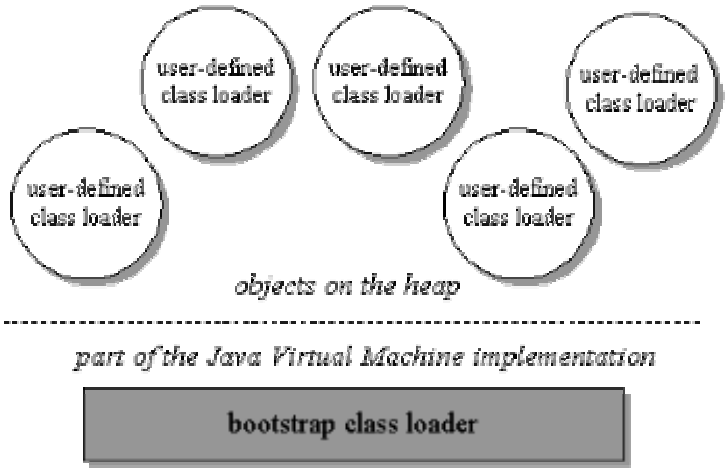

Abbildung 19 Das Class Loader Modell der JVM, Quelle: [Ven00]

Darüber hinaus erlaubt dieses Modell eine dynamische und flexible Erweiterung der Java Applikation zur run-time, da zur Compile-Zeit noch nicht feststehen muss welche Klassen geladen werden müssen. Dazu kommt, dass die JVM Kommunikation nur zwischen solchen Klassen erlaubt, die vom selben Class Loader stammen und ermöglicht somit unterschiedliche „Name Spaces“ innerhalb ein und derselben Applikation. Als praktisches Beispiel seien an dieser Stelle alle Web-Browser genannt, die mit Hilfe von user-defined Class Loaders Klassen für Applets aus verschiedenen Sourcen in unterschiedlichen Name-Spaces über das Internet einbinden und dadurch wechselweise voneinenander geschützt sind.

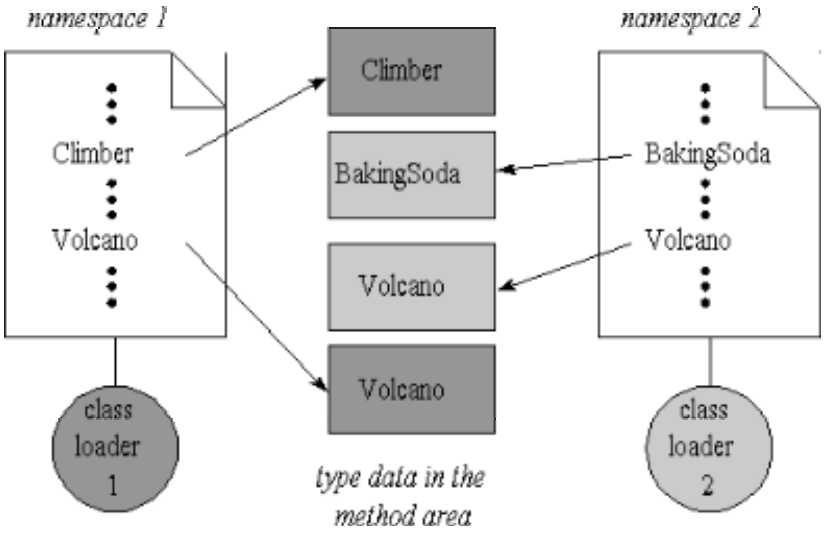

Abbildung 20 Ein Beispiel der Class Loader Architektur, Quelle: [Ven00]

4.2.2.4. Die JVM Execution Engine

Die zweite wesentliche Kernfunktion der Java Virtual Machine, nämlich die Ausführung des Bytecodes, wird von der Execution Engine, bzw. deren Ausprägungen, den Threads, durchgeführt. Die Java Spezifikation legt dafür das „Instruction set“, also die Menge der Befehle im Detail fest, wobei auch hier das „Was“ sehr genau beschrieben und das „Wie“ dem Implementierer der JVM überlassen wird. Konkret beschränkt sich die Befehlsmenge auf eine kleine Zahl mehr oder weniger einfacher Instruktionen, die je nach verwendeter Technik interpretiert, just-in-time kompiliert oder in irgendeiner anderen Weise ausgeführt werden. Zusätzlich zu den so entstehenden existieren noch weitere, für die Java Anwendung „unsichtbare“ Threads die beispielsweise zur garbage collection herangezogen werden, die ja immer und unabhängig von der Applikation arbeiten muss.

Die bereits angesprochene Menge an Instruktionen, die der erzeugte Bytecode enthält ist standardisiert und besteht immer aus einem Opcode und optionalen Operanden, die ihrerseits je nach Befehl Daten, Sprungadressen oder andere Referenzen enthalten können. Wesentlich dabei ist im Fall von Daten, dass diese nicht unbedingt direkt an dem Opcode anschließen müssen, sondern auch in anderen Datenbereichen der JVM zu finden sein können, zB im Operanden-Stack. Die Abarbeitung des Bytecodes eines Threads wird dabei erst beendet, wenn von der Startmethode zurückgekehrt oder durch eine nicht-abgefangene Exception abgebrochen wird. Die Instruktionen werden dabei grundsätzlich in der Reihenfolge ihres Auftretens im Bytestream behandelt, mit Ausnahme von expliziten oder impliziten Sprungbefehlen wie „goto“, „return“ oder einer Exception, wobei hier nach einer entsprechenden „catch“ Klausel gesucht wird.

Adaptive Optimierung

Bereits in einem der Vorkapitel wurde diese Technik der Execution Engine kurz angeschnitten und soll im Folgenden etwas genauer betrachtet werden. Neben einer Menge an möglichen Strategien den generierten Bytecode auszuführen entpuppt sich die Adaptive Optimierung einerseits als kostengünstige (in Bezug auf den Speicherverbrauch) und andererseits im höchsten Maße performanzsteigernde Möglichkeit, der Java nachgesagten Trägheit bei der Programmausführung effizient entgegenzuwirken. Im Wesentlichen als eine Mischung zwischen reiner Interpretation und vollständiger oder just-in-time Kompilierung zeigt sich auch bei der Adaptiven Optimierung, dass der Mittelweg oft golden ist. Der große Vorteil dieser Technik liegt

darin, dass auf Informationen, die erst zur Laufzeit zur Verfügung stehen zurückgegriffen wird, um genau jene Teile des Bytecodes in Native-Code zu kompilieren, die am häufigsten verwendet werden. Untersuchungen zeigen, dass durchschnittlich zwischen 80 – 90% der Run-Time nur 10 – 20% des Codes ausgeführt werden. Auf Basis dieses Wissen monitort die JVM den zu Beginn ganz „normal" interpretierten Bytecode, um diesen so genannten „Hot Spot" zu identifizieren. Erst einmal gefunden wird ein Background-Job angestoßen, der zur Laufzeit den Hot Spot in Native-Code kompiliert und vor allem in höchstem Maße optimiert (vergleichbar mit der statischen C++ Optimierung) und damit besten Performanz garantiert. Dazu kommt, dass dieser Vorgang sich immer dann wiederholt, wenn sich der Hot Spot innerhalb des Bytecodes verschieben sollte, weswegen in der JVM eine Kopie des ursprünglichen Bytecodes gehalten wird, um diesen gegebenenfalls (dh wenn dieser aus der Hot Spot Region herausfallen sollte) rekonstruieren zu können.
Auch bei den angesprochenen Optimierungen werden Run-Time Informationen herangezogen, um optimale Performance zu erreichen. Als Technik wird Inlining verwendet, bei dem Methodenaufrufe durch den Code der gerufenen Funktion ersetzt werden, um sinnvolle Optimierung der Statements erst zu ermöglichen. Während statisch kompilierte, im höchste Maße objektorientierte Programme den Optimierer dabei vor meist unlösbare Aufgaben stellen (Gründe dafür ist die naturgemäß große Zahl an Methodenaufrufen bzw. das Konzept der dynamische Bindung), wird bei der Adaptiven Optimierung Inlining nur dort verwendet wo zur Laufzeit feststeht, welche Methoden am häufigsten benötigt werden und für jeden dieser Fälle der Code eigens optimiert.

Threading Model

Wie auch bei allen anderen Teilen von Java ist auch beim Threading Modell größter Wert auf Plattformunabhängigkeit gelegt worden, was zwar grundsätzlich begrüßenswert ist, allerdings in Bezug auf Threads durchaus auch Nachteile mit sich bringt. Grundsätzlich stehen dem Implementierer der JVM zwei Möglichkeiten offen: das Mapping der Java Threads auf das bestehende, vom Betriebssystem abhängige Native-Threading-Model oder die Entwicklung eines eigenen Modells. Aus Portabilitätsgründen existiert in der Spezifikation aber lediglich ein einfaches Prioritätssystem in 10 Stufen ohne bestimmte Vorgaben in Bezug auf Schedulingmechanismen etc., dh konkret als Vorgabe: der Thread mit der höheren

Priorität muss vor dem mit niedrigerer Wichtigkeit ausgeführt werden – auf alles Weiterführende darf sich der Anwendungsentwickler nicht verlassen. Bei Multi-Threading Applikationen liegt es daher am Programmierer, Aufgaben wie Synchronisation zwischen den Threads selbst in die Hand zu nehmen und im Code vorzusehen. Dabei unterstützt wird er von zwei Mechanismen der JVM: Object Locking und Thread wait bzw. notify. Ersteres bewahrt vor Inkonsistenzen wenn zwei unabhängige Threads auf gemeinsamen Daten arbeiten und letzteres wird dazu eingesetzt, um Threads zu koordinieren, die gemeinsam eine Aufgabe zu lösen haben. In beiden Fällen handelt es sich um Daten, die im Shared-Main Memory der JVM abgelegt sind, wie zB Variablen von Objekten, Array-Daten, etc. wobei alle Operationen auf primitiven Datentypen (Integer, Byte, etc.) atomar ausgeführt werden und somit kein Synchronisationsproblem hervorrufen. Um aber alle anderen Datentypen bzw. Objekte sinnvoll synchronisieren zu können, sind nach *[Ven00]* zwei grundlegende low-level Regel für Threads spezifiziert, die festlegen, wann ein Thread obligatorisch bzw. optional:

- Werte vom Main Memory in den Thread eigene Arbeitsspeicher kopieren und
- diese Werte zurück in den Hauptspeicher geschrieben werden müssen

Diese Vorgangsweise garantiert zum einen eine wiederhol- und vorhersehbare Arbeitsweise der JVM und zum anderen erlaubt sie dem einzelnen Programmierer dennoch gewisse Flexibilität in Bezug auf die Ausnützung besonderer Vorteile vorhandener Hardware. Sie entbindet ihn aber nicht von der Aufgabe, Synchronisation bei Multi Threading Anwendungen (wie zB Web Applikationen) vorzusehen, um inkonsistente Datenbestände zu verhindern.

4.3. Java Servlets

Im speziellen auf Web-Applikationen ausgerichtet existiert eine Erweiterung des Java API, die so genannten Servlets. Im Wesentlichen handelt es sich dabei um Server-seitige Applikationen, die eingebettet in eine Java Virtual Machine als Schnittstelle zwischen dem Web-Browers und der eigentlichen Web-Anwendung unterschiedlichste Aufgaben übernehmen können. Anfragen an einen Web-Server, zB durch Klicken auf einen Link, werden zunächst an das Servlet weitergeleitet, das diese Anfrage entgegennimmt, interpretiert, die nötigen Aktionen entweder selbst vornimmt oder veranlasst und als Ergebnis ein fertiges HTML-Dokument an den Client zurücksendet. Daraus lassen sich a priori einige Vorteile ableiten:

- Servlets sind **portabel**: da sie Java-Programme sind, die zur Ausführung lediglich eine JVM benötigen, die auf allen Plattformen zur Verfügung steht
- Servlets sind **unabhängig** vom Web-Browser: da sie auf am Server laufen und als Output einfache HTML Seiten generieren
- Servlets sind **sicher**: da alle Sicherheitsmechanismen der JVM greifen (Typsicherheit, Exception-Handling, etc.)
- Servlets sind **effizient**: sind sie erst einmal im Speicher des Webservers bleiben sie dort als Singleton und bearbeiten alle Anfragen durch einfache Threads (im Unterschied zu „herkömmlicher“ Common-Gate-Interface/CGI Programmierung, wo ganze Prozesse erzeugt werden)

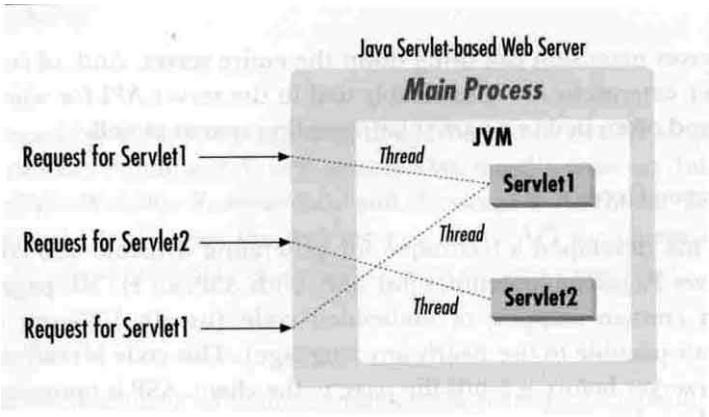

Abbildung 21 Das Multithreading der Java Virtual Machine, Quelle: [Hun98]

Alle wesentlichen Web-Server unterstützen Java-Servlets, was ihnen beträchtliche Flexibilität verleiht, denn durch diese Technologie können beliebige Erweiterung der

Funktionalität vorgenommen werden, die durch Java-Programmierung implementierbar sind. Ein einfacher Mailserver beispielsweise könnte dadurch auch Virenscanning vornehmen oder Junk-Mails filtern. Servlets bieten somit die beste Basis für eine komplexe Web-Applikation und insbesondere für einen Kernel einer solchen Anwendung, der als zentrale Schnittstelle designed wurde.

4.3.1. Der Lebenszyklus von Java Servlets

Der Lebenszyklus von Java Servlets beginnt mit der ersten es betreffenden Anfrage an den Webserver. Das Servlet wird geladen, initialisiert und bleibt im Speicher bis der Webserver aus unterschiedlichsten Gründen das Servlet „zerstört“, dh die vorgegebene „destroy“ Methode, die jedes Servlet besitzen muss, aufruft und es aus dem Memory entfernt. Die einzige Möglichkeit bewusst auf diesen Vorgang einzuwirken ist, manuell das Servlet neu zu laden, was insbesondere dann von Bedeutung ist, wenn sich das darunterliegenden class-file geändert haben sollte. Durch diese Methode können demnach beispielsweise Updates leicht eingespielt werden ohne dabei den Web-Server an sich stoppen zu müssen.

4.4. Der WeLearn Kernel

4.4.1. Einleitung

Im Rahmen des web- und objektorientierten Gesamtkonzeptes von WeLearn wurde der Kernel als Java Servlet implementiert und dient somit, wie im Vorkapitel besprochen als Schnittstelle zum Anwender am Web-Browser einerseits und zur eigentlichen Java-Applikation andererseits. Auch wenn Teile der Funktionalitäten eines Betriebssystem Kernels von der Java Virtual Machine in der jeweiligen Implementierung übernommen werden (Memory Management, Thread Scheduling, Security Belange, etc.), komplettiert der WeLearn Kernel aus Sicht der Web-Anwendung diese Dienste auf einer höheren Ebene und übernimmt daher folgende Kernaufgaben:

- Initialisierung des Systems
- Message Passing zwischen den Objekten
- Scheduling von High-Level Tasks
- Erzeugung des Outputs für den Anwender

Im Allgemeinen kann also festgehalten werden, dass der Vergleich eines Kernels eines Betriebssystems mit dem „dualen“ System JVM plus WeLearn Kernel zulässig ist, auch wenn die gesetzten Ziele und Aufgabenbereiche durchaus differieren, was in dieser Arbeit noch näher zu erläutern sein wird.

4.4.2. Initialisierung des Systems

Einer der Kernaufgaben des WeLearn Kernels nach der erstmaligen Installation des Systems bzw. beim Start des Kernel-Servlets liegt in der Initialisierung der einzelnen Komponenten. Diese Aufgaben beziehen sich auf core-components wie den Persistenzmanager, den Logger, das Uploading Modul und das virtuelle Dateisystem. Während letzteres lediglich beim aller ersten Aufruf des Servlets initialisiert werden muss, besteht im Kontrast dazu die Notwendigkeit, bei jedem Systemstart, dh zu dem Zeitpunkt an dem das Kernel-Servlet vom Web-Server instanziiert wird, zu garantieren, dass die restlichen Komponenten einsatzbereit sind.

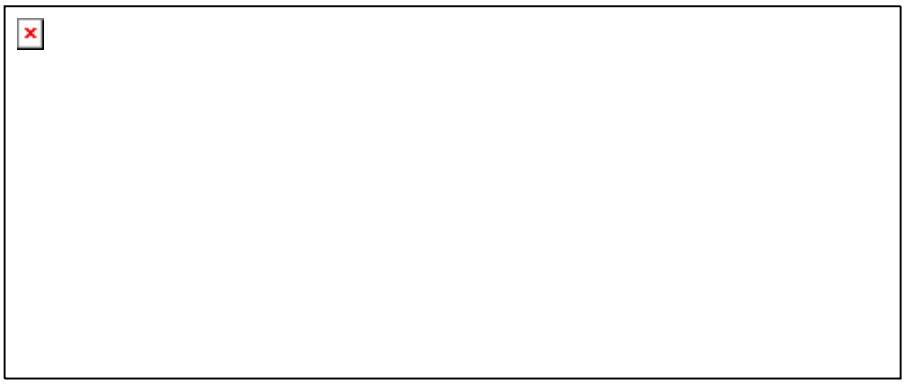

Abbildung 22 Die vier Teile der WeLearn Kernel Initialisierung

4.4.2.1. Der Persistenzmanager

Persistenz wird in *[Tec02]* definiert als *„the storage of an object on a disk or other permanent storage device"*. Persistenz ist damit Kernaufgabe jedes Systems, das mit dynamischen, veränderbaren Daten umzugehen hat. In großen Datenbanksystemen übernimmt diese Aufgabe der Datenbankmanager, der zusätzlich zum reinen Speichern der Informationen auch sicherstellt, dass die Konsistenz gewahrt bleibt. Auch im WeLearn Framework werden sämtliche für die Funktionalität relevanten bzw. von den Benutzern erzeugten Objekte wie Diskussionsforen, InfoBoards oder Kursmaterialien persistiert. Naheliegend wäre nun für diese Anforderung einen public domain Datenbank heranzuziehen (was in WeLearn Release 1 auch der Fall gewesen ist), es zeigt sich aber im Echtbetrieb, dass diese frei verfügbare Software eklatante Performancedefizite bei einer vielzahl an konkurrierenden Zugriffen aufweist, was aber in Multi-User, Multi-Threading Web-Applikationen in der Natur der Sache liegt (zB bei Chats). Darüber hinaus war eines der Designkriterien im Entwurf dieses Systems, eine vollintegrierte Lehr- und Lernplattform zur Verfügung zu stellen, die möglichst unabhängig von Drittanbietern ist.

Aus diesem Grund wurde für die Speicherung jeglicher Art von Objekten in WeLearn Release 2 ein eigener Persistenzmanager entwickelt. Im Wesentlichen umfasst dieser ähnliche Funktionalität wie ein Datenbankmanager in kommerziellen Systemen, allerdings mit dem Unterschied, dass als Speichermedium das Filesystem des OS dient und keine Datenbank.

Der Persistenzmanager überwacht durch einen integrierten Lockingmechanismus den Zugriff auf Objekte und stellt sicher, dass diese zu jedem Zeitpunkt persistent gehalten sind. Aus Performance Gründen wird darüber hinaus ein interner Cache verwendet, der nach der Write-Through Strategie arbeitet, dh veränderte Objekte werden sofort auf das Speichermedium zurück geschrieben (vgl. Kapitel 4.5.2 Grundlegende Unterschiede zum OS Kernel).

Während der Initialisierungsphase wird dabei ein Filesystem Ordner erzeugt, der sämtliche serialisierten WeLearn-Objekte enthält. Zusätzlich besteht die Möglichkeit, festzulegen ob der Cache aktiviert werden soll.

4.4.2.2. Der Logger

Als Logger wird das Standard Logging Modul der Apache Group Log4j *[Apa02]* verwendet, der an beliebigen Stellen im Quellcode beliebige Informationen aufzeichnen kann. Im WeLearn System wird er speziell dazu verwendet, um die Anfragen an das Kernel Servlet zu protokollieren. Dabei wird u.a. festgehalten, um welchen Typ es sich dabei handelt („get“ oder „post“), welcher Befehl abgesetzt wird und welchem Systemobjekt dieses Kommando weitergeleitet wird (in Form der OID) bzw. diverse zusätzliche Informationen. Ein typischer Eintrag könnte demnach so aussehen:
"*GET*/Kernel?dwm_retrieve_area=dwm_bottom&*krnlcmd=operate*&*oid=2a18fd20523 bf70d866f8b418559d9b7d*&rc=1&inst=1 HTTP/1.1" 200 751“.

4.4.2.3. Das Uploading Modul

Wie bereits angesprochen wird zur reibungslosen Abwicklung von e-Learning Kursen bei WeLearn auf ein internes, virtuelles Dateisystem zurückgegriffen (auf das im folgenden Kapitel näher eingegangen wird). Es stellt keine Abbildung des realen Filesystems dar, sondern ist stark kurs- und benutzerorientiert ausgelegt. Um nun „reale“ Dateien wie PDFs, DOCs, aber auch CPS Packete in dieses Dateisystem einbinden zu können existiert das Uploading Modul. Intuitiv benutzbar besteht die Möglichkeit uploading vorzunehmen, wobei in der Initialisierungsphase lediglich ein Folder im Filesystem erzeugt wird, in den sämtliche Dateien eingespielt werden, unabhängig davon wo sie im virtuellen Dateisystem zu finden sein werden.

4.4.2.4. Das virtuelle Dateisystem

Das virtuelle Dateisystem ist die Basis für die Arbeit mit WeLearn, da mit Hilfe dieses Moduls sowohl die Administration der Kurse als auch des Systems selbst sowie die User-seitig die Anwendung gewährleistet werden. Ausgehend von einem Wurzelknoten werden in einer Baumstruktur alle internen WeLearn-Objekte wie Foren, Folder, User, Gruppen, etc. aber auch alle externen Files abgebildet, die durch uploads in WeLearn Objekte „eingebettet" und dem virtuellen Dateisystem hinzugefügt wurden. Dieser Vorgang hat zur Folge, dass bei nicht WeLearn-Objekten auf dem realen Speichermedium schließlich zwei Dateien existieren:

- Die **tatsächlich hinaufgeladenene Datei** im (von der Init-Routine des Uploading Moduls erzeugten) „Upload" Folder
- Das **serialisierte WeLearn Objekt** im (von der Init-Routine des Persistenzmanager erzeugten) „Object-Root" Folder, das lediglich eine Referenz in Form eines absoluten Pfads enthält

Der Vorteil davon ist, dass somit beliebige Strukturen für unterschiedlichste Anwendungsgebiete erzeugt werden können, unabhängig vom tatsächlichen Aufbau des Dateisystems im Hintergrund. Zusammenhänge zwischen internen WeLearn-Objekten werden einerseits durch eine einfache Vater-Sohn Beziehung bzw. Querlinks durch die systemweit eindeutige Object-ID realisiert.

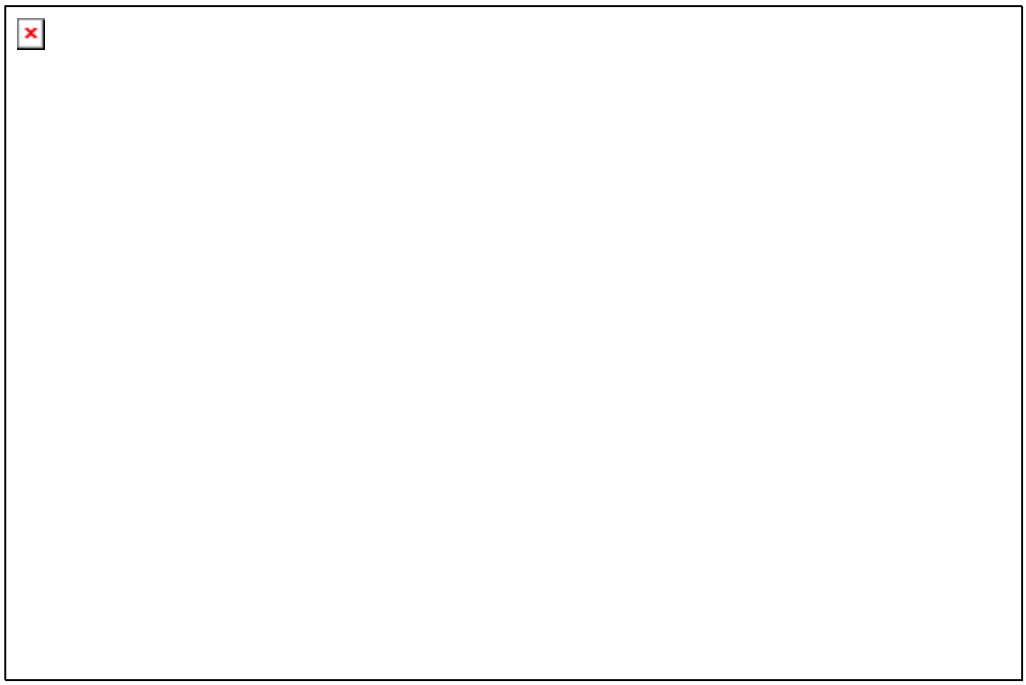

Abbildung 23 Das Virtuelle Dateisystem von WeLearn, ein Beispiel

Bei der Initialisierung wird zunächst sichergestellt, dass der Wurzelknoten des virtuellen Dateisystems existiert. Danach müssen standardisierte „Grundobjekte“ erstellt werden, die die absolute Basisfunktionalität zur Verfügung stellen wie der Users- und Groups Folder, einige User wie „Everybody“, „System“ oder „Owner“ bzw. der Rechteordner, in dem die ACLs (Access Control List) der WeLearn Objekte abgelegt werden. Da WeLearn ohne die Existenz dieser Objekte nicht lauffähig ist, ist der Kernel selbst für die Generierung verantwortlich.

Anschließend wird das Rechtesystem (vgl. *[Lei03]*) gestartet und dem User „System“ werden Administratorrechte eingeräumt, weil dieser als Basis für das nun auszuführende Setup-Script dienen wird (was im Übrigen die einzige Aufgabe dieses Benutzers darstellt). Sämtliche Möglichkeiten der Adaptierung und Gestaltung des WeLearn Systems stehen dabei zur Verfügung und können in dieses Script miteinfließen. Diese Methode ein System aufzubauen verleiht WeLearn eine weitere Nuance der Flexibilität: dadurch können einerseits unterschiedlichste Basis-Szenarien mit ausgeliefert werden und andererseits in Zukunft der Kunde selbst vorab seine Wünsche zB per Konfigurationswizard dem System mitteilen ohne es dabei ein einziges mal tatsächlich administrieren zu müssen.

4.4.3. Message Passing

Eine der Kernaufgaben jedes Kernel ist die Bereitstellung von effizienten Kommunikationsmechanismen zwischen den einzelnen Komponenten sei es des Betriebssystems oder der E-Learning Plattform. Bei Windows2000/XP wird diese Anforderung von den Local Procedure Calls (LPCs) übernommen. In Rahmen des WeLearn Systems führt diese Aufgabe der Kernel selbst aus. Jede Anfrage des Browsers, zB das Klicken eines Links zu einem Powerpoint File, wird an das Kernel Servlet weitergeleitet und nach Überprüfung der Validität der Session aus der sie stammt, bearbeitet. Dabei muss in Betracht gezogen werden, dass obwohl der Benutzer sich im virtuellen Dateisystem von WeLearn befindet und damit in einer in gewissem Sinn irrealen Welt, trotzdem die „gewohnte“ Funktionalität des Browser bestehen bleiben sollte. Was am Beispiel des Links nichts anderes heißt, als dass das vom Servlet zurückgegebene Resultat kein WeLearn Objekt oder ähnliches, sondern das reine Powerpoint File sein muss (worauf, je nach Browsereinstellung und Verfügbarkeit der Software, Powerpoint gestartet und die entsprechende Datei angezeigt wird).

Darüber hinaus ergibt sich aus der Architektur des Systems die Notwendigkeit all jene Dateitypen besonders zu behandeln, die Links beinhalten bzw. sich aus verschiedenen miteinander verknüpften Teilen bestehen, also insbesondere HTML Dateien mit eingebetteten Bildern und Links zu weiteren HTML Files. Diese beinhalten nämlich relative Pfade zu ihren Objekten (zB könnte ein HTML Tag der ein Bild einbindet so aussehen: <img src=“/pictures/mypic.gif“>) was bei Nichtunterscheidung im Kernel zu Problemen führt.

Ein Beispiel: angenommen eine HTML Datei enthält obigen Pfad zu einer Grafik und ist im virtuellen Dateisystem im Pfad „/home/ernie/index.html“ abgelegt. Ein Benutzer öffnet nun diese web Seite. Der Kernel behandelt dieses Objekt auf folgende Art und Weise: der Pfad der Webseite wird dem Kernel übergeben. Dadurch wird zunächst die Site unter „/home/ernie/index.html“ im virtuellen Dateisystem gesucht (und gefunden). Allerdings wird auch das verknüpfte Element unter dem absoluten Pfad („/pictures/mypic.gif“) gesucht und nicht gefunden, da ja auf Grund der relativen Adressierung im Code der HTML Seite das Bild in „/home/ernie/pictures/mypic.gif“ liegt.

Im Wesentlichen werden daher im Kernel zwei Typen von Requests unterschieden:

- Data requests für HTML und strukturähnliche Dateien
- Alle übrigen requests

Abbildung 24 Abläufe im WeLearn Kernel (schematisch)

Handelt es sich bei der Anfrage also um keinen data-request, dh eine WeLearn Objekt möchte ein Kommando an sich selbst oder eine anderes WeLearn Objekt senden, wertet der Kernel diesen Befehl aus und leitet ihn an das entsprechende Zielobjekt weiter. Dabei können insgesamt drei verschiedene solcher Befehle auftauchen:

- **Operate**: der Defaultwert, dh sollte die Anfrage kein Kommando enthalten, wird ein Operate command erzeugt. Der Operate Befehl veranlasst das Objekt

an das er versandt wird zumindest dazu, sich erneut darzustellen wird aber meist auch zur Interaktion mit dem Benutzer verwendet. Ein Beispiel: beim Login wird nach der Prüfung der eingegebenen Informationen das Ergebnis (Successful oder Failed) zurückgeliefert und grafisch dargestellt. Dabei sind an dieses Objekt zwei aufeinander folgende Operate Kommandos ergangen: das erste um die Eingabemaske darzustellen und das zweite um die Korrektheit zu prüfen und das Ergebnis zu publizieren.

- **Change Properties**: Jedes Objekt in WeLearn besitzt Eigenschaften, wie Name, Owner, bzw. für das Objekt typische Werte, zB bei Personen die Gruppenmitgliedschaften. Wird nun an ein Objekt der Change Properties Befehl weitergeleitet, besteht die Möglichkeit dessen Eigenschaften gleichsam von einer Metaebene zu ändern.
- **Get Stream**: Wird explizit (zB zum Download) der referenzierte Inhalt eines WeLearn Objekts angefordert (zB ZIP, ppt, pdf Dateien), wird ein GetStream Kommando versandt, das das gewünschte Ergebnis zurückliefert.

Im Speziellen beim Operate Kommando wird ersichtlich, dass WeLearn Objekte alle für ihre Arbeit nötige Funktionalität kapseln, selbst für die Erzeugung des Outputs und auch, wie in einem Folgekapitel noch besprochen werden wird, für die Prüfung der Zugriffsrechte verantwortlich sind. Der Kernel leitet einerseits die erzeugten Kommandos an die Objekte weiter ohne dabei über die Session-Validität hinausgehende Checks durchzuführen (vgl. QNX). Andererseits wird der dadurch generierte, abstrakte Output (entspricht der Struktur der auszugebenden HTML Seite) an die so genannte Render-Engine gesandt, die daraus low-level HTML Seiten gewinnt, die letztlich zum Browser zurückgesandt werden. Das eventuell dadurch entstehende Sicherheitsrisiko ist dennoch minimal, da auf die Security-Mechanismen der JVM vertraut werden kann.
Diese Regel wird nur in einem Punkt verletzt: immer dann, wenn die Bearbeitung der Eigenschaften eines Objekts beendet werden sollen. In diesem Fall wird nicht ein (wie auch immer lautendes) „Close“ Kommando an das Objekt gesandt (im konkreten Fall wird das zuvor geöffnete Eigenschaftsfenster wieder geschlossen), sondern im Kernel per Java Script die Aktion beendet, was als kleine Unsauberkeit zu verstehen ist.

4.4.4. Scheduling

Scheduling in Betriebsystemen im speziellen in modernen Multi-tasking OS wird meist in mehrere Bereiche unterteilt, die sich durch die Zeitspanne in der sie arbeiten von einander abgrenzen. Es sind daher long term, medium term und short term Scheduling zu unterscheiden, die nach *[Cal95]* wie folgt definiert sind:

- **Long Term Scheduling**: *„determines which programs are admitted to the system for execution and when, and which ones should be exited“*
- **Medium Term Scheduling**: *“determines when processes are to be suspended and resumed”*
- **Short Term Scheduling**: *“determines which of the ready processes can have CPU resources, and for how long”*

Dies ist an dieser Stelle deshalb wesentlich festzuhalten, da genau jene Betrachtungsweise einen Vergleich der WeLearn Umgebung mit Windows2000/XP in dieser Frage zulässt und sinnvoll macht. Während ein herkömmliches Betriebssystem alle drei Formen des Scheduling subsumiert, wird im zweiten Fall das short term (bzw. thread dispatching) und teilweise das medium term scheduling von der Java Virtual Machine übernommen. Der WeLearn Kernel unterstützt durch sein Scheduling Modul ein Mischform aus medium und long term Scheduling.
Im Detail bedeutet dies, dass der stetig laufende Prozess des Task Schedulers die Aufgabe hat bestimmte, vom Administrator oder System festgelegte Aktionen anzustoßen hat, die (im Normalfall, aber nicht notwendigerweise) als Backgroundjobs, also ohne Bildschirmausgabe, abgearbeitet werden. Als griffiges Beispiel für einen solchen Task wäre die garbage collection zu nennen, die allerdings im vorliegenden WeLearn System noch zu implementieren sein wird. Das Scheduling Modul selbst stellt über eine standardisierte Schnittstelle Werkzeuge zur Einbuchung und Administration von beliebigen Tasks zur Verfügung, die zu bestimmten Zeitpunkten als Kommandos für die entsprechenden WeLearn Objekte über den Kernel weitergereicht werden.

4.4.5. Output-Generierung

Eine Kernaufgabe eines Java Servlets und damit auch des WeLearn Kernels liegt in der Generierung des Outputs für den anfragestellenden Webbrowser. Dies kann auf zwei verschiedene Arten geschehen: entweder der Kernel produziert den Output selbst oder er veranlasst sein ihn umgebendes System dazu, diesen Task für ihn zu übernehmen. Da der WeLearn Kernel Teil eines komplexen Zusammenspiels unterschiedlichster WeLearn Objekte ist und vor dem Hintergrund einer möglichst flexiblen Erweiterbarkeit des Systems, kann und soll dieser auch im Sinne einer objektorientierten Denkweise den Output nicht selbst erzeugen, sondern diese Aufgabe den jeweiligen Objekte überlassen, die „selbst“ am besten wissen, wie sie darzustellen sind. Aus diesem Grund existiert die „Renderengine“ als Teil des Gesamtkonzepts, die den abstrakt generierten Output der WeLearn Objekte (den sie mit eigens entwickelten Java Klassen erzeugen) in konkrete HTML Statements umwandelt. Vorteil dieser Methode ist nicht nur, dass der Kernel unberührt von etwaigen Erweiterungen der WeLearn Objekt Kollektion ist, sondern auch, dass innerhalb der Renderengine diverse Kompatibiltätsprüfungen vorgenommen werden können und damit garantiert werden kann, dass die Ausgabe auf unterschiedlichsten Browsern (Internet Explorer, Navigator, Mozilla, etc.) von unterschiedlichsten Anbietern nahezu identisch ist.

Einzige Anforderung an den Kernel ist es daher, zunächst den WeLearn Objekte die Kommandos weiterzureichen und damit die Generierung des Outputs anzustoßen und anschließend diese Ausgabe an die Renderengine weiterzuleiten.

4.4.6. Die Gesamtarchitektur

In den obigen Kapiteln wurde in Bezug auf WeLearn sehr stark auf die Funktionalität und Aufgabe einzelner Komponenten Wert gelegt. An dieser Stelle sollen nun die betrachteten Teile zu einer Gesamtarchitektur rund um den WeLearn Kernel zusammengesetzt werden, um eine Überblick auf hoher Ebene zu geben, wie diese Teile miteinander verbunden sind bzw. miteinander kommunizieren.

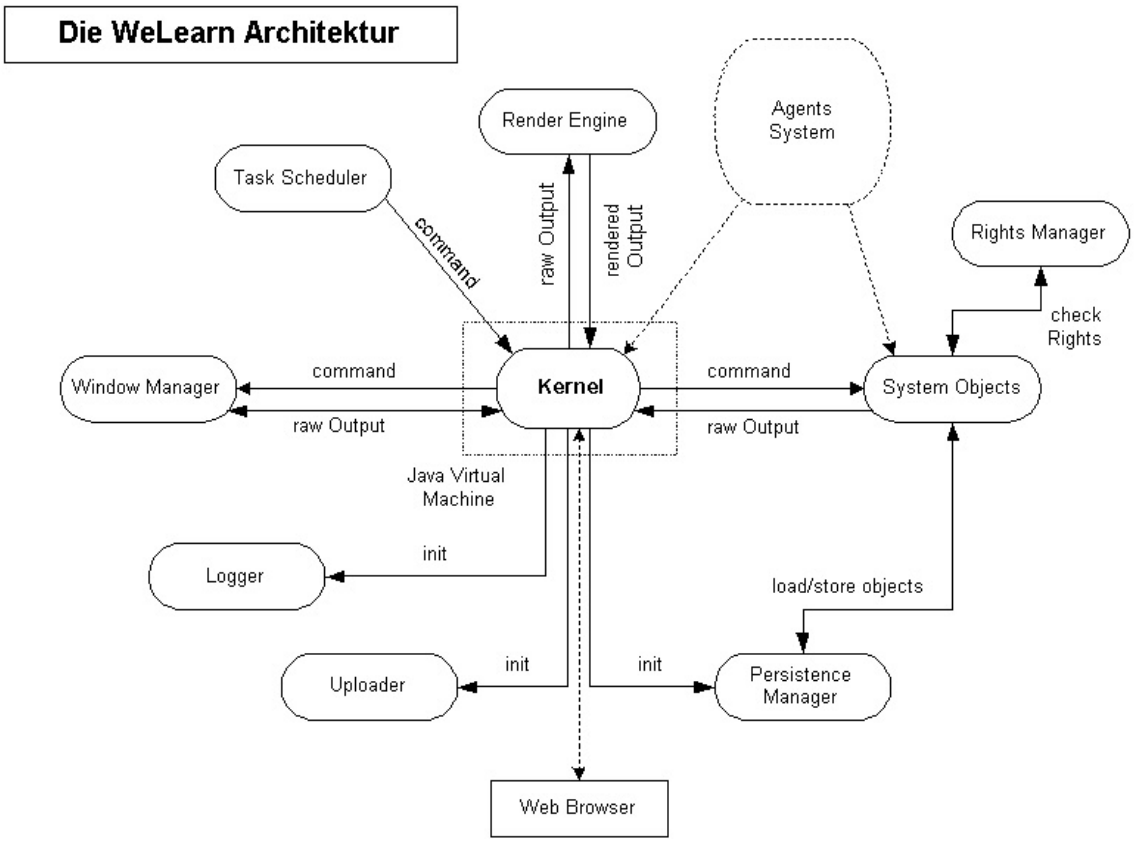

Abbildung 25 Die WeLearn Architektur

Wie in Abbildung 25 zu ersehen ist kommunizieren alle wesentlichen Module des WeLearn Systems mit Ausnahme des Rights Managers direkt mit dem Kernel. Der Windows Manager hat dabei im Wesentlichen die Aufgabe den Output so zu steuern, dass alle Frames automatisch refreshed werden, die von einer Aktion betroffen sind bzw. überprüft der Rights Manager die Rechte, die für ein spezielles Systemobject vergeben worden sind.

Am Beispiel einer einfachen Browseranfrage wird das Zusammenspiel aller Komponenten deutlich:

1. Browser sendet Anfrage an Kernel Servlet
2. Kernel interpretiert Anfrage und leitet diese weiter an den Window Manager weiter
3. Window Manager entscheidet vorab ob für ihn dieses Kommando später von Bedeutung ist, setzt aber keine Aktionen
4. Kernel leitet die Anfrage an das entsprechende Systemobject weiter
5. Systemobject überprüft ob der Benutzer die entsprechenden Rechte besitzt bzw. welche Rechte ihm zugestanden werden durch Anfrage an den Rightsmanager
6. Je nach aus Punkt 5 resultierenden Rechten, wird der Raw Output erzeugt und zurück an den Kernel gesandt
7. Kernel leitet den Raw Output an den Window Manager weiter
8. Window Manager bearbeitet Raw Output und legt fest welche Frames refreshed werden müssen und sendet veränderten Raw Output zurück an Kernel
9. Kernel leitet den Raw Output an die Render Engine weiter
10. Render Engine erzeugt die entsprechende HTML Seite
11. Kernel Servlet leitet die HTML Seite an den Browser weiter

Anhand dieses Beispiels ist einerseits klar, dass sich die Aufgaben des Kernels lediglich auf die Vermittlung von Nachrichten zwischen den Modulen beschränken und andererseits wird deutlich, dass die einzelnen Komponenten eng miteinander verbunden sind und sie ihre Aufgaben nur im Kollektiv wahrnehmen können. Eine Designentscheidung dabei war sicherlich, dass die Systemobjects selbst die Aufgabe haben, zu überprüfen ob ein Benutzer die entsprechenden Rechte zur Ausführung der Aktion besitzt. Dies führt zwar notwendigerweise zu einem erhöhten Programmieraufwand bzw. –disziplin bei der Entwicklung neuer Systemobjects, hält aber den Kernel selbst schlank und beeinflusst damit seine Wartbarkeit und Robustheit positiv, was insgesamt für die Stabilität des System maßgeblich ist.
In Abbildung 25 sind auch mögliche Schnittstellen zu Agentensystemen schematisch dargestellt, auf die in einem der Folgekapitel näher eingegangen werden wird.

4.5. Spezielle Anforderungen an den WeLearn Kernel zur Skalierung

4.5.1. Einführung

In den Vorkapiteln wurde großer Wert darauf gelegt, die verschiedenen (Betriebs)Systeme in ihren unterschiedlichen Eigenschaften isoliert zu beschreiben und insbesondere die Funktion des Kernels hervorzuheben. Bei genauerer Betrachtung kristallisierten sich daraus bereits einige grundlegende Anforderungen als Konsequenz der web-basierten Umgebung für den WeLearn Kernel heraus:

- Umgang mit einer Vielzahl von konkurrierenden Benutzern bzw. Threads
- Flexible Erweiterungsmöglichkeiten müssen angeboten werden insbesondere in Richtung Agenten
- Scheduling von besonderen High-Level Tasks
- Umgang mit den Eigenheiten vitaler Umgebungssysteme wie der JVM oder dem Webserver
- Generierung von einheitlichem, standardisiertem Output, da dieser auf verschiedensten Zielsystemen (Webbrowsern) dargestellt werden können muss

Alle diese Anforderungen benötigen besondere Behandlungen, die teilweise in den Kernel bzw. das Gesamtsystem integriert werden können wie zum Beispiel die Synchronisation von beliebig vielen Benutzeranfragen, die aber auch mit den das WeLearn System umgebenden Systemen zusammenhängen (JVM, Webserver) und somit spezieller Betrachtung bedürfen. Insbesondere die Verwendung eines Java Servlets für den Kernel unterscheidet sich im Verhalten eklatant von dem Kernel eines Betriebssystems.

4.5.2. Grundlegende Unterschiede zum OS Kernel

Bei jedem OS wird von einer Prämisse ausgegangen: der Kernel stellt ein Stück Software dar, das nie aus dem Speicher der Maschine entfernt wird und somit zu jeder Zeit bei beliebigen Systemzuständen zur Verfügung steht, seinen eigenen, inneren Zustand immer kennt und Anfragen damit, basierend auf der „Vergangenheit“, stets positiv und sinnvoll behandelt. Ganz im Gegensatz dazu steht die Grundphilosophie eines Servlets. Hier kann das Servlet nicht selbst entscheiden oder bestimmen, dass es immer zur Verfügung steht, denn es wird vom Webserver gestartet und befindet sich anschließend in dessen Memory und unterliegt somit der Speicherverwaltung des Webservers, auf den das Servlet keinerlei Einfluss besitzt. Es kann also sein, dass während der Webserver selbst wochenlang durchläuft zwischenzeitlich ein, fünf oder zehn mal das Kernel Servlet der WeLearn Plattform geladen (und damit gestartet wird mit allen dazugehörigen Initialisierungen, etc) und wieder beendet und aus dem Speicher entfernt wird. Trotz dieser Tatsache soll die Funktionsweise für den Benutzer selbstverständlich unverändert und transparent bleiben. Es muss also einerseits garantiert werden können, dass das System zu jeder Zeit konsistent gehalten ist, da immer damit zu rechnen ist, dass unerwartet der Kernel nicht mehr zur Verfügung steht, und andererseits beim Wiederanlauf des Kernels der vorherige Zustand wiederhergestellt werden kann. Aus dieser Überlegung heraus haben sich einige (einschränkende) Anforderungen an den Kernel selbst aber auch an einzelne Teile des Gesamtsystems ergeben, zum Beispiel:

- Der Cache des Persistenzmanagers muss mit einer Write-Through Strategie arbeiten, weil anderenfalls Daten unwiderruflich verloren gehen könnten
- Die bei Java Servlets grundsätzlich vorgesehene destroy-methode (die beim Shutdown ausgeführt wird) enthält keine Statements, weil Java generell nicht garantiert, dass solche methoden aufgerufen werden. Diese Tatsache bedeutet allerdings, dass der Kernel selbst keine Datenstrukturen im Speicher halten darf, die bei plötzlicher Beendigung desselben Inkonsistenzen des Gesamtsystems hervorrufen könnten
- Der interne WeLearn Scheduler kann sich nicht auf eigene Zeitangaben stützen, sondern muss stets zB auf die Systemzeit vertrauen

Diese Beispiele zeigen bereits, dass nicht nur die Entwicklung des WeLearn Kernel keinesfalls straight forward geschehen kann, sondern auch alle anderen Teile des Systems nur durch präzise geplantes Vorgehen in der Entwicklung solche (in Wirklichkeit) k.o. Kriterien minimieren können.
Darüber hinaus müssen die Initialisierungsroutinen, von denen bereits die Rede war, insbesondere auf dieses Verhalten Rücksicht nehmen und flexibel auf den aktuellen Systemzustand reagieren können.
Allein die Entscheidung für den Kernel des WeLearn Systems ein Java Servlet zu verwenden hat also gravierende Auswirkung auf beinahe alle anderen Teile des Systems sowohl in der Entwurfs- als auch in der Implementierungsphase.

4.5.3. Die Startphase des Kernel Servlets

Die Startphase des Kernel Servlets hat deshalb besondere Bedeutung, da sie unter Umständen mehrfach und zu unbestimmten Zeitpunkten passieren kann. Dies geschieht genau dann, wenn einerseits das Servlet nicht im Speicher des Webservers vorhanden ist, und zugleich ein oder mehrere Benutzer Anfragen an den WeLearn Kernel stellen. Insbesondere gefordert sind hier die Initialisierungsroutinen der einzelnen WeLearn Komponenten, die sowohl beim gänzlichen Neustart des Systems als auch beim Aufsetzen auf beliebige Systemzustände korrekt funktionieren müssen. Es muss daher zu Beginn des Kernelstarts, im Unterschied zu herkömmlichen Betriebssystemen (wo der Start im Wesentlichen immer gleich verläuft), festgestellt werden ob bereits ein System existiert; wenn ja, werden bestimmt Basisinitialisierungen übersprungen (zB das Erzeugen von Filesystem Ordnern) und lediglich die Existenz und Funktionstüchtigkeit aller vitalen Teile des Systems sichergestellt (Persistenzmanager, Uploader, Logger, virtuelles Filesystem, richtige Initialisierung aller Kernelattribute wie eine Referenz zum „Upload“ Folder, etc).
Die Startphase ist auch deshalb entscheidend, weil hier aus Gründen der Responsetime-Minimierung nur die aller notwendigsten Klassen geladen werden sollen, alle anderen werden bei Bedarf dann ohnehin nachgeladen von der Java Virtual Machine.

4.5.4. Erweiterbarkeit des Gesamtsystems

Das WeLearn Learning and Teaching Environment ist auf Grund des angestrebten eingeschränkten „open source” Status, insbesondere in Bezug auf mögliche Erweiterungen gefordert. Dies kann im Wesentlichen zwei Bereiche betreffen:

- **Intern**: Erweiterung der Funktionalität (Systemobjects), der unterstützten Standards, etc.
- **Extern**: Agentenschnittstellen

4.5.4.1. Interne Erweiterungen

Das Zusammenspiel der einzelnen Systemobjekte ergibt gesamt gesehen die Funktionalität des WeLearn Systems. Der integrierte FileManager des virtuellen Dateisystems ist genauso ein Systemobjekt wie die Diskussionsforen oder die CPS Paketdarstellung. Eine Erweiterung in diesem Bereich wirkt sich also direkt auf den Funktionsumfang aus und damit auf die Flexibilität und Benutzerfreundlichkeit, was wiederum positiv auf die Akzeptanz einwirkt. Dennoch sind bei solchen Erweiterungen einige Dinge zu beachten, die bei der Implementierung von Bedeutung sind. Wie in Abbildung 25 ersichtlich ist, kommunizieren diese Objekte nämlich nicht nur mit dem Kernel, sondern auch mit dem RightsManager und dem PersistenceManager. Es sind daher folgende Dinge zu beachten:

- **Capabilities**: das SystemObject bestimmt selbst, welche Kommandos es akzeptiert bzw. welche Grundeigenschaften es besitzt über die so genannten Capabilities. Dabei muss für jeden gültigen Befehl die entsprechende Implementierung vorgenommen werden. Selbstverständlich sind auch beliebige (sinnvolle) Kombinationen möglich:
    - Browsable: Objekte, die browsable sind, können im virtuellen Dateisystem Kinder besitzen, zB Folders.
    - Executable: Objekte, die executable sind besitzen Funktionalität die ohne weitere Benutzerinteraktion einmal ausgeführt werden, zB der Befehl „copy“.
    - Operable: die wichtigste Capability in Bezug auf Benutzerinteraktion. Objekte, die operable sind können (auch über mehrere Schritte hinweg) mit dem Benutzer interagieren, besitzen meist einen inneren Zustand und führen abhängig von diesem Zustand und der Benutzereingabe diverse Funktionen aus. Darüber hinaus wird ebenfalls zustandsabhängig der Output generiert.
    - ChangeProperties: Jedes Systemobjekt akzeptiert im Default dieses Kommando zum Ändern der Objekteigenschaften.
    - Duplicable: Zeigt an, das ein Objekt duplizierbar ist. Ebenfalls „true“ im Default.

    - Streamable: Objekte die Streamable sind besitzen eine Referenz auf externe Dateien wie ppt-slides. Dieses Kommando wird im WeLearn daher für die FileWrapper-Systemobjekte verwendet.
- **RightsManager**: Jedes Systemobjekt ist selbst dafür zuständig, zu überprüfen ob ein Benutzer die erforderlichen Rechte besitzt, um verschiedene Kommandos auszuführen. Diese Prüfung geschieht durch den RightsManager, von dem systemweit nur eine einzige Instanz existiert, die von allen Objekten benutzt werden kann.
- **PersistenceManager**: Der PersistenzManager speichert und lädt alle Systemobjekte. Er ist transaktionsorientiert und muss exklusiven Zugriff zumindest auf einzelne Felder des Systemobjekts garantieren. Es ist daher nötig bei der Implementierung der Zugriffsmethoden auf diese Felder, diese in Transaktionen des PersistenzManagers zu „verpacken“, um Inkonsistenzen zu vermeiden.
- **Kernel**: Der Kernel sendet Kommandos zu den Systemobjekten, die von diesen verarbeitet werden und das Ergebnis dieser Arbeit als Output zurückgeliefert wird.

Systemobjekte spielen also die zentrale Rolle in der Interaktion mit dem Benutzer einerseits, aber auch im Zusammenspiel wesentlicher Systemkomponenten und müssen daher mit großer Sorgfalt entwickelt werden. Unsachgemäßer Umgang kann zu Sicherheitslöchern und unangenehmen Seiteneffekten führen.

4.5.4.2. Agentenschnittstellen

Bevor die Schnittstelle zwischen WeLearn und Agentensystemen näher betrachtet wird, sollte der Begriff „Agent“ per se bestimmt sein, der nach *[OMG00]* wie folgt definiert wird: *„An agent ia a computer program that acts autonomously on behalf of a person or organisation. Each agent has its own thread of execution so tasks can be performed on its own initiative.“* Bei der Schnittstellengestaltung zu solchen Systemen muss, dieser Beschreibung folgend, demnach davon ausgegangen werden, dass keinerlei Informationen über Absicht oder Implementierung des Agenten zur Verfügung stehen und daher eine möglichst generalistische Lösung bei der Realisierung angestrebt werden. Besonderes Augenmerk wird darüber hinaus auf Sicherheitsaspekte zu legen sein, weswegen die Agentenanbindung nicht gänzlich gelöst vom restlichen WeLearn Framework gesehen werden kann. Insbesondere dann nicht, wenn Wert auf den letzten Absatz des Vorkapitels gelegt wird. Externe Software, die im allgemeinen Fall als nicht vertrauenswürdig zu bewerten ist, könnte durch dem Umstand, dass keine Zentralen Securtiy – Mechanismen im Kernel vorgesehen sind, unerlaubten Zugriff auf Systemressourcen erlangen ohne dass das System es bemerkt.

Optimistischer betrachtet erweitern Agenten das Einsatzgebiet und die Flexibilität des Frameworks erheblich und bieten daher Mehrwert für alle Anwender. Chancen und Risken einer solchen Agentenanbindung werden aber erst dann ersichtlich, wenn die technischen Realisierungsmöglichkeiten in Betracht gezogen sind. Zwei Ansätze zeigen sich dabei als überlegenswert (vgl. Abbildung 25):

- **Anbindung direkt an den Systemobjekten**: Diese Variante geht davon aus, dass direkt auf die im realen Dateisystem abgelegten Systemobjekte zugegriffen werden kann. Jede Schnittstelle, die sich dieser Methode bedient, müsste demnach auch selbst dafür Sorge tragen, dass zB alle Zugriffsrechte überprüft werden. Dazu kommt, dass gleichzeitig beliebige Anfragen an beliebige Systemobjekte, aus Sicht der Agentenschnittstelle unbemerkt, „passieren“ können, was unweigerlich zu Problemen in Bezug auf Synchronisation und Persistenz führt, die wahrscheinlich nur durch erheblichen Zusatzaufwand gelöst werden können. Besser ist aus meiner Sicht daher die:
- **Anbindung am Kernel**: Die Vorteile der Anbindung der Agenten am Kernel liegen klar auf der Hand: Agenten kommunizieren mit System-Objekten auf dem „normalen“ Weg, dh über das integrierte MessagePassing bzw. die bereits

beschriebenen Abläufe garantieren Persistenz und die Prüfung der Rechte. Die Agenten fügen sich problemlos in das Gesamtsystem ein.

Da die Agentenkommunikation allerdings asynchron und zusätzlich nicht über WeLearn Kommandos abläuft, wird ein eigenes „externes Programm“, das vom Kernel gestartet werden soll, die Schnittstelle zwischen WeLearn und beliebigen Agenten darstellen müssen. Dieses externe Programm könnte mit dem WeLearn Kernel einerseits und mit den Agenten andererseits zB per XML kommunizieren. Solche XML Daten werden direkt auf Port 80, also per HTTProtokoll ausgetauscht oder mit Hilfe von HTML getunnelt, um Irritation bei diversen Webservern zu vermeiden, die das Weiterleiten von reinem XML nicht unterstützen.

Wesentlich ist allerdings, dass dieses externe Programm ständig als „Andockstation“ für Agenten läuft, also auch dann, wenn das Java Servlet (aus bereits beschriebenen Gründen) nicht aktiv ist!!! Dies wird neben den Security-Aspekten wohl der entscheidenste Teil der Implementierung der Agentenschnittstelle sein. Es muss garantiert werden, dass Agenten zu jeder Zeit auf das System Zugriff haben.

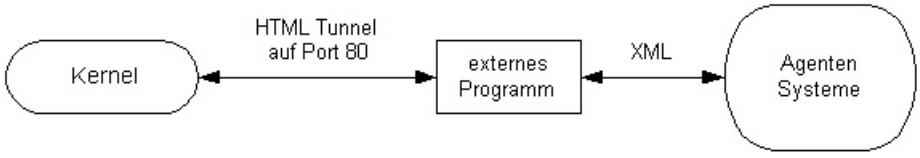

Abbildung 26 Mögliche Agentenanbindung über den WeLearn Kernel

Die sich aus einer wie auch immer implementierten Agentenanbindung ergebenden Möglichkeiten der Erweiterung des Systems werden jedenfalls zur Akzeptanz und Verbreitung des Frameworks maßgeblich beitragen.

4.5.5. Scheduling

Vorgesehen nach Fertigstellung des Gesamtsystems ist auch ein Task – Scheduler, der eventgesteuert, das System asynchron veranlasst, bestimmte Aktionen durchzuführen. Als Beispiel bereits angeführt wurde die garbage collection, gestartet als Backgroundjob, es können dadurch aber beliebige Aufgaben wahrgenommen werden. Denkbar wäre in diesem Zusammenhang eine Art Notifikationsmechanismus per email, sollten neue Einträge oder Antworten in einem Forum bestehen, oder eine Art persönliche Mailbox innerhalb des Systems, die vom User administrierbar diesen über Neuigkeiten in Kursmaterialien informiert. In wie weit dieser Scheduleransatz von ähnlichen Agentensystemen übernommen werden kann, die den Task Scheduler obsolet werden lassen, wird die konkrete Entwicklung zeigen. Sinnvoll erscheint allerdings, einen einheitlichen Ansatz zu wählen, um Doppelgleisigkeiten zu vermeiden.

4.5.6. Die Shutdownphase des Kernel Servlets / Webservers

Wie bereits in vorigen Kapiteln besprochen kommt der Shutdownphase sowohl des Kernel Servlets als auch des Webservers an sich besondere Bedeutung in Bezug auf Stabilität und Konsistenz des WeLearn Systems zu. Wird das Servlet um Webserver aus dem Speicher genommen, so hat dies per se keine negativen Auswirkungen auf die Konsistenz des Systems, da in diesem Fall davon ausgegangen werden kann, dass keine Sessions über einen gewissen Zeitraum aktiv waren und somit auch keine laufenden Transaktionen existieren, die Probleme bereiten könnten.
Anders verhält sich die Sache, wenn aus welchen Gründen auch immer der Webserver selbst heruntergefahren wird. Hier sind im Grunde zwei Situationen zu unterscheiden:

- Shutdown bei aktiven Transaktionen
- Shutdown ohne aktive Transaktionen

Letzteres kann zwar für einen Benutzer unangenehm sein, der aus für ihn unersichtlichen Gründen in Mitten der Arbeit mit dem System keinen Zugriff mehr erhält und den „berühmten“ Fehler 404 („Diese Seite kann nicht angezeigt werden“) geliefert bekommt, es hat aber keine negativen Auswirkungen auf das System an sich.
Ganz anderes stellt sich die Situation dar bei aktiven Transaktionen, denn hier muss unterschieden werden zwischen

- Graceful Shutdown
- Non Graceful Shutdown

Im Falle des Graceful Shutdowns werden, bevor der Server herunterfährt, alle aktiven Requests fertig serviciert und somit im Falle von WeLearn alle Transaktionen beendet, was einem konsistenten Systemzustand gleichkommt, da alle verändertet Systemobjekte zu diesem Zeitpunkt persistiert worden sind. Sollte allerdings ein Non Graceful Shutdown angestoßen werden während aktive Transaktionen Systemobjekte verändert aber noch nicht gespeichert haben, kann es zu groben Inkonsistenzen kommen. Sie haben im worst case zur Folge, dass das System nicht wieder angefahren werden kann.
Diese „Entscheidung“, Graceful/Non Graceful Shutdown des Webservers, kann aber nicht vom WeLearn System selbst beeinflusst werden, sondern liegt allein am Web-Administrator. Es ist daher von unschätzbarer Wichtigkeit, diesen auf diese Tatsache in

geeigneter Form Aufmerksam zu machen, sei es durch Betriebshandbücher, Schulungen, etc., damit (auch gewisse Unerfahrenheit im Umgang) keine negativen Auswirkungen im laufenden Wartungsbetrieb des WeLearn Systems hat, was naturgemäß die Akzeptanz der Plattform erheblich beeinflussen würde.

4.5.7. Standardisierter Output

Benutzer haben unterschiedliche Vorraussetzungen und Zugänge zu e-Learning. Dies betrifft sowohl die Vorbildung im Sinne eines selbstgesteuerten Lernens und Akzeptanz desselben als auch die technischen Möglichkeiten. Eine e-Learning Plattform wird demnach in verschiedensten Umfeldern eingesetzt, von Informatikinstituten der Universität mit hohem Wissenstand in diesem Gebiet bis zur Pflichtschule wo e-Learning als Schulversuch eingesetzt wird (ohne dabei eine qualitative Wertung vornehmen zu wollen).

Wesentlich in beiden Fällen ist allerdings ein standardisierter Output in Bezug auf das optische Erscheinungsbild, das eine gewisse Intuitivität und damit Benutzerfreundlichkeit zulässt, ganz gleich welcher Bereich der Plattform gerade betrachtet oder welcher Browser verwendet wird. Im WeLearn System zeigt die RenderEngine verantwortlich für beide Anforderungen. Sie garantiert zum einen, dass der Output (sprich die HMTL Datei, die an den Browser geschickt wird) von allen gängigen Browsern gleich interpretiert und damit angezeigt wird, und zum zweiten wird den Systemobjekten selbst nur die Möglichkeit eingeräumt die Struktur und den Inhalt der Webseiten zu bestimmen, nicht aber die daraus resultierende HTML Seite zu generieren. Diese Aufgabe kommt einzig und allein der RenderEngine zu, die vom Systemobjekt über den Kernel den „raw Output“ bekommt und zu „echten“ HTML Seiten zusammenstellt.

Dieses Konzept ist auch in Bezug auf Wartbarkeit und Erweiterung zB der unterstützten Browser mehr als nur hilfreich (und im Übrigen in allen Betriebssystemen auf die eine oder andere Weise implementiert).

5 Ausblick

Die Akzeptanz eines e-Learning Systems hängt selbstverständlich ab von gewissermaßen generellen Kriterien wie der offerierten Benutzerfreundlichkeit, intuitiven Anwendung oder Administrierbarkeit, aber es darf auch auf den Kostenfaktor und die unterschiedlichsten Einsatzgebiete nicht vergessen werden, die insbesondere bei der vorliegenden Zielgruppe (Schulen, Universitäten, Fachhochschulen, etc.) eine gewichtige Rolle spielen.

WeLearn wird diesen Anforderung durch den angestrebte Open-Source Gang, den flexiblen Adaptierungsmöglichkeiten der bestehenden Module und die allgemeine Kostenfreiheit im Wesentlichen gerecht, allerdings existiert noch viel Potenzial in der Weiterentwicklung der WeLearn internen Architektur.

5.1. Schwachstellen in der WeLearn Architektur

Schwachstellen eines Systems hängen stark vom Zugang und der Sichtweise ab. Benutzer werden andere Mängel identifizieren als Systemadministratoren oder Implementierer. Für zukünftige Entwicklungen des Systems in Richtung flexibler Erweiterbarkeit in möglichst alle Richtungen (interne Funktionalität, externe Schnittstellen) sind besonderes die internen Architekturentscheidungen und deren Konsequenzen von Bedeutung:

- **Rechte** werden nicht im Kernel (wie bei WeLearn Release 1), sondern von den Systemobjekten selbst überprüft (vgl. Abbildung 25): dies bedeutet einen erheblichen Mehraufwand bei der Entwicklung von neuen Systemobjekten, da Anfragen vom Kernel ohne „Kontrolle“ einfach weitergeleitet werden. Dazu kommt, dass der Implementierer auch genau über die Funktionsweise des RightsManagers bescheid wissen muss.

  Änderungen der Schnittstelle des RechteManagers müssen in allen (!) Systemobjekten nachgezogen werden, was einen massiv erhöhten Wartungsaufwand zur Folge hat.

  Wünschenswert wäre hier eine weitgehende Zentralisierung der Rechteprüfung im Kernel – auch im Hinblick auf Erweiterungen wie Agentenschnittstellen.
- Systemobjekte erzeugen den abstrakten **Output** selbst: Systemobjekte kapseln nicht nur Funktionalität, sondern beinhalten auch gleichzeitig die Generierung des abstrakten Outputs (der dann vom Kernel an die RenderEngine weitergereicht wird). Diese Vermengung führt zu „aufgeblähten“ und damit undurchsichtigen Codestücken, die nur schwer wartbar und erweiterbar sind (im Regelfall nur vom Entwickler selbst).

  Hier würde eine Trennung zwischen Funktionalität und Outputerzeugung nicht nur die Wartbarkeit erhöhen und Doppelgleisigkeiten vermeiden, sondern erzeugte auch eine klare Strukturvorgabe für die Entwicklung von zusätzlichen Modulen von WeLearn.
- Generell ist viel **Know-How** zur Entwicklung von neuen SystemObjekten nötig: Um neue Systemobjekte implementieren zu können, ist das Wissen um die Funktionalität vieler Bereiche des Systems nötig. So muss zum Beispiel der Umgang mit dem Persistenzmanager im Detail gekannt werden, um überhaupt das System dazu zu bewegen, ein Objekt abzuspeichern und nicht eine Flut von

(für Außenstehende wenig verständlichen) Exceptions zu erzeugen. Die Problematik rund um den RightsManager wurde bereits angesprochen.

- Alle **persistierten Dateien** werden in einem einzigen FileSystem Ordner abgelegt: Dies führt bei unterschiedlichen FileSystem-Architekturen im Hintergrund zu verschiedenen Antwortzeiten des Systems und lässt damit keine generell gültige Aussage mehr über die Leistungsgrenze (im Sinne einer Obergrenze von persistierten Dateien) von WeLearn zu.
  Intelligente Ordnerstrukturen, zB mit Hilfe der Idee des Hashings, könnte hier Abhilfe schaffen und so einen Flaschenhals vermeiden.
- In **FileWrapper Objekten** wird der absolute Pfad der referenzierten Datei abgelegt: dies macht eine einfache Portierung eines Systems beinahe unmöglich oder zumindest äußerst schwierig, da am Zielsystem dieselben Ordner und Laufwerksstrukturen existieren müssen.

Diese angesprochenen Defizite können aus meiner Sicht zum überwiegenden Teil nur durch erheblichen Mehraufwand in WeLearn Release 2 behoben werden und sind daher als Basis für zukünftige Entwicklungen zu verstehen, wobei auch festgehalten werden muss, dass es sich hier um Designentscheidungen handelt, die die Implementierungsebene betreffen und für den externen Benutzer weitgehend unsichtbar bleiben.

5.2. Entwicklungspotenzial

Auf Grund der Tatsache, dass WeLearn partiell als Open-Source Lehr- Lernplattform konzipiert wurde, wird die Verbreitung und Akzeptanz stark von den Möglichkeiten der individuellen Adaptierung und Erweiterung abhängen. Ich sehe daher das größte und entscheidenste Entwicklungspotential in der Zurverfügungstellung von Schnittstellen zu Agentensystemen einerseits und (für die Entwickler) einfachen, internen Funktionsergänzungen andererseits.

Eine weitere Überlegung zu WeLearn Release 2 betrifft das Setup Script. Hier könnte tatsächlich durch einen geschickt implementierten Konfigurationswizard, aus einem ursprünglich als Hilfsmittel zur Erzeugung von wesentlichen „Basisobjekten" für WeLearn gedachten Script, ein Tool geschaffen werden, mit dem bereits bei der Installation des Systems verschiedenste Lernszenarien abgebildet werden. Zum Beispiel könnten User, Folder, Foren, etc. vorab angelegt werden und „Standardszenarien" mit der WeLearn Software mitausgeliefert werden.

Die grundsätzlichen Vorraussetzung für den Erfolg von WeLearn sind jedenfalls gegeben:

- Kostenfreiheit (unter bestimmten Vorraussetzungen)
- Publizierte Schnittstellen
- Unterstützung des IMS Standard
- Professionelles Umfeld

Aus meiner Sicht liegt es nun daran, die gewonnenen Erfahrungen sowohl von Benutzer- als auch von Implementierungsseite zu nutzen und zu einem Gesamtpaket zu kombinieren, das nicht mehr als ein Basisframework zur Verfügung stellt, das aber auf einfache Art und Weise jegliche gewünschte Weiterentwicklungen erlaubt.

6 Anhang

6.1. Code Listing: WeLearn Release 2 Kernel

```
/*
 * Kernel.java
 * Created on 12. August 2002, 12:25
 */

package at.jku.fim.welearn.system;

import at.jku.fim.welearn.objects.Folder;
import at.jku.fim.welearn.objects.user.LdapGroupsFolder;
import at.jku.fim.welearn.objects.Shell;
import at.jku.fim.welearn.objects.user.Person;
import at.jku.fim.welearn.objects.user.UsersFolder;
import at.jku.fim.welearn.objects.file.FileSettings;
import at.jku.fim.welearn.objects.scheduler.*;
import at.jku.fim.welearn.objects.scheduler.impl.SimpleSchedulerFactory;
import at.jku.fim.welearn.objects.scheduler.test.BackupTask;
import at.jku.fim.welearn.objects.user.LdapUsersFolder;
import at.jku.fim.welearn.objects.winman.WindowManager;
import at.jku.fim.welearn.objects.winman.DefaultWindowManager;
import at.jku.fim.welearn.rendering.elements.SystemObjectOutput;
import at.jku.fim.welearn.rendering.elements.SingleOutput;
import at.jku.fim.welearn.rendering.elements.basic.JavaScript;
import at.jku.fim.welearn.rendering.engine.RenderEngine;
import at.jku.fim.welearn.rendering.engine.html.StandardDesign;
import at.jku.fim.welearn.system.object.AccessViolationException;
import at.jku.fim.welearn.system.object.InvalidOidException;
import at.jku.fim.welearn.system.object.ObjectIdentifier;
import at.jku.fim.welearn.system.object.SpecialObjectException;
import at.jku.fim.welearn.system.object.SystemObject;
import at.jku.fim.welearn.system.object.command.ChangeProperties;
import at.jku.fim.welearn.system.object.command.Execute;
import at.jku.fim.welearn.system.object.command.Operate;
import at.jku.fim.welearn.system.object.command.GetStream;
import at.jku.fim.welearn.system.rights.RightsManager;
import at.jku.fim.welearn.system.rights.Permission;
import at.jku.fim.welearn.system.rights.Acl;
import at.jku.fim.welearn.system.rights.PermissionException;
import at.jku.fim.welearn.persistence.ObjectNotFoundException;
import at.jku.fim.welearn.persistence.PersistenceException;
import at.jku.fim.welearn.persistence.PersistenceManager;
import at.jku.fim.welearn.persistence.disk.DiskManager;
import at.jku.fim.welearn.system.multipart.MultipartRequestWrapper;
import java.io.*;
import java.util.*;
import java.net.*;
```

```
import javax.servlet.*;
import javax.servlet.http.*;
import org.apache.log4j.Logger;
import org.apache.log4j.BasicConfigurator;
import org.apache.log4j.PropertyConfigurator;
```

```
/**
 * Receives all HTTP requests and dispatches them to SystemObjects.
 */

public class Kernel extends HttpServlet {
    /* The class-specific logger instance */
    private static final Logger logger = Logger.getLogger( Kernel.class );

    /* The short description message for this servlet. */
    private static final String SHORT_DESCRIPTION =
            "WeLearn R2 - (c) 2002 by FIM";

    /* The moniker used to identify a "data" request */
    public static final String DATA_REQUEST_MONIKER = "wldata";

    /* The name of the special SystemObject <code>USERS_FOLDER</code>. */
    private static final String USERS_FOLDER_NAME = "users";

    /* The name of the special SystemObject <code>USER_EVERYBODY</code>. */
    private static final String USER_EVERYBODY_NAME = "everybody";

    /* The name of the special SystemObject <code>USER_OWNER</code>. */
    private static final String USER_OWNER_NAME = "owner";

    /* The name of the special SystemObject <code>USER_SYSTEM</code>. */
    private static final String USER_SYSTEM_NAME = "system";

    /* The name of the special SystemObject <code>LDAPUSERS_FOLDER</code>. */
    private static final String LDAPUSERS_FOLDER_NAME = "ldapusers";

    /* The name of the special SystemObject <code>GROUPS_FOLDER</code>. */
    private static final String GROUPS_FOLDER_NAME = "groups";

    /* The name of the special SystemObject <code>LDAPGROUPS_FOLDER</code>. */
    private static final String LDAPGROUPS_FOLDER_NAME = "ldapgroups";

    /* The name of the special SystemObject <code>RIGHTS_FOLDER</code>. */
    private static final String RIGHTS_FOLDER_NAME = "rights";

    /* The request count query string parameter. */
    private static final String REQUEST_COUNT = "rc";

    /* The name of the environment variable <code>USERNAME</code>. */
    public static final String USERNAME_VAR = "USERNAME";

    /* The name of the environment variable <code>LANG</code>. */
    public static final String LANG_VAR = "LANG";

    /* The query string used for the kernel command parameter. */
    public static final String COMMAND_MONIKER = "krnlcmd";

    /* The kernel command paramater OPERATE. */
```

```
public static final String OPERATE_CMD = "operate";

/* The kernel command parameter CHANGE PROPERTIES. */
public static final String CHANGE_PROPERTIES_CMD = "chgprp";

/* The kernel command parameter GET RESOURCE. */
public static final String GET_STREAM_CMD = "stream";

/* The oid query string parameter. */
public static final String OID_PARAM = "oid";

/* The instanced quere string parameter. */
public static final String INSTANCE_PARAM = "inst";

/* The name of the active list used as session attribute. */
public static final String ENVIRONMENT = "environment";

/* The init parameter determining the default uploading directory. */
public static final String UPLOAD_ROOT = "UPLOAD_ROOT";

/* The init parameter determining the root dir of the PM. */
private static final String DISKPM_ROOT = "DISKPM_ROOT";

/* The init parameter determining the usage of the disk-pm cache. */
private static final String DISKPM_CACHE = "DISKPM_CACHE";

/* The init parameter determining the timeout for locking. */
private static final String LOCKING_TIMEOUT = "LOCKING_TIMEOUT";

/* The SystemObject to be executed when no oid is specified. */
private static final String STARTUP = "STARTUP";

/* The default window manager. */
private static final String DEFAULT_WINDOW_MANAGER = "system/winman";

/* Initialization setting for the window manager */
private static final String WINDOW_MANAGER = "WINDOW_MANAGER";

/* The default value for <code>STARTUP</code>. */
private static final String DEFAULT_STARTUP = "/system/startup";

/* The path and name of the setup script. */
private static final String SETUP_SCRIPT = "SETUP_SCRIPT";

/* The default value for the <code>SETUP_SCRIPT</code> context parameter.*/
private static final String DEFAULT_SETUP_SCRIPT = "setup.scr";

/* The default value for the <code>SETUP_SCRIPT</code> context parameter.*/
private static final String FILE_SETTINGS= "FileSettings";

/* The default timeout value for locking. */
private static final int DEFAULT_TIMEOUT = 5000;
```

```
/* The ObjectIdentifier of the start object. */
private ObjectIdentifier startId;

/* The RenderEngine used by the kernel to produce output. */
private RenderEngine renderEngine = new StandardDesign();

/* The root directory for file uploading */
private File uploadDirectory;

/* The scheduler instance (we need it to shutdown the sceduler upon exit). */
private Scheduler taskScheduler;
```

```
/**
 * Initializes the servlet.
 * @param config the <code>ServletConfig</code> object that contains
 * information used to configure  this servlet.
 * @throws ServletException if an exection occurs that interrupts the
 * servelt's normal operation
 */

public void init(ServletConfig config) throws ServletException {
    super.init(config);

    /* store the "proper" path to this servlet in the associated context */
    getServletContext().setAttribute( "welearn.servlet.name",
                                      config.getServletName() );

    /* initialize the logger module */
    initLogger();

    /* initialize the uploader module */
    initUploader();

    /* initilaize the PersistenceManager */
    initPm();

    /* initalize the system. */
    if ( !isSystemInitialized() ) {
        initializeSystem();
    }

    /* get the ObjectIdentifier of the start object */
    String start = getInitParameter(STARTUP);
    if ( start == null ) {
        start = DEFAULT_STARTUP;
    }

    SystemPath systemPath = new SystemPath(ObjectIdentifier.ROOT_FOLDER);
    SystemObject sysobj;
    try {
        sysobj = systemPath.findObject(start);
    }
    catch (PersistenceException e) {
        throw new ServletException("error loading the start object", e);
    }
    catch (InvalidPathException e) {
        throw new ServletException("invalid path to the start object", e);
    }

    if ( sysobj == null ) {
        throw new ServletException("start object not found");
    }
    startId = sysobj.getOid();
   }
```

```
/**
  * Destroys the servlet.
  */

public void destroy() {
}
```

```
/**
 * Processes requests for both HTTP <code>GET</code> and <code>POST</code>
 * methods.
 * @param request servlet request
 * @param response servlet response
 * @throws ServletException if an exection occurs that interrupts the
 * servelt's normal operation
 * @throws IOException if an input or output error is detected when the
 * servlet handles the request
 */

protected void processRequest(HttpServletRequest request,
                              HttpServletResponse response)
    throws ServletException, java.io.IOException
{
    if ( !request.isRequestedSessionIdValid() ) {

        HttpSession forcedSession = request.getSession();
        forcedSession.setMaxInactiveInterval( 20*60 );
        String url = response.encodeRedirectURL( request.getServletPath() );
        if ( url.endsWith( Kernel.DATA_REQUEST_MONIKER ) ) {
            url = url.substring( 0, url.lastIndexOf( Kernel.DATA_REQUEST_MONIKER ) ) +
                  request.getSession().getServletContext().getAttribute(
                                                          "welearn.servlet.name" ) ;
        }
        causeRedirect( request, response, url );
        return;

    }

    /* initialize the current session and get the active-list */
    HttpSession session = request.getSession();

    /* Check if the session is new; if so, don't try to interpret any
       parameters it may be carrying along (might be the result of an
       expired session's request. */
    if ( !session.isNew() ) {
        if ( isDataRequest(request) ) {
            this.handleDataRequest(request, response);
            return;
        }
        /* 2002.08.19 - alpar : Handle multipart requests containing files */
        String requestType = request.getHeader( "Content-Type" );
```

```
        /* if the type if multipart/form-data, wrap the original request */
        if ((requestType != null) && requestType.startsWith( "multipart/form-data" )) {
            try {
                MultipartRequestWrapper wrapper = new MultipartRequestWrapper( request,
                                                    uploadDirectory.getAbsolutePath() );
                request = wrapper;
            }
            catch ( Exception unused ) { }
        }

        /* get the command to perform */
        String cmd = request.getParameter(COMMAND_MONIKER);
        if ( cmd == null ) {
            cmd = OPERATE_CMD;
        }
        if ( cmd.equals(OPERATE_CMD) ) {
            doOperate(request, response);
        } else if ( cmd.equals(CHANGE_PROPERTIES_CMD) ) {
            doChangeProperties(request, response);
        } else if ( cmd.equals(GET_STREAM_CMD) ) {
            doGetStream(request, response);
        }
    } else {
        /* We have a new session; either a new user, or an expired session.
           Force an Operate command, which should further make sure that the
           rest of the details involved are handled properly */
           doOperate(request, response);
    }

}
```

```
/**
 * Utility method that decides whether a request is a special "data" request
 * @request the <code>HttpServletRequest</code> to be checked
 * @return <code>true</code> if the parameter is a "data" request,
 *         <code>false</code> otherwise
 */

private boolean isDataRequest(HttpServletRequest request) {
    if ( request.getRequestURI().indexOf("/" + DATA_REQUEST_MONIKER + "/") != -1 )
        return true;
    return false;
}
```

```
/**
 * Handles a "data" request by simulating a <code>GET_STREAM_CMD</code>
 * command. This method will dispatch a <code>GetStream</code> command to
 * the <code>SystemObject</code> that is extracted from the request path.
 * @param request servlet request
 * @param response servlet response
 * @throws ServletException if an exection occurs that interrupts the
 * servelt's normal operation
 * @throws IOException if an input or output error is detected when the
 * servlet handles the request
 */

private void handleDataRequest(HttpServletRequest request,
                               HttpServletResponse response)
    throws ServletException, IOException
{
    /* --- Populate a SessionParams object --- */
       Ensure a valid and populated session */
    HttpSession session = request.getSession();
    initializeSession(session);

    /* Extract the oid string and data resource path from the request URL */
    String requestURI = request.getRequestURI();
    System.out.println( "URI: " + requestURI );

    /* oid */
    int oidStart = requestURI.lastIndexOf( DATA_REQUEST_MONIKER );

    /* start of oid */
    oidStart += DATA_REQUEST_MONIKER.length() + 1;

    /* end of oid */
    int oidEnd = requestURI.indexOf('/', oidStart);
    String oidStr = requestURI.substring(oidStart, oidEnd);

    /* start of instance */
    int instanceStart = oidEnd + 1;
```

```
/* end of instance */
int instanceEnd = requestURI.indexOf('/', instanceStart);
int instance = Integer.valueOf( requestURI.substring( instanceStart,
                                instanceEnd ) ).intValue();
/* data path */
String dataPath = requestURI.substring( instanceEnd + 1 );
dataPath = new java.net.URLDecoder().decode( dataPath, "ISO-8859-1" );

/* Convert the oid string to an actual oid */
ObjectIdentifier oid;
if ( oidStr != null ) {
    try {
        oid = new ObjectIdentifier(oidStr);
    } catch (InvalidOidException e) {
        throw new ServletException("invalid oid in request", e);
    }
} else {
   throw new ServletException("Error while extracting an oid from the request URL: "
                              + requestURI);
}

/* create and initialize the SessionParams structure. */
SessionParams sp = new SessionParams();
PersistenceManager pm = PersistenceManager.instance();

sp.environment = (Environment) session.getAttribute(ENVIRONMENT);
sp.activeList = sp.environment.getActiveList();

if ( instance < 0 ) {
    try {
        sp.sysobj = pm.load(oid, true);
    } catch (PersistenceException e) {
         throw new ServletException("unable to load SystemObject", e);
    }
    sp.instance = sp.activeList.add(sp.sysobj);
} else {
    sp.sysobj = sp.activeList.get(oid, instance);
    if ( sp.sysobj == null ) {
        throw new ServletException( "specified SystemObject not found (Instance: "
                                    + instance + ", Oid: " + oid + ")");
    }
    sp.instance = instance;
}

/* --- Create a GetStream command --- */
int rcParam = getIntParameter(request, REQUEST_COUNT);
int rc = sp.activeList.requestCount(sp.sysobj);
boolean reloaded = (rcParam >= 0) && (rc >= 0) && (rcParam < rc);
sp.activeList.increaseRequestCount(sp.sysobj);
InetReferenceFactory factory = makeFactory(sp, request);
factory.addParameter(REQUEST_COUNT, Integer.toString(rc + 1));
factory.addParameter(COMMAND_MONIKER, OPERATE_CMD);
```

```
    GetStream getStream = new GetStream();
    getStream.setEnvironment(sp.environment);
    getStream.setSession(request.getSession());
    getStream.setReloaded(reloaded);
    getStream.setReferenceFactory(factory);
    getStream.setRequest(request);

    /* add the extra data path parameter */
    getStream.setDataPath( dataPath )
    getStream.setResourcePath("resources/");

    try {
        sp.sysobj.doCommand(getStream);
    }
    catch (AccessViolationException e) {
        throw new ServletException("error sending doCommand(GetStream)", e);
    }
    catch (PersistenceException e) {
        throw new ServletException("error sending doCommand(GetStream)", e);
    }

    renderEngine.render(getStream.getOutput(), response);
}
```

```
/**
 * Handles a <code>GET_STREAM_CMD</code> command sent to the kernel.
 * This method will dispatch a <code>GetStream</code> command to the
 * requested <code>SystemObject</code>.
 * @param request servlet request
 * @param response servlet response
 * @throws ServletException if an exection occurs that interrupts the
 * servelt's normal operation
 * @throws IOException if an input or output error is detected when the
 * servlet handles the request
 */

private void doGetStream(HttpServletRequest request,
                         HttpServletResponse response)
    throws ServletException, IOException
{
    SessionParams sp = getSessionParam(request);
    int rcParam = getIntParameter(request, REQUEST_COUNT);
    int rc = sp.activeList.requestCount(sp.sysobj);
    boolean reloaded = (rcParam >= 0) && (rc >= 0) && (rcParam < rc);
    sp.activeList.increaseRequestCount(sp.sysobj);
    InetReferenceFactory factory = makeFactory(sp, request);
    factory.addParameter(REQUEST_COUNT, Integer.toString(rc + 1));
    factory.addParameter(COMMAND_MONIKER, OPERATE_CMD);
    GetStream getStream = new GetStream();
    getStream.setEnvironment(sp.environment);
    getStream.setSession(request.getSession());
    getStream.setReloaded(reloaded);
    getStream.setReferenceFactory(factory);
    getStream.setRequest(request);
    getStream.setResourcePath("resources/");

    try {
        sp.sysobj.doCommand(getStream);
    }
    catch (AccessViolationException e) {
        throw new ServletException("error sending doCommand(GetStream)", e);
    }
    catch (PersistenceException e) {
        throw new ServletException("error sending doCommand(GetStream)", e);
    }
        renderEngine.render(getStream.getOutput(), response);
}
```

```
/**
 * Handles a <code>OPERATE_CMD</code> command send to the kernel.
 * This method will initiate a <code>Operate</code> command to the
 * requested <code>SystemObject</code>.
 * @param request servlet request
 * @param response servlet response
 * @throws ServletException if an exection occurs that interrupts the
 * servelt's normal operation
 * @throws IOException if an input or output error is detected when the
 * servlet handles the request
 */

private void doOperate(HttpServletRequest request,
                       HttpServletResponse response)
    throws ServletException, IOException
{
    SessionParams sp = getSessionParam(request);
    int rcParam = getIntParameter(request, REQUEST_COUNT);
    int rc = sp.activeList.requestCount(sp.sysobj);
    boolean reloaded = (rcParam >= 0) && (rc >= 0) && (rcParam < rc);
    sp.activeList.increaseRequestCount(sp.sysobj);
    InetReferenceFactory factory = makeFactory(sp, request);
    factory.addParameter(REQUEST_COUNT, Integer.toString(rc + 1));
    factory.addParameter(COMMAND_MONIKER, OPERATE_CMD);
    Operate operate = new Operate();
    operate.setEnvironment(sp.environment);
    operate.setSession(request.getSession());
    operate.setReloaded(reloaded);
    operate.setReferenceFactory(factory);
    operate.setRequest(request);
    operate.setResourcePath("resources/");

    try {
        /* Hooked in the window manager processing */
        sp.environment.getWindowManager().preprocess( operate, sp.sysobj );
        sp.sysobj.doCommand(operate);
        sp.environment.getWindowManager().handleOutput( operate, sp.sysobj );
    }
    catch (AccessViolationException e) {
        throw new ServletException("error sending doCommand(Operate)", e);
    }
    catch (PersistenceException e) {
        throw new ServletException("error sending doCommand(Operate)", e);
    }
    renderEngine.render(operate.getOutput(), response);
}
```

```
/**
 * Handles a <code>CHANGE_PROPERTIES_CMD</code> command send to the kernel.
 * This method will initiate a <code>ChangeProperties</code> command to the
 * requested <code>SystemObject</code>.
 * @param request servlet request
 * @param response servlet response
 * @throws ServletException if an exection occurs that interrupts the
 * servelt's normal operation
 * @throws IOException if an input or output error is detected when the
 * servlet handles the request
 */

private void doChangeProperties(HttpServletRequest request,
                                HttpServletResponse response)
    throws ServletException, IOException
{
    SessionParams sp = getSessionParam(request);
    int rcParam = getIntParameter(request, REQUEST_COUNT);
    int rc = sp.activeList.requestCount(sp.sysobj);
    boolean reloaded = (rcParam >= 0) && (rc >= 0) && (rcParam < rc);
    sp.activeList.increaseRequestCount(sp.sysobj);
    InetReferenceFactory factory = makeFactory(sp, request);
    factory.addParameter(REQUEST_COUNT, Integer.toString(rc + 1));
    factory.addParameter(COMMAND_MONIKER, CHANGE_PROPERTIES_CMD);
    ChangeProperties chgprop = new ChangeProperties();
    chgprop.setEnvironment(sp.environment);
    chgprop.setSession(request.getSession());
    chgprop.setReloaded(reloaded);
    chgprop.setReferenceFactory(factory);
    chgprop.setRequest(request);
    chgprop.setResourcePath("resources/");

    try {
        sp.sysobj.doCommand(chgprop);
        if ( chgprop.isInit() ) {
            sp.sysobj.doCommand(chgprop);
        }
    }
    catch (AccessViolationException e) {
        throw new ServletException("error sending doCommand(Operate)", e);
    }
    catch (PersistenceException e) {
        throw new ServletException("error sending doCommand(Operate)", e);
    }

    SystemObjectOutput output;
    if ( chgprop.isFinished() ) {
        sp.activeList.remove(sp.sysobj);
        SingleOutput singleOutput = new SingleOutput(sp.sysobj);
        JavaScript js = new JavaScript();
        js.addSourceCode("self.close()");
        singleOutput.addElement(js);
```

```
            output = singleOutput;
        } else {
            output = chgprop.getOutput();
        }
        renderEngine.render(output, response);
    }
```

```
/**
 * Retrieves the specified parameter of type integer from the request.
 * @param request servlet request
 * @param name the name of the parameter to retrieve
 * @return the integer value of the specified parameter or -1 if the
 * specified parameter has not been found or is not an integer
 */

private int getIntParameter(HttpServletRequest request, String name) {
    String str = request.getParameter(name);
    int instance;

    if ( str != null ) {
        try {
            instance =  Integer.parseInt(str);
        } catch (NumberFormatException e) {
            instance =  -1;
        }
    } else {
        instance = -1;
    }

    return instance;
}
```

```
/**
 * Creates a new <code>InetReferenceFactory</code> according to the
 * specified session parameters.
 * @param sp the current session parameters
 * @param request servlet request
 * @return the newly craeted <code>InetReferenceFactory</code>
 * @throws ServletException if an exection occurs that interrupts the
 * servelt's normal operation
 */

private InetReferenceFactory makeFactory(SessionParams sp, HttpServletRequest request)
    throws ServletException
{
    URI baseUri;
    try {
        baseUri = new URI(new String(request.getRequestURL()));
    }
    catch ( URISyntaxException e) {
        throw new ServletException(e);
    }

    return new InetReferenceFactory(baseUri, sp.sysobj.getOid(),
                                    sp.instance);
}
```

```
/**
 * Creates a new <code>SessionParam</code> object holding important
 * This method also tries to retrieve the requested SystemObject and
 * instance number from the query string; if no object is specified,
 * the start object is used.
 * parameters of the current session.
 * @param request servlet request
 * @return the newly created <code>SessionParam</code> object.
 * @throws ServletException if an exection occurs that interrupts the
 * servelt's normal operation
 */

private SessionParams getSessionParam(HttpServletRequest request)
    throws ServletException
{
    /* try to get the oid and instance */
    ObjectIdentifier oid;
    int instance;

    /* Brought initialization to the top, so that we
       can be sure that we have a valid user env.,
       containing a valid window manager before we
       proceed. */

    /* get and initialize the session */
    HttpSession session = request.getSession();
    initializeSession(session);
    String oidStr = request.getParameter(OID_PARAM);

    /* Check whether we have a valid session and OID */
    if ( !session.isNew() && (oidStr != null) ) {
    try {
        oid = new ObjectIdentifier(oidStr);
    } catch (InvalidOidException e) {
        throw new ServletException("invalid oid in request", e);
    }
    instance = getIntParameter(request, INSTANCE_PARAM);
    } else {
        /* Changed so that the window manager stored in
           the user's environment is used instead of the
           start object */
        Environment env = (Environment)session.getAttribute( ENVIRONMENT );
        WindowManager winman = env.getWindowManager();
        oid = winman.getOid();
        instance = env.getActiveList().instance( winman.getSystemObject() );
    }

    /* create and initialize the SessionParams structure. */
    SessionParams sp = new SessionParams();
    PersistenceManager pm = PersistenceManager.instance();

    sp.environment = (Environment) session.getAttribute(ENVIRONMENT);
```

```
    sp.activeList = sp.environment.getActiveList();
    if ( instance < 0 ) {
        /* -1 has become a "magic number" that means "create if
           !!! necessary", i.e., if an instance of the object exists
           !!! in the active list, that one will be used instead of a
           !!! new one being created +7
        if (instance == -1) {
            sp.sysobj = sp.activeList.get(oid);
            if ( sp.sysobj != null ) {
                // an instance exists, use it
                sp.instance = sp.activeList.instance( sp.sysobj );
            } else {
                // no instance available, try creating a new one
                try {
                    sp.sysobj = pm.load(oid, true);
                } catch (PersistenceException e) {
                    throw new ServletException("unable to load SystemObject", e);
                }
                sp.instance = sp.activeList.add(sp.sysobj);
            }

        } else {
            /* !!! Any other negative instance numebr, except -1 causes
               !!! the previously default behaviour to be invoked */
            try {
                sp.sysobj = pm.load(oid, true);
            } catch (PersistenceException e) {
                throw new ServletException("unable to load SystemObject", e);
            }
            sp.instance = sp.activeList.add(sp.sysobj);
        }
    } else {
        sp.sysobj = sp.activeList.get(oid, instance);
        if ( sp.sysobj == null ) {
            throw new ServletException(
                "specified SystemObject not found (Instance: "
                + instance + ", Oid: " + oid + ")");
        }
        sp.instance = instance;
    }
    return sp;
}
```

```
/**
 * Checks if the current session has already an <code>Environment</code>
 * and creates one if not.
 * @param session the current </code>HttpSession</code> object
 */

private void initializeSession(HttpSession session) {
    Environment sessionEnv = (Environment)session.getAttribute(ENVIRONMENT);
    if ( sessionEnv == null ) {
        Environment environment = new Environment();
        environment.setActiveList(new ActiveList());
        session.setAttribute(ENVIRONMENT, environment);

        /* set the userid to the user 'Everybody' */

        /* changed so that there is a central point for
           "logging in" users */
        try {
            changeUser(ObjectIdentifier.USER_EVERYBODY, environment);
        }
        catch (PersistenceException unused) {
            logger.error( "Unable to set the current user to 'everybody'.", unused);
        }
    }
}
```

```
/**
 * Changes the current user to the specified one.
 * This method will load the user profile of the specified person
 * and initialize the given <code>Environement</code> object according
 * to the new user.
 * @param userId the <code>ObjectIdentifier</code> of the new user.
 * @param env the <code>Environement</code> object to be initialized
 * according to the new user.
 * @throws PersistenceException in case of any error during a persistence
 * operation
 */

public static void changeUser(ObjectIdentifier userId, Environment env)
    throws PersistenceException
{
    PersistenceManager pm = PersistenceManager.instance();
    env.setUserId(userId);

    Person user = (Person) pm.load(userId);
    env.addVariable(USERNAME_VAR, user.getLogin());

    /* varables like LANG, ... */
    env.addVariable(LANG_VAR, "de");

    /* preliminary code for supporting the window manager */
    try {
        if (env.getWindowManager() == null) {
            String winman = DEFAULT_WINDOW_MANAGER;
            SystemPath systemPath = new SystemPath(ObjectIdentifier.ROOT_FOLDER);
            WindowManager windowManager = (WindowManager)systemPath.findObject(winman,
                                                                               true);
            if ( windowManager == null ) {
                logger.error("Window manager not found");
                throw new PersistenceException("Window manager not found: " + winman);
            }
            env.setWindowManager( windowManager );
            env.getActiveList().add( windowManager.getSystemObject() );
            windowManager.initialize();
        }
    }
    catch (PersistenceException ex) {
        logger.error( "Unable to initialize the window manager for the current user's
                      environment.", ex);
        throw new PersistenceException( "Unable to initialize the window manager for the
                                        current user's environment.", ex);
    }
    catch (InvalidPathException ex) {
        logger.error( "Unable to initialize the window manager for the current user's
                      environment.", ex);
        throw new PersistenceException( "Unable to initialize the window manager for the
                                        current user's environment.", ex);
    }
```

```
    catch (AccessViolationException ex) {
        logger.error( "Unable to initialize the window manager for the current user's
                      environment.", ex);
        throw new PersistenceException( "Unable to initialize the window manager for the
                                        current user's environment.", ex);
    }
}
```

```
/**
 * Handles the HTTP <code>GET</code> method.
 * @param request servlet request
 * @param response servlet response
 * @throws ServletException if an exception occurs that interrupts the
 * servlet's normal operation
 * @throws IOException if an input or output error is detected when the
 * servlet handles the request
 */

protected void doGet(HttpServletRequest request,
                     HttpServletResponse response)
    throws ServletException, java.io.IOException
{
    processRequest(request, response);
}
```

```
/**
 * Handles the HTTP <code>POST</code> method.
 * @param request servlet request
 * @param response servlet response
 * @throws ServletException if an exception occurs that interrupts the
 * servlet's normal operation
 * @throws IOException if an input or output error is detected when the
 * servlet handles the request
 */

protected void doPost(HttpServletRequest request,
                      HttpServletResponse response)
    throws ServletException, java.io.IOException
{
    processRequest(request, response);
}
```

```
/**
 * Returns a short description of the servlet.
 * @return a short description of the servlet
 */
public String getServletInfo() {
    return SHORT_DESCRIPTION;
}
```

```
/**
 * Initializes the uploading module (basically, determines the default
 * path to which files should be uploaded).
 * @throws ServletException if the corresponding servlet initialization
 * parameter is missing, or if the directory does
 * not exist and cannot be created
 */

private void initUploader() throws ServletException {
    /* Get the directory from the initialization file */
    String pmRootDir = getServletContext().getInitParameter(UPLOAD_ROOT);

    if ( pmRootDir == null ) {
        logger.fatal("Missing servlet parameter: " + UPLOAD_ROOT);
        throw new ServletException("Missing servlet parameter: " + UPLOAD_ROOT);
    }
    File rootDir = new File(pmRootDir);
    if ( !rootDir.exists() && !rootDir.mkdir() ) {
        logger.fatal( "Unable to create the root directory for file uploading: '" +
                     pmRootDir + "'");
        throw new ServletException( "Unable to create the root directory "
                                   + "for file uploading: '" + pmRootDir + "'");
    }
    this.uploadDirectory = rootDir;
}
```

```
/**
  * Initializes the logging module (currently log4j).
  */

private void initLogger() {
    String prefix =  getServletContext().getRealPath("/WEB-INF");
    String file = getInitParameter("LOGGER_INIT_FILE");
    if (file != null) {

        /* if the initialization file is set, read from it
        while ( file.startsWith( File.separator ) ) {
            file = file.substring( 1 );
        }
        File testPath = new File(prefix + File.separator + file);
        System.out.println( testPath );
        if ( testPath.exists() ) {
            try {
                PropertyConfigurator.configure( testPath.getAbsolutePath() );
            }
            catch (RuntimeException unused) {

                /* should only happen if there exists a socket target and
                 it is not accessible */
            }

            /* Whether the configuration was successful or not, there is
               nothing more to do here */
            return;
        }
    }
    /* no initialization file, or problems reading it; try a basic configuration */
    BasicConfigurator.configure();
}
```

```
/**
 * Initializes the PersistenceManager if necessary.
 * @throws ServletException if an exception occurs that interrupts the
 * servlet's normal operation
 */
private void initPm() throws ServletException {

    if ( PersistenceManager.isInitialized() ) {
        /* The PM is already initilized, we have nothing to do. */
        return;
    }

    /* Get the root directory and create it if it does not exists already. */
    String pmRootDir = getInitParameter(DISKPM_ROOT);

    if ( pmRootDir == null ) {
        throw new ServletException("Missing servlet parameter: "
                                   + DISKPM_ROOT);
    }
    File rootDir = new File(pmRootDir);

    if ( !rootDir.exists() && !rootDir.mkdir() ) {
        throw new ServletException("Unable to create the root directory of"
            + " the Disk-PersistenceManager: '" + pmRootDir + "'");
    }

    String cacheStr = getInitParameter(DISKPM_CACHE);
    boolean useCache;
    useCache = (cacheStr == null) || (!cacheStr.equalsIgnoreCase("off"));
    DiskManager dm;
    try {
        dm = new DiskManager(rootDir);
        dm.setCachingEnabled(useCache);
    } catch (PersistenceException e) {
        throw new ServletException("error initializing PersistenceManager", e);
    }
    String timeoutStr = getInitParameter(LOCKING_TIMEOUT);
    int timeout;
    if ( timeoutStr != null ) {
        try {
            timeout = Integer.parseInt(timeoutStr);
        } catch (NumberFormatException e) {
            throw new ServletException("the servlet parameter '"
                                       + LOCKING_TIMEOUT + "' is invalid: '" +
                                       timeoutStr + "'");
        }
    } else {
        timeout = DEFAULT_TIMEOUT;
    }
    PersistenceManager.setBasicIo(dm);
    PersistenceManager.setLockingTimeout(timeout);
}
```

```
/**
 * Returns <code>true</code> if the system has been initialized before.
 * @return <code>true</code> if the system has been initialized before,
 * otherwise <code>false</code>
 * @throws ServletException if an exception occurs that interrupts the
 * servlet's normal operation
 */

private boolean isSystemInitialized() throws ServletException {
    PersistenceManager pm = PersistenceManager.instance();
    SystemObject rootFolder;
    try {
        rootFolder = pm.load(ObjectIdentifier.ROOT_FOLDER);
    }
    catch (ObjectNotFoundException e ) {
        rootFolder = null;
    }
    catch (PersistenceException e) {
        throw new ServletException(
            "Unable to access root folder of the virtual file system", e);
    }

    if ( rootFolder == null ) {
        return false;
    }
    return true;
}
```

```
/**
 * Initializes the system by creating a root folder and executing the
 * <code>SETUP_SCRIPT</code>.
 * @throws ServletException if an exception occurs that interrupts the
 * servlet's normal operation
 */

private void initializeSystem() throws ServletException {

    /* create a root folder if necessary. */
    ensureVirtualFileSystemRoot();

    /* create all special system objects. */
    createSpecialObjects();

    /* set permissions for the root folder */
    RightsManager rm = RightsManager.instance();
    try {
        PersistenceManager pm = PersistenceManager.instance();
        SystemObject rootFolder = pm.load(ObjectIdentifier.ROOT_FOLDER, true);
        rm.setPermission(ObjectIdentifier.USER_SYSTEM, rootFolder,
                         Permission.READ, true);
        rm.setPermission(ObjectIdentifier.USER_SYSTEM, rootFolder,
                         Permission.WRITE, true);
        rm.setPermission(ObjectIdentifier.USER_SYSTEM, rootFolder,
                         Permission.EXECUTE, true);
        rm.setPermission(ObjectIdentifier.USER_SYSTEM, rootFolder,
                         Permission.VISIBLE, true);
        rm.setPermission(ObjectIdentifier.USER_SYSTEM, rootFolder,
                         Permission.CHANGE_RIGHTS, true);
    }
    catch (PersistenceException e) {
        throw new ServletException( "unable to set permissions on the root folder", e);
    }
    catch (AccessViolationException e) {
        throw new ServletException( "unable to set permissions on the root folder", e);
    }
    catch (PermissionException e) {
        throw new ServletException("unable to set permissions on the root folder", e);
    }

    /* get the setup script. */
    String setupPath = getInitParameter(SETUP_SCRIPT);
    if ( setupPath == null ) {
        setupPath = DEFAULT_SETUP_SCRIPT;
    }

    /* Convert and adapt path (should not be necessary) */
    setupPath = setupPath.replace('\\', '/');
    if ( setupPath.charAt(0) == '/' ) {
        setupPath = setupPath.substring(1);
    }
```

```
    /* Load the script into a string object. */
    ServletContext sc = getServletContext();
    setupPath = sc.getRealPath("WEB-INF/" + setupPath);
    File setup = new File(setupPath);
    String script = new String();
    try {
        BufferedReader br = new BufferedReader(new FileReader(setup));
        String line = br.readLine();
        while ( line != null ) {
            script += line + "\n";
            line = br.readLine();
        }
        br.close();
    }
    catch (FileNotFoundException e) {
        throw new ServletException("setup-script '" + setupPath
                                   + "' not found");
    }
    catch (IOException e) {
        throw new ServletException("error reading setup-script", e);
    }

    /* execute the script. */
    Environment env = new Environment();
    env.setUserId( ObjectIdentifier.USER_SYSTEM );
    Execute execute = new Execute();
    execute.setEnvironment(env);
    execute.setCurrentPath(new SystemPath(ObjectIdentifier.ROOT_FOLDER));
    List params = new ArrayList();
    params.add("shell");
    params.add(script);
    execute.setParameters(params);
    Shell shell = new Shell();
    try {
        shell.doCommand(execute);
    }
    catch (PersistenceException e) {
        throw new ServletException("error executing setup-script", e);
    }
    catch (AccessViolationException e) {
        throw new ServletException("error executing setup-script", e);
    }

    if ( !execute.isSuccessful() ) {
       throw new ServletException("error executing setup-script: "
                                   + (String) execute.getResult());
    }
}
```

```
/**
 * Creates the root folder of the virtual file system if necessary.
 * @throws ServletException if an exception occurs that interrupts the
 * servlet's normal operation
 */

private void ensureVirtualFileSystemRoot() throws ServletException {
    PersistenceManager pm = PersistenceManager.instance();

    SystemObject rootFolder;
    try {
        rootFolder = pm.load(ObjectIdentifier.ROOT_FOLDER);
    }
    catch (ObjectNotFoundException e ) {
        rootFolder = null;
    }
    catch (PersistenceException e) {
        throw new ServletException(
            "Unable to access root folder of the virtual file system", e);
    }
    if ( rootFolder != null ) {
        return;
    }

    /* No root folder present, create a new one. */
    try {
        rootFolder = SystemObject.specialObject(
            Folder.class, ObjectIdentifier.ROOT_FOLDER, new AccessKey());
        rootFolder.setAcl( new Acl() );

        pm.storeRoot(rootFolder);
    }
    catch (SpecialObjectException e) {
        throw new ServletException(
            "unable to create root folder of the virtual file system", e);
    }
    catch (PersistenceException e) {
        throw new ServletException(
            "unable to create root folder of the virtual file system", e);
    }
    catch (AccessViolationException e) {
        throw new ServletException(
            "unable to create root folder of the virtual file system", e);
    }
}
```

```
/**
 * Creates some "special" objects and adds them to the virtual file system.
 * These "special" objects are SystemObjects having a well known
 * ObjectIdentifier. Therefore they can be accessed from anywhere, anytime.
 * @throws ServletException if an exception occurs that interrupts the
 * servlet's normal operation
 */

private void createSpecialObjects()
    throws ServletException
{
    PersistenceManager pm = PersistenceManager.instance();
    AccessKey accessKey = new AccessKey();
    SystemObject root;
    SystemObject sysobj;

    try {
        root = pm.load(ObjectIdentifier.ROOT_FOLDER, true);
    } catch (PersistenceException e) {
        throw new ServletException("could not load root folder", e);
    }

    /* ------------------------------------------------------------------
       create the special folders for users, groups and rights.
       ------------------------------------------------------------------ */
    try {
        /* the users folder */
        sysobj = SystemObject.specialObject(UsersFolder.class,
                                            ObjectIdentifier.USERS_FOLDER,
                                            accessKey);
        sysobj.setName(USERS_FOLDER_NAME);
        sysobj.setAcl(root);
        root.addChild(sysobj);

        /* the user EVERYBODY */
        Person everybody = (Person) SystemObject.specialObject(Person.class,
                                            ObjectIdentifier.USER_EVERYBODY,
                                            accessKey);
        everybody.setName(USER_EVERYBODY_NAME);
        everybody.setLogin(USER_EVERYBODY_NAME);
        everybody.setEnabled(false);
        everybody.setAcl(sysobj);
        sysobj.addChild(everybody);

        /* the user SYSTEM */
        Person system = (Person) SystemObject.specialObject(Person.class,
                                                ObjectIdentifier.USER_SYSTEM,
                                                accessKey);
        system.setName(USER_SYSTEM_NAME);
        system.setLogin(USER_SYSTEM_NAME);
        system.setEnabled(false);
        system.setAcl(sysobj);
```

```
            sysobj.addChild(system);

            /* the user OWNER */
            Person owner = (Person) SystemObject.specialObject(Person.class,
                                                ObjectIdentifier.USER_OWNER,
                                                accessKey);
            owner.setName(USER_OWNER_NAME);
            owner.setLogin(USER_OWNER_NAME);
            owner.setEnabled(false);
            owner.setAcl(sysobj);
            sysobj.addChild(owner);

            /* the groups folder */
            sysobj = SystemObject.specialObject(Folder.class,
                                                ObjectIdentifier.GROUPS_FOLDER,
                                                accessKey);
            sysobj.setName(GROUPS_FOLDER_NAME);
            sysobj.setAcl(root);
            root.addChild(sysobj);

            /* the rights folder */
            sysobj = SystemObject.specialObject(Folder.class,
                                                ObjectIdentifier.RIGHTS_FOLDER,
                                                accessKey);
            sysobj.setName(RIGHTS_FOLDER_NAME);
            sysobj.setAcl(root);
            root.addChild(sysobj);

            /* the file settings object */
            sysobj = SystemObject.specialObject(FileSettings.class,
                                                ObjectIdentifier.FILE_SETTINGS,
                                                accessKey);
            sysobj.setName(FILE_SETTINGS);
            sysobj.setAcl(root);
            root.addChild(sysobj);
            ((FileSettings)sysobj).initialize( this.getServletContext() );
        }
        catch (SpecialObjectException e) {
            throw new ServletException( "unable to create special object", e);
        }
        catch (PersistenceException e) {
            throw new ServletException( "unable to store special object", e);
        }
        catch (AccessViolationException e) {
            throw new ServletException( "unable to store special object", e);
        }
    }
```

```
/**
 * Used to hold all necesary parameters according to the current session.
 */

private class SessionParams {
    /* The requested <code>SystemObject</code>. */
    public SystemObject sysobj;

    /* The instance number of the requested <code>SystemObject</code>. */
    public int instance;

    /* Holds all active SystemObjects of the current session. */
    public ActiveList activeList;

    /* The Environment object of the current session. */
    public Environment environment;
}
```

```
/**
 * A access key class for the kernel.
 * This class is used to identification key for the
 * <code>SystemObject.specialObject</code> method.
 */

public class AccessKey {
    /* Creates a new instance of AccessKey. */
    private AccessKey() {}
}
```

```
/**
 * Prepares a response that ensures the browser will be redirected to the
 * specified URL. Furthermore, it makes sure that the new URL will be shown
 * at the outer-most containing frame, in the case of nested frames.
 */

void causeRedirect(HttpServletRequest request, HttpServletResponse response, String url)
    throws IOException {
        PrintWriter out = response.getWriter();
        out.println("<html><head><title>WeLearn 2</title></head><body>");
        out.println("<script language=\"Javascript\">");
        out.println("  <!--");
        out.println("  top.location='" + url + "';");
        out.println("  //-->");
        out.println("</script>");
        out.println("<noscript>");
        out.println("</noscript>");
        out.println("</body></html>");
    }

}
```

6.2. Das Setup Script

```
# WeLearn R2 Setup-Script

# create basic file structure
mkobj at.jku.fim.welearn.objects.utilities.MakeDirectory mkdir
mkdir system
mkdir /system/bin
mkdir /system/sbin
mkdir /home

# create basic command utilities
mkobj at.jku.fim.welearn.objects.utilities.Copy system/bin/cp
mkobj at.jku.fim.welearn.objects.utilities.Delete system/bin/del
mkobj at.jku.fim.welearn.objects.utilities.Move system/bin/mv
mkobj at.jku.fim.welearn.objects.utilities.SetProperty system/bin/setprop
mkobj at.jku.fim.welearn.objects.utilities.ListDirectory system/bin/ls
mkobj at.jku.fim.welearn.objects.utilities.SetLink system/bin/ln
/system/bin/mv mkdir system/bin

# create and initialize template objects
/system/bin/mkdir /system/templates
mkobj at.jku.fim.welearn.objects.Folder /system/templates/Folder
mkobj at.jku.fim.welearn.objects.menus.Menu /system/templates/Menu
mkobj at.jku.fim.welearn.objects.Url /system/templates/Url
mkobj at.jku.fim.welearn.objects.forum.DiscussionForum /system/templates/Forum
mkobj at.jku.fim.welearn.objects.InfoText /system/templates/InfoText
mkobj at.jku.fim.welearn.objects.InfoBoard /system/templates/InfoBoard
mkobj at.jku.fim.welearn.objects.user.Group /system/templates/Group
mkobj at.jku.fim.welearn.objects.menus.Desktop /system/templates/Desktop

# create start object
mkobj at.jku.fim.welearn.objects.user.Login /system/startup

# create the window manager
mkobj at.jku.fim.welearn.objects.winman.DefaultWindowManager /system/winman

# create some command line utilities for user and group administration
mkobj at.jku.fim.welearn.objects.utilities.AddUser /system/sbin/adduser
mkobj at.jku.fim.welearn.objects.utilities.AddGroup /system/sbin/addgroup
mkobj at.jku.fim.welearn.objects.utilities.GroupMember /system/sbin/groupmember
mkobj at.jku.fim.welearn.objects.utilities.ChangeRight /system/sbin/changeright
mkobj at.jku.fim.welearn.objects.utilities.ShowRights /system/sbin/showrights

# create a mini scenario
mkobj at.jku.fim.welearn.objects.menus.Menu /Courses
mkobj at.jku.fim.welearn.objects.menus.Menu /Courses/Propaedeutikum
mkobj at.jku.fim.welearn.objects.menus.Menu /Courses/Betriebssysteme
mkobj at.jku.fim.welearn.objects.Cli /system/cli
mkobj at.jku.fim.welearn.objects.menus.TopMenu /system/topmenu
mkobj at.jku.fim.welearn.objects.user.Login /system/topmenu/Login
```

```
mkobj at.jku.fim.welearn.objects.menus.Menu /system/topmenu/Home
mkobj at.jku.fim.welearn.objects.filemanager.FileManager
    /system/topmenu/Home/FileManager
mkobj at.jku.fim.welearn.objects.utilities.ChangePassword
    /system/topmenu/Home/ChangePassword
mkobj at.jku.fim.welearn.objects.HardLink /system/topmenu/Courses
    /system/bin/setprop /system/topmenu/Courses object /Courses
mkobj at.jku.fim.welearn.objects.menus.Menu /system/topmenu/Links
mkobj at.jku.fim.welearn.objects.Url /system/topmenu/Links/FIM
    /system/bin/setprop /system/topmenu/Links/FIM url http://www.fim.uni-linz.ac.at
mkobj at.jku.fim.welearn.objects.Url /system/topmenu/Links/Google
    /system/bin/setprop /system/topmenu/Links/Google url http://www.google.at
mkobj at.jku.fim.welearn.objects.Url /system/topmenu/Links/Wörterbuch
    /system/bin/setprop /system/topmenu/Links/Wörterbuch url http://dict.leo.org
mkobj at.jku.fim.welearn.objects.menus.Menu /system/topmenu/Help
mkobj at.jku.fim.welearn.objects.forum.DiscussionForum /system/topmenu/Help/Help-Forum
mkobj at.jku.fim.welearn.objects.InfoText /system/topmenu/Help/Information
    /system/bin/setprop /system/topmenu/Help/Information headline Help
    /system/bin/setprop /system/topmenu/Help/Information message Online-Hilfe-Seiten

# create some groups
/system/sbin/addgroup Users
/system/sbin/addgroup Administrators
/system/sbin/addgroup LVALeiter
/system/sbin/addgroup CRM

# set some rights for the groups Users and Administrators
/system/sbin/changeright -what / -group /groups/Administrators -grant rwxv
/system/sbin/changeright -what / -group /groups/Administrators -grant chgrights
/system/sbin/changeright -what / -group /groups/Administrators -grant edit
/system/sbin/changeright -what / -group /groups/Administrators -grant attach
/system/sbin/changeright -what / -owner -grant edit
/system/sbin/changeright -what / -owner -grant attach

#initialize skeleton directory
/system/bin/mkdir /skel
    /system/sbin/changeright -what /skel -ownacl
    /system/sbin/changeright -what /skel -group /groups/Users -deny rwxv
    /system/sbin/changeright -what /skel -owner -grant rwxv

# create some users
/system/sbin/adduser -l Sesam -f Ernie -login ernie -pwd ernie -home /home -skel /skel
/system/sbin/adduser -l Sesam -f Bert -login bert -pwd bert -home /home -skel /skel
/system/sbin/adduser -l Admin -login admin -pwd admin -home /home -skel /skel

# specify group membership
/system/sbin/groupmember -g /groups/Administrators -l admin
/system/sbin/groupmember -g /groups/Users -l ernie
/system/sbin/groupmember -g /groups/Users -l bert

/system/sbin/changeright -what /home -ownacl
/system/sbin/changeright -what /home -group /groups/Users -grant vx
```

```
/system/sbin/changeright -what /system/topmenu -ownacl
/system/sbin/changeright -what /system/topmenu -group /groups/Users -grant vx
/system/sbin/changeright -what /system/topmenu -person everybody -grant v

/system/sbin/changeright -what /system/topmenu/Login -ownacl
/system/sbin/changeright -what /system/topmenu/Login -person everybody -grant x

/system/sbin/changeright -what /system -ownacl
/system/sbin/changeright -what /system -group /groups/Users -grant x

/system/sbin/changeright -what /system/templates -ownacl
/system/sbin/changeright -what /system/templates -group /groups/Users -grant vr

/system/sbin/changeright -what /system/topmenu/Help/Help-Forum -ownacl
/system/sbin/changeright -what /system/topmenu/Help/Help-Forum -group
    /groups/Users -grant vrw

/system/sbin/changeright -what /Courses -ownacl
/system/sbin/changeright -what /Courses -group /groups/Users -grant vrx

# create and initialize icons used by the system
/system/bin/mkdir /system/icons
mkobj at.jku.fim.welearn.system.object.Pictogram /system/icons/folder.ico
    /system/bin/setprop /system/icons/folder.ico smallImage folder_closed_16x16.gif
    /system/bin/setprop /system/icons/folder.ico mediumImage folder_closed_32x32.gif
    /system/bin/setprop /system/icons/folder.ico largeImage folder_closed_48x48.gif

mkobj at.jku.fim.welearn.system.object.Pictogram /system/icons/sysfolder.ico
    /system/bin/setprop /system/icons/sysfolder.ico smallImage sysfolder_16x16.gif
    /system/bin/setprop /system/icons/sysfolder.ico mediumImage sysfolder_32x32.gif
    /system/bin/setprop /system/icons/sysfolder.ico largeImage sysfolder_48x48.gif

mkobj at.jku.fim.welearn.system.object.Pictogram /system/icons/person.ico
    /system/bin/setprop /system/icons/person.ico smallImage group_16x16.gif
    /system/bin/setprop /system/icons/person.ico mediumImage group_32x32.gif
    /system/bin/setprop /system/icons/person.ico largeImage group_48x48.gif

mkobj at.jku.fim.welearn.system.object.Pictogram /system/icons/sysobj.ico
    /system/bin/setprop /system/icons/sysobj.ico smallImage sysobj_16x16.gif
    /system/bin/setprop /system/icons/sysobj.ico mediumImage sysobj_32x32.gif
    /system/bin/setprop /system/icons/sysobj.ico largeImage sysobj_48x48.gif

mkobj at.jku.fim.welearn.system.object.Pictogram /system/icons/menu.ico
    /system/bin/setprop /system/icons/menu.ico smallImage menu_16x16.gif
    /system/bin/setprop /system/icons/menu.ico mediumImage menu_32x32.gif
    /system/bin/setprop /system/icons/menu.ico largeImage menu_48x48.gif

mkobj at.jku.fim.welearn.system.object.Pictogram /system/icons/cps.ico
    /system/bin/setprop /system/icons/cps.ico smallImage cps_16x16.gif
    /system/bin/setprop /system/icons/cps.ico mediumImage cps_32x32.gif
    /system/bin/setprop /system/icons/cps.ico largeImage cps_48x48.gif
```

```
mkobj at.jku.fim.welearn.system.object.Pictogram /system/icons/forum.ico
    /system/bin/setprop /system/icons/forum.ico smallImage forum_16x16.gif
    /system/bin/setprop /system/icons/forum.ico mediumImage forum_32x32.gif
    /system/bin/setprop /system/icons/forum.ico largeImage forum_48x48.gif

mkobj at.jku.fim.welearn.system.object.Pictogram /system/icons/url.ico
    /system/bin/setprop /system/icons/url.ico smallImage url_16x16.gif
    /system/bin/setprop /system/icons/url.ico mediumImage url_32x32.gif
    /system/bin/setprop /system/icons/url.ico largeImage url_48x48.gif

mkobj at.jku.fim.welearn.system.object.Pictogram /system/icons/hello.ico
    /system/bin/setprop /system/icons/hello.ico smallImage hw_16x16.gif
    /system/bin/setprop /system/icons/hello.ico mediumImage hw_32x32.gif

mkobj at.jku.fim.welearn.system.object.Pictogram /system/icons/unknown.ico
    /system/bin/setprop /system/icons/unknown.ico smallImage unknown_16x16.gif
    /system/bin/setprop /system/icons/unknown.ico mediumImage unknown_32x32.gif
    /system/bin/setprop /system/icons/unknown.ico largeImage unknown_48x48.gif

mkobj at.jku.fim.welearn.system.object.Pictogram /system/icons/pdf.ico
    /system/bin/setprop /system/icons/pdf.ico smallImage pdf_16x16.gif
    /system/bin/setprop /system/icons/pdf.ico mediumImage pdf_32x32.gif
    /system/bin/setprop /system/icons/pdf.ico largeImage pdf_48x48.gif

mkobj at.jku.fim.welearn.system.object.Pictogram /system/icons/ppt.ico
    /system/bin/setprop /system/icons/ppt.ico smallImage powerpnt_16x16.gif
    /system/bin/setprop /system/icons/ppt.ico mediumImage powerpnt_32x32.gif
    /system/bin/setprop /system/icons/ppt.ico largeImage powerpnt_48x48.gif

mkobj at.jku.fim.welearn.system.object.Pictogram /system/icons/doc.ico
    /system/bin/setprop /system/icons/doc.ico smallImage doc_16x16.gif
    /system/bin/setprop /system/icons/doc.ico mediumImage doc_32x32.gif
    /system/bin/setprop /system/icons/doc.ico largeImage doc_48x48.gif

mkobj at.jku.fim.welearn.system.object.Pictogram /system/icons/xls.ico
    /system/bin/setprop /system/icons/xls.ico smallImage xls_16x16.gif
    /system/bin/setprop /system/icons/xls.ico mediumImage xls_32x32.gif
    /system/bin/setprop /system/icons/xls.ico largeImage xls_48x48.gif

mkobj at.jku.fim.welearn.system.object.Pictogram /system/icons/html.ico
    /system/bin/setprop /system/icons/html.ico smallImage htm_16x16.gif
    /system/bin/setprop /system/icons/html.ico mediumImage htm_32x32.gif
    /system/bin/setprop /system/icons/html.ico largeImage htm_48x48.gif

mkobj at.jku.fim.welearn.system.object.Pictogram /system/icons/zip.ico
    /system/bin/setprop /system/icons/zip.ico smallImage zip_16x16.gif
    /system/bin/setprop /system/icons/zip.ico mediumImage zip_32x32.gif
    /system/bin/setprop /system/icons/zip.ico largeImage zip_48x48.gif

mkobj at.jku.fim.welearn.system.object.Pictogram /system/icons/txt.ico
    /system/bin/setprop /system/icons/txt.ico smallImage txt_16x16.gif
```

```
    /system/bin/setprop /system/icons/txt.ico mediumImage txt_32x32.gif
    /system/bin/setprop /system/icons/txt.ico largeImage txt_48x48.gif

mkobj at.jku.fim.welearn.system.object.Pictogram /system/icons/gif.ico
    /system/bin/setprop /system/icons/gif.ico smallImage image_16x16.gif
    /system/bin/setprop /system/icons/gif.ico mediumImage image_32x32.gif
    /system/bin/setprop /system/icons/gif.ico largeImage image_48x48.gif

mkobj at.jku.fim.welearn.system.object.Pictogram /system/icons/jpg.ico
    /system/bin/setprop /system/icons/jpg.ico smallImage image_16x16.gif
    /system/bin/setprop /system/icons/jpg.ico mediumImage image_32x32.gif
    /system/bin/setprop /system/icons/jpg.ico largeImage image_48x48.gif

mkobj at.jku.fim.welearn.system.object.Pictogram /system/icons/wav.ico
    /system/bin/setprop /system/icons/wav.ico smallImage audio_16x16.gif
    /system/bin/setprop /system/icons/wav.ico mediumImage audio_32x32.gif

mkobj at.jku.fim.welearn.system.object.Pictogram /system/icons/mp3.ico
    /system/bin/setprop /system/icons/mp3.ico smallImage audio_16x16.gif
    /system/bin/setprop /system/icons/mp3.ico mediumImage audio_32x32.gif

mkobj at.jku.fim.welearn.system.object.Pictogram /system/icons/avi.ico
    /system/bin/setprop /system/icons/avi.ico smallImage video_16x16.gif
    /system/bin/setprop /system/icons/avi.ico mediumImage video_32x32.gif
```

7 Abbildungsverzeichnis

8 Literatur

[Tan02] *A. S. Tanenbaum, M. van Steen, „Distributed Systems“, ISBN 0-13-088893-1, 2002*

[IMS01] *IMS Global Learning Consortium, “IMS Content Packaging Information Model”, http://imsglobal.org, 2001*

[Scheu02] *T. Scheuermann , University of North Carolina, Department of Computer science, 2002, http://www.cs.unc.edu*

[Lied96] *J. Liedtke, “Towards Real Microkernels”, ACM, 1996*

[QNX02] *QNX Website, http://www.qnx.de*

[Sil02] *A. Silberschatz, P. B. Galvin, G. Gagne “Operating System Concepts”, ISBN 0-471-41743-2, 2002*

[Sol00] *D. A. Solomon, M. E. Russinovich, „Inside Microsoft Windows 2000“, Third Edition, ISBN 0-7356-1021-5, 2000*

[Div02] *R. Divotkey, D. Remplbauer, “Design und Implementierung einer webbasierten Lernumgebung”, 2002*

[JRM02] *R. Divotkey, J. R. Mühlbacher, S. Reisinger, D. Remplbauer, „The WeLearn Distance Teaching Framework“, EDEN, Annual Conference, P8308-314, Granada 2002*

[Ven00] *B. Venners, “Inside Java Virtual Machine”, 2nd Edition, ISBN 0-07135-0943, 2000*

[Rus01] *M. E. Russionovich, D. A. Solomon, „Windows XP, Kernel Improvement, Create a More Robust, Powerful and Scaleable OS“, MSDN Magazine, December 2001*

[Hol01] *A. I. Holub, „Inside the Java VM“, 2001, http://www.holub.ccom*

[Lal98] *A. Lal, J. Formechelli, “The Java Virtual Machine”, Kent State University, 1998*

[Ven96] *B. Venners, “Under the Hood”, 1996, http://www.javaworld.com*

[Hal01] *M. Hall, “Servlets und JavaServer Pages”, ISBN 3-8272-5945-2, 2001*

[Hun98] *Jason Hunter, “Java Servlet Programming”, ISBN 1-56592-391-X, 1998*

[Cal95] *Francesco Callari, “Operating Systems”, McGill University, 1995, http://www.cim.mcgill.ca*

[Tec02] *TechWeb, http://www.techweb.com/encyclopedia, 2002*

[Apa02] *Apache Software Foundation, http://www.apache.org/, 2002*

[Lei03] *R. Leitner, „Rechtssysteme von Betriebssystemen - Eine Bestandsaufnahme und deren Umsetzung im virtuellen Dateisystem der Telecoaching - Plattform WeLearn“, 2003*

[OMG00] *Object Management Group, „Mobile Agent System Interoperability Facilities Specification, http://www.omg.org, 2000*

9 Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Linz, im Jänner 2003

Oliver Kronawittleithner

10 Lebenslauf

STARHEMBERGSTRASSE 66/6/39 • A-4020 LINZ • AUSTRIA
TELEFON 0699 14 00 60 46 • FAX 0699 44 00 60 46 • E-MAIL OLIVERK@LIWEST.AT

OLIVER KRONAWITTLEITHNER

PERSÖNLICHE INFORMATION

- Familienstand: ledig
- Staatsangehörigkeit: Österreich
- Geburtsdatum: 3. Juni 1978
- Geburtsort: Linz
- Eltern: Reinhold, Berufsmusiker
  Edith, Ordinationshilfe

AUSBILDUNG

1984 - 1988 Volkschule 21 Linz

1988 - 1996 BRG Hamerlingstrasse Linz

- Schwerpunkt Informatik

seit 1996 Johannes Kepler Universität Linz

- Informatik, BWL

BERUFSERFAHRUNG

1997 - 2002 Rosenbauer International AG
Freier Mitarbeiter

seit 1999 Gewerbetreibender EDV - Trainer

seit 2002 FIM, Universität Linz
Projektmitarbeiter

PRAKTIKA

1993 Voest-Alpine-Industrieanlagenbau Linz
- Entwicklung und Änderung von Anlagenzeichnungen mit AutoCAD

1994 Voest-Alpine-Industrieanlagenbau Linz
- Entwicklung und Änderung von Anlagenzeichnungen mit AutoCAD

1996 Rosenbauer International AG Leonding
- im Fertigungsbetrieb

1997 Rosenbauer International AG Leonding
- im Bürobetrieb

2002 Praktikum bei Dr. Paul Rübig, mdep Brüssel

SPRACHKENNTNISSE

1995	Sprachwoche Englisch	London
1995	Sprachwoche Spanisch	Barcelona

MITGLIEDSCHAFTEN

K.Ö.St.V. Severina Linz
Obmann-Stv. *SS 1998*
Obmann *WS 1998/99*

AktionsGemeinschaft Linz
Obmann *1999 – 2001*

Junge ÖVP Oberösterreich *seit 2001*
Referent für Bildung

Art of Arts – Eventmanagement
Vize-Präsident *seit 2001*

Junge Kunst, Kultur und Medien *seit 2002*
Obmann

Kindwelt Oberösterreich *seit 2002*
Landesobmann

ÖFFENTLICHE ÄMTER

seit 1999	Österreichische Hochschülerschaft Linz
	Mitglied des Akademischen Senats
	Mitglied der Universitätsversammlung
	Mitglied der Fakultätsvertretung TNF
	Mitglied diverser Institutskonferenzen
	Mandatar der Universitätsvertretung
	Studienrichtungsvertretung Informatik (Vorsitzender seit 2001)
1999 - 2001	*Mandatar des Fakultätskollegiums TNF*
seit 2001	Österreichische Hochschülerschaft
	Mandatar der Bundesvertretung

WEITERBILDUNG

1999	Rhetorik und Kommunikation (JAS)
2000	Konfliktmanagement (Perfact Training)
2000	Führen und Motivieren (Perfact Training)
2000	Emotionale Intelligenz (Perfact Training)
2000	Leadership (Perfact Training)
2000	„Power Academy“ (Perfact Training + Outdoor Consult)
2001	Ausbildung zum Kommunikationstrainer
2002	NLP Grundausbildung