INSTITUT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Fortgeschrittenenpraktikum

Portierung eines bestehenden CORBA-Systemmanagementagenten auf IBM OS/2 3.0

Bearbeiter: Gregor Hölscher

Aufgabensteller: Prof. Dr. Heinz-Gerd Hegering

Betreuer: Alexander Keller

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

In den vergangenen Jahren hat die Datenkommunikation immer weiter an Bedeutung gewonnen, und beschränkt sich nicht mehr nur auf große Rechenzentren. Vielmehr entstanden in Unternehmen große Rechnernetze mit verschiedenen Komponenten.
Diese heterogenen Netze von Rechnern müssen natürlich verwaltet und überwacht werden. Hierunter fallen z.B. die Überwachung, Steuerung und Verwaltung von Systemressourcen, aber auch die Verwaltung von Usern.
Um diese Aufgaben bewältigen zu können, wurde der Aufgabe des Systemmanagements in der letzten Zeit eine immer größere Rolle zugesprochen.
Neben diesen großen Netzen entstanden aber auch eine Vielzahl von kleinen PC-Netzen in kleinen und mittelständischen Betrieben. Auch für diese war und ist es wichtig, ein gutes und relativ kostengünstiges Systemmanagement zu haben.
An dieser beschriebenen Situation kann man sehr gut erkennen, daß der Trend immer mehr weg von den Großrechnern und hin zu den Client-Server-basierten Strukturen führt. Durch diese Heterogenität steigt aber auch die Komplexität des Managements.

Da man aber nicht in der Lage ist, ein Systemmanagement für ein komplettes System zu entwickeln, das auch in anderen Bereichen bzw. Netzen einsetzbar ist, sollen Techniken des Netzmanagements eingesetzt werden, um die Effektivität des Systemmanagements zu steigern.
Dies hat zur Folge, daß für bestimmte Workstation- oder auch PC-Netze sogenannte Systemmanagementagenten entwickelt werden sollen, die dem Netzadministrator die Überwachung, Steuerung, usw. erleichtern.

Um dieses Ziel auch mit größtmöglichem Erfolg erreichen zu können, werden bei der Realisierung Managementarchitekturen eingesetzt. Bei diesen existieren mehrere Produkte mit unterschiedlichen Managementprotokollen, wobei sich in letzter Zeit die Architektur der OMG-Gruppe, genannt *Common Object Request Broker Architecture* (CORBA), durchaus Vorteile verschaffen konnte. Die *Object Management Group* (OMG) wurde 1989 von Softwareentwicklern, Netzwerkbetreibern, Hardwareproduzenten und kommerziellen Anwendern von Computersystemen als internationaler, aber nicht profitorientierter Zusammenschluß gegründet.
Diese Architektur, auch als *Object Management Architecture* (OMA) bezeichnet, ist ein offener Ansatz, um verteilte, objektorientierte Systeme in heterogenen Netzen zu entwickeln. Dies ermöglicht die Zusammenarbeit von Anwendungen verschiedener Hersteller, unabhängig davon, für welches Betriebssystem bzw. für welche Hardware und vor allem in welcher Programmiersprache diese entwickelt wurden.

Ziel dieser Arbeit ist es, einen CORBA-basierten Systemmanagementagenten für das Betriebssystem OS/2 3.0 zu entwickeln. Als Grundlage bzw. zur Unterstützung dieser Arbeit wird ein Objektmodell herangezogen, das im Fortgeschrittenenpraktikum von Holger Sirtl [Sirt96] entstanden ist. Denn mit diesem entstandenen Objektmodell wurde schon ein Systemmanagementagent für Unix auf der Basis von CORBA implementiert. Dieser Agent soll jetzt auf OS/2 3.0 portiert werden.

2 Entwicklungsschritte

Im folgenden Kapitel soll näher auf die einzelnen Arbeitsschritte eingegangen werden, die nötig sind, um den Systemmanagementagenten von UNIX nach OS/2 3.0 zu portieren. Bevor die einzelnen Objektklassen näher betrachtet und auf ihre Verwendung hin überprüft werden, soll noch ein Einblick in die Entwicklung und die Historie des Agenten unter UNIX gewährt werden. Abschließend wird noch ein Blick auf die Realisierung des neuen Systemmanagementagenten geworfen.

2.1 Ausgangssituation

Am Beginn aller Arbeiten stand die MNM-UNIX-MIB. Sie bildete die Basis für alle weiteren durchgeführten Projekte. Bevor näher auf die weiteren Punkte eingegangen wird, soll der Begriff *Management Information Base* (MIB) erläutert werden. In der Abbildung 2-1 ist ein typisches Einsatzszenario einer MIB dargestellt.

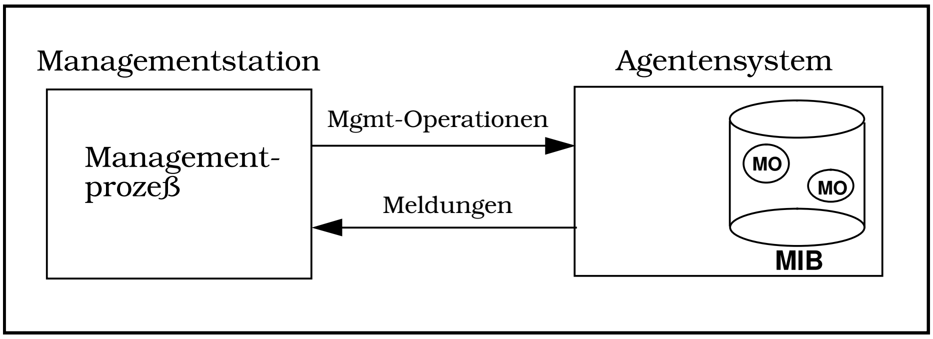

Abbildung 2-1: Einsatzszenario einer MIB

Damit ein Management von einzelnen Netzkomponenten möglich ist, sind diese mit Agenten ausgestattet, die mit einem zentralen Managementsystem kommunizieren. Dafür muß festgelegt werden, welche Informationen für das Management relevant sind, und wie sie in der Datenbasis der jeweiligen Kommunikationspartner abgelegt sind. Diese Aufgabe übernimmt die *Management Information Base* (MIB). Sie legt fest, welche Daten abfragbar oder manipulierbar sind und wie die einzelnen Datenobjekte (*Managed Objects*, MO) eindeutig auffindbar sind. Dies geschieht mittels eines Registrierungsbaums. Auf die einzelnen MIB-Variablen im Baum kann über einen eindeutigen Objektidentifier zugegriffen werden. Für eine Variable wird der Name, der Typ, der Zugriff, der Status, eine Beschreibung und der Ort im Baum angegeben.
Neben dieser MIB waren auch schon die einzelnen C-Prozeduren, die die Systemmanagementaufrufe unter UNIX realisieren, vorhanden.

Mit diesen zwei Vorgaben sollte ein Agent entwickelt werden, der den gestellten Anforderungen für das Management entsprach.
Dieses Problem sollte in einem Fortgeschrittenenpraktikum mit dem Titel “Objektorientierte Modellierung von Workstations für ein integriertes Systemmanagement” gelöst werden.
Als Hilfsmittel wurde das CASE-Tool *Software through Pictures* (StP) auf der Basis der *Object Modeling Technique* (OMT) zur Verfügung gestellt. Die OMT-Version von StP ist ein Multi-User-Werkzeug, das das Zeichnen der Modelle und das Navigieren zwischen verschiedenen Systemsichten komfortabel unterstützt. Darüber hinaus bietet StP Konsistenzprüfungen, Code- und Dokumentationsgenerierung, Versionskontrolle u.v.m. an. Damit stand also ein mächtiges Werkzeug für computerunterstützte Softwarekonstruktion zur Verfügung, das auch in allen Phasen durch leistungsfähige Editoren unterstützt wurde. Am Ende der Arbeit, die von Holger Sirtl [Sirt96] durchgeführt wurde, war ein optimiertes CORBA-konformes Objektmodell für Workstations entstanden.
Mit Hilfe dieses objektorientierten Modells und den gegebenen C-Prozeduren wurde dann ein UNIX-Systemmanagementagent für das Betriebssystem AIX 3.2.5 implementiert.

Die Aufgabe des vorliegenden Fortgeschrittenenpraktikums ist die Portierung dieses entwickelten Systemmanagementagenten auf das IBM Betriebssystem OS/2 3.0. Die notwendige Infrastruktur für diese Aufgabe wurde durch das IBM *SOMObjects Developer Toolkit* und den darin enthaltenen CORBA-konformen *Object Request Broker* (ORB) zur Verfügung gestellt. Zur Compilierung der einzelnen C-Systemprozeduren wurde die OS/2-Version des GNU-C-Compilers *gcc* (EMX-Version) verwendet.

Die vorhandenen C-Prozeduren für die Systemmanagementaufrufe unter UNIX können leider nicht verwendet werden. Somit müssen die meisten Aufrufe neu implemetiert werden. Hierfür stellt OS/2 3.0 eine Vielzahl von APIs zur Verfügung. Jedoch ist leider sehr schnell feststellbar, daß mit den vorhandenen APIs nicht alle UNIX-Aufrufe realisierbar sind und somit die Funktionalität des Agenten bei weitem nicht mit dem UNIX-Agenten vergleichbar ist. Diese Tatsache ist aber auch schon damit zu erklären, daß man von einem Multi-User-Betriebssystem zu einem Single-User-Betriebssystem gewechselt hat. Somit liegt die Funktionalität des Systemmanagementagenten unter OS/2 hauptsächlich im Bereich Monitoring.

2.2 Überarbeitungsphase der gegebenen Unterlagen

Wie zuvor schon erläutert, ist die Arbeitsgrundlage für die Portierung die optimierte objektorientierte MIB (siehe Abbildung 2-2) für UNIX-Workstations, die durch das Fortgeschrittenenpraktikum von Holger Sirtl [Sirt96] entstanden ist. Sie besteht aus einzelnen Objektklassen, die wiederum Attribute enthalten. Ebenso sind für einige Objektklassen Operationen definiert. Nachdem es eine objektorientierte MIB ist, gibt es Vererbung, Assoziationen und Aggregationen zwischen den Objektklassen.

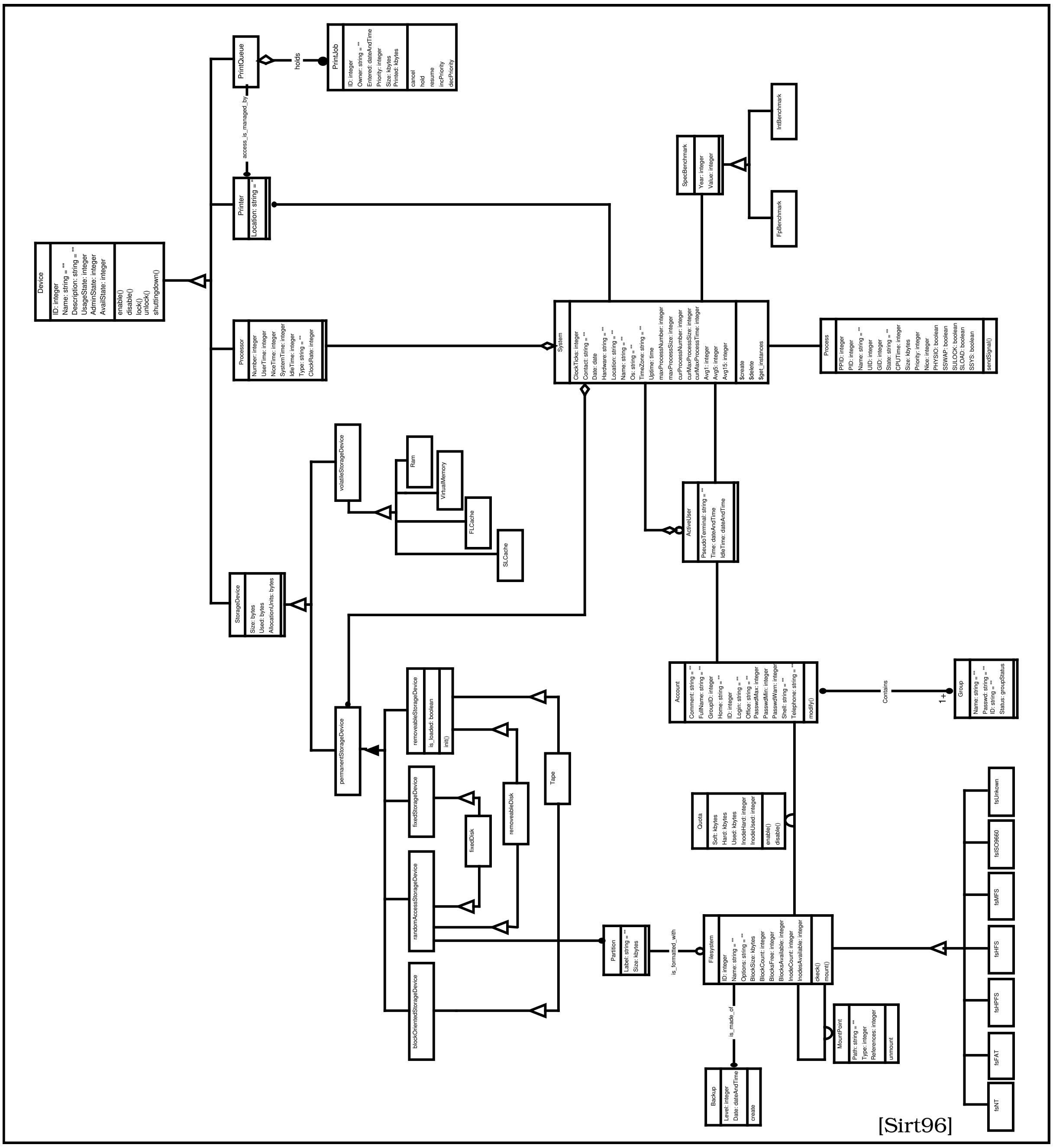

Abbildung 2-2: optimierte objektorientierte System-MIB

Als erster Schritt werden nun die einzelnen Objektklassen daraufhin überprüft, ob sie auch unter OS/2 3.0 sinnvoll einsetzbar sind.
Die nun folgende Betrachtung bezieht sich zuerst auf die einzelnen Objektklassen. Anschließend werden die einzelnen Attribute und Operationen auf ihre Verwendung überprüft.
Auf den ersten Blick stellt man sehr schnell fest, daß der größte Unterschied zwischen beiden Systemen in der Benutzerverwaltung liegt.
Unter UNIX können auf einer Workstation mehrere User mit eigenem Login und Passwort angelegt und zu bestimmten Gruppen zusammengefaßt werden. Daneben können bestimmte Systemressourcen durch den Systemadministrator verwaltet bzw. zugeteilt werden. Für diese Aufgaben stehen dem Verwalter spezielle Systemaufrufe zur Verfügung.

Betrachtet man jetzt OS/2 3.0, so kann man erkennen, daß eine Benutzerverwaltung mit Passwortvergabe nicht realisierbar ist. Auch eine Gruppierung von Benutzern zu bestimmten Benutzergruppen ist unter OS/2 nicht möglich.
Somit müssen die Objektklassen `ActiveUser`, `Account` und `Group` ersatzlos gestrichen werden. Auch die Objektklasse `Quota`, die unter UNIX die Verteilung von Systemressourcen steuert, wird nicht realisiert, da auch diese Funktionalität bei OS/2 nicht vorhanden ist.

Betrachtet man die Objektklasse `Filesystem` und vor allem die Objektklassen, die im engeren Zusammenhang damit stehen, genauer, so wird auch hier sehr bald klar, daß einige Klassen dem Rotstift zum Opfer fallen müssen. Unter UNIX sind die einzelnen möglichen Typen des Filesystems als eigene Klassen definiert.
Da man bei OS/2 nur zwischen dem Typ "FAT" und "HPFS" unterscheidet, wird der Filesystemtyp als Attribut in der Objektklasse `Filesystem` definiert.

Daneben werden auch die Objektklassen `MountPoint` und `Backup` bei der Realisierung nicht weiter berücksichtigt, da mit den vorhandenen APIs von OS/2 keine vernünftige Lösung realisierbar ist.

Ebenso fällt die Objektklasse `SpecBenchmark` inklusive ihrer Subklassen weg, da auch hier an eine Realisierung unter OS/2 nicht zu denken ist.

Die letzten Hindernisse treten auf, wenn man die Subklassen der `permanentStorageDevice`-Klasse einer genaueren Prüfung unterzieht. Als erstes muß demzufolge die Unterscheidung weggelassen werden, ob die vorhandenen Speichermedien fest installiert oder auswechselbar sind. Daneben hat man leider auch nicht die Möglichkeit zu überprüfen, ob die Speichermedien block- oder zeichenorientiert arbeiten.

Aus diesen Gründen wird eine neue Objektklasse `Disk` eingeführt, die soweit wie möglich die Aufgabe der bisherigen Subklassen unterhalb der `permanentStorage-Device`-Klasse übernimmt.

Die letzten zwei Klassen, die noch gestrichen werden mußten, findet man unterhalb der `volatileStorageDevice`-Klasse. Es sind die Klassen `SLCache` und `FLCache`, da auch für diese Klassen keine Unterstützung von OS/2 zur Verfügung steht.

Abschließend kann man sagen, daß doch eine relativ große Zahl von Objektklassen unter OS/2 weggefallen sind. Dies liegt aber hauptsächlich daran, daß der schon oben beschriebene Wechsel von einem Multi-User-Betriebssystem zu einem Single-User-Betriebssystem durchgeführt wurde. Von den 39 ursprünglichen Objektklassen in der objektorientierten System-MIB sind unter dem Betriebssystem OS/2 3.0 noch 15 Objektklassen vorhanden. Dies ist eine doch deutliche Verringerung gegenüber der bisherigen MIB, die sich natürlich auch später bei der Realisierung des Systemmanagementagenten bezüglich der Funktionalität bemerkbar macht.

Nachdem die Betrachtung der einzelnen Objektklassen abgeschlossen ist, sollen jetzt zunächst die einzelnen Attribute und Operationen in den verbliebenen Objektklassen auf ihre Verwendungsmöglichkeit überprüft werden.
Auch hier stellt man sehr bald fest, daß auch in diesem Bereich eine Vielzahl von Attributen nicht unterstützt werden kann, da von der Seite des Betriebssystems keine Unterstützung geliefert wird.

Startet man mit der Betrachtung in der obersten Klasse, der Vaterklasse `Device`, so müssen alleine hier fünf Attribute, nämlich ID, Description, UsageState, AdminState und AvailState gestrichen werden. Dies ist nötig, da man für diese Attribute unter OS/2 keine Systemprozeduren zur Verfügung hat. Als Attribut bleibt nur Name erhalten, der auch die Funktionalität des Attributs ID übernimmt. Die Funktionalität der einzelnen Attribute ist im Anhang B aufgeführt.

In der Objektklasse `Device` wird dagegen kein Attribut gestrichen, sondern ein neues Attribut State eingeführt, das dem Systemadministrator erlaubt, sich die prozentuale Auslastung der einzelnen Speichermedien ausgeben zu lassen.

Auch in der Objektklasse `Printer` wurde ein weiteres Attribut eingeführt, um den Informationsgehalt dieser Klasse noch zu steigern. Es ist das Attribut PrinterStatus. Damit erhält man Informationen über den derzeitigen Status des ausgewählten Druckers.

Im Gegensatz dazu steht die Objektklasse Processor, bei der sechs von sieben Attributen gestrichen werden müssen. Es sind die Attribute Number, UserTime, NiceTime, SystemTime, IdleTime und ClockRate. Diese stellen hauptsächlich Informationen bezüglich der CPU zur Verfügung und werden von OS/2 leider nicht in dem Umfang unterstützt, wie es bei UNIX der Fall ist. Eine genaue Beschreibung der einzelnen Attribute ist auch hier wieder im Anhang B zu finden.

Die Probleme mit den Abfragen bezüglich der Auslastung und Verwendung von CPU-Zeiten finden sich auch in zwei anderen Objektklassen wieder. So muß in der Objektklasse System das Attribut curMaxProcessTime und in der Objektklasse Process das Attribut CPUTime weggelassen werden. Dies sind aber leider nicht die einzigen Veränderungen.

In der Klasse System werden noch die Attribute Contact, Location, maxProcessNumber, curMaxProcessSize, Avg1, Avg5, Avg15 und in der Klasse Process die Attribute UID, GID, Nice, PHYSIO, SSWAP, SULOCK, SLOAD und SSYS gestrichen.
Diese Veränderungen sind teilweise wieder auf den Unterschied zwischen Multi-User- und Single-User-Betriebssysteme zurückzuführen und andererseits sind, wie in den meisten Fällen, keine adäquaten Systemaufrufe seitens des Betriebssystems OS/2 3.0 vorhanden.

In den beiden nun folgenden Objektklassen wurden teilweise Attribute ergänzt bzw. die Attribute komplett überarbeitet, um sie der vorgegebenen Situation besser anpassen zu können.

Die vorhandenen Attribute in der Objektklasse PrintJob wurden umbenannt, um die jeweilige Funktion noch zu verdeutlichen und aus den vorhandenen Attributen entstanden folgende: JobId, JobOwner, JobEntered, JobPrio, JobSize und JobPrinted.
Daneben wurden noch vier weitere Attribute hinzugefügt, nämlich: QueueName, QueueState, PrinterName und PrinterLocation. Auch diese Attribute dienen noch dazu, den Informationsgehalt dieser Klasse zu steigern.

Die Objektklasse Filesystem wurde dagegen völlig neue konzipiert, da eine Übernahme der alten Struktur aus oben genannten Gründen nicht möglich war. In dieser Klasse gibt es jetzt vier Attribute. FSName gibt den Namen des jeweiligen Filesystems wieder. Mit den Attributen FSBlockSize und FSBlocksFree wird eine Aussage über die Größe der Blöcke und die noch vorhandenen Blöcke des Filesystems getroffen. Das letzte Attribut FSType übernimmt die Aufgabe, dem Anwender den jeweils verwendeten Typ des Filesystems auszugeben.

An dem Attribut FSType kann man sehr gut erkennen, daß man die bei der ersten Überprüfung weggelassenen Klassen nicht einfach vergessen darf, sondern versuchen sollte, diese Klassen soweit wie möglich und nötig, eventuell als Attribute in einer anderen Klasse, wieder einzusetzen.
Abschließend kann man sagen, daß man eine große Anzahl von Attributen nicht wieder verwenden kann. Vergleicht man die Anzahl der Attribute in den jeweiligen Klassen, so sind es bei UNIX 66 und bei OS/2 nur 40 einzelne Attribute.

Betrachtet man noch die Operationen in den jeweiligen Klassen, so ist erkennbar, daß hier die meisten Operationen übernommen werden können. Nur in der Klasse `System` wurden die Operationen create, delete und get_instance gestrichen, da eine Realisierung unter OS/2 3.0 nicht möglich ist.

Somit kann man am Ende der Überarbeitungsphase sagen, daß eine vollständige Portierung der Funktionalität des Systemmanagementagenten von UNIX nach OS/2 3.0 nicht möglich ist. Gründe sind hierfür hauptsächlich fehlende Unterstützung bezüglich der Systemaufrufe von Seiten des Betriebssystems OS/2 3.0 und natürlich der schon angesprochene Unterschied zwischen Multi-User- und Single-User-Betriebssystem. Es besteht jedoch trotzdem die Hoffnung, daß durch die neue Version OS/2 4.0 eine Möglichkeit besteht, weitere Klassen bzw. Attribute des UNIX-Systemmanagementagenten zu realisieren.

2.3 Realisierungsphase

Aus den bisher gewonnenen Erkenntnissen bei der Überprüfung ist es nun möglich, eine neue optimierte und objektorientierte MIB zu erstellen. In der folgenden Abbildung 2-3 ist die MIB in der OMT-Notation von Rumbaugh dargestellt.

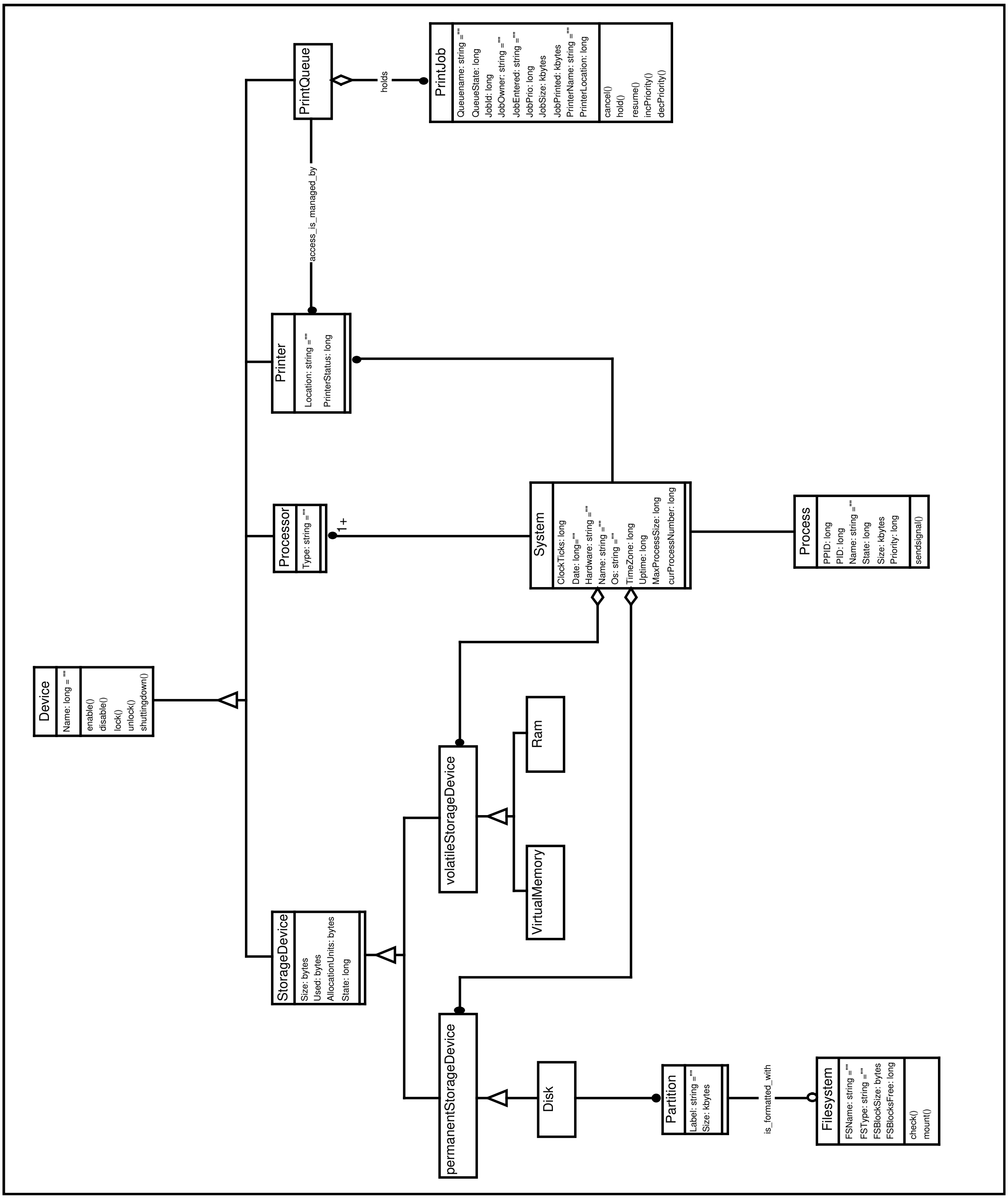

Abbildung 2-3: Objektmodell für den CORBA-Systemmanagementagenten

Diese hier abgebildete objektorientierte MIB ist die Grundlage für das weitere Vorgehen, um einen lauffähigen Systemmanagementagenten für das Betriebssystem OS/2 3.0 zu entwickeln.

Der erste Schritt ist, die einzelnen Objektklassen in einer IDL-Datei zu definieren, d.h. jede Klasse dieser MIB erhält eine eigene Datei und der Dateiname ist gleichzeitig auch der Klassenname. Die *Interface Definition Language* (IDL) dient als Verbindung zwischen Klassenimplementierungen, die in unterschiedlichen Programmiersprachen geschrieben wurden. Mit Hilfe dieser Sprache ist man in der Lage, die Attribute und Methoden einer Klasse zu beschreiben, um damit auch später Objekte dieser Klasse ansprechen zu können. Dies wird auch als *Public Interface* bezeichnet. Zusätzlich können in den IDL-Dateien die exakten Beziehungen zwischen den einzelnen Klassen modelliert werden. Damit werden Vererbung, Assoziationen und Aggregationen in der IDL-Datei sichtbar und sind auch später im Agenten vollständig einsetzbar. Eine Aufstellung aller IDL-Dateien dieser MIB ist im Anhang C enthalten.

Nachdem man alle einzelnen Objektklassen in IDL-Dateien umgesetzt hat, müssen diese mit dem SOM-Compiler kompiliert werden.
Der hierfür notwendige Befehl lautet: **sc -s“c;ih;h“ *<dateiname.idl>***.
Wird der SOM-Compiler mit diesem Befehl aufgerufen, so werden zwei Header-Dateien und eine Implementierungsvorlage in der gewünschten Programmiersprache, im vorliegenden Fall die Sprache C, generiert. Die Implementierungsvorlage wird dabei auch als Template bezeichnet.
Die beiden Header-Dateien enthalten in den .h-Dateien die Header-Bindings für die Clients und in den .ih-Dateien die jeweiligen für die Objektimplementierung. In der .c-Datei ist die Implementierungsvorlage für die jeweilige Objektklasse enthalten.

Nachdem damit die erste Grundlage für den Agenten gelegt ist, muß im nächsten Schritt die Implementierungsvorlage ergänzt bzw. gefüllt werden. Eine genaue Beschreibung dieser Vorgehensweise wird im Kapitel 3.2 vorgestellt.
Hat man auch diesen Arbeitsschritt abgeschlossen, besteht die letzte Aufgabe darin, ein Clientprogramm zu schreiben, das die erzeugten Klassen schließlich auch verwendet. Hierzu müssen im Clientprogramm, wie schon oben beschrieben, die Header-Dateien mit der Endung „.h“ verwendet werden.
Ist auch dies geschehen, wird durch das Kompilieren und Linken der erstellten Dateien ein lauffähiges Clientprogramm erzeugt. Dies ist dann ohne weitere Arbeiten sofort ausführbar.
Um diese doch sehr theoretische Abhandlung genauer zu erläutern, wird im nächsten Kapitel eine Objektklasse aus der MIB herausgenommen und an dieser werden dann beispielhaft die oben beschriebenen Entwicklungschritte nochmals erläutert.

3 Implementierungsphase

Dieses Kapitel wird auf der Basis der Objektklasse `System` die Implementierungsphase beispielhaft aufzeigen. Dabei wird auch auf einige Befehle in der IDL-Syntax näher eingegangen.

3.1 IDL-Dateien unter SOM

Wie schon erwähnt, handelt es sich bei der SOM-IDL um eine Klassenbeschreibungssprache, die dem CORBA-Standard der OMG entspricht und deren Syntax sich an der Programmiersprache C++ orientiert. Somit sind auch alle wesentlichen Funktionen für objektorientierte Systeme, wie Vererbung usw. enthalten. Zusätzlich erweitert SOM-IDL den CORBA-Standard in zwei wesentlichen Punkten. Einerseits werden Pointer eingeführt, die dazu dienen die Kompatibilität zu SOM 1.0 zu gewährleisten, andererseits gibt es Implementierungsangaben, die die Austauschbarkeit von Modulen ermöglichen.
Als Ausgangspunkt für die weiteren Betrachtungen wird die Klasse `System` (siehe Abbildung 3-1) aus der schon erstellten objektorientierten MIB herausgenommen.

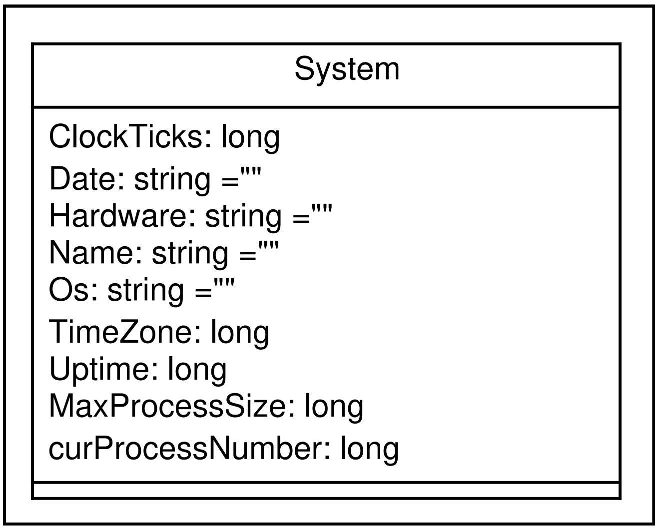

Abbildung 3-1: Ausschnitt aus dem Objektmodell

Aus dieser abgebildeten Objektklasse soll jetzt eine IDL-Datei entstehen. Als Dateiname wird die Bezeichnung der Objektklasse übernommen. In diesem Fall sind neun Attribute zu definieren und natürlich die Abhängigkeiten der Objektklasse in der gesamten MIB. Diese sind in der Abbildung 2-3 dargestellt. Methoden sind in dieser Klasse keine definiert.

Wie schon erwähnt, ist die SOM-IDL sehr an die Programmiersprache C++ angelehnt und so ist es nicht verwunderlich, daß im ersten Abschnitt der Datei ein ähnlicher Aufbau zu erkennen ist, weshalb darauf auch nicht näher eingegangen wird.

Grundsätzlich kann man sagen, daß die Schnittstellenbeschreibung einer SOM-Klasse aus zwei großen Teilen besteht. Der eine Teil enthält CORBA-spezifische Eigenschaften, der zweite Teil die SOM-spezifischen.
In der Abbildung 3-2 ist der CORBA-spezifische Teil der Klasse `System` dargestellt.

```
#ifndef _System_idl_
#define _System_idl_

// class declarations
interface System;
// class declarations end

// class definition
interface System : SOMObject
{
//class members
        readonly attribute long ClockTicks;
        attribute string Date;
        readonly attribute string Hardware;
        readonly attribute string Name;
        readonly attribute string Os;
        readonly attribute long TimeZone;
        readonly attribute long UpTime;
        readonly attribute long MaxProcessSize;
        readonly attribute long curProcessNumber;
        readonly attribute SOMObject assnPrinter;
        readonly attribute SOMObject assnProcess;
        readonly attribute SOMObject aggrProcessor;
        readonly attribute SOMObject aggrvolatileStorageDevice;
        readonly attribute SOMObject aggrpermanenStorageDevice;
// class members end
```

Abbildung 3-2: CORBA-spezifischer Teil der Datei System.idl

In diesem Teil der Datei wird die Klasse benannt, hier `System`. Da eine SOM-Klasse aber immer von mindestens einer Klasse abgeleitet werden muß, wird auch die Vaterklasse angegeben. Sollte dies durch Vererbung geschehen, ist es sofort eindeutig. Im vorliegenden Fall erkennt man jedoch, daß diese Klasse von keiner anderen Klasse erbt. Somit wird die Klasse `System` von der Basisklasse aller SOM-Klassen abgeleitet, nämlich von `SOMObject`. Diese Klasse enthält alle Basisroutinen, die notwendig sind, um Objekte zu erzeugen und zu löschen.
Mit den folgenden Befehlen „attribute“ werden die in der MIB beschriebenen Attribute gekennzeichnet.
Dabei stehen folgende Datentypen zur Verfügung: Fließkommatypen, Zeichenkettentypen, Aufzählungstypen, Feldtypen, und Objekttypen.
Zu diesem Zeitpunkt wird auch schon festgelegt, auf welche Art man auf die Attribute zugreifen kann, d.h. nur lesend (*readonly*) oder auch schreibend.

In der vorliegenden Beispielklasse erkennt man bei den letzten fünf Attributen noch eine Besonderheit. Diese dort aufgeführten Attribute sind in der Objektklasse nicht direkt definiert, sondern entstehen dadurch, daß man die Beziehungen dieser Klasse mit anderen Klassen modelliert.
So spiegelt die Zeile

readonly attribute SOMObject assnPrinter

eine Assoziation zwischen der Klasse `System` und der Klasse `Printer` wieder.
Damit wird eine semantische Beziehung zwischen zwei Klassen dargestellt, die in einem ER-Modell einer *relationship* entsprechen würde, also einer Beziehung zwischen zwei Entitäten eines Systems.

Daneben gibt es dann noch die Aggregation, die oft auch als “part of“-Beziehung bezeichnet wird. Damit wird ausgedrückt, daß eine Instanz einer Klasse Instanzen einer anderen Klasse enthält. Diese Beziehung wird in der SOM-IDL ebenfalls durch die Angabe bei den Attributen ausgedrückt.
Ein Beispiel ist folgende Zeile in der Klasse `System`:

readonly attribute SOMObject aggrProcessor.

Sollten in der Objektklasse noch Methoden definiert sein, müssen diese auch in diesem Teil angegeben werden. Dabei müssen die Eingabeparameter in der Prozedur mit “in“, die Ein- und Ausgabeparameter mit “inout“ und die Ausgabeparameter mit “out“ bezeichnet werden. Im vorliegenden Beispiel sind keine Methoden definiert, jedoch enthalten einige IDL-Dateien im Anhang C Beispiele für diesen Fall.

Der SOM-spezifische Teil wird durch **#ifdef __SOM-IDL__ ... #endif** und zusätzlich durch das Schlüsselwort *implementation* gekennzeichnet. Dies ist auch in der Abbildung 3-3 erkennbar, die nur den SOM-spezifischen Teil der IDL-Datei enthält.

```
#ifdef __SOMIDL__
implementation{
        dllname = „System.dll“;
        releaseorder: ClockTicks, Date, Hardware, Name, Os, TimeZone,
                      Uptime, maxProcessSize, curProcessNumber, assnPrinter,
                      assnProcess, aggrProcessor, aggrvolatileStorageDevice,
                      aggrpermanentStorageDevice, _get_ClockTicks, _get_Date,
                      _set_Date, _get_Hardware,_get_Name, _get_Os,
                      _get_TimeZone, _get_UpTime, _get_MaxProcessNumber,
                      get_MaxProcessSize,_get_curProcessNumber,
                      _get_assnPrinter, _get_assnProcess, _get_aggrProcessor,
                      _get_aggrvolatileStorageDevice,
                      _get_aggrpermanentStorageDevice;
        ClockTicks: noget;
        Date: noget, noset;
        Hardware: noget;
        Name: noget;
        Os: noget;
        TimeZone: noget;
        UpTime: noget;
        maxProcessSize: noget;
        curProcessNumber: noget;
        assnPrinter: noget;
        assnProcess: noget;
        assnProcessor: noget;
        aggrvolatileStorageDevice: noget;
        aggrpermanentStorageDevice: noget;
        somInit: override;
        somUninit: override;
};
#endif /* __SOMIDL__ */
```

Abbildung 3-3: SOM-spezifischer Teil der Datei System.idl

Mit dem Statement *releaseorder* werden für jedes Attribut die Einträge für **_get_(Attribut)** und **_set_(Attribut)**, soweit vorhanden, und die Methoden der Reihenfolge nach aufgeführt. Mit dieser Reihenfolge wird dann durch den SOM-Compiler eine Referenzierungstabelle aufgestellt, über die dann der Zugriff auf die Attribute und Methoden gesteuert wird.
Der Vorteil dieser Vorgehensweise liegt darin, daß trotz struktureller Änderungen eine Binärunabhängigkeit erhalten bleibt und somit eine neue Übersetzung bei Codeänderungen nicht notwendig ist. Somit können neue Methoden ergänzt und schon vorhandene verändert werden, ohne daß eine Neukompilierung vorgenommen werden muß.

Im letzten Teil dieser Datei wird für jedes Attribut festgelegt, ob ein Methodenrumpf automatisch für die jeweilige Methode erzeugt werden soll oder nicht. In unserem Fall wird durch "noget" der Compiler dazu veranlaßt diesen Methodenrumpf zu erzeugen, der später nur noch durch den entsprechenden C-Code ergänzt werden muß. Das Statement "noset" bewirkt das gleiche, nur handelt es sich hier um eine set-Methode.
Das letzte Statement, das in diesem Zusammenhang auftreten kann, ist "override". Dies bedeutet, daß die angegebene Methode in einer anderen Klasse definiert und in dieser Klasse nur abgeleitet ist, und überschrieben werden soll.
Die beiden letzten Methoden in diesem Abschnitt stellen die Basisfunktionalität zur Verfügung, die notwendig ist, um das Erzeugen und Löschen von Klassen zu bewirken.

Nachdem nun die Objektklasse `System` vollständig in eine Datei, die der SOM-IDL Syntax entspricht, umgesetzt wurde, muß diese nun mit dem SOM-Compiler kompiliert werden. Der hierzu notwendige Befehl lautet bei Verwendung der Programmiersprache C wie folgt:

sc -s"c;ih;h" <System.idl>.

Damit werden nun zwei Header-Dateien und die Implementierungsvorlage für die Sprache C erzeugt.
Für den Fall, daß man an der IDL-Datei nochmals Änderungen vornehmen muß, werden bei der erneuten Übersetzung mit dem SOM-Compiler nur die Stellen verändert, die sich in der IDL-Datei verändert haben. Somit bleibt ein schon bestehender Code auch nach der Übersetzung vollständig erhalten. Dies ist ein großer Vorteil, vor allem wenn man schon bei manchen Methodenrümpfen den noch nötigen C-Code ergänzt hat. Auf eine nähere Betrachtung der beiden Header-Dateien muß leider aus Platzgründen verzichtet werden. Die Implementierungsvorlage wird im folgenden Kapitel näher betrachtet.

3.2 C-Code für die Systemprozeduren

Als Arbeitsgrundlage für die nun anstehende Arbeit dient hauptsächlich die durch den SOM-Compiler erzeugte Implementierungsvorlage für die Sprache C.
Jedoch ist es vorher notwendig, die einzelnen Systemprozeduren zu implementieren. Im vorliegenden Fall konnte man die UNIX-Systemprozeduren leider nicht als Grundlage für die Implementierung verwenden, da die APIs unter OS/2 3.0 keine Ähnlichkeiten mit denen von UNIX aufwiesen.
Deshalb war es von Anfang an nötig, alle Systemprozeduren neu zu implementieren. Da aber sehr viele Systemaufrufe zwar bei UNIX, aber nicht bei OS/2 möglich waren, mußten viele Aufrufe leider gestrichen werden.
Um einen Eindruck zu vermitteln, wie eine Systemprozedur bei OS/2 implementiert wird, ist in der folgenden Abbildung eine Methode aus der Objektklasse, und zwar get_sysDate, dargestellt.

```
/* get_sysDate.c */

#define INCL_DOSDATEMTIME
#include <stdio.h>
#include <os2.h>
#include „get_sysDate.h“

long * get_sysDate()
{
        DATETIME DateTime;


        /* Abfragen der aktuellen Systemzeit */
        DosGetDateTime (&DateTime);
        return &DateTime
}
```

Abbildung 3-4: Systemprozedur get_sysDate.c

Hat man für alle definierten Attribute Prozeduren in der Sprache C verfaßt, so müssen die einzelnen Systemaufrufe in der Implementierungsvorlage ergänzt werden. Daneben müssen noch einige weitere Veränderungen bzw. Ergänzungen durchgeführt werden.
Diese angesprochenen Veränderungen sind in der nun folgenden Darstellung in *kursiver* und **fetter** Schrift hervorgehoben. Weitere Erläuterungen werden nach der Abbildung 3-5 besprochen.

Zusätzlich soll dort auch der grundsätzliche Aufbau der Implementierungsvorlage behandelt werden. Aus Platzgründen sind in diesem Beispiel aber nur die zwei Methoden get_sysDate und set_sysDate abgebildet. Die anderen Methoden werden jedoch ebenso implementiert.

```
/* This file was generated by the SOM Compiler and Emitter Framework.
* Generated using template emitter:
*   SOM Emitter emitctm: 2.23.1.9
*/

#ifndef SOM_Module_system_Source
#define SOM_Module_system_Source
#endif
#define System_Class_Source

#include „System.ih“
#include „get_sysClockTick.h“
#include „get_sysDate.h“
#include „set_sysDate.h“
#include „get_sysHardware.h“
#inlcude „get_sysOs.h“
#include „get_Timezone.h“
#inlcude „get_sysUpTime.h“
#inlcude „get_MaxProcessSize.h“
#inlcude „get_curProcessNumber.h“

/*
* Method from the IDL attribute statement:
* „attribute long Date“
*/

SOM_Scope long SOMLINK _get_Date(System *somSelf, Environment *ev)
{
        long Date;
        SystemData *somThis = SystemGetData(somSelf);
        SystemMethodDebug(„System“, „_get_Date“);
        Date = get_sysDate();
        /* Return statement to be customized: */
        return (Date);
}
/*
* Method from the IDL attribute statement:
* „attribute string Date“
*/

SOM_Scope void SOMLINK _set_Date(System *somSlef, Environment *ev, string Date)
{
        SystemDate *somThis = SystemGetData(somSelf);
        SystemMethodDebug(„System“, „_set_Date“);
        set_sysDate(); }
/*
* class members end
*/
```

Abbildung 3-5: Implementierungsvorlage der Klasse System

Betrachtet man jetzt einen generierten Methodenrumpf, so stellt man fest, daß in der SOM-Methode ein Pointer auf das Objekt (somSelf) und daneben noch für die Fehlerbehandlung eine Environment-Variable (ev) übergeben wird. Um später auf die Datenstruktur des jeweiligen Objekts zugreifen zu können, wird auch hier ein Pointer (somThis) erzeugt.
Eine weitere Zeile im obigen Methodenrumpf enthält z.B. die Anweisung „SystemMethodDebug(„System“, „_get_date“)“. Sie stellt ein Makro dar, das vom Anwender als Aufruf einer Debug-Funktion verwendet werden kann.

Diese Verbindungen, in der Abbildung 3-5 zwischen der Programmiersprache C und den sprachneutralen binären Klassen, nennt man allgemein *Bindings*. Spricht man von der Implementierung der Klassen, so spricht man *Implementation Bindings* und *Usage Bindings* für die Anwendungen, die diese Klassen dann auch verwenden.

Mit Abschluß dieses Vorgangs haben wir aus den Objekten in der objektorientierten MIB SOM-Objekte erstellt. Der Zugriff auf diese Objekte kann dann auf drei verschiedenen Wegen geschehen:
Offset Resolution: Dieser Aufruf entspricht in etwa dem Aufruf einer virtuellen Funktion bei der Programmiersprache C++. Man kann sagen, daß dieser Aufruf der Standardweg bei der Verwendung von SOM mit C ist. Hierfür müssen zur Übersetzungszeit der Name der Klasse und der Name der Methode bekannt sein. Daneben muß die aufzurufende Methode Bestandteil der IDL-Beschreibung einer Klasse sein, d.h. SOM-Klassen können genauso wie C-Funktionen verwendet werden.

Name-Lookup Resolution: Eine Methode wird mit Hilfe des SOM-APIs aufgerufen. Voraussetzung ist, daß der Name der aufzurufenden Methode oder einer Klasse erst zur Laufzeit bekannt ist.

Dispatch-Function Resolution: Dies ist der flexibelste, aber auch der teuerste Weg bezüglich der Laufzeit, da man mit jeder Methode jedes SOM-Objekt aufrufen kann. Sogar die Parameterübergabe wird während der Laufzeit durchgeführt.

Hat man den oben beschriebenen Vorgang für alle IDL-Dateien abgeschlossen, ist das Ziel, einen Systemmanagementagenten zu entwickeln, fast erreicht. Der letzte Schritt, der jetzt noch notwendig ist, besteht darin, ein Clientprogramm zu schreiben, das die definierten Objektklassen verwendet.

3.3 Clientprogramm

Der letzte Arbeitsschritt, der notwendig ist, um einen lauffähigen Systemmanagementagenten zu erhalten, liegt darin, ein Clientprogramm zu implementieren. Dieser Client soll dann in der Lage sein, die in den vorherigen Kapiteln definierten Objektklassen zu verwenden. Grundsätzlich benötigt man dazu fünf Schritte bzw. Befehlszeilen.

Der erste Schritt zur Verwendung ist das Einbinden der Header-Datei der jeweiligen gewünschten SOM-Klasse. Dadurch wird der Klassentyp bekanntgemacht und kann im weiteren Vorgehen verwendet werden.

Die Deklaration der gewünschten Variablen ist der nächste Schritt auf dem Weg zum Clientprogramm. Dazu dient der Aufruf **<Klassenname> obj**, der einen Zeiger auf das zu instanzierende Objekt darstellt. Die zweite wichtige Variable, die zu deklarieren ist, ist ein Zeiger auf eine Umgebungsvariable **ev**. Die Initialisierung geschieht in diesem Fall mit dem SOM-API *somGetGlobalEnvironment*. Die Variable wird dabei jedem Methodenaufruf einer SOM-Klasse mitgegeben, um damit die Fehlerbehandlung zu bewerkstelligen.

Danach wird die Variable **obj** vom Typ <KlassenName> mit dem SOM-Makro **<KlassenNamen>New()** instanziert. Zu beachten ist dabei, daß bei der Instanzierung von SOM-Klassen keine Initialisierungsvariablen übergeben werden können, da es für SOM-Objekte keine Konstruktoren mit Parameterübergabe gibt. Jedoch wird eine leere Parameterklammer angegeben, um die Analogie zu C++ zu gewährleisten, die an mehreren Stellen der IDL-Definition zu erkennen ist. Es gibt aber trotzdem noch eine Möglichkeit, dieses Problem zu lösen. Dafür definiert man dann eine Metaklasse, die ja als Teil von SOM-Klassen zur Verwaltung von Instanzen verwendet wird.

Im nächsten Schritt kann man die Attribute der jeweiligen Klassen bzw. die dazugehörigen Methoden verwenden. Für diese Methodenauflösung wird sehr oft die *Offset Resolution* herangezogen, da sie sehr effizient arbeitet. Dabei werden die Methoden mit **_<MethodenName>(<ObjektName>, ev, <Parameterübergabe>)** aufgerufen. Vergleicht man diesen Aufruf mit dem C-Funktionsaufruf, so kann man sagen, daß der C-Aufruf zweimal schneller arbeitet. Dies kommt daher, daß die Methoden von SOM über eine eigene Methodentabelle adressiert werden.

Nachdem man mit diesen instanzierten Objekten gearbeitet hat, ist es aber auch notwendig, diese wieder freizugeben. Auch in diesem Fall wird ein SOM-API zur Verfügung gestellt. Der Aufruf lautet: **_somFree(obj)**, wobei **obj** die zu Beginn initialisierte Objektvariable ist.

Um diese theoretische Abhandlung über einen Client nochmals zu verdeutlichen, ist in der Abbildung 3-6 ein Beispielclient enthalten.

```
#include „system.h“          /* include the Headerfile for System */

main()
{
        /*declare a variable (obj) that is a pointer
        * to an instance of the System class */
        System obj;

        /* declare a variable (ev) that is a pointer
        * to an environment variable */
        Environment *ev;
        ev = somGetGlobalEnvironment();

        /* create an instance of the System class
        * and store a pointer to it in obj */
        obj = SystemNew();

        /* Execute the „get_sysClockTick“ method */
        _get_sysClockTick(obj, ev);

        /* Free the instance */
        _somFree(obj);
        return(0);
}
```

Abbildung 3-6: Beispielclient für die Klasse `System`

Damit liegt nun ein prototypisches Clientprogramm vor, das zum Abschluß noch mit der zugehörigen .c-Datei der Objektklasse `System` kompiliert und gelinkt werden muß. Die Befehle hierfür lauten wie folgt:

set LIB=d:\toolkit\som\lib;d:\SOFTWARE\ibmcpp\lib und

icc -I. -I\d:\som\include -Fe system main.c system.c somtk.lib.

Bei diesen Befehlen ist zu beachten, daß sie auf die beim Fortgeschrittenenpraktikum verwendete Umgebung abgestimmt sind.

In der ersten Befehlszeile werden Pfadinformationen angegeben, vor allem das Verzeichnis für das SOM-Paket. Dies ist notwendig, da die definierten Klassen nicht als Bibliotheken definiert wurden. Dies erlaubt aber dann, daß das Clientprogramm und die Klassenimplementierung gleichzeitig kompiliert werden können. Voraussetzung dafür ist aber noch, daß beide Implementierungen in der gleichen Programmiersprache geschrieben sind, was in vorliegendem Fall auch geschehen ist.

Die zweite Befehlszeile enthält dann den endgültigen Aufruf zum Kompilieren und Linken. Im vorliegenden Beispiel werden die Objektklasse `System` und das Clientprogramm "main.c" kompiliert und gelinkt.
Ist dieser Vorgang ohne Fehlermeldungen erfolgreich abgeschlossen, so kann man jetzt das geschriebene Clientprogramm aufrufen.
Ergänzt man dieses prototypische Clientprogramm soweit, daß alle definierten Objektklassen enthalten sind, so ist das Ziel erreicht, ein funktionsfähiger Systemmanagementagent auf Basis von CORBA für das Betriebssystem OS/2 zu erhalten.

4 Zusammenfassung und Ausblick

Betrachtet man die Arbeit rückblickend, so stellt man fest, daß die Funktionalität des Systemmanagementagenten doch sehr stark eingeschränkt ist.
Dies liegt, wie bereits auch schon erläutert, hauptsächlich daran, daß mit OS/2 3.0 ein Single-User Betriebssystem vorliegt. Ein zweiter Grund ist, daß nicht alle notwendigen APIs zur Verfügung gestellt werden, z.B. wie Abfragen von CPU-Zeiten.

Daneben sind auch immer wieder Probleme aufgetreten, wenn es darum ging, das Netzwerk zu überprüfen. So ist es z.B. während des Fortgeschrittenenpraktikums nicht gelungen, eine Verbindung zum Netzwerkdrucker herzustellen. Die Objektklassen wurden deshalb auch nur für einen lokalen Drucker getestet. Somit dürften beim Netzwerkdrucker in Bezug auf die Anwendung der speziellen Objektklassen auch keine Probleme auftreten.

Nachdem aber während des Fortgeschrittenenpraktikums die neue Version des Betriebssystems erschienen ist, besteht die Hoffnung, daß sich damit vor allem die Probleme mit dem Netzwerk und der Anbindung des Netzwerkdruckers lösen lassen. Denn der entwickelte Systemmanagementagent dient ja nicht nur dazu, einen lokalen Computer zu überwachen, sondern hauptsächlich dazu, von einer Managementstation aus, mehrere Stationen im Netz zu überwachen. Vielleicht werden unter OS/2 4.0 auch weitere APIs zur Verfügung gestellt, die es dem Systemmanager erlauben, nicht nur passiv ein Netz zu überwachen, sondern auch aktiv auf einzelne Stationen zuzugreifen. Diese Ergänzung würden auch keine größeren Probleme aufwerfen, da man nur die vorhandenen Objektklassen erweitern bzw. neue Klassen einführen müßte. Denn auch die weiteren Tätigkeiten, die noch notwendig sind, werden durch den SOM-Compiler auf eine sehr angenehme und effektive Art unterstützt.

Somit kann man abschließend zu diesem Fortgeschrittenenpraktikum sagen, daß es sich durchaus lohnt, einen Systemmanagementagenten auf der Basis von CORBA zu entwickeln. Vor allem durch die Verwendung des *SOM Developer Toolkits* wird die Arbeit stark erleichtert und man ist damit auch in der Lage, sehr schnell auf neue Entwicklungen zu reagieren, da durch die CORBA-Architektur eine spätere Ergänzungder Funktionalität des Systemmanagementagenten leicht durchzuführen ist. Ebenso ist man in der Lage, unterschiedliche Programmiersprachen zu verwenden. So besteht zukünftig auch die Möglichkeit, die Programmiersprache Java einzusetzen. Damit wird die Gestaltung und Implementierung einer graphischen Oberflächen erleichtert, da unter Java sehr viele Klassen zur Verfügung gestellt werden, die den Entwickler bei der Implementierung der Oberfläche unterstützen.

Somit sollte man in der Zukunft auf jeden Fall darauf achten, daß die CORBA-Architektur nicht vernachlässigt wird, da sie mit ihrer Funktionalität eine sehr angenehme Arbeitsumgebung bietet und sie den Softwareentwickler in vielen Bereichen sehr gut unterstützt. Trotzdem darf man dabei aber nicht vergessen, daß das SOM Paket schon sehr umfangreich ist, obwohl noch lange nicht alle Punkte des CORBA-Standards verwirklicht wurden.
Abschließend kann man sagen, daß mit CORBA eine zukunftsträchtige Architektur vorliegt, die vor allem im Systemmanagementbereich große Möglichkeiten bietet, da mit dieser Architektur eine Plattform- und Betriebssystemunabhängigkeit erreicht werden kann.

Anhang A: OS/2 Systemmanagementagent

Device_group:

Name	r	long	get_devName	Name und Beschreibung von Devices (DosDevConfig)

Filesystem_group:

FSName	r	string	get_FSName	Name des Filesystems (DosQueryFSAttach)
FSType	r	string	get_FSType	Typ des Filesystems (DosQueryFSAttach)
FSBlockSize	r	bytes	get_FSBlockSize	Blockgröße des Systems (DosQueryFSInfo)
FSBlocksFree	r	long	get_FSBlocksFree	Anzahl der freien Blöcke im gemounteten System (DosQueryFSInfo)

Partition_group:

Label	r	string	get_partitionLabel	Beschreibung der Partition (DosQueryFSInfo)
Size	r	kbytes	get_partitionSize	Größe der Partition (DosQueryFSInfo)

Printer_group:

Location	r	string	get_printerLocation	Ort des Druckers (lokal oder remote) (SplEnumPrinter)
PrinterStatus	r	long	get_printerStatus	Printerstatus (SplQueryDevice)

Processor_group:

Type	r	string	get_cpuType	Typ der CPU

Printqueue_group / Printjob_group:

Queuename	r	string	get_queueName	Name der Queue (SplEnumJob)
Queuestatus	r	long	get_queueState	Status der Queue (SplEnumJob)
JobId	r	long	get_queueJobId	Job-Identifier (SplEnumJob)
JobOwner	r	string	get_queueJobOwner	Name Jobbesitzer
JobEntered	r	string	get_queueJobEntered	Uhrzeit des Eintritts in die Queue (SplEnumJob)
JobPrio	r	long	get_queueJobPrio	Priorität des Jobs (SplEnumJob)
JobSize	r	kbytes	get_queueJobSize	Größe des Jobs (SplEnumJob)
JobPrinted	r	kbytes	get_queueJobPrinted	wieviele Minuten wurden bisher gedruckt (SplEnumDevice)
PrinterName	r	string	get_queuePrinterName	Druckername (nicht bei lokalen Druckern) (SplEnumQueue)
PrinterLocation	r	long	get_queuePrinter Location	Ort des Druckers (SplEnumPrinter)

Process_group:

PPID	r	long	get_processPPID	ParentProzeßIdentifier (DosExecPgm + PSTAT /C)
PID	r	long	get_processPID	ProzeßIdentifier (DosExecPgm + PSTAT /C)
Name	r	string	get_processName	Aufrufname des Prozesses (DosExecPgm + PSTAT /C
State	r	long	get_processState	Prozeßzustand (DosExecPgm + PSTAT /C)
Priority	r	long	get_processPriority	Priorität (DosExecPgm + PSTAT /C)
Signal	w	long	set_processSignal	angegebenes Signal zum Prozeß senden (sendsignal())

StorageDevice_group:

Size	r	bytes	get_stoSize	Speichergröße (DosQueryFSInfo)
Used	r	bytes	get_stoUsed	verwendeter Speicher (DosQueryFSInfo)
AllocationUnits	r	bytes	get_stoAllocationUnits	Größe einer allokierten Speichereinheit (DosQueryFSInfo)
State	r	long	get_stoState	Prozentsatz des verwendeten Speichers (DosQueryFSInfo)

System_group:

ClockTicks	r	long	get_sysClockTick	clock ticks in tenths of a millisecond (DosQuerySysInfo)
Date	rw	long	get_sysDate set_sysDate	lokales Datum / Zeit (DosGetDateTime/ DosSetDateTime)
Hardware	rw	string	get_sysHardware	Bezeichnung der Hardware (DosDevConfig)
Name	r	string	get_sysName	System/Domainname (DosExecPgm + hostname)
Os	r	string	get_sysOs	Betriebssystem (DosQuerySysInfo)
TimeZone	r	long	get_timezone	Zeitzone (DosGetDateTime)
UpTime	r	long	get_sysUptime	Laufzeit des Rechners (DosQuerySysInfo)
MaxProcess Size	r	long	get_MaxProcessSize	maximaler Speicher für einen Prozeß in kbytes (DosQuerySysInfo)
curProcess Number	r	long	get_curProcessNumber	Anzahl geladener Prozesse (DosExecPGM + PSTAT /C)

Anhang B: Unix-Systemmanagementagent

Account group:

Comment	rw	string	get_userComment	Logininformation, die benötigt wird, um Batchjobs an Mainframes zu übermitteln
FullName	rw	string	get_userFullName	der volle Name des Benutzers
GroupID	rw	long	get_userGroupID	Gruppenidentifikator, der die Benutzergruppe identifiziert
Home	rw	string	get_userHome	Homeverzeichnis des Benutzers
ID	rw	long	get_userID	Benutzeridentifikator, der den Zugang zum BS durch eine ID identifiziert
Office	rw	string	get_userOffice	Büro des Benutzers
Passwd	rw	long	get_userPasswd	Paßwort des Benutzers
Shell	rw	string	get_userShell	Loginshell des Benutzers
Telephone	rw	string	get_userTelephone	Telefonnummer des Benutzers

ActiveUser group:

Terminal	r	string	get_whoDevice	Device des Benutzers, von dem er sich eingelogt hat
Time	r	string	get_whoTime	Uhrzeit des ersten Einlogens
System	r	string	get_whoWhere	Host, der zum Einlogen benutzt wurde
Name	rw	string	get_whoName	Name des eingelogten Benutzers

Device group:

Name	rw	string	get_devName	Name des Geräts
Desciption	r	string	get_devDescr	Beschreibung des Geräts
DeviceType	r	string	get_devType	Typ des Geräts in der MIB
DeviceState	r	string	get_devStatus	Devicestatus

Disk group:

Capacity	r	kbytes	get_diskCapacity	Größe des Laufwerks
State	r	kbytes	get_diskState	freier Platz auf dem Laufwerk
Busy	r	long	get_diskBusy	Anzahl der time ticks, die das Laufwerk aktiv war
Media	r	long	get_diskMedia	Typ der Disk

Filesystem group:

FSName	r	string	get_FSName	Name des Filesystems
FSType	r	string	get_FSType	Typ des Filesystems
FSBlockSize	r	bytes	get_FSBlockSize	Blockgröße des FS
FSBlocksFree	r	long	get_FSBlocksFree	Anzahl der freien Blöcke im FS
FSMountPoint	r	string	get_FSMountPoint	MountPoint des FS

Group group:

ID	rw	long	get_groupID set_groupID	Gruppenidentifikator des Systems
Passwd	rw	string	get_groupPasswd set_groupPasswd	Paßwort für die Gruppe
Status	rw	long	get_groupStatus set_groupStatus	Status der Gruppe

Partition group:

Label	r	string	get_partitionLabel	Beschreibung der Partition
Size	r	kbytes	get_partitionSize	Größe der Partition

Printer group:

Location	rw	string	get_printerLocation set_printerLocation	Ort des Druckers
Name	r	string	get_printerName	Druckername
OpStatus	r	long	get_printerOpStatus	gibt an, ob für User erreichbar oder nicht
UsState	r	long	get_printerUsState	gibt an, ob der Drucker arbeitet oder nicht
AdStatus	r	long	get_printerAdStatus	gibt an, ob drucken erlaubt ist oder nicht

Printqueue / PrintJob group:

Queuename	r	string	get_queueName	Name der Queue
Status der Queue	r	long	get_queueStatus	Status der Queue
JobId	r	long	get_queueJobId	Job-Identifier
JobOwner	r	string	get_queueJobOwner	Name des Jobbesitzer
JobEntered	r	string	get_queueJobEntered	Datum und Uhrzeit des Eintritts in die Queue
JobPrio	rw	long	get_queueJobPrio set_queueJobPrio	Priorität des Jobs
JobSize	r	kbytes	get_queueJobSize	Größe des Jobs
JobAction	w	long	set_queueJobAction	Aktionen, die fürs System-management reserviert sind
PrinterName der Queue	r	string	get_queuePrinterName	Druckername (hostname:printername)
Action	w	long	set_queueAction	Aktionen, die für das System-management reserviert sind
PrinterLocation	r	long	get_queuePrinter Location	Printer lokal / remote

Process group:

PID	rw	long	get_processPID	Prozeß-Identifier
PPID	rw	long	get_processPPID	ParentProzeßIdentifier
Name	r	string	get_processName	Aufrufname des Prozesses
UID	r	long	get_processUID	UserID des Prozeßbesitzers
GID	r	long	get_processGID	GruppenID des Prozeß-besitzers
State	r	long	get_processState	Prozeßzustand
CPUTime	r	long	get_processCPUTime	verbrauchte CPU durch den Prozeß
Size	r	kbytes	get_processSize	Größe des allokierten Speichers für einen Prozeß
Priority	r	long	get_processPri	Priorität
Nice	rw	long	get_processNice set_processNice	wiederbesetzen mit dem angegebenen Wert
Signal	rw	long	get_processSignal set_processSignal	angegebenes Signal zum Prozeß senden
PHYSIO	r	boolean	get_processFlagSPHYSIO	true = physical I/O
SSWAP	r	boolean	get_processFlagSSWAP	true = process swapped out
SULOCK	r	boolean	get_processFlagSULOCK	true = no swapped out for this process
SLOAD	r	boolean	get_processFlagSLOAD	true = process loaded in main memory
SSYS	r	boolean	get_processFlagSSYS	true = swapper or pager

Processor group:

UserTime	r	long	get_cpuUserTime	Prozentsatz von der Zeit, in dem der Prozeß im Usermode war
NiceTime	r	long	get_cpuNiceTime	Prozentsatz von der Zeit, in dem der Prozeß im Usermode war
SystemTime	r	long	get_cpuSystemTime	Prozentsatz von der Zeit, in dem der Prozeß im System-modus war
IdleTime	r	long	get_cpuIdleTime	Prozentsatz von der Zeit, die der Prozessor unbenützt war
Type	r	string	get_cpuType	Typ der CPU
ClockRate	r	long	get_cpuClockRate	CPU Taktrate

Quota group:

Soft	rw	kbytes	get_soft	Soft-Quota
Hard	rw	kbytes	get_hard	Hard-Quota
Used	rw	kbytes	get_used	verwendeter Plattenplatz
InodeSoft	rw	long	get_InodeSoft	Inode Soft-Quota
InodeHard	rw	long	get_InodeHard	Inode Hard-Quota
InodeUsed	r	long	get_InodeUsed	verwendete Inodes

Ram group:

State	r	long	get_stoState	gibt an, wieviel Prozent des SPeichers benützt werden

StorageDevice group:

Size	r	bytes	get_stoSize	Speichergröße
Used	r	bytes	get_stoUsed	verwendeter Speicher
Allocation Units	r	bytes	get_stoAllocationUnits	allokierter Speicher
State	r	long	get_stoState	Prozentsatz des verwendeten Speichers

System group:

ClockTicks	r	long	get_sysClockTick	clock ticks in thents of a millisecond
Contact	rw	long	get_sysContact set_sysContact	Kontaktperson
Date	rw	string	get_sysDate set_sysDate	lokales Datum / Zeit
Hardware	rw	string	get_sysHardware	Bezeichnung der Hardware
Location	rw	string	get_sysLocation set_sysLocation	physischer Standort
Name	r	string	get_sysName	System-/Domainname
Os	r	string	get_sysOs	Betriebssystem
TimeZone	r	string	get_timezone	Zeitzone
UpTime	r	long	get_sysUptime	Laufzeit des Rechners
maxProcess Number	r	long	get_maxProcessNumber	maximale Zahl der Prozesse, die unterstützt werden
maxProcess Size	r	long	get_maxProcessSize	maximaler Speicher für einen Prozeß in kbytes
curProcess Number	r	long	get_curProcessNumber	Anzahl geladener Prozesse
curMaxProcess Size	r	long	get_curMaxProcessSize	Größe des größten Prozesses in kbytes
curMaxProcess Time	r	long	get_curMaxProcessTime	CPU-Time des längsten aufge-rufenen Prozesses in Sekunden
Avg1	r	long	get_loadAvg1	Anzahl der laufenden Prozesse in der letzten Minute
Avg5	r	long	get_loadAvg5	Anzahl der laufenden Prozesse in den letzten fünf Minuten
Avg15	r	long	get_loadAvg15	Anzahl der laufenden Prozesse in den letzten fünfzehn Minuten

Virtualm group:

state	r	long	get_stoState	gibt an, wieviel Prozent des Speichers benützt werden

Anhang C: IDL-Dateien des OS/2-Agenten

Device.idl:

```
#ifndef _Device_idl_
#define _Device_idl_

#include <somobj.idl>
#include <mcollect.idl>

// class declarations
interface Device;
// class declarations end

// class definition
interface Device : somf_MCollectible
{
// class members
            readonly attribute long Name;
            void enable();
            void disable();
            void lock();
            void unlock();
            void shuttingdown();
// class members end

#ifdef __SOMIDL__
implementation
{
            dllname = „System.dll";
            releaseorder : Name, enable, disable, lock, unlock, shuttingdown, _get_Name;
            Name: noget;
};
#endif /* __SOMIDL__ */

};
// class definition end

// footer
#endif
// footer end
```

Disk.idl:

```
#include <somobj.idl>
#include <tdeq.idl>
```

```
#ifndef _Disk_idl_
#define _Disk_idl_

#include „permanentStorageDevice.idl“

// class declarations
interface Disk;
// class declarations end

// class definition
interface Disk : permanentStorageDevice
{
// stp class members
		readonly attribute SOMObject assnPartition;
// stp class members end

#ifdef __SOMIDL__
implementation
{
		dllname = „System.dll“;
		releaseorder : assnPartition, _get_assnPartition;
		assnPartition: noget;
		_get_AllocationUnits : override;
		get_Used : override;
		get_Size : override;
		get_State: override;
};
#endif /*__SOMIDL__*/
};
// class definition end

// footer
#endif
// footer end
```

Filesystem.idl:

```
#include <somobj.idl>
#include <tdeq.idl>

interface Partition;

#ifndef _Filesystem_idl_
#define _Filesystem_idl_
// class declarations
interface Filesystem;
// class declarations end
```

```
// class definition
interface Filesystem : SOMObject
{
typedef long bytes;
// stp class members
                readonly attribute string FSName;
                readonly attribute string FSType;
                readonly attribute bytes FSBlockSize;
                readonly attribute long FSBlocksFree;
                readonly attribute SOMObject assnPartition;
                readonly attribute SOMObject is_formatted_with;
                void check();
                void mount();
// stp class members end

#ifdef __SOMIDL__
implementation {
                dllname = „System.dll“;
                releaseorder : FSName, FSType, FSBlockSize, FSBlocksFree, assnPartition,
                is_formatted_with, check, mount, _get_FSName, _get_FSType, _get_FSMountPoint,
                _get_FSBlockSize, _get_FSBlocksFree, _get_assnPartition,
                _get_is_formatted_with;
                FSName: noget;
                FSType: noget;
                FSBlockSize: noget;
                FSBlocksFree: noget;
                assnPartition: noget;
                is_formatted_with: noget;
};
#endif /*__SOMIDL__*/
};
// class definition end
// footer
#endif
// footer end
```

Partition.idl:

```
#include <somobj.idl>
#include <tdeq.idl>

interface Disk;
interface Filesystem;

#ifndef _Partition_idl_
#define _Partition_idl_

// class declarations
```

```
interface Partition;
// class declarations end
// class definition
interface Partition : SOMObject
{
                typedef long kbytes;
// stp class members
                readonly attribute string Label;
                readonly attribute kbytes Size;
                attribute SOMObject assnDisk;
                attribute SOMObject assnFilesystem;
                attribute SOMObject is_formatted_with;
// stp class members end

#ifdef __SOMIDL__
implementation {
                dllname = „System.dll“;
                releaseorder : Label, Size, assnDisk, assnFilesystem, is_formatted_with,
                _get_Label, _get_Size, _get_assnDisk, _set_assnDisk, _get_assnFilesystem,
                _set_assnFilesystem, _get_is_formatted_with, _set_is_formatted_with;
                Label: noget;
                Size : noget;
                assnDisk: noget;
                assnFilesystem: noget;
                is_formatted_with: noget;
};
#endif /*__SOMIDL__*/
};
// class definition end

// footer
#endif
// footer end
```

permanentStorageDevice.idl:

```
#ifndef _permanentStorageDevice_idl_
#define _peramanentStorageDevice_idl_

#include „StorageDevice.idl“

//class declarations
interface permanentStorageDevice;
interface System;
// class declarations end

// class definition
interface permanentStorageDevice : StorageDevice
```

```
{

// class members
                attribute SOMObject aggrSystem;
// class members end
#ifdef __SOMIDL__
implementation {
                dllname="System.dll";
                releaseorder : aggrSystem, _get_aggrSystem, _set_aggrSystem;
                aggrsystem: noget;
};
#endif /* __SOMIDL__ */
};
// class definition end

// footer
#endif
// footer end
```

Printer.idl:

```
#ifndef _Printer_idl_
#define _Printer_idl_

#include „Device.idl“

// class declarations
interface Printer;
// class declarations end

interface Printer : Device {
// class members
                readonly attribute string Location;
                readonly attribute string PrinterStatus;
                attribute SOMObject assnSystem;
                attribute SOMObject access_is_managed_by;
// class members end

#ifdef __SOMIDL__
implementation {
                dllname = „System.dll“;
                releaseorder : Location, PrinterStatus, assnSystem, access_is_managed_by,
                _get_Location, _get_PrinterStatus, _get_assnSystem, _set_assnSystem,
                _get_access_is_managed_by, _set_access_is_managed_by;
                Location: noget;
                PrinterStatus: noget;
                assnSystem: noget;
                access_is_managed_by: noget;
```

```
};
#endif /* __SOMIDL__ */

};
// class definition end

//footer
#endif
//footer end
```

PrintJob.idl:

```
#ifndef _PrinterJob_idl_
#define _PrinterJob_idl_

#include <somobj.idl>
#include „Device.idl“
#include „Printer.idl“
#include „PrintQueue.idl“

// class declarations
interface PrintJob;
interface PrintQueue;
// class declarations end

// class definition
interface PrintJob : PrintQueue {
            typedef long kbytes;
// class members
            readonly attribute string QueueName;
            readonly attribute long QueueStatus;
            readonly attribute long JobID;
            readonly attribute string JobOwner;
            readonly attribute string JobEntered;
            readonly attribute long JobPrio;
            readonly attribute kbytes JobSize;
            readonly attribute kbytes JobPrinted;
            readonly attribute string PrinterName;
            readonly attribute string PrinterLocation;
            attribute SOMObject aggrPrintQueue;
            void cancel();
            void hold();
            void resume();
            void incPriority();
            void decPriority();
// class members end
```

```
#ifdef __SOMIDL__
implementation {
        dllname = „System.idl“;
        releaseorder : QueueName, QueueStatus, JobID, JobOwner, JobEntered, JobPrio,
        JobSize, JobPrinted, PrinterName, PrinterLocation, aggrPrinterQueue, cancel,
        hold, resume, incPriority, decPriority, _get_QueueName, _get_QueueStatus,
        _get_JobID, _get_JobOwner, _get_JobEntered, _get_JobPrio, _get_JobSize,
        _get_JobPrinted, _get_PrinterName, _get_PrinterLocation, _get_aggrPrintQueue,
        _set_aggrPrintQueue;
        QueueName: noget;
        QueueStatus: noget;
        JobID: noget;
        JobOwner: noget;
        JobEntered: noget;
        JobPrio: noget;
        JobSize: noget;
        JobPrinted: noget;
        PrinterName: noget;
        PrinterLocation: noget;
        aggrPrintQueue: noget;
};
#endif /* __SOMIDL__ */

};
// class definition end

// footer
#endif
// footer end
```

PrintQueue.idl:

```
#ifndef _PrintQueue_idl_
#define _PrintQueue_idl_

#include „Device.idl“
#include „Printer.idl“

// class declarations
interface PrintQueue;
interface Printer;
// class declarations end
//class definition
interface PrintQueue : Device {
// class members
        readonly attribute Printer access_is_managed_by;
        readonly attribute SOMObject holds;
// class members end
```

```
#ifdef __SOMIDL__
implementation {
                    dllname = „System.dll“;
                    releaseorder : access_is_managed_by, holds, _get_access_is_managed_by,
                    _set_access_is_managed_by, _get_holds;
                    access_is_managed_by: noget;
                    holds: noget;
};
#endif /* __SOMIDL__ */

};
// class definition end

//footer
#endif
// footer end
```

Process.idl:

```
#include <somobj.idl>
#include <mcollect.idl>

interface System;

#ifndef _System_idl_
#define _System_idl_

// class declarations
interface Process;
// class declarations end

//class definition
interface Process : somf_MCollectible {
                    typedef long kbytes;
//class members
                    readonly attribute long PPID;
                    readonly attribute long PID;
                    readonly attribute string Name;
                    readonly attribute string State;
                    readonly attribute long Priority;
                    attribute System assnSystem;
                    void sendSignal(in long signo);
// class members end
#ifdef __SOMIDL__
implementation {
                    dllname="System.dll";
                    releaseorder : PPID, PID, Name, State, Priority, assSystem, sendSignal, _get_PPID,
                    _get_PID, _get_Name, _get_State, _get_Priority, _get_assnSystem, _set_assnSystem;
```

```
                    PPID: noget;
                    PID: noget;
                    Name: noget;
                    State: noget;
                    Priority: noget;
};
#endif /*__SOMIDL__*/
};
// class definition end

// footer
#endif
// footer end
```

Processor.idl:

```
#ifndef _Processor_idl_
#define _Processor_idl_

#include <somobj.idl>
#include „Device.idl“

//class declarations
interface Processor;
interface System;
// class declarations end

// class definition
interface Processor : Device {
// class members
                    readonly attribute string Type;
                    attribute System aggrSystem;
// class members end

#ifdef __SOMIDL__
implementation {
                    dllname =“System.dll“;
                    releaseorder : Type, aggrSystem, _get_Type, _get_aggrSystem, _set_aggrSystem;
                    Type: noget;
                    aggrSystem: noget;
};
#endif /* __SOMIDL__ */
};
// class definition end

// footer
#endif
// footer end
```

Ram.idl:

```
#ifndef _Ram_idl_
#define _Ram_idl_
#include „volatileStorageDevice.idl“
// class declarations
interface Ram;
// class declarations end
// class definition
interface Ram : volatileStorageDevice
{
// stp class members
// stp class members end

#ifdef __SOMIDL__
implementation {
                dllname = „System.dll“;
                _get_Used : override;
                _get_Size : override;
                _get_State: override;
};
#endif /*__SOMIDL__*/
};
// class definition end

// footer
#endif
// footer end
```

StorageDevice.idl:

```
#ifndef _StorageDevice_idl_
#define _StorageDevice_idl_

#include <somobj.idl>
#include „Device.idl“

// class definition
interface StorageDevice : Device {
                typedef long bytes;
// class members
                readonly attribute bytes Size;
                readonly attribute bytes Used;
                readonly attribute bytes AllocationUnits;
                readonly attribute long State;
// class members end
```

```
#ifdef __SOMIDL__
implementation {
		dllname = „System.dll“;
		releaseorder : Size, Used, AllocationUnits, State, _get_Size, _get_Used,
		_get_AllocationUnits, _get_State;
		Size: noget;
		Used: noget;
		AllocationUnits: noget;
		State: noget; };
#endif /* __SOMIDL__ */
};
// class definition end
// footer
#endif
// footer end
```

System.idl:

```
#include <somobj.idl>
#include <tdeq.idl>

interface Printer;
interface Process;
interface Processor;
interface volatileStorageDevice;
interface permanentStorageDevice;
interface somf_TDeque;

#ifndef _System_idl_
#define _System_idl_

// class declarations
interface System;
// class declarations end
// class definition
interface System : SOMObject {
// class members
		readonly attribute long ClockTicks;
		attribute long Date;
		readonly attribute string Hardware;
		readonly attribute string Name;
		readonly attribute string Os;
		readonly attribute long TimeZone;
		readonly attribute long UpTime;
		readonly attribute long MaxProcessSize;
		readonly attribute long curProcessNumber;
		readonly attribute SOMObject assnPrinter;
		readonly attribute SOMObject assnProcess;
```

```
                readonly attribute SOMObject aggrProcessor;
                readonly attribute SOMObject aggrvolatileStorageDevice;
                readonly attribute SOMObject aggrpermanentStorageDevice;
// class members end

#ifdef __SOMIDL__
implementation {
                dllname="System.dll";
                releaseorder : ClockTicks, Date, Hardware, Name, Os, TimeZone, Uptime, maxProcessSize,
                curProcessNumber, assnPrinter, assnProcess, aggrProcessor, aggrvolatileStorageDevice,
                aggrpermanentStorageDevice, _get_ClockTicks, _get_Date, _set_Date, _get_Hardware,
                _get_Name, _get_Os, _get_TimeZone, _get_UpTime, _get_MaxProcessNumber,
                _get_MaxProcessSize, _get_curProcessNumber, _get_assnPrinter, _get_assnProcess,
                _get_aggrProcessor, _get_aggrvolatileStorageDevice, _get_aggrpermanentStorageDevice;
                ClockTicks: noget;
                Date: noget, noset;
                Hardware: noget;
                Name: noget;
                Os: noget;
                TimeZone: noget;
                Uptime: noget;
                maxProcessSize: noget;
                curProcessNumber: noget;
                assnPrinter: noget;
                assnProcess: noget;
                aggrProcessor: noget;
                aggrvolatileStorageDevice: noget;
                aggrpermanentStorageDevice: noget;
                somInit: override;
                somUninit: override; };
#endif /*__SOMIDL__*/
};
// class definition end

// footer
#endif
// footer end
```

VirtualMemory.idl:

```
#ifndef _VirtualMemory_idl_
#define _VirtualMemory_idl_

#include „volatileStorageDevice.idl“

// class declarations
interface VirtualMemory;
// class declarations end
```

```
// class definition
interface VirtualMemory : volatileStorageDevice {
// class members
// class members end

#ifdef __SOMIDL__
implementation {
                dllname = „System.dll“;
                _get_Used : override;
                _get_Size : override;
                _get_State: override;
};
#endif /*__SOMIDL__*/
};
// class definition end

// footer
#endif
// footer end
```

volatileStorageDevice.idl:

```
#ifndef _volatileStorageDevice_idl_
#define _volatileStorageDevice_idl_

#include „StorageDevice.idl“

//class declarations
interface volatileStorageDevice;
interface System;
// class declarations end
// class definition
interface volatileStorageDevice : StorageDevice {
// class members
                attribute SOMObject aggrSystem;
// class members end

#ifdef __SOMIDL__
implementation {
                dllname=“System.dll“;
                releaseorder : aggrSystem, _get_aggrSystem, _set_aggrSystem;
                aggrSystem: noget;
};
#endif /* __SOMIDL__ */
};
// class definition end
// footer
#endif
// footer end
```

Anhang D: C-Code der Systemprozeduren

/* get_devName.c */

```
#define INCL_DOSDEVICES
#define INCL_DOSERRORS
#include <os2.h>
#include <stdio.h>
#include „get_devName.h"

LONG get_devName () {
		LONG	erg[3];
		BYTE	Model = 0,
			PrinterPorts = 0,
			RS232Ports = 0,
			Floppy = 0,
			Printers = 0;
		APIRET rc = NO_ERROR;

		rc = DosDevConfig(&Model, DEVINFO_MODEL);
		if (rc != NO_ERROR)
		{
			return rc /* Error code */;
		}

		erg[0]= DosDevConfig(&PrinterPorts, DEVINFO_PRINTER);
		erg[1] = DosDevConfig(&Printers, DEVINFO_PRINTER);
		erg[2] = DosDevConfig(&RS232Ports, DEVINFO_RS232);
		rc = DosDevConfig(&Floppy, DEVINFO_FLOPPY);
		if (Floppy >=1)
			erg[3] = Floppy;
		else
			erg[3]=0;

		return &erg;
}
```

/* get_FSName.c */

```
#define INCL_DOSFILEMGR		/* File manager values */
#define INCL_DOSERRORS		/* DOS error values */
#include <os2.h>
#include <stdio.h>
#include <string.h>
#include „get_FSName.h"

CHAR *get_FSName(CHAR * szDeviceName)
{
		ULONG ulOrdinal = 0;		/* Ordinal of entry in name list */
		PBYTE  pszFSDName = NULL;	/* pointer to FS name */
		PBYTE prgFSAData = NULL;	/* pointer to FS data */
		APIRET rc = NO_ERROR;		/* Return code */
		BYTE fsqBuffer[sizeof(FSQBUFFER2) + (3 * CCHMAXPATH)] = {0};
		ULONG cbBuffer = sizeof(fsqBuffer);
		PFSQBUFFER2 pfsqBuffer = (PFSQBUFFER2) fsqBuffer;
		CHAR *typestring;
		CHAR erg[2];
```

```
        rc = DosQueryFSAttach (szDeviceName,            /* Logical drive of attached FS */
                               ulOrdinal,               /* ignored for FSAIL_QUERYNAME */
                               FSAIL_QUERYNAME,         /* Return data for a Drive or Device */
                               pfsqBuffer,              /* returned data */
                               &cbBuffer);              /* returned data length */


        if (rc != NO_ERROR)
        {
                return rc;              /* error code */
        }
        else
        {
                pszFSDName = pfsqBuffer->szName + pfsqBuffer->cbName + 1;
                prgFSAData = pszFSDName + pfsqBuffer->cbFSDName + 1;

                if (pfsqBuffer->iType == 3)
                                typestring = „local drive“;
                else
                                typestring = „remote drive attached to the filesystem driver“;

                erg[0] = pfsqBuffer->szName;
                erg[1] = typestring;
                erg[2] = prgFSAData;
        }
        return &erg;
}
```

/* get_FSType.c */

```
#define INCL_DOSFILEMGR     /* File manager values */
#define INCL_DOSERRORS      /* DOS error values */
#include <os2.h>
#include <stdio.h>
#include <string.h>
#include „get_FSType.h“

CHAR *get_FSType(CHAR *szDeviceName)
{
        ULONG ulOrdinal = 0;            /* Ordinal of entry in name list */
        PBYTE pszFSDName = NULL;        /* pointer to FS name */
        PBYTE prgFSAData = NULL;        /* pointer to FS data */
        APIRET rc = NO_ERROR;           /* Return code */
        BYTE fsqBuffer[sizeof(FSQBUFFER2) + (3 * CCHMAXPATH)] = {0};
        ULONG cbBuffer = sizeof(fsqBuffer);
        PFSQBUFFER2 pfsqBuffer = (PFSQBUFFER2) fsqBuffer;
        CHAR *typestring;
        CHAR erg[1];

        rc = DosQueryFSAttach (szDeviceName,            /* Logical drive of attached FS */
                               ulOrdinal,               /* ignored for FSAIL_QUERYNAME */
                               FSAIL_QUERYNAME,         /* Return data for a Drive or Device */
                               pfsqBuffer,              /* returned data */
                               &cbBuffer);              /* returned data length */

        if (rc != NO_ERROR)
        {
                return rc;              /* error code */
        }
        else
```

```
	{
		pszFSDName = pfsqBuffer -> szName + pfsqBuffer -> cbName + 1;
		prgFSAData = pszFSDName + pfsqBuffer -> cbFSDName + 1;

		erg[0] = pfsqBuffer->szName;
		erg [1] = pszFSDName;
	}
	return &erg;
}
```

/* get_FSBlockSize.c */

```
#define INCL_DOSFILEMGR		/* File Manager values */
#define INCL_DOSMISC
#define INCL_DOSERRORS		/* DOS Error values */
#include <os2.h>
#include <stdio.h>
#include „get_FSBlockSize.h“

LONG get_FSBlockSize(LONG laufwerk)
{
	ULONG aulFSInfoBuf[40] = {0};	/* File system info buffer */
	APIRET rc = NO_ERROR;		/* Return code */
	ULONG ergebnis;
	ULONG ulDriveNum = 0;		/* Drive number */
	ULONG ulDriveMap = 0;		/* Mapping of valid drives */
	ULONG i = 0;			/* A loop index */
	LONG erg[1];

	rc = DosQueryCurrentDisk (&ulDriveNum, &ulDriveMap);

	if (rc != NO_ERROR)
	{
		return rc;
	}

	rc = DosQueryFSInfo(laufwerk,			/* Drive number */
				FSIL_ALLOC,		/* Level 1 allocation info */
				(PVOID)aulFSInfoBuf,	/* Buffer */
				sizeof(aulFSInfoBuf));	/* Size of buffer */

	if (rc != NO_ERROR)
	{
		return rc;
	}
	else
	{
		ergebnis = aulFSInfoBuf[2] * (aulFSInfoBuf[1] * (USHORT)aulFSInfoBuf[4]);
		erg[0] = aulFSInfoBuf[2];
		erg[1] = ergebnis;
	}

	DosExit(EXIT_THREAD, aulFSInfoBuf[3]);/* Return available allocation units to the initiating process */
	return &erg;
}
```

/* get_FSBlocksFree.c */

```
#define INCL_DOSFILEMGR       /* File Manager values */
#define INCL_DOSMISC
#define INCL_DOSERRORS        /* DOS Error values */
#include <os2.h>
#include <stdio.h>
#include „get_FSBlocksFree.h“

LONG get_FSBlocksFree(LONG laufwerk)
{
        ULONG aulFSInfoBuf[40] = {0};/* File system info buffer */
        APIRET rc = NO_ERROR;         /* Return code */
        ULONG ergebnis;
        ULONG ulDriveNum = 0;         /* Drive number */
        ULONG ulDriveMap = 0;         /* Mapping of valid drives */
        ULONG i = 0;                  /* A loop index */
        LONG erg[1];

        rc = DosQueryCurrentDisk (&ulDriveNum, &ulDriveMap);
        if (rc != NO_ERROR)
        {
                return rc;
        }
        rc = NO_ERROR;

        rc = DosQueryFSInfo(laufwerk,                             /* Drive number */
                            FSIL_ALLOC,                /* Level 1 allocation info */
                            (PVOID)aulFSInfoBuf,       /* Buffer */
                            sizeof(aulFSInfoBuf));     /* Size of buffer */

        if (rc != NO_ERROR)
        {
                return rc;
        }
        else
        {
                ergebnis = aulFSInfoBuf[3] * (aulFSInfoBuf[1] * (USHORT)aulFSInfoBuf[4]);
                erg[0] = aulFSInfoBuf[3];
                erg[1] = ergebnis;
        }
                DosExit(EXIT_THREAD, aulFSInfoBuf[3]);     /* Return available allocation units to the initiating process */
                return &erg;
}
```

/* get_partitionLabel.c */

```
#define INCL_DOSDEVICES       /* Device values */
#define INCL_DOSFILEMGR       /* File manager values */
#define INCL_DOSERRORS        /* DOS error values */
#include <os2.h>
#include <stdio.h>
#include „get_partitionLabel.h“

CHAR *get_partitionLabel(LONG laufwerk)
{
        typedef struct _FSINFOBUF {
                ULONG           ulVolser;       /* Volume serial Number */
                VOLUMELABEL     vol;            /* Volume label */
        } FSINFOBUF;
```

```
		typedef FSINFOBUF *PFSINFOBUF;
		ULONG ulDriveNumber = 0;	/* Drive number */
		FSINFOBUF VolumeInfo = {0};	/* File system info buffer */
		ULONG aulFSInfoBuf[40] = {0};	/* File system info buffer */
		APIRET rc = NO_ERROR;		/* Return code */
		ULONG ulDriveNum = 0;		/* Drive number */
		ULONG ulDriveMap = 0;		/* Mapping of valid drives */
		USHORT usNumDrives = 0;	/* Data return buffer */
		ULONG   ulDataLen    = sizeof (USHORT);/* Data return buffer length */
		CHAR erg[2];

		/* Request a count of the number of partitionable disks in the system */
		rc = DosPhysicalDisk(INFO_COUNT_PARTITIONABLE_DISKS,
						&usNumDrives,
						ulDataLen,
						NULL,
						0L);

		if (rc != NO_ERROR)
		{
			return rc;
		}
		else
		{
			erg[0] = usNumDrives;
		}
		rc = NO_ERROR;

		rc = DosQueryCurrentDisk (&ulDriveNum, &ulDriveMap);

		if (rc != NO_ERROR)
		{
			return rc;
		}
		rc = NO_ERROR;
		ulDriveNumber = laufwerk;	/* specify drive */

		rc = DosQueryFSInfo(ulDriveNumber,
						FSIL_VOLSER,		/* Request volume information */
						&VolumeInfo,		/* Buffer for information */
						sizeof(FSINFOBUF));	/* Size of buffer */
		if (rc != NO_ERROR)
		{
			return rc;
		}
		else
		{
			erg[1] = VolumeInfo.vol.szVolLabel;
			erg[2] = VolumeInfo.ulVolser);
		}
		return &erg;
}
```

/* get_partitionSize.c */

```
#define INCL_DOSFILEMGR	/* File Manager values */
#define INCL_DOSMISC
#define INCL_DOSERRORS	/* DOS Error values */
#include <os2.h>
#include <stdio.h>
#include „get_partitionSize.h“
```

```
LONG get_partitionSize(LONG laufwerk)
{
	ULONG	aulFSInfoBuf[40] = {0};	/* File system info buffer */
	APIRET	rc = NO_ERROR;		/* Return code */
	LONG	ergebnis;
	ULONG	ulDriveNum = 0;		/* Drive number */
	ULONG	ulDriveMap = 0;		/* Mapping of valid drives */

	rc = DosQueryCurrentDisk (&ulDriveNum, &ulDriveMap);

	if (rc != NO_ERROR)
	{
		return rc;
	}
	else
	{
		rc = DosQueryFSInfo(laufwerk,			* Drive number */
					FSIL_ALLOC,		/* Level 1 allocation info */
					(PVOID)aulFSInfoBuf,	/* Buffer */
					sizeof(aulFSInfoBuf));	/* Size of buffer */

	if (rc != NO_ERROR)
	{
		return rc;
	}
	else
	{
		ergebnis = aulFSInfoBuf[2] * (aulFSInfoBuf[1] * (USHORT)aulFSInfoBuf[4]);
	}
	DosExit(EXIT_THREAD, aulFSInfoBuf[3]);	/* Return available allocation units to the initiating process */
	return ergebnis;
	}
}
```

/* get_printerLocation.c */

```
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS
#include <os2.h>
#include <stdio.h>
#include „get_printerLocation.h“

CHAR *get_printerLocation()
{
	PVOID	pBuf;
	ULONG	fsType;
	ULONG	cbBuf;
	ULONG	cRes;
	ULONG	cTotal;
	ULONG	cbNeeded;
	SPLERR	splerr = 0;
	PPRINTERINFO pRes;
	CHAR erg[40];
	fsType = SPL_PR_QUEUE | SPL_PR_DIRECT_DEVICE | SPL_PR_QUEUED_DEVICE | SPL_PR_LOCAL_ONLY;
	splerr = SplEnumPrinter(NULL, 0, fsType, NULL, NULL, &cRes, &cTotal, &cbNeeded, NULL);

	if (splerr != 0)
	{
		DosAllocMem(&pBuf, cbNeeded, PAG_READ | PAG_WRITE | PAG_COMMIT);
		cbBuf = cbNeeded;
```

```
			splerr = SplEnumPrinter(NULL, 0, fsType, pBuf, cbBuf, &cRes, &cTotal, &cbNeeded, NULL);
			pRes = (PPRINTERINFO)pBuf;

			while (cRes--)
			{
				if (pRes[cRes].pszLocalName == NULL)
				{
				erg[cRes] = pRes[cRes].pszPrintDestinationName;
				}
				else
				{
				erg[cRes] = pRes[cRes].pszPrintDestinationName, pRes[cRes].pszLocalName);
				}
			}
		DosFreeMem(pBuf);
		return &erg;
		}
		else
		{
			return (splerr);
		}
		DosExit(EXIT_PROCESS, 0);
}
```

/* get_printerStatus.c */

```
#define INCL_BASE
#define INCL_DOSMEMMGR
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS
#include <os2.h>
#include <stdio.h>
#include „get_printerStatus.h"

CHAR *get_printerStatus(CHAR *PName)
{
		SPLERR		splerr;
		ULONG		cbBuf;
		ULONG		cbNeeded;
		ULONG		ulLevel;
		PSZ		pszComputerName;
		PSZ		pszPrintDeviceName;
		PVOID		pBuf;
		PPRDINFO3	pprd3;
		CHAR		erg[1];
		pszComputerName = (PSZ)NULL;
		pszPrintDeviceName = PName;
		ulLevel = 3;
		splerr = SplQueryDevice(pszComputerName, pszPrintDeviceName, ulLevel, (PVOID)NULL, 0L, &cbNeeded);

		if(!DosAllocMem(&pBuf, cbNeeded, PAG_READ | PAG_WRITE | PAG_COMMIT))
		{
			cbBuf = cbNeeded;
			splerr = SplQueryDevice(pszComputerName, pszPrintDeviceName, ulLevel, pBuf, cbBuf,  &cbNeeded);
			pprd3=(PPRDINFO3)pBuf;
			erg[0] = pprd3->pszPrinterName;
			if ( pprd3->fsStatus & PRJ_DEVSTATUS)
						erg[1] = „Job deleted.";
			return &erg;
```

```
			switch (pprd3->fsStatus & PRD_STATUS_MASK)
			{
				case 0 : erg[1] = „Job Status: Processing “;
				 break;
				case 1 : erg[1] = „Job Status: Not processing, or paused“;
			}
			switch (pprd3->fsStatus & PRJ_DEVSTATUS)
			{
				case 4 : erg[1] = „Job Status: Job complete “;
				  break;
				case 8 : erg[1] = „Job Status: Intervention required “;
				  break;
				case 16 : erg[1] = „Job Status: Error occurred “;
				   break;
				case 32 : erg[1] = „Job Status: Print device offline “;
				 break;
				case 64 : erg[1] = „Job Status: Print device paused “;
				   break;
				case 128 : erg[1] = „Job Status: Raise alert “;
				     break;
				case 256 : erg[1] = „Job Status: Print device out of paper “;
			}

			DosFreeMem(pBuf);
			return &erg;
		}
		DosExit(EXIT_PROCESS, 0);
		return (splerr);
}
```

/* get_cpuType.c */

```
#include <stdio.h>
#include „get_cpuType.h“

CHAR *get_cpuType()
{
		CHAR *erg;
		return erg = „CPUType of this local computer: Pentium P90\n“;
}
```

/* get_queueName.c */

```
#define INCL_BASE
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS

#include <os2.h>
#include <stdio.h>
#include „get_queueName.h“

CHAR *get_queueName()
{
		SPLERR		splerr;
		ULONG		cbBuf;
		ULONG		cTotal;
		ULONG		cReturned;
```

```
		ULONG		cbNeeded;
		ULONG		ulLevel;
		ULONG		i;
		PSZ		pszComputerName;
		PBYTE		pBuf;
		PPRQINFO3	prq;
		CHAR		erg[20];
		ulLevel = 4L;
		pszComputerName = (PSZ)NULL;
		splerr = SplEnumQueue(pszComputerName, ulLevel, pBuf, 0L, &cReturned, &cTotal, &cbNeeded, NULL);

		if (splerr == ERROR_MORE_DATA)
		{
			if(!DosAllocMem(&pBuf, cbNeeded, PAG_READ | PAG_WRITE | PAG_COMMIT))
			{
				cbBuf = cbNeeded;
				splerr = SplEnumQueue(pszComputerName, ulLevel, pBuf, cbBuf, &cReturned, &cTotal, &cbNee-
ded, NULL);
				if (splerr == NO_ERROR)
				{
				  /* Set pointer to the beginning of the buffer */
				  prq = (PPRQINFO3)pBuf;

				  for (i=0; i < cReturned; i++)
				  {
				     erg[i] = prq->pszName;
				  }
				}
				DosFreeMem(pBuf);
				return &erg;
			}
		}
		else
		{
			return (splerr);
		}
		DosExit(EXIT_PROCESS, 0);
}
```

/* get_queueState.c */

```
#define INCL_BASE
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS

#include <os2.h>
#include <stdio.h>
#include „get_queueState.h“

LONG get_queueState()
{
		SPLERR		splerr;
		ULONG		cbBuf;
		ULONG		cTotal;
		ULONG		cReturned;
		ULONG		cbNeeded;
		ULONG		ulLevel;
		ULONG		i;
		PSZ		pszComputerName;
		PBYTE		pBuf;
```

```
		PPRQINFO3	prq;
		LONG		erg[20];
		ulLevel = 4L;
		pszComputerName = (PSZ)NULL;
		splerr = SplEnumQueue(pszComputerName, ulLevel, pBuf, 0L, &cReturned, &cTotal, &cbNeeded, NULL);

		if (splerr == ERROR_MORE_DATA)
		{
			if(!DosAllocMem(&pBuf, cbNeeded, PAG_READ | PAG_WRITE | PAG_COMMIT))
			{
				cbBuf = cbNeeded;
				splerr = SplEnumQueue(pszComputerName, ulLevel, pBuf, cbBuf, &cReturned, &cTotal, &cbNeeded,
NULL);
				if (splerr == NO_ERROR)
				{
					/* Set pointer to the beginning of the buffer */
					prq = (PPRQINFO3)pBuf;

					for (i=0;i < cReturned;i++)
					{
					erg[i] = prq->fsStatus;
					}
				}
				DosFreeMem(pBuf);
				return &erg;
			}
		} /* end if Q level given */
		else
		{
			return(splerr);
		}
		DosExit(EXIT_PROCESS, 0);
}
```

/* get_queueJobId.c */

```
#define INCL_BASE
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS

#include <os2.h>
#include <stdio.h>
#include „get_queueJobId.h“

LONG get_queueJobId()
{
		SPLERR		splerr;
		USHORT		jobCount;
		ULONG		cbBuf;
		ULONG		cTotal;
		ULONG		cReturned;
		ULONG		cbNeeded;
		ULONG		ulLevel;
		ULONG		i,j;
		PSZ		pszComputerName;
		PBYTE		pBuf;
		PPRQINFO3	prq;
		PPRJINFO2	prj2;
		LONG		erg[40];
		ulLevel = 4L;
```

```
        pszComputerName = (PSZ)NULL;
        splerr = SplEnumQueue(pszComputerName, ulLevel, pBuf, 0L, &cReturned, &cTotal, &cbNeeded, NULL);
        if (splerr == ERROR_MORE_DATA)

        {
            if(!DosAllocMem(&pBuf, cbNeeded, PAG_READ | PAG_WRITE | PAG_COMMIT))
            {
              cbBuf = cbNeeded;
              splerr = SplEnumQueue(pszComputerName, ulLevel, pBuf, cbBuf, &cReturned, &cTotal, &cbNeeded,
              NULL);
              if (splerr == NO_ERROR)
              {
                /* Set pointer to the beginning of the buffer */
                prq = (PPRQINFO3)pBuf;

                /* Save the count of jobs. There are this many PRJINFO2 */
                /* structures following the PRQINFO3 structure */
                jobCount = prq->cJobs;
                /* Increment the pointer past the PRQINFO3 structure */
                prq++;
                /* Set a pointer to point to the first PRJINFO2 structure */
                prj2 = (PPRJINFO2)prq;
                for (j=0; j<jobCount; j++)
                {   erg[i] = prj2->uJobId;
                    /* Increment the pointer to point the next structure */
                    prj2++;
                } /* end for jobCount */
                prq = (PPRQINFO3)prj2;
              }
              DosFreeMem(pBuf);
              return &erg;
            }
        } /* end if Q level given */
        else
        {
        return(splerr);
        }
        DosExit(EXIT_PROCESS, 0);
}
```

/* get_queueJobOwner.c */

```
#define INCL_BASE
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS

#include <os2.h>
#include <stdio.h>
#include „get_queueJobOwner.h“

CHAR *get_queueJobOwner()
{
        SPLERR      splerr;
        USHORT      jobCount;
        ULONG       cbBuf;
        ULONG       cTotal;
        ULONG       cReturned;
        ULONG       cbNeeded;
        ULONG       ulLevel;
        ULONG       i,j;
```

```
        PSZ             pszComputerName;
        PBYTE           pBuf;
        PPRQINFO3       prq;
        PPRJINFO2       prj2;
        CHAR            erg[40,40];
        ulLevel = 4L;
        pszComputerName = (PSZ)NULL;
        splerr = SplEnumQueue(pszComputerName, ulLevel, pBuf, 0L, &cReturned, &cTotal, &cbNeeded, NULL);
        if (splerr == ERROR_MORE_DATA)
        {
                if(!DosAllocMem(&pBuf, cbNeeded, PAG_READ | PAG_WRITE | PAG_COMMIT))
                {
                    cbBuf = cbNeeded;
                    splerr = SplEnumQueue(pszComputerName, ulLevel, pBuf, cbBuf, &cReturned, &cTotal, &cbNeeded,
NULL);
                    if (splerr == NO_ERROR)
                    {
                        /* Set pointer to the beginning of the buffer */
                        prq = (PPRQINFO3)pBuf;

                        /* Save the count of jobs. There are this many PRJINFO2 */
                        /* structures following the PRQINFO3 structure */
                        jobCount = prq->cJobs;

                        /* Increment the pointer past the PRQINFO3 structure */
                        prq++;
                        /* Set a pointer to point to the first PRJINFO2 structure */
                        prj2 = (PPRJINFO2)prq;
                        for (j=0; j<jobCount; j++)
                        {
                            erg[j,j] = prj2->uJobId, prj2->pszUserName;
                            /* Increment the pointer to point the next structure */
                            prj2++;
                        } /* end for jobCount */
                        prq = (PPRQINFO3)prj2;
                    }
                    DosFreeMem(pBuf);
                    return &erg;
                }
        }
        else
        {
                return(splerr);
        }
        DosExit(EXIT_PROCESS, 0);
        return(splerr);
}
```

/* get_queueJobEntered.c */

```
#define INCL_BASE
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS

#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include „get_queueJobEntered.h“
```

```
CHAR *get_queueJobEntered()
{
        SPLERR      splerr;
        USHORT      jobCount;
        ULONG       cbBuf;
        ULONG       cTotal;
        ULONG       cReturned;
        ULONG       cbNeeded;
        ULONG       ulLevel;
        ULONG       j;
        PSZ         pszComputerName;
        PBYTE       pBuf;
        PPRJINFO2   prj2;
        PPRQINFO3   prq;
        CHAR        erg[40,40,40,40];
        double x;
        long hh, mm, ss;
        ulLevel = 4L;
        pszComputerName = (PSZ)NULL;
        splerr = SplEnumQueue(pszComputerName, ulLevel, pBuf, 0L, &cReturned, &cTotal, &cbNeeded, NULL);

        if (splerr == ERROR_MORE_DATA)
        {
            if(!DosAllocMem(&pBuf, cbNeeded, PAG_READ | PAG_WRITE | PAG_COMMIT))
            {
              cbBuf = cbNeeded;
              splerr = SplEnumQueue(pszComputerName, ulLevel, pBuf, cbBuf, &cReturned, &cTotal, &cbNeeded,
              NULL);
              if (splerr == NO_ERROR)
              {
                /* Set pointer to the beginning of the buffer */
                prq = (PPRQINFO3)pBuf;
                /* Save the count of jobs. There are this many PRJINFO2 */
                /* structures following the PRQINFO3 structure */
                jobCount = prq->cJobs;
                printf(„Job count in this queue is %d\n\n“, jobCount);
                /* Increment the pointer past the PRQINFO3 structure */
                prq++;
                /* Set a pointer to point to the first PRJINFO2 structure */
                prj2 = (PPRJINFO2)prq;
                for (j=0; j<jobCount; j++)
                {
                  x = ((prj2->ulSubmitted / 3600.) / 24.);
                  x = ((x - labs(x)) * 24.);
                  hh = labs(x);
                  x = ((x - hh) * 60.);
                  mm = labs(x);
                  x = ((x - mm) * 60.);
                  ss = labs(x);
                  erg[j,j,j,j] = prj2->uJobId, hh, mm, ss;
                  /* Increment the pointer to point the next structure */
                  prj2++;
                } /* end for jobCount */
              prq = (PPRQINFO3)prj2;
              }
              DosFreeMem(pBuf);
              return &erg;
            }
        } /* end if Q level given */
        else
        {
            return (splerr);}
        DosExit(EXIT_PROCESS, 0);
}
```

/* get_queueJobPrio.c */

```
#define INCL_BASE
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS

#include <os2.h>
#include <stdio.h>
#include „get_queueJobPrio.h“

LONG get_queueJobPrio()
{
        SPLERR          splerr;
        USHORT          jobCount;
        ULONG           cbBuf;
        ULONG           cTotal;
        ULONG           cReturned;
        ULONG           cbNeeded;
        ULONG           ulLevel;
        ULONG           i,j;
        PSZ             pszComputerName;
        PBYTE           pBuf;
        PPRQINFO3       prq;
        PPRJINFO2       prj2;
        LONG            erg[40,40]
        ulLevel = 4L;
        pszComputerName = (PSZ)NULL;
        splerr = SplEnumQueue(pszComputerName, ulLevel, pBuf, 0L, &cReturned, &cTotal, &cbNeeded, NULL);
        if (splerr == ERROR_MORE_DATA)
        {
            if(!DosAllocMem(&pBuf, cbNeeded, PAG_READ | PAG_WRITE | PAG_COMMIT))
            {
              cbBuf = cbNeeded;
              splerr = SplEnumQueue(pszComputerName, ulLevel, pBuf, cbBuf, &cReturned, &cTotal, &cbNeeded,
              NULL);
              if (splerr == NO_ERROR) {
                /* Set pointer to the beginning of the buffer */
                prq = (PPRQINFO3)pBuf;
                /* Save the count of jobs. There are this many PRJINFO2 */
                /* structures following the PRQINFO3 structure */
                jobCount = prq->cJobs;
                /* Increment the pointer past the PRQINFO3 structure */
                prq++;
                /* Set a pointer to point to the first PRJINFO2 structure */
                prj2 = (PPRJINFO2)prq;
                for (j=0; j<jobCount; j++)
                {
                  erg[j,j] = prj2-> uJobId, prj2->uPriority;
                  /* Increment the pointer to point the next structure */
                  prj2++;
                } /* end for jobCount */
                prq = (PPRQINFO3)prj2;
              }
              DosFreeMem(pBuf);
              return &erg;
            }
        } /* end if Q level given */
        else
        { return (splerr);
        }
        DosExit(EXIT_PROCESS, 0);
}
```

/* get_queueJobSize.c */

```
#define INCL_BASE
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS

#include <os2.h>
#include <stdio.h>
#include „get_queueJobSize.h“

LONG get_queueJobSize()
{
        SPLERR      splerr;
        USHORT      jobCount;
        ULONG       cbBuf;
        ULONG       cTotal;
        ULONG       cReturned;
        ULONG       cbNeeded;
        ULONG       ulLevel;
        ULONG       j;
        PSZ         pszComputerName;
        PBYTE       pBuf;
        PPRQINFO3   prq;
        PPRJINFO2   prj2;
        LONG        erg[40,40]
        ulLevel = 4L;
        pszComputerName = (PSZ)NULL;
        splerr = SplEnumQueue(pszComputerName, ulLevel, pBuf, 0L, &cReturned, &cTotal, &cbNeeded, NULL);
        if (splerr == ERROR_MORE_DATA)
        {
            if(!DosAllocMem(&pBuf, cbNeeded, PAG_READ | PAG_WRITE | PAG_COMMIT))
            {
              cbBuf = cbNeeded;
              splerr = SplEnumQueue(pszComputerName, ulLevel, pBuf, cbBuf, &cReturned, &cTotal, &cbNeeded,
              NULL);
              if (splerr == NO_ERROR)
              {
                /* Set pointer to the beginning of the buffer */
                prq = (PPRQINFO3)pBuf;
                /* Save the count of jobs. There are this many PRJINFO2 */
                /* structures following the PRQINFO3 structure */
                jobCount = prq->cJobs;
                /* Increment the pointer past the PRQINFO3 structure */
                prq++;
                /* Set a pointer to point to the first PRJINFO2 structure */
                prj2 = (PPRJINFO2)prq;
                for (j=0; j<jobCount; j++)
                {
                  erg[j,j] = prj2->uJobId, prj2->ulSize;
                  /* Increment the pointer to point the next structure */
                  prj2++;
                } /* end for jobCount */
                prq = (PPRQINFO3)prj2;
              }
              DosFreeMem(pBuf);
              return &erg; }
        }
        else
        { return (splerr);
        }
        DosExit(EXIT_PROCESS, 0);
        return(splerr); }
```

/* get_queueJobPrinted.c */

```
#define INCL_BASE
#define INCL_DOSMEMMGR
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS

#include <os2.h>
#include <stdio.h>
#include „get_queueJobPrinted.h"

LONG get_queueJobPrinted()
{	ULONG		cbBuf;
	ULONG		cTotal;
	ULONG		cReturned;
	ULONG		cbNeeded;
	ULONG		ulLevel = 3L;
	ULONG		i;
	SPLERR		splerr;
	PSZ		pszComputerName;
	PBYTE		pBuf;
	PPRDINFO3	pprd3;
	LONG		erg[40,40];
	pszComputerName = (PSZ)NULL;
	splerr = SplEnumDevice(pszComputerName, ulLevel, pBuf, 0L, &cReturned, &cTotal, &cbNeeded, NULL);
	if (splerr == ERROR_MORE_DATA )
	{
		if (!DosAllocMem( &pBuf, cbNeeded, PAG_READ|PAG_WRITE|PAG_COMMIT))
		{
			cbBuf = cbNeeded;
			splerr = SplEnumDevice(pszComputerName, ulLevel, pBuf, cbBuf, &cReturned, &cTotal, &cbNeeded,
			NULL);
			if (splerr == NO_ERROR)
			{
				for (i=0; i < cReturned; i++)
				{
					pprd3 = (PPRDINFO3)pBuf+i;
					erg[i,i] = pprd3->uJobId, pprd3->time);
				}
			}
			DosFreeMem(pBuf);
			return &erg;
		}
	}
	else
	{ return (splerr);
	}
	DosExit( EXIT_PROCESS, 0);
}
```

/* get_queuePrinterName.c */

```
#define INCL_BASE
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS
#include <os2.h>
#include <stdio.h>
#include „get_queuePrinterName.h"
```

```
CHAR *get_queuePrinterName()
{
        SPLERR      splerr;
        USHORT      jobCount;
        ULONG       cbBuf;
        ULONG       cTotal;
        ULONG       cReturned;
        ULONG       cbNeeded;
        ULONG       ulLevel;
        ULONG       i;
        PSZ         pszComputerName;
        PBYTE       pBuf;
        PPRQINFO3   prq;
        CHAR        erg[20];
        ulLevel = 4L;
        pszComputerName = (PSZ)NULL;
        splerr = SplEnumQueue(pszComputerName, ulLevel, pBuf, 0L, &cReturned, &cTotal, &cbNeeded, NULL);
        if (splerr == ERROR_MORE_DATA)
        {
            if(!DosAllocMem(&pBuf, cbNeeded, PAG_READ | PAG_WRITE | PAG_COMMIT))
            {
                cbBuf = cbNeeded;
                splerr = SplEnumQueue(pszComputerName, ulLevel, pBuf, cbBuf, &cReturned, &cTotal, &cbNeeded,
                NULL);
                if (splerr == NO_ERROR)
                {
                    /* Set pointer to the beginning of the buffer */
                    prq = (PPRQINFO3)pBuf;
                    /* cReturned has the count of the beginning of PRQINFO3 structures */
                    for (i = 0; i<cReturned; i++)
                    {
                        erg[i] = prq->pszPrinters;
                    } /* end for cReturned */
                }
                DosFreeMem(pBuf);
                return &erg;
            }
        } /* end if Q level given */
        else
        { return (splerr);
        }
        DosExit(EXIT_PROCESS, 0);
}
```

/* get_queuePrinterLocation.c */

```
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS
#include <os2.h>
#include <stdio.h>
#include „get_queuePrinterLocation.h“

CHAR *get_queuePrinterLocation()
{
        PVOID       pBuf;
        ULONG       fsType;
        ULONG       cbBuf;
        ULONG       cRes;
        ULONG       cTotal;
        ULONG       cbNeeded;
```

```
        SPLERR          splerr = 0;
        PPRINTERINFO    pRes;
        CHAR            erg[20];
        fsType = SPL_PR_QUEUE | SPL_PR_DIRECT_DEVICE | SPL_PR_QUEUED_DEVICE | SPL_PR_LOCAL_ONLY;
        splerr = SplEnumPrinter(NULL, 0, fsType, NULL, NULL, &cRes, &cTotal, &cbNeeded, NULL);

        if (splerr != 0)
        {
                DosAllocMem(&pBuf, cbNeeded, PAG_READ | PAG_WRITE | PAG_COMMIT);
                cbBuf = cbNeeded;
                splerr = SplEnumPrinter(NULL, 0, fsType, pBuf, cbBuf, &cRes, &cTotal, &cbNeeded, NULL);
                pRes = (PPRINTERINFO)pBuf;
                while (cRes--)
                {
                    if (pRes[cRes].pszLocalName == NULL)
                    {
                        erg[cRes] = pRes[cRes].pszPrintDestinationName;
                    }
                    else
                    {   erg[cRes] = pRes[cRes].pszPrintDestinationName, pRes[cRes].pszLocalName;
                    }
                }
        DosFreeMem(pBuf);
        return &erg;
        }
        else
        {
                return (splerr);
        }
        DosExit(EXIT_PROCESS, 0);
}
```

/* get_processPPID.c */

```
#define INCL_DOSPROCESS
#define INCL_DOS
#define INCL_DOSERRORS
#include <stdio.h>
#include <os2.h>
#include „get_processPPID.h“

LONG get_processPPID()
{
        UCHAR           LoadError[CCHMAXPATH] ={0};
        RESULTCODES     ChildRC = {0};
        PID             pidChild = 0;
        PSZ             Args = „PSTAT\0 /C\0“;
        PSZ             Envs = NULL;
        APIRET          rc = NO_ERROR;
        rc = DosExecPgm(LoadError, sizeof(LoadError), EXEC_ASYNC, Args, Envs, &ChildRC, „PSTAT.EXE“);
        return rc;
}
```

/* get_processPID.c */

```
#define INCL_DOSPROCESS
#define INCL_DOS
#define INCL_DOSERRORS
```

```
#include <stdio.h>
#include <os2.h>
#include „get_processPID.h“

LONG get_processPID()
{
        UCHAR           LoadError[CCHMAXPATH] ={0};
        RESULTCODES     ChildRC = {0};
        PID             pidChild = 0;
        PSZ             Args = „PSTAT\0 /C\0“;
        PSZ             Envs = NULL;
        APIRET          rc = NO_ERROR;

        rc = DosExecPgm(LoadError, sizeof(LoadError), EXEC_ASYNC, Args, Envs, &ChildRC, „PSTAT.EXE“);

        return rc;
}
```

/* get_processName.c */

```
#define INCL_DOSPROCESS
#define INCL_DOS
#define INCL_DOSERRORS
#include <stdio.h>
#include <os2.h>
#include „get_processName.h“

CHAR *get_processName()
{
        UCHAR                   LoadError[CCHMAXPATH] ={0};
        RESULTCODES             ChildRC = {0};
        PID                     pidChild = 0;
        PSZ                     Args = „PSTAT\0 /C\0“;
        PSZ                     Envs = NULL;
        APIRET                  rc = NO_ERROR;
        rc = DosExecPgm(LoadError, sizeof(LoadError), EXEC_ASYNC, Args, Envs, &ChildRC, „PSTAT.EXE“);
        return rc;
}
```

/* get_processState.c */

```
#define INCL_DOSPROCESS
#define INCL_DOS
#define INCL_DOSERRORS
#include <stdio.h>
#include <os2.h>
#include „get_processState.h“

LONG get_processState()
{       UCHAR           LoadError[CCHMAXPATH] ={0};
        RESULTCODES     ChildRC = {0};
        PID             pidChild = 0;
        PSZ             Args = „PSTAT\0 /C\0“;
        PSZ             Envs = NULL;
        APIRET          rc = NO_ERROR;
        rc = DosExecPgm(LoadError, sizeof(LoadError), EXEC_ASYNC, Args, Envs, &ChildRC, „PSTAT.EXE“);
        return rc;
}
```

/* get_processPriority.c */

```
#define INCL_DOSPROCESS
#define INCL_DOS
#define INCL_DOSERRORS
#include <stdio.h>
#include <os2.h>
#include „get_processPriority.h“

LONG get_processPriority()
{
        UCHAR           LoadError[CCHMAXPATH] ={0};
        RESULTCODES     ChildRC = {0};
        PID             pidChild = 0;
        PSZ             Args = „PSTAT\0 /C\0“;
        PSZ             Envs = NULL;
        APIRET          rc = NO_ERROR;

        rc = DosExecPgm(LoadError, sizeof(LoadError), EXEC_ASYNC, Args, Envs, &ChildRC, „PSTAT.EXE“);
        return rc;
}
```

/* get_stoSize.c */

```
#define INCL_DOSFILEMGR/* File Manager values */
#define INCL_DOSMISC
#define INCL_DOSERRORS/* DOS Error values */
#include <os2.h>
#include <stdio.h>
#include „get_stoSize.h“

LONG get_stoSize(LONG antwort, LONG laufwerk)
{
        ULONG   aulFSInfoBuf[40] = {0};         /* File system info buffer */
        APIRET  rc = NO_ERROR;                  /* Return code */
        ULONG   ergebnis;
        ULONG   ulDriveNum = 0;                 /* Drive number */
        ULONG   ulDriveMap = 0;                 /* Mapping of valid drives */
        ULONG   erg[1];

        if (antwort == 1)
        {
                ULONG aulSysInfo[QSV_MAX] = {0};
                APIRET rc = NO_ERROR;

                rc = DosQuerySysInfo(1L, QSV_MAX, (PVOID)aulSysInfo, sizeof(ULONG)*QSV_MAX);
                if (rc!=NO_ERROR)
                {
                    printf(„DosQuerySysInfo error: return code = %u\n“, rc);
                    return rc;
                }
                else
                {
                return &aulSysInfo;
                }
        }
        else
        {
        rc = DosQueryCurrentDisk (&ulDriveNum, &ulDriveMap);
```

```
                if (rc != NO_ERROR)
                {
                        rc = DosQueryFSInfo(laufwerk,   /* Drive number */
                        FSIL_ALLOC,                     /* Level 1 allocation info */
                        (PVOID)aulFSInfoBuf,            /* Buffer */
                        sizeof(aulFSInfoBuf));          /* Size of buffer */

                if (rc != NO_ERROR)
                {
                            printf(„DosQueryFSInfo error: return code = %u\n“, rc);
                            return rc;
                }
                else
                {
                            ergebnis = aulFSInfoBuf[2] * (aulFSInfoBuf[1] * (USHORT)aulFSInfoBuf[4]);
                            erg[0] = aulFSInfoBuf[2];
                            erg[1] = ergebnis;
                            return &erg;
                }
                DosExit(EXIT_THREAD, aulFSInfoBuf[3]);/* Return available allocation units to the initiating process */
                }
}
```

/* get_stoUsed.c */

```
#define INCL_DOSFILEMGR       /* File Manager values */
#define INCL_DOSERRORS        /* DOS Error values */
#include <os2.h>
#include <stdio.h>
#include „get_stoUsed.h“

LONG get_stoUsed()
{
                ULONG aulFSInfoBuf[40] = {0};       /* File system info buffer */
                APIRET rc = NO_ERROR;               /* Return code */
                ULONG ergebnis;
                ULONG ulDriveNum = 0;               /* Drive number */
                ULONG ulDriveMap = 0;               /* Mapping of valid drives */
                ULONG erg[1];
                rc = DosQueryCurrentDisk (&ulDriveNum, &ulDriveMap);
                if (rc != NO_ERROR)
                {
                        return rc;
                }
                rc = DosQueryFSInfo(laufwerk,                   /* Drive number */
                                FSIL_ALLOC,                     /* Level 1 allocation info */
                                (PVOID)aulFSInfoBuf,            /* Buffer */
                                sizeof(aulFSInfoBuf));          /* Size of buffer */
                if (rc != NO_ERROR)
                {
                        return rc;
                }
                else
                {
                        ergebnis = (aulFSInfoBuf[2]-aulFSInfoBuf[3]) * (aulFSInfoBuf[1] * (USHORT)aulFSInfoBuf[4]);
                        erg[0] = aulFSInfoBuf[2]-aulFSInfoBuf[3];
                        erg[1] = ergebnis;
                        return &erg;
                }
                DosExit(EXIT_THREAD, aulFSInfoBuf[3]);/* Return available allocation units to the initiating process */
}
```

/* get_stoAllocationUnits.c */

```
#define INCL_DOSFILEMGR          /* File Manager values */
#define INCL_DOSERRORS           /* DOS Error values */
#include <os2.h>
#include <stdio.h>
#include „get_stoAllocationUnits.h“

LONG get_stoAllocationUnits(LONG laufwerk)
{
        ULONG   aulFSInfoBuf[40] = {0};  /* File system info buffer */
        APIRET  rc = NO_ERROR;           /* Return code */
        ULONG   ergebnis;
        ULONG   ulDriveNum = 0;          /* Drive number */
        ULONG   ulDriveMap = 0;          /* Mapping of valid drives */
        ULONG   erg;
        rc = DosQueryCurrentDisk (&ulDriveNum, &ulDriveMap);

        if (rc != NO_ERROR)
        {
                return rc;
        }
        rc = DosQueryFSInfo(laufwerk,                  /* Drive number */
                        FSIL_ALLOC,                    /* Level 1 allocation info */
                        (PVOID)aulFSInfoBuf,           /* Buffer */
                        sizeof(aulFSInfoBuf));         /* Size of buffer */

        if (rc != NO_ERROR)
        {
                return rc;
        }
        else
        {
                erg = aulFSInfoBuf[1] * (USHORT)aulFSInfoBuf[4];
                /* (Sectors per allocation unit) x (Bytes per sector) */
                return erg;
        }
        DosExit(EXIT_THREAD, aulFSInfoBuf[3]);/* Return available allocation units to the initiating process */
}
```

/* get_stoState.c */

```
#define INCL_DOSFILEMGR      /* File Manager values */
#define INCL_DOSERRORS       /* DOS Error values */
#include <os2.h>
#include <stdio.h>
#include „get_stoState.h“

LONG get_stoState(LONG laufwerk)
{
        ULONG   aulFSInfoBuf[40] = {0};  /* File system info buffer */
        APIRET  rc = NO_ERROR;           /* Return code */
        ULONG   ulDriveNum = 0;          /* Drive number */
        ULONG   ulDriveMap = 0;          /* Mapping of valid drives */
        LONG    ergbenis_3;
        float   ergebnis_1, ergebnis_2;
        rc = DosQueryCurrentDisk (&ulDriveNum, &ulDriveMap);
        if (rc != NO_ERROR)
        { return rc;
        }
```

```
        rc = DosQueryFSInfo(laufwerk,                      /* Drive number*/
                            FSIL_ALLOC,                    /* Level 1 allocation info */
                            (PVOID)aulFSInfoBuf,           /* Buffer */
                            sizeof(aulFSInfoBuf));         /* Size of buffer */

        if (rc != NO_ERROR)
        {
                return rc;
        }
        else
        {
                ergebnis_1 = (aulFSInfoBuf[2]-aulFSInfoBuf[3]) * (aulFSInfoBuf[1] * (USHORT)aulFSInfoBuf[4]);
                /* used bytes on disk */

                ergebnis_2 = aulFSInfoBuf[2] * (aulFSInfoBuf[1] * (USHORT)aulFSInfoBuf[4]);
                /* all bytes on disk */

                ergebnis_3 = (ergebnis_1 / ergebnis_2) * 100;
                return ergebnis_3;
        }
        DosExit(EXIT_THREAD, aulFSInfoBuf[3]);/* Return available allocation units to the initiating process */
}
```

/* get_sysClockTicks.c */

```
#define INCL_DOSFILEMGR
#define INCL_DOSMISC
#include <os2.h>
#include „get_sysClockTicks.h“

LONG get_sysClockTicks()
{
        ULONG   SysVar[23];         /* Puffer für alle Systemvariablen */
        ULONG   TimerIntervall;     /* physikalische Tick-Dauer */
        /* Bestimmen aller Systemvariablen */
        DosQuerySysInfo (QSV_MAX_PATH_LENGTH,          /* Startindex */
                         QSV_MAX_COMP_LENGTH,          /* EndIndex (alle) */
                         &SysVar,                      /* Puffer (23x4 Byte) */
                         sizeof (SysVar));             /* Puffergröße */
        /* Physikalische Tick-Dauer */
        TimerIntervall = SysVar[QSV_TIMER_INTERVAL-1];
        return TimerIntervall;
}
```

/* get_sysDate.c */

```
#define INCL_DOSDATEMTIME
#include <stdio.h>
#include <os2.h>
#include „get_sysDate.h“

LONG *get_sysDate()
{
        DATETIME DateTime;
        /* Abfragen der aktuellen Systemzeit */
        DosGetDateTime (&DateTime);
        return &DateTime;
}
```

/* set_sysDate.c */

```
#define INCL_DOSDATEMTIME
#define INCL_DOSERRORS
#include <stdio.h>
#include <os2.h>
#include „set_sysDate.h“

LONG set_sysDate(DATETIME DateTime)
{
        APIRET      rc = NO_ERROR;
        rc = DosGetDateTime(&DateTime);
        if (rc != NO_ERROR)
        {
            return rc;
        }
        rc = DosSetDateTime(&DateTime);
        if (rc != NO_ERROR)
        {
            printf(„DosSetDateTime error : return code = %u\n“, rc);
            return rc;
        }
        rc = DosGetDateTime(&DateTime);

        if (rc != NO_ERROR)
        {
            return rc;
        }
        else
        {   return &DateTime;
        }
}
```

/* get_sysHardware.c */

```
#define INCL_DOSDEVICES
#define INCL_DOSERRORS
#include <os2.h>
#include <stdio.h>
#include „get_sysHardware.h“

CHAR *get_sysHardware()
{
        BYTE    Model       = 0,
                SubModel    = 0,
                Adapter     = 0,
                CoProc      = 0;
        CHAR    erg[1];
        APIRET rc           = NO_ERROR;
        rc = DosDevConfig(&Model, DEVINFO_MODEL);
        if (rc != NO_ERROR)
        {
            return rc;
        }
        erg[0] = DosDevConfig(&SubModel, DEVINFO_SUBMODEL);
        erg[1] = DosDevConfig(&Adapter, DEVINFO_ADAPTER);
        erg[2] = DosDevConfig(&CoProc, DEVINFO_COPROCESSOR);
        return &erg;
}
```

/* get_sysName.c */

```
#define INCL_DOSPROCESS
#define INCL_DOS
#define INCL_DOSERRORS
#include <stdio.h>
#include <os2.h>

#include „get_sysName.h“

CHAR *get_sysName()
{
        UCHAR               LoadError[CCHMAXPATH] ={0};
        PSZ                 Args = NULL;
        PSZ                 Envs = NULL;
        RESULTCODES         ChildRC = {0};
        APIRET              rc = NO_ERROR;

        rc = DosExecPgm(LoadError, sizeof(LoadError), EXEC_SYNC, Args, Envs, &ChildRC, „hostname.exe“);
        return rc;
}
```

/* get_sysOs.c */

```
#define INCL_DOSFILEMGR
#define INCL_DOSMISC
#include <os2.h>
#include „get_sysOs.h“

CHAR *get_sysOS()
{
        ULONG SysVar[23];           /* Puffer für alle Systemvariablen */
        ULONG Version;              /* Angabe des Betriebssystems */
        CHAR * erg;
        /* Bestimmen aller Systemvariablen */
        DosQuerySysInfo (QSV_MAX_PATH_LENGTH,         /* Startindex */
                         QSV_MAX_COMP_LENGTH,         /* EndIndex (alle) */
                         &SysVar,                     /* Puffer (23x4 Byte) */
                         sizeof (SysVar));            /* Puffergröße */

        Version = SysVar[QSV_VERSION_MINOR-1];
        if (Version == 00)
                return erg = “OS/2 2.0”;
        if (Version == 10)
                return erg =”OS/2 2.1”;
        if (Version == 11)
                return erg =”OS/2 2.11”;
        else
                return erg =”OS/2 3.0”;
}
```

/* get_timezone.c */

```
#define INCL_DOSDATEMTIME
#include <stdio.h>
#include <os2.h>
#include „get_timezone.h“
```

```
LONG *get_timezone()
{           DATETIME DateTime;
            CHAR * erg;

            /* Abfragen der aktuellen Systemzeit */
            DosGetDateTime (&DateTime);

            return = DateTime.timezone;
}
```

/* get_sysUptime.c */

```
#define INCL_DOSFILEMGR
#define INCL_DOSMISC
#include <os2.h>
#include „get_sysUptime.h“

LONG get_sysUptime()
{
            ULONG SysVar[23];           /* Puffer für alle Systemvariablen */
            ULONG TimerIntervall;       /* physikalische Tick-Dauer */
             LONG Minuten;              /* Anzahl der Minuten seit letztem Systemstart */

            /* Bestimmen aller Systemvariablen */
            DosQuerySysInfo (QSV_MAX_PATH_LENGTH,       /* Startindex */
                             QSV_MAX_COMP_LENGTH,       /* EndIndex (alle) */
                             &SysVar,                   /* Puffer (23x4 Byte) */
                             sizeof (SysVar));          /* Puffergröße */

            /* Zeit seit Systemstart bestimmen (Minuten) */
            Minuten = (SysVar[QSV_MS_COUNT-1] / 1000) / 60;
            return Minuten;
}
```

/* get_MaxProcessSize.c */

```
#define INCL_DOSFILEMGR/* File Manager values */
#define INCL_DOSMISC
#define INCL_DOSERRORS/* DOS Error values */
#include <os2.h>
#include <stdio.h>
#include „get_MaxProcessSize.h“

LONG get_MaxProcessSize()
{
            ULONG     aulSysInfo[QSV_MAX] = {0};    /* File system info buffer */
            APIRET    rc = NO_ERROR;                /* Return code */
            LONG      erg;
            rc = DosQuerySysInfo(1L, QSV_MAX, (PVOID)aulSysInfo, sizeof(ULONG)*QSV_MAX);
            if (rc!=NO_ERROR)
                  return rc;
            }
            else
            {
                  erg = aulSysInfo[QSV_MAXPRMEM-1] + aulSysInfo[QSV_MAXSHMEM-1]));
                  return erg;
            }
}
```

/* get_curProcessNumber.c */

```
#define INCL_DOSPROCESS
#define INCL_DOS
#define INCL_DOSERRORS
#include <stdio.h>
#include <os2.h>
#include „get_curProcessNumber.h“

LONG get_curProcessNumber()
{
        UCHAR           LoadError[CCHMAXPATH] ={0};
        RESULTCODES     ChildRC = {0};
        PID             pidChild = 0;
        PSZ             Args = „PSTAT\0 /C\0“;
        PSZ             Envs = NULL;
        APIRET          rc = NO_ERROR;

        rc = DosExecPgm(LoadError, sizeof(LoadError), EXEC_ASYNC, Args, Envs, &ChildRC, „PSTAT.EXE“);

        return rc;
}
```

Abbkürzungsverzeichnis

API	Application Programming Interface
CORBA	Common Object Request Broker Architecture
ER	Entity Relationship
FAT	File Allocation Table
HPFS	High Performance File System
IDL	Interface Definition Language
LRZ	Leibniz Rechenzentrum
MIB	Management Information Base
MO	Managed Object
OMA	Object Management Architecture
OMG	Object Management Group
OMT	Object Modeling Technique
ORB	Object Request Broker
SOM	System Object Model
StP	Software through Pictures

Literaturverzeichnis

[Heck95] Hecker D., Götz H.-J.; "OS/2 Warp Version 3, Integrationsplattform"; Franzis, 1995

[Heg93] Hegering H-G., Abeck S.; "Integriertes Netz- und Systemmanagement"; Addison-Wesley, 1993

[Heil95] Heilbronner S., Keller A.; "Weiterentwicklung der Agenten zum Management von Endsystemen und systemnahen Anwendungen"; Universität München, Dez. 1995

[Heil96] Heilbronner S., Keller A., Neumaier B.; "Integriertes Netz - und Systemmanagement mit modularen Agenten"; In: Proceedings of SIWORK'96, Zürich, Switzerland, 1996

[Müsb94] Müscheborn H.-J.; "OS/2 System- und Netzwerkprogrammierung"; tewi, 1994

[Sirt96] Sirtl H.; "Objektorientierte Modellierung von Workstations für ein integriertes Systemmanagement", Fortgeschrittenen-Praktikum, Technische Universität München, 1996

[SOM95] SOMobjects Developer Toolkit, "Programmer's Guide, Volume 1: SOM and DSOM", SOMobjects Version 3.0, IBM 1995