Diplomarbeit zum Thema

SNMP-basiertes Drucker-Inventarsystem

Sommer 2005

Verfasser:	Holger Peters, Praktische Informatik
	Karla Margarita Picazo Luna, Praktische Informatik
Hochschule:	Fachhochschule Braunschweig Wolfenbüttel
Erstprüfer:	Prof. Dr. Frank Klawonn
Zweitprüfer:	Dipl. Ing. Markus Windelen

Kurzfassung

Mit der ständig steigenden Anzahl von Computern in Unternehmen steigt gleichzeitig auch die Anzahl der verwendeten Drucker in denselben Unternehmen. Auch bei den Druckern wird es zunehmend schwieriger, auf herkömmlichem Wege den Überblick zu behalten. Daher wäre ein automatisches System wünschenswert, das die im Betrieb befindlichen Geräte sowohl erfasst, als auch wichtige Informationen über diese sammelt und diese in ansprechender Form den Administratoren zur Verfügung stellt. Die zentrale Verfügbarkeit solcher Informationen kann bei der Abschätzung der Wirtschaftlichkeit und beim rechtzeitigen Bereitstellen von zugehörigem Verbrauchsmaterial hilfreich sein.

Im Rahmen dieser Diplomarbeit wird eine Erweiterung zu einem bereits bestehenden Rechner-Inventarsystem entwickelt, das auch entsprechende Informationen über Netzwerkdrucker in einer Datenbank sammelt. Diese Informationen können den Systembetreuern ebenfalls über ein vergleichbares Web-Interface bereitgestellt werden.

Gemäß §25 (6) der Diplomprüfungsordnung versichern wir hiermit, dass diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt wurden.

Ort, Datum	Karla Margarita Picazo Luna
Ort, Datum	Holger Peters

Inhaltsverzeichnis

Kurzfassung ... 2
1. Einleitung ... 6
2. Vorstellung des Client-Inventarsystems ... 8
3. Lösungsansatz ... 10
3.1. Wahl der Entwicklungsumgebung ... 11
4. Allgemeine Grundlagen ... 12
4.1. Grund der Entwicklung vom .NET Technologie ... 12
4.2. Die Lösung von Microsoft: die .NET Technologie ... 12
4.3. Komponenten des .NET Framework ... 13
4.4. ADO.NET ... 15
4.4.1. Übersicht über ADO.NET ... 15
4.4.2. ADO.NET Architektur ... 15
4.4.3. ADO.NET Komponenten ... 16
4.4.4. Die wichtigsten Klassen des .NET Framework-Datenproviders ... 17
4.5. Das DataSet ... 22
4.6. ASP.NET ... 24
4.6.1. Dynamischen Webseiten ... 24
4.6.2. Überblick über ASP.NET ... 25
4.6.3. Die Architektur von ASP.NET ... 25
4.6.4. Webformulare ... 27
4.6.5. Web Services ... 27
4.7. Verteilte Anwendungen ... 28
4.7.1. Web Services allgemein ... 28
4.7.2. SOAP ... 29
4.7.3. WSDL ... 29
4.7.4. ASP.NET Web Services ... 30
4.7.5. .NET Remoting ... 30
4.8. Windows-Dienst ... 32
4.9. Multithreading ... 34
4.9.1. Nebenläufige Prozesse ... 34
4.9.2. Nebenläufige Threads ... 35
4.9.3. Multithreading unter .NET ... 35
4.9.4. Gemeinsame Ressourcen ... 36
4.10. XML Konfigurationsdatei ... 37
5. SNMP ... 39
5.1. Was ist SNMP? ... 39
5.1.1. Historisches ... 39
5.1.2. Versionen ... 40
5.1.3. Wichtige Begriffe zu SNMP ... 41
5.1.4. SNMP im OSI Schichtenmodell ... 41
5.2. Wie funktioniert SNMP? ... 42
5.2.1. Verfügbare Funktionen ... 43
5.2.2. Protocol Data Units ... 44
5.2.3. Ports ... 44
5.2.4. Management Information Base ... 45
5.2.5. Objektbezeichner ... 45
5.2.6. Scalar und Columnar Variablen ... 46
5.2.7. Sicherheit ... 46
5.3. Die SNMP Bibliothek ... 47
5.3.1. Einfaches Anwenden der Bibliothek ... 48

5.3.2. Fortgeschrittenere Anwendungsmöglichkeiten ... 51
5.4. Abfragerealisierung ... 53
5.4.1. Auszulesende Informationen ... 53
5.4.2. Der Auslesevorgang ... 56
5.4.3. Probleme beim Auslesen ... 59
6. Der Drucker-Inventar-Dienst ... 62
6.1. Allgemeines ... 62
6.2. Die Threads ... 64
6.2.1. Der Main-Thread ... 64
6.2.2. Der Scheduler-Thread ... 64
6.2.3. Der Restart-Thread ... 65
6.2.4. Der Datenbank-Thread ... 65
6.2.5. Der Auslese-Thread ... 65
6.3. Die Hauptklassen ... 67
6.3.1. Der Scheduler ... 67
6.3.2. Die DBQueue ... 67
6.3.3. Die AusleseQueue ... 67
6.3.4. Der Drucker ... 68
6.3.5. DITransfer ... 68
6.3.6. Die Konfiguration ... 68
6.4. Performanceüberlegungen ... 69
6.4.1. Beobachtungen ... 69
6.4.2. Erläuterungen ... 71
6.4.3. Empfehlungen ... 71
7. Die Datenbank ... 73
7.1. SQL und Server: Eine kurze Zusammenfassung ... 73
7.2. Der Datenbank Server ... 73
7.3. Transact-SQL ... 75
7.3.1. Allgemein ... 75
7.4. Übersicht Drucker-Datenbank ... 75
7.5. Verwendete Datenbankfunktionen ... 76
7.5.1. Primärschlüssel ... 76
7.5.2. UNIQUE Einschränkung ... 77
7.5.3. Fremdschlüssel ... 77
7.5.4. Kaskadieren von Fremdschlüsseln ... 77
7.5.5. 1:n Beziehung ... 78
7.5.6. Gespeicherte Prozeduren ... 78
7.5.7. Indizes ... 79
7.5.8. Trigger ... 79
7.5.9. Transaktionen ... 79
7.5.10. Isolationsstufen in Microsoft SQL Server 2000 ... 80
7.6. Aufbau der Datenbank ... 81
7.7. Einfügen eines Datensatzes in der Datenbank ... 84
7.8. Löschen eines kompletten Datensatzes ... 85
7.9. Löschen und Aktualisieren von Datensätzen mit 1:n Beziehungen ... 86
7.10. Aktualisieren der Datenbank ... 86
7.11. Datenbankgröße ... 86
8. Das Web-Interface ... 88
8.1. Anforderungen ... 88
8.2. Voraussetzungen ... 88
8.3. Abfragemöglichkeiten ... 89
8.4. Zugriffberechtigung ... 90

8.5. Code-Archiv ... 90
8.5.1. CSV-Export ... 90
8.5.2. PDF-Export ... 91
8.6. Administrativer Bereich ... 91
8.7. Aufbau der ASP-Seiten ... 92
8.8. Der Web Service ... 94
9. Abschluss ... 95
9.1. Geplante Erweiterungen ... 95
9.2. Abschließende Worte (Karla) ... 96
9.3. Abschließende Worte (Holger) ... 96
Anlagenteil ... 97
A. Programmierhinweise ... 98
A.1. Allgemeine Anmerkungen ... 98
A.2. Übersicht Drucker-Inventardienst ... 98
A.2.1. Klassenübersicht ... 99
A.3. Übersicht Web-Service ... 100
A.4. Übersicht Webseiten ... 101
A.5. Übersicht Datenbank ... 102
A.5.1. Datenbankdiagramm ... 102
A.5.2. Beschreibung der Tabellen ... 103
A.5.3. Beschreibung der gespeicherten Prozedur ... 105
A.5.4. Einfügen eines Datensatzes ... 106
A.5.5. Erwartete Datenbankgröße ... 107
B. Installation ... 108
B.1. Server-Installation ... 108
B.2. Konfiguration der Datenbank ... 108
B.3. Konfiguration des Web-Interfaces ... 109
B.3.1. Datenbankzugriff ... 109
B.3.2. Überwachung und Steuerung des Drucker-Inventar-Dienstes ... 109
B.3.3. NTFS Berechtigungen ... 110
C. Wartung ... 110
D. Bedienungshandbuch der Webseiten ... 111
D.1. Allgemeine Abfragen ... 111
D.2. Verwaltung ... 117
Abbildungsverzeichnis ... 119
Abkürzungsverzeichnis ... 120
Literaturverzeichnis ... 122

1. Einleitung

Mit zunehmender Zahl ist es für größere Firmen kaum noch möglich, den Überblick über Anzahl, Standort und Zustand der Rechner, wie auch im Fall dieses Projektes der Drucker, im Netzwerk zu behalten.
Ohne genauere Kenntnisse über die betreffenden Geräte ist es für die Systembetreuer sehr schwierig, Probleme zu erkennen und zu beheben. Da für die meisten Druckermodelle jeweils unterschiedliche Gerätetreiber, Ersatzeile und Verbrauchsmaterialien benötigt werden, kann das passende Material in der Regel auch nicht beim ersten Versuch mitgeführt werden. Auch von den Anwendern können solche detaillierten Kenntnisse nicht erwartet werden.

Bisher ist es bei den verantwortlichen Administratoren üblich, Informationen über die von ihnen betreuten Geräte manuell selbst zu sammeln und in eigenen Systemen, beispielsweise Excel-Tabellen, zu halten. Diese Daten sind daher nur bedingt aktuell und genau.
Da diese Informationen innerhalb der verschiedenen Verantwortungsbereiche meist nicht einheitlich geführt werden, ist es nicht möglich, diese an zentraler Stelle automatisch zusammenzufassen.

Von Interesse sind solche zentral gehaltenen Informationen auch vom wirtschaftlichen Aspekt her. Jeder Drucker kostet das Unternehmen sowohl in Anschaffung als auch im Unterhalt Geld. Die Auslastung der jeweiligen Drucker, beispielsweise anhand der gedruckten Seiten, zu kennen, ermöglicht es, kaum genutzte, unwirtschaftliche Geräte ausfindig zu machen.

Diese Diplomarbeit setzt sich mit den oben genannten Aspekten auseinander. Sie entstand im Rahmen eines Praxissemesters in der IT-Abteilung (IVS – „Informationsverarbeitungsstelle“) des Volkswagen-Werkes in Salzgitter. Die Informationsverarbeitungsstelle betreut ein Netzwerk, das aus ca. 20 Servern und ca. 2000 Clients besteht. Außerdem sind noch ca. 400 Drucker an dieses Netzwerk angeschlossen. Diese Drucker umfassen sowohl mehrere verschiedene Modelle als auch unterschiedliche Hersteller.

Für die Clients besteht bereits ein automatisches Inventarsystem, welches im Rahmen einer anderen Diplomarbeit entstand. Aufbauend auf diesem System wurde von den verantwortlichen VW-Mitarbeitern eine Erweiterung dieses Systems gewünscht, das auch Netzwerkdrucker automatisch erfassen und relevante Informationen über diese sammeln kann. Diese Informationen sollten den zuständigen Systembetreuern über ein Web-Interface zur Verfügung gestellt werden. Das Web-Interface sollte möglichst nahtlos in das bereits vorhandene eingebunden werden.

Aufgrund dieser Anforderungen und Voraussetzungen war eine Eigenentwicklung für diese Erweiterung unumgänglich.

Die Diplomarbeit lässt sich in die folgenden Abschnitte unterteilen:
Zuerst wird das bereits bestehende Client-Inventarsystem kurz vorgestellt und danach der gewählte Lösungsansatz für die Erweiterung dargestellt, um einen Überblick über das Projekt zu erhalten.
Im Anschluss werden die allgemeinen Grundlagen der im Rahmen des Projektes verwendeten Technologien (.NET, Windows-Dienst, Multithreading, etc.) erläutert. Aufgrund der zentralen Bedeutung für das Auslesen der Druckerinformationen wurde dem SNMP-Protokoll danach ein eigenes Kapitel zugestanden. Hier werden sowohl Grundlagen, wie auch die konkrete Realisierung behandelt.
Daraufhin werden der eigentliche Drucker-Inventar-Dienst und dessen Arbeitsweise ausführlich behandelt. Als nächstes folgt das Kapitel „Datenbank“ in dem die Speicherung der erfassten Informationen erläutert wird und schließlich wird das Web-Interface des Systems beschrieben.
Es folgen noch ein kurzer Ausblick auf mögliche Erweiterungen und ein Schlusswort, gefolgt von den Anlagen zu dieser Arbeit.

2. Vorstellung des Client-Inventarsystems

Wie bereits erwähnt basiert das Projekt, das im Rahmen dieser Diplomarbeit bearbeitet wurde, auf einem bereits bestehenden Client-Inventarsystem. Dieses Vorgängersystem wird hier kurz dargestellt.

Die Aufgabe des Vorgängerprojekts ist es, die Clients des Firmennetzwerks zu inventarisieren. Dazu sollen nützliche Informationen von den Clients ausgelesen und in einer Datenbank gesammelt werden. Die gesammelten Informationen sollen den zuständigen Systembetreuern schließlich über ein einfach zu bedienendes Web-Interface zur Verfügung gestellt werden.

Das System ist prinzipiell aus drei Teilen aufgebaut: dem TCP-Client, dem Inventar-Server und dem Web-Interface.

Der TCP-Client ist ein kleines Tool, welches auf den Clients ausgeführt wird. Dazu ist es auf einer allgemein zugänglichen Netzwerkfreigabe gespeichert und wird über das Anmeldeskript beim Systemstart der Clients aufgerufen. Das Tool meldet sich bzw. den Client dann beim Inventar-Server zum Auslesen an.

Der Inventar-Server ist ein Windows-Dienst, der auf einem der Netzwerkservern läuft. Dieser sammelt alle Anmeldungen der Clients zunächst in einer Warteschlange. Die Warteschlange wird von mehreren so genannten Auslesethreads abgearbeitet. Jeder dieser Threads versucht, zunächst eine Verbindung mit dem jeweiligen Client aufzubauen. Danach wird versucht die relevanten Informationen über den Client mittels der WMI-Technologie auszulesen. Die gewonnenen Informationen werden schließlich in einer Datenbank gespeichert.

Das Web-Interface umfasst sowohl eine Reihe von aktiven Webseiten als auch einen Webservice. Die Webseiten stellen mehrere Übersichten und auch Detailansichten für die in der Datenbank gespeicherten Informationen zur Verfügung. Außerdem gibt es administrative Seiten, über die die Benutzerverwaltung, die Konfigurationseinstellungen und die Fernsteuerung des Inventar-Servers, mittels des Webservices, gehandhabt werden.

Da in dem Firmennetzwerk vor Ort praktisch nur Microsoft Systeme verwendet werden, wurde im Client-Inventarsystem auch auf Microsoft Strukturen zurückgegriffen, nur der TCP-Client ist eine mit dem Borland C++ Compiler erstellte Applikation. Ansonsten wurde auf die neue .NET Architektur von Microsoft zurückgegriffen. Der Inventarserver wurde in VB.NET erstellt, die Datenbankzugriffe geschehen mittels ADO.NET, und die Webseiten benutzen ASP.NET. Als Webserver wurde der MS IIS (Microsoft Internet Information Server) in der Version 6 eingesetzt und als Datenbank wurde auf den MS SQL Server 2000 in der Version 8 zurückgegriffen.

3. Lösungsansatz

Auch wenn die Aufgabenstellungen der Projekte recht ähnlich sind, musste für das Drucker-Inventarsystem ein neuer Lösungsansatz gefunden werden, da bei Netzwerkdruckern ganz andere Voraussetzungen bestehen als bei normalen Windows Clients.
Der erste Unterschied ist, dass die Drucker nicht über ein Anmeldeskript verfügen und sich daher auch nicht selbstständig bei einem Server anmelden können. Das Ausführen einer Clientanwendung auf einem Drucker ist ebenfalls nicht ohne weiteres möglich.
Ein weiterer Unterschied ist, dass auf den Druckern in der Regel kein Windows Betriebssystem läuft und daher ein Auslesen mittels der WMI-Technologie nicht möglich ist.

Obwohl in beiden Fällen keine Kenntnisse über den genauen Bestand der zu inventarisierenden Geräte und deren Adressen im Netzwerk zur Verfügung stehen, ergibt sich gegenüber dem Client Inventarsystem ein neues Problem. Die auszulesenden Drucker müssen erst einmal im Netzwerk gefunden werden. Außerdem muss eine Alternative zum Auslesen per WMI gefunden werden.

Als einzige sinnvolle Alternative zu WMI bietet sich nur das Auslesen über das Simple Network Management Protocol (SNMP) an, da es einen hersteller- und modellübergreifenden Standard bietet. Andere Auslesemethoden sind in der Regel sehr spezifisch und daher wenig praktikabel, um alle vorhandenen Netzwerkdrucker zu erfassen. Um sie zu finden, wird das Netzwerk serverseitig komplett durchsucht und es wird versucht, gefundene Geräte als Drucker zu identifizieren.

Der Inventarisierungsablauf sieht somit etwa folgendermaßen aus:
Der Drucker-Inventar-Dienst sucht zunächst in einem definierten Netzwerkbereich nach Druckern. Dazu wird an alle IP-Adressen in diesem Bereich ein ICMP Paket (Ping) geschickt, um potentiell auslesbare Geräte zu finden. Bei den gefundenen aktiven Geräten wird anhand deren Hostnamen festgestellt, ob diese Drucker sind. Schließlich werden die Drucker per SNMP ausgelesen, die gewonnenen Daten werden dann in einer SQL-Datenbank abgelegt.
Weil nicht immer alle Drucker eingeschaltet bzw. online sind, können mit einem solchen Durchgang nicht unbedingt alle Drucker erfasst werden. Daher muss der Vorgang mehrfach wiederholt werden. Das Einleiten dieser Vorgänge übernimmt der Dienst selbstständig zu definierten Zeiten.

Außerdem ist auch ein Aktualisierungsdurchgang vorgesehen, in dem lediglich die Daten der bereits bekannten Drucker neu ausgelesen werden, um die Daten in der Datenbank aktuell zu halten. Hierbei werden gleichzeitig veraltete Druckereinträge, die nach einer bestimmten Zeit nicht mehr ausgelesen werden konnten, entfernt.
Zur Darstellung der erfassten Informationen dient ein Web-Interface. Hier sind Übersichten zu bestimmten häufig verlangten Abfragen zu finden, z.B. eine Auflistung aller gefundenen Drucker, die sich nach der Herstellerfirma filtern lässt, die Auflistung aller MAC-Adressen der Drucker oder auch eine Statistik über die insgesamt gedruckten Seiten aller Drucker. Eine Detailansicht aller ausgelesenen Daten der einzelnen Drucker ist ebenfalls möglich. Außerdem lässt sich der Drucker-Inventar-Dienst von hier aus fernsteuern und konfigurieren.

Statt den Inventar-Server zu erweitern, war es aufgrund der prinzipiellen Unterschiede beim Inventarisierungsablauf sinnvoll den Drucker-Inventar-Dienst als eigenständigen Dienst zu implementieren. Zur Speicherung der Daten wurde eine neue Datenbank erstellt, die jedoch auch problemlos zusätzlich zu der Datenbank des Client-Inventarsystems auf demselben Datenbankserver eingerichtet werden kann. Das Web-Interface wurde jedoch übernommen und erweitert, was eine besonders einheitliche Darstellung des gesamten Inventarsystems ermöglicht.

3.1. Wahl der Entwicklungsumgebung

Da das Client-Inventarsystem bereits auf Microsofts .NET Architektur aufgebaut wurde, lag es nahe, für die Erweiterung ebenso auf .NET zu setzen. So wurde auch der Drucker-Inventar-Dienst überwiegend in VB.NET erstellt. Dasselbe gilt für die Webseiten, die mittels ASP.NET entwickelt wurden. Die Kommunikation der Webseiten bzw. des Webservices geschieht mittels .NET Remoting.
Als Datenbankserver wird derselbe Server wie beim Vorgängerprojekt benutzt, ein MS SQL Server 2000 (Version 8), auf den über die ADO.NET Schnittstelle zugriffen wird. Auch als Webserver wird derselbe MS IIS (Microsoft Internet Information Server) in der Version 6 eingesetzt.
Entwickelt wurde überwiegend mit dem Microsoft Visual Studio .NET 2003, die Webseiten wurden mithilfe vom Macromedia Dreamweaver MX 2004 erstellt.
Zum Austesten der Möglichkeiten der SNMP Abfragen und zum Verifizieren der Ergebnisse während der Entwicklungsphase, war die Trialversion von NetworkView 3.1[1] sehr hilfreich.

[1] http://www.networkview.com/

4. Allgemeine Grundlagen

4.1. Grund der Entwicklung vom .NET Technologie

Im Laufe der Zeit lässt sich beobachten, dass die PCs oder Großrechner nicht mehr die Infrastruktur allein ausmachen. Das Internet, bei dem Millionen von Rechnern mittels Kommunikationsleitungen Verbindung mit Millionen von Webservern aufnehmen und Informationen austauschen können, spielt ebenfalls eine große Rolle. Viele Firmen benutzen das Internet, um komplette Dienstleistungen wie Verwaltung und Transaktionen auszuführen. Auf der anderen Seite nutzen nicht nur Rechner diesen Dienst; sondern auch mobile Geräte wie Handy (WAP, SMS, MMS, etc.) erlauben bereits heute, bestimmte Informationen abzurufen oder zu verteilen. Auch weitere portable Geräte wie PDA's (Personal Digital Assistent) können Dienste des Internets nutzen, und zukünftig werden weitere Geräte wie Spielkonsolen, Fernsehen, etc mit Kommunikationsfunktionen ausgestattet sein.

Microsoft hat sich Gedanken gemacht, wie sich Anwendungen für das Internet entwickeln lassen, die ohne Probleme von unterschiedlichen Geräten genutzt werden können. Außerdem ist es hilfreich, wenn die Entwicklung von Anwendungen für die jeweiligen Geräte mit einer Plattform und der Zusammenarbeit verschiedener Programmiersprachen möglich ist.

4.2. Die Lösung von Microsoft: die .NET Technologie

Die Lösung von Microsoft für diese Herausforderung ist die .NET Technologie. Die Definition von Microsoft für diese Technologie lautet:

„Es handelt sich um eine Plattform, die die notwendigen Werkzeuge und Technologien zur Verfügung stellt, um verteilte Webanwendungen zu entwickeln, die standardisierte Webprotokolle nutzen können. Diese Anwendungen ermöglichen die Kommunikation mit vielen Geräten wie (Mobiltelefon, PDA, etc.) .NET ist auch eine Plattform, die eine Integration zwischen verschiedenen Programmiersprachen und Laufzeitdiensten erlaubt."

Die Architektur von Microsoft .NET besteht aus vier Bausteinen.

- Visual Studio .NET ist eine Microsoft-Enwickungsplattform, mit der Anwendungen durch das .NET Framework erstellt werden können. Das Ziel von Visual Studio .NET ist die Unterstützung und Vereinfachung beim Entwicklungsprozess.

- .NET Framework ist eine Sammlung von Komponenten, die die Infrastruktur zur Entwicklung von .NET Anwendungen anbietet. Um .NET Anwendungen auf dem Zielgerät ausführen zu können, werden Teile des .NET Frameworks benötigt.
- .NET Enterprise Server sind die von Microsoft bereitgestellten Server wie zum Beispiel Windows 2000-Server, Microsoft Application Center, Exchange-Server oder Microsoft SQL Server, mit denen Web-basierte Anwendungen entwickelt und bereitgestellt werden können. Neben den .NET Servern stellt Microsoft auch .NET-Clients bereit oder entwickelt diese, um neben Windows-PCs auch weitere Geräte wie die Xbox, Mobiltelefon, PDA, etc. zu unterstützen. .NET Framework ist einer dieser Clients.
- .NET Building Block Services stellen eine Sammlung von Webdiensten (XML Web Services) dar. Diese Webdienste sind Anwendungen, die auf Webservern laufen und von Anwendungen genutzt werden können. Die Daten zwischen Webdienst und Anwendung werden durch standardisierte Internetprotokolle wie SOAP oder HTTP ausgetauscht.

Die obige Zusammenstellung zeigt, dass .NET eine komplexe Sammlung von Technologien, Anwendungen und Entwicklungsumgebungen ist, die unter einer gemeinsamen Architektur integriert wurde.

4.3. Komponenten des .NET Framework

Die wichtigsten Komponenten des .NET Framework sind zum einen die CLR (Common Language Runtime), die als Ablaufumgebung ermöglicht, dass die für das .NET Framework geschriebenen Anwendungen auf einer Zielmaschine laufen können. Wenn die CLR nur einmal auf dem Zielrechner installiert wird, werden alle mit einer der .NET Programmiersprachen wie (C#, Visual Basic, JScript, etc.) für das .NET Framework erstellten Anwendungen auf dieser Maschine laufen.

Die Klassenbibliothek ist die zweite Komponente, die .NET Framework neben der Laufzeitumgebung bereitstellt. Diese Klassenbibliothek wird von den Anwendungen und den .NET Programmiersprachen genutzt. Das Neue bei .NET ist, dass diese Klassenbibliothek nicht nur auf eine Sprache beschränkt ist, sondern auch andere Programmiersprachen gemeinsam nutzen kann. Ein C# Programm z.B. benutzt die gleiche Klassenbibliothek wie Visual Basic oder C++. Diese große Bibliothek ist in hierarchische Namensräume gegliedert. Ein Namensraum gibt dem nutzenden Programm also genau vor, wo es in der betreffenden Bibliothek die benötigte Komponente findet.

Wenn eine Dateioperation in einem Programm durchgeführt wird, wird das Programm die entsprechenden Funktionen und die benutzten Datentypen für die Variablen verwenden.
Bei der Ausführung des Programms werden die Klassen instantiiert und die Instanzen können als Objekte mit Methoden und Eigenschaften angesprochen werden. Etwas Besonderes beim .NET Framework ist das CTS (Common Type System). Dieses Modell deklariert die Regeln nach denen die CLR Typen verwaltet werden. Für diesen Zweck sind im .NET Framework die Basistypen und komplexe Typen für die Variablen und Konstanten definiert.

In .NET lassen sich Anwendungen in verschiedenen Programmiersprachen erstellen. Außerdem können in unterschiedlichen Sprachen erstellte Programmteile innerhalb der Anwendung kombiniert werden. Die CLS (Common Language Specification) ist das Regelwerk, welches den Austausch von Typen aus unterschiedlichen Programmiersprachen steuert.

Eine weitere Komponente ist ein Satz von ADO.NET (Active Data Object) Klassen, mit denen man auf Daten zugreifen kann. ADO.NET stellt Funktionen zum Zugriff auf Datenbanken (SQL Server, Access, etc) und XML-Daten zur Verfügung. Mit diesen Funktionen können die XML-Daten gelesen, bearbeitet und gespeichert werden. Weitere Informationen über ADO.NET werden zu einem späteren Zeitpunkt dargestellt.

Die letzte Komponente, die das .NET Framework anbietet, ist der ASP.NET (Active Server Page). Dieser Teil stellt Funktionen wie Web-Services und Windows User Interface bereit. ASP.NET ist dabei ein eigenes Framework, welches zur Entwicklung von Webanwendungen benutzt werden kann. ASP.NET kommt auf einem Server (Internet Information Server 5.0) zum Einsatz. Mit ASP.NET können Webformulare erstellt oder Web-Dienste entwickelt werden, die auf einem Server laufende Anwendungen darstellen. Diese Anwendungen tauschen Information mit anderen Anwendungen durch offene Internetstandards wie das HTTP-Protokoll oder XML aus. Auf ASP.NET wird im späteren Verlauf dieser Arbeit noch genauer eingegangen.

4.4. ADO.NET

Im Rahmen des Projektes wird das Übergeben der Zugriffsdaten auf die Datenbank durch ADO.NET ausgeführt.

4.4.1. Übersicht über ADO.NET

ADO.NET ist eine Reihe von Klassen, die Datenzugriffsdienste für .NET-Programmierer zur Verfügung stellen.
ADO.NET unterstützt das Erstellen der z.B. von Anwendungen, Tools, Sprachen oder Internetbrowsern verwendeten Front-End-Datenbankclients sowie von Geschäftsobjekten mittlerer Ebenen. ADO.NET ermöglicht den konsistenten Zugriff auf Datenquellen wie Microsoft SQL Server sowie OLE DB und XML. Zum Verändern, Aktualisieren und Abrufen von Daten können Verbraucheranwendungen durch Datenfreigabe eine Verbindung mit dieser Datenquelle mittels ADO.NET herstellen.

ADO.NET enthält den .NET Framework-Datenprovider für die Verbindung mit einer Datenbank, um das Erzeugen und Abrufen von Ergebnissen auszuführen. Die Klassen von ADO.NET befinden sich in der Datei System.Data.dll und sind in die XML-Klassen integriert, die sich in der Datei Sytem.XML.dll befinden.

Der Namensraum System.Data besteht aus zwei grundlegenden Komponenten:

- Den verwalteten Anbietern (Managed Providers), die die Verbindung zu einer Datenquelle ermöglichen. Diese Anbieter führen die entsprechenden Befehle auf der Datenquelle aus und lesen Daten aus dem Speicher oder schreiben sie in ein DataSet.
- Dem DataSet und seinen unterstützenden Klassen, mit denen Daten verbindungslos manipuliert werden können.

4.4.2. ADO.NET Architektur

Die traditionelle Datenverarbeitung basierte grundlegend auf einem verbindungsbasierten Modell mit zwei Ebenen. Auf Grund der Verwendung von immer mehr Architekturen mit mehreren Ebenen für die Datenverarbeitung wechseln Programmierer zu einer nicht verbundenen Lösung, um eine bessere Skalierbarkeit für die Anwendungen zu erreichen.

4.4.3. ADO.NET Komponenten

Die Komponenten von ADO.NET wurden konzipiert, um den Datenzugriff von der Datenbearbeitung zu trennen. ADO.NET umfasst zwei Hauptkomponenten:

Das ADO.NET-DataSet ist die Hauptkomponente der nicht verbundenen Architektur von ADO.NET. Das DataSet wurde explizit für den Datenzugriff unabhängig von Datenquellen entwickelt. Deswegen kann es mit mehreren, unterschiedlichen Datenquellen, mit XML-Daten oder zum Verwalten von lokalen Anwendungsdaten verwendet werden. Das DataSet enthält eine oder mehrere DataTable-Objekte, die aus Datenzeilen und –spalten sowie aus Primärschlüsseln, Fremdschlüsseln, Einschränkungen und Beziehungsinformationen über Daten in dem DataTable-Objekten bestehen.

Das zweite Hauptelement ist der .NET Framework-Datenprovider, der für die Erstellung einer Verbindung mit einer Datenbank, zur Ausführung von Befehlen und zum Abrufen von Ergebnissen, verwendet wird. .NET Framework verfügt über den .NET Framework-Datenprovider für SQL Server, OLE DB und ODBC. Das folgende Diagramm zeigt die Komponenten der ADO.NET Architektur.

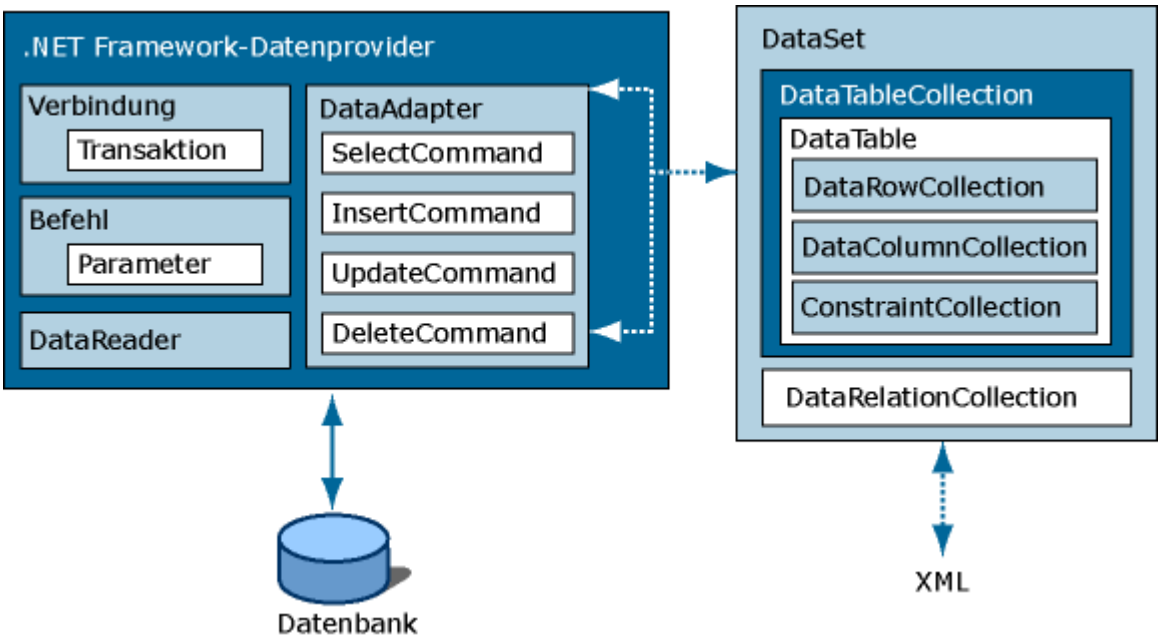

Abbildung 1.: ADO.NET-Architektur

Für das Projekt stand ein Microsoft SQL Server 2000 zur Verfügung. Deshalb wird der .NET Framework-Datenprovider für SQL Server genutzt, der ein eigenes Protokoll für die Kommunikation mit SQL Servern verwendet.

Dieses Protokoll ist kompakt und funktioniert sehr gut, da es für den direkten Zugriff auf einen SQL Server optimiert wurde, so dass das Hinzufügen einer OLE DB- oder ODBC-Schicht (Open Database Connectivity) nicht erforderlich ist.
Um auf die Klassen der .NET Framework-Datenprovider für SQL Server zugreifen zu können, muss der System.Data.SqlClient Namensraum in die Anwendung importiert werden. Dazu muss die Angabe Imports System.Data.SqlClient eingefügt werden.

4.4.4. Die wichtigsten Klassen des .NET Framework-Datenproviders

Zu den wichtigsten Klassen des .NET Framework-Datenprovider für SQL Server gehören:

Klasse SqlConnection

Diese Klasse sorgt für eine offene Verbindung zu einem SQL Server. Bei einem Client-/Server-Datenbanksystem entspricht dies einer Netzwerkverbindung mit dem Server. Diese wird durch einen Connection-String hergestellt, der den Namen des Datenbankservers, den Datenbanknamen, sowie Informationen zur Authentifizierung enthält. Zur Verbesserung der Leistung beim Verbinden mit einer Microsoft SQL Server-Datenbank wird SqlConnection in Verbindung mit SqlDataAdapter und SqlCommand verwendet.

Ein Beispiel zum Aufbau einer Datenbank-Verbindung könnte folgendermaßen aussehen:

```
Dim Connection as String = "Server=Dbserver; Database=druckertest; UserID=student;
                            Password=student"
Dim verbindung as New SqlClient.SqlConnection(Connection)

'Verbindung öffnen
  verbindung.Open()

   'Datenbankoperationen durchführen

'Verbindung schließen
  verbindung.Close()
```

Klasse SqlCommand

Mit dieser Klasse können Zugriffe auf Datenbankbefehle ausgeführt werden, um Daten zurückzugeben, zu ändern oder gespeicherte Prozeduren durchzuführen, sowie Parameterinformationen zu senden oder abzurufen. Dafür enthält diese Klasse die folgenden Methoden für das Ausführen von Befehlen in einer SQL Server-Datenbank:

- ExecuteNonQuery:

  Führt Befehle aus, die keine Ergebnisse zurückgeben. Hierbei geht es um Insert- oder Update-Anweisungen, bzw. um die Datenmanipulation bei gespeicherten Prozeduren. Diese Methode gibt eine ganze Zahl zurück, die der Anzahl der Zeilen entspricht, auf die sich die ausgeführte Anweisung oder gespeicherte Prozedur ausgewirkt hat.

  Bei der Ausführung mehrerer Anweisungen entspricht der zurückgegebene Wert der Summe der Datensätze, die von allen ausgeführten Anweisungen beeinflusst wurden.

  Ein Beispiel der ExecuteNonQuery Methode:

```
Dim mycommand As SqlCommand = verbindung.CreateCommand()
mycommand = New SqlClient.SqlCommand("INSERT INTO drucker (Druckername)
                                      VALUES ('Drucker1') ", verbindung)
mycommand.ExecuteNonQuery()
```

- ExecuteScalar:

  Diese Methode gibt nur die erste Spalte der ersten Zeile des Datensatzes als Objekt zurück. Alle anderen Daten bleiben unbeachtet. Die ExecuteScalar Methode wird häufig für Select-Anweisungen mit Aggregatfunktionen wie Count( ), Sum( ) oder Avg( ) verwendet.

  Beispiel:

```
Dim mycommand as SqlCommand = verbindung.CreateCommand()
mycommand = New SqlClient.SqlCommand("SELECT COUNT(Drucker_ID) FROM drucker",
                                      verbindung)
Dim Anzahl as Integer
Anzahl = mycommand.ExecuteScalar
```

- ExecuteReader:

  Wenn die Ergebnisse als Datenstream zurückgegeben werden, wird die ExecuteReader-Methode zur Rückgabe eines DataReader-Objekts verwendet. Diese Methode wird im Wesentlichen bei der Datenbank für normale Select-Anfragen benutzt.

  Folgendes Codebeispiel zeigt die Verwendung der ExecuteReader Methode:

```
Dim mycommand as SqlCommand = verbindung.CreateCommand()
mycommand = New SqlClient.SqlCommand("SELECT Name from drucker", verbindung)
Dim myreader as SqlDataReader = mycommand.ExecuteReader()
```

Klasse SqlDataReader

Die Klasse SqlDataReader stellt ein Verfahren zum Lesen eines Vorwärtsstreams von Zeilen aus einer Datenbank zur Verfügung. Die Ergebnisse werden bereits während der Ausführung der Abfrage zurückgegeben und im Netzwerkpuffer auf dem Client gespeichert, bis sie mit der Read-Methode der SqlDataReader Klasse abgefragt werden. Mit der Ausführung des DataReader-Objekts kann die Leistung der Anwendung erhöht werden.

Zum Einen werden Daten abgerufen, sobald sie verfügbar sind. Dadurch muss man nicht warten, bis die vollständigen Ergebnissen der Abfrage zurückgegeben werden.

Zum Anderen wird die Belastung des Systems reduziert, da immer nur eine Zeile gleichzeitig gespeichert wird. Die beste Leistung wird erreicht, wenn die vom DataReader-Objekt bereitgestellten Methoden verwendet werden, mit denen auf Spaltenwerte in den systemeigenen Datentypen zugegriffen werden kann (GetDateTime, GetDouble, GetString, usw.). Nach der Verwendung des DataReader-Objekts soll die Close-Methode aufgerufen werden, da während ein DataReader-Objekt geöffnet ist, die Datenbankverbindung exklusiv von diesem DataReader-Objekt verwendet wird. Befehle können erst dann für das Connection-Objekt durchgeführt werden, wenn das DataReader Ausgangobjekt geschlossen ist.

Ein Beispiel für ein DataReader-Objekts ist:

```
'Kommando ausführen
 Dim catCMD As SqlCommand = New SqlCommand("SELECT Druckername FROM main",
                                           nwindConn)
'Verbindung öffnen
 nwindConn.Open()

'Daten wiedererlangen
 Dim myReader As SqlDataReader = catCMD.ExecuteReader()
 Do While myReader.Read()
     Console.Writeline(vbTab & "{0}", myReader.GetString(0))
 Loop

'DataReader schließen
 myReader.Close()

'Verbindung schließen
 nwindConn.Close()
```

Klasse SqlAdapter

Diese Klasse stellt Datensätze von SQL Servern für ein DataSet-Objekt bereit. Es erlaubt das Auswählen, Schreiben, Aktualisieren und Löschen von Daten im DataSet. Das ADO.Net-DataSet ist eine speicherresidente Darstellung von Daten, die ein konsistentes relationales und von der Datenquelle unabhängiges Programmiermodell zur Verfügung stellt.

Ein DataSet stellt komplette Datengruppen einschließlich Tabellen, Einschränkungen und Beziehungen zwischen Tabellen dar. Über das DataSet werden später genauere Informationen gegeben. Ein DataAdapter-Objekt wird verwendet, um Daten aus einer Datenquelle abzurufen und Tabellen in ein DataSet zu füllen. Der DataAdapter nutzt das Connection-Objekt des .NET Framework-Datenproviders, um eine Verbindung zu einer Datenquelle herzustellen.

Mit den Command-Objekten werden Daten aus der Datenquelle abgerufen und Änderungen an die Datenquelle zurückgegeben.

Das folgende Codebeispiel zeigt, wie ein DataSet mit einem DataAdapter gefüllt wird:

```
'Verbindung durch den String herstellen
 Dim Connection as String = "Server=Dbserver; Database=druckertest; UserID=student;
                            Password=student"
 Dim verbindung as New SqlClient.SqlConnection(Connection)

'Befehl ausführen
 Dim mycommand as SqlCommand = verbindung.CreateCommand()
 mycommand = New SqlClient.SqlCommand("SELECT druckername from drucker",
                                      verbindung)

'Verbindung öffnen
 verbindung.Open()

'DataAdapter erstellen
 Dim myAdapter as SqlDataAdapter
 myAdapter = New SqlDataAdapter(mycommand, verbindung)

'DataSet erstellen
 Dim myDataSet as New DataSet

'DataSet durch den DataAdapter füllen
 myAdapter.Fill(myDataSet, "Drucker")

'Verbindung schließen
 verbindung.Close()
```

Klasse SqlParameter

Diese Klasse wird zur Übergabe von Parametern an SqlCommand-Objekte verwendet. Jeder Parameter hat grundsätzlich die folgenden Eigenschaften: Die Erste ist sein Name, dann sein Datentyp. Dieser wird durch die Eigenschaft DbType abgerufen oder festgelegt. Mit der Angabe Direction kann man bestimmen, ob es sich um einen In-, Out-, oder In/Out-Parameter handelt. Wenn die Richtung des Parameters nicht angegeben wird, wird diese als Input-Parameter interpretiert. Zum Schluss wird der Wert des Parameters durch die Angabe Value abgerufen oder festgelegt.

Ein Beispiel zur Verwendung dieser Klasse könnte folgendermaßen aussehen:

```
'Befehl ausführen
 Dim mycommand as SqlCommand = verbindung.CreateCommand()
 mycommand = New SqlClient.SqlCommand("INSERT INTO drucker (druckername) VALUES
                                      (@name)", verbindung)

'Parameter übergeben
 mycommand.Parameters.Add("@name", SqlDbType.VarChar, 30)
 mycommand.Parameters("@Druckername").Direction = ParameterDirection.Input
 mycommand.Parameters("@Druckername").Value = "Drucker1"
 mycommand.ExecuteNonQuery()
```

Klasse SqlTransaction

Mit dieser Klasse können (über ADO.NET) Transaktionen gesteuert werden, die in einer SQL Server-Datenbank ausgeführt werden sollen. Durch die BeginTransaction-Methode des SqlConnection-Objekts lässt sich eine lokale Transaktion initialisieren.

Nach Beginn einer Transaktion ist es möglich mit der Transaction-Eigenschaft des Command-Objekts einen Befehl in diese Transaktion einzutragen. Anschließend, wenn das Programm fehlerfrei abgelaufen ist, können die Änderungen durch einen Commit-Befehl gespeichert werden. Tritt ein Fehler auf, dann lassen sich alle Aktualisierungen auf den Ausgangsstatus vor Beginn der Transaktion durch den Rollback-Befehl zurücksetzen.

Das folgende Codebeispiel demonstriert die Transaktionslogik unter Verwendung von ADO.NET mit Microsoft SQL Server:

```
'Verbindung durch den String herstellen
 Dim Connection as String = "Server=Dbserver; Database=druckertest; UserID=student;
                            Password=student"
 Dim verbindung as New SqlClient.SqlConnection(Connection)

'Verbindung öffnen
 verbindung.Open()

'Definiert eine Transaktion von Transact-SQL, die in der Datenbank durchgeführt
'wird
 Dim mytrans As SqlTransaction

'Start der Transaktion
 mytrans = verbindung.BeginTransaction()
 mycommand.Connection = verbindung

'Befehl ausführen
 Dim mycommand as SqlCommand = verbindung.CreateCommand()
 mycommand = New SqlClient.SqlCommand("INSERT INTO drucker (Druckername) VALUES
                                       ('Drucker1') ", verbindung)

'Die Anweisung der Transaktion hinzufügen
 mycommand.Transaction = mytrans

 Try
   'Versuch den Befehl auszuführen
    mycommand.ExecuteNonQuery()

   'Bestätigung der Transaktion der Datenbank
    mytrans.Commit()
    Console.Writeline("Die Daten sind in der Datenbank gespeichert")

 Catch e As Exception
   'Wenn ein Fehler aufgetreten wird
    mytrans.Rollback()
 End Try

'Verbindung schließen
 verbindung.Close()
```

Mit der Verwendung dieser Klassen wird in diesem Projekt die komplette Datenbankkommunikation durchgeführt. Die Klasse SqlConnection zum Beispiel wurde sowohl vom Inventar-Dienst als auch vom Web-Dienst für die Erstellung der Datenbankverbindung verwendet. Die Parameterinformation wurde mit Hilfe der Klasse SqlParameter an SqlCommand-Objekte übergegeben.

Mit dieser Information konnten im Inventar-Dienst die verschiedenen Abfragen und Datenbankoperationen durch die von der Klasse SqlCommand bereitgestellte Befehle ausgeführt werden. Die Transaktionen wurden im Inventar-Dienst durch die Verwendung der Klasse SqlTransaction ausgeführt. Zur Visualisierung der Ergebnisse im Web-Dienst wurde die Klasse SqlDataAdapter in Kombination mit DataSets verwendet.

4.5. Das DataSet

Ein DataSet ist ein verbindungsloser Schreib-/Lese-Container, der eine oder mehrere Datentabellen und ihre Relationen enthalten kann. Die DataTables, DataRelation und Constraints in einem DataSet können programmgesteuert erstellt und die Tabellen mit Daten gefüllt werden. Das DataSet ermöglicht automatisch verbindungslosen Datenzugriff. Jede Tabelle in einem DataSet entspricht einem DataTable-Objekt, das eine Sammlung von DataRow-Objekten und DataColumn-Objekten enthält. Es gibt auch Sammlungen für die in diesen Tabellen verwendeten Primärschlüssel, Constraints und Voreinstellungen sowie die Parent/Child Beziehungen zwischen den Tabellen. Schließlich gibt es für jede Tabelle ein DefaultView-Objekt, das für die Erzeugung eines basierenden DataView-Objekts auf der Tabelle verwendet wird. Damit können die Daten durchsucht, gefiltert, anderweitig manipuliert oder zur Darstellung an ein Steuerelement gebunden werden.

Im folgenden Beispiel wird ein DataSet erstellt, ein neues DataTable-Objekt zum DataSet hinzugefügt und anschließend drei DataColumn-Objekte in die Tabelle eingefügt. Schließlich wird eine Spalte für den Primärschlüssel festgelegt:

```
'DataSet erstellen
 Dim druckDataSet As DataSet = New DataSet("Drucker")

'Tabelle hinzufügen
 Dim mainTable As DataTable = druckDataSet.Tables.Add("Daten")

'Spalten einfügen
 Dim pkCol As DataColumn = mainTable.Columns.Add("DruckerID",
                           Type.GetType("System.Int32"))

 mainTable.Columns.Add("Name", Type.GetType("System.String"))
 mainTable.Columns.Add("Hersteller", Type.GetType("System.String"))

'Spalte als Primärschlüssel festlegen
 mainTable.PrimaryKey = New DataColumn() {pkCol}
```

Jetzt können Daten zur Tabelle hinzugefügt werden:

```
 Dim date1 as DataRow
 date1 = druckDataSet.Tables("Daten").NewRow
 date1 = ("DruckerID") = 12
 date1 = ("Name") = "Drucker1"
 date1 = ("Hersteller") = "Epson"
```

Ein DataSet kann mit mehreren, unterschiedlichen Datenquellen, wie XML-Daten oder zur Verwaltung lokaler Anwendungsdaten verwendet werden. Um Daten aus einer Datenquelle abzurufen und Tabellen in einem DataSet zu füllen, wird ein DataAdapter-Objekt verwendet. Das ist praktischer als ein DataSet selbst zu definieren und zu füllen. Das DataAdapter-Objekt löst außerdem Änderungen, die am DataSet vorgenommen wurden, in der Datenquelle auf. Dieses benutzt das Connection-Objekt des .NET Framework Datenproviders, um eine Verbindung zu einer Datenquelle herzustellen. Mit den Command-Objekten werden Daten aus der Datenquellen abgerufen und Änderungen an die Datenquelle zurückgegeben.

Die SelectCommand-Eigenschaft des DataAdapter-Objekts ist ein Command-Objekt, das Daten aus der Datenquelle abruft. Mit der Fill-Methode des DataAdapter-Objekts wird ein DataSet mit den Ergebnissen der SelectCommand-Eigenschaft gefüllt. Ein Beispiel zum Befüllen eines DataSets durch diese Methode könnte folgendermaßen aussehen:

```
'Verbindung durch den String herstellen
 Dim Connection as String = "Server=Dbserver; Database=druckertest; UserID=student;
                            Password=student"
 Dim verbindung as New SqlClient.SqlConnection(Connection)

'Befehl ausführen
 Dim mycommand as SqlCommand = verbindung.CreateCommand()
 mycommand = New SqlClient.SqlCommand("SELECT Herstellername from main",
                                      verbindung)

'Verbindung öffnen
 verbindung.Open()

'DataAdapter definieren
 Dim myAdapter as SqlDataAdapter
 myAdapter = New SqlDataAdapter(mycommand, verbindung)

'DataSet erstellen
 Dim myDataSet as New DataSet

'DataSet durch den DataAdapter füllen
 myAdapter.Fill(myDataSet, "Hersteller")

'Verbindung schließen
 verbindung.Close()
```

Zur Übernahme von Änderungen in die Datenquelle aus einem DataSet wird die Update-Methode des DataAdapter-Objekts aufgerufen. Als Argumente verwendet diese Methode eine Instanz vom DataSet, die die vorgenommenen Änderungen, ein optionales DataTable-Objekt oder einen DataTable-Namen enthält. Mit DataTable wird die Tabelle identifiziert, aus der die Änderungen abgerufen werden. Wenn die Update-Methode aufgerufen wird, wird das DataAdapter-Objekt die vorgenommenen Änderungen verarbeiten und den entsprechenden Befehl (INSERT, UPDATE oder DELETE) ausführen.

Bei der Verwendung des DataSets lässt sich beobachten, dass die Daten einfacher und ohne Verbindung zum Datenspeicher bearbeitet werden können. Anschließend wird die Verbindung wiederhergestellt und die eventuellen Änderungen können in den Datenspeicher übernommen werden.

4.6. ASP.NET

Bevor näher auf das Thema ASP.NET eingegangen wird, wird nachfolgend das Konzept von dynamischen Webseiten kurz erläutert.

4.6.1. Dynamischen Webseiten

Mit der steigenden Verbreitung und damit größeren Bedeutung des Internets haben sich auch die Anforderungen verändert. In vielen Fällen ist es nicht mehr genug, eine statische Datei auszuliefern. Unterschiedliche Browsersoftware, personalisierte Seiten, Inventarsysteme, die auf eine Datenbank gestützt sind, und Internetseiten mit sich ständig ändernden, aktuellen Inhalten müssen technisch umgesetzt werden. Diese gestiegenen Anforderungen führen zur Erstellung der dynamischen Webseiten.

Dynamische Webseiten können client- oder serverseitig realisiert werden. In einer clientseitigen Ausführung wird immer noch eine sich nicht verändernde Datei vom Server ausgeliefert, deren Inhalt erst auf dem Client durch aktive Bestandteile wie z.B. Skripte nachträglich modifiziert wird. Die Skripte, die in der Regel in Sprachen wie Java, JavaScript oder VBScript geschrieben werden, laufen dabei komplett im Browser des Users ab. Im Gegensatz zur clientseitigen Ausführung wird in der serverseitigen Ausführung die angeforderte Datei nicht sofort an den Client ausgeliefert, da sie erst durch ein Programm verändert oder sogar bei der Anforderung durch ein Programm erzeugt werden kann. Der anzuzeigende Inhalt kann dabei auch aus einer Datenbank kommen, wie z.B. im Rahmen dieses Projekts, bei dem die Daten aus der Drucker-Datenbank geladen werden. Eine URL bezeichnet also nicht mehr unbedingt eine konkrete Datei, sondern vielmehr eine vom Server bereitzustellende Informationseinheit. Sprachen für die serverseitige Generierung von dynamischen Webseiten sind beispielsweise CGI, PHP, ASP, sowie die neue Architektur ASP.NET.

4.6.2. Überblick über ASP.NET

ASP.NET ist eine neue Architektur, die zwei neue Programmiermodelle beinhaltet, sowie zahlreiche neue Dienste, die auf dem .NET Framework basieren.
Diese neuen Programmiermodelle sind Webformulare und Web Services. Anwendungen können in einer beliebigen .NET-kompatiblen Sprache geschrieben werden, einschließlich Visual Basic.NET, C# und JScript.NET. Gleichzeitig befinden sich hier die Vorteile der .NET Technologie wie die Sprachinteroperabilität, dem CLR (Common Language Runtime)-Dienst sowie starke Typisierung, Typsicherheit, frühe Bindung und Vererbung.

ASP.NET besteht aus einer Reihe verwalteter Klassen, die im Namensraum System.Web enthalten sind. Diese stellt folgende Dienste bereit:

- Konfigurationsdateien, die auf XML basieren. Jede Website kann eine Konfigurationsdatei enthalten, mit der bestimmte Merkmale wie die Authentifizierung und das Zustandsmanagement gesteuert werden können.
- Zustandsverwaltung auf Sitzungs- und Anwendungsebene, die über Servergrenzen hinweg in Webformen eingesetzt werden können.
- Caching-Engine, welche zu einer Verbesserung der Leistungsfähigkeit und besserer Skalierbarkeit führt.
- Durch die Tracing-Funktion können die Leistung des Codes überprüft und entsprechende Nachrichten definiert werden.
- Sicherheit: ASP.NET stellt Authentifizierungsanbieter, Authentifizierungssemantiken und Impersonationsmechanismen zur Verfügung, die zusammengenommen eine flexible und erweiterbare Sicherheitsarchitektur bilden.
- Erweiterbarkeit: Entwicklern können z.B. ihre eigenen verwalteten Klassen schreiben, um bestimmte Ressourcen zu verarbeiten, oder auch eigene Module integrieren. Damit können Entwickler erweiterte Funktionen schreiben, die zuvor eine Domäne der C++ Entwickler war.

4.6.3. Die Architektur von ASP.NET

Grundlegend besteht eine ASP.NET-Anwendung aus einer Anzahl von Dateien, die sich in einem virtuellen Verzeichnis und darin enthaltenen Unterverzeichnissen auf einem IIS (Internet Information Server) Webserver befinden. Die ASP.NET-Anwendung wird zusammen mit einer Anwendungsdomäne kompiliert und läuft in dieser.

Alle ASP.NET-Seiten der Anwendung werden demnach in einem Assembly kompiliert, das im Verzeichnis *bin* innerhalb des virtuellen Verzeichnisses abgelegt wird.

Die ASP.NET-Laufzeit vermittelt zwischen den Anforderungen des Clients und der eigentlichen Seitenverarbeitung. Ein von einem Client an eine ASP.NET-Datei (ASPX, ASCX oder ASMX) gerichteter Aufruf wird vom IIS mit der registrierten ISAPI-Erweiterung bearbeitet. Zum Schluss werden die benötigten Assemblies geladen.
Die Laufzeit von ASP.NET verarbeitet im nächsten Schritt die Anfrage und verwendet dabei die eigenen Konfigurationsinformationen, um vorbereitende Maßnahmen zu treffen. Wenn diese Maßnahmen abgeschlossen sind, werden die ASP.NET-Klassen innerhalb des Assembly instanziiert und Methoden aufgerufen. Diese erzeugen letztendlich den HTML-Code, der zurück an den Client geschickt wird.

Eine Aufgabe der ASP.NET-Laufzeit ist die Anwendung von Konfigurationsinformationen, die aus der XML-Konfigurationsdatei auf Computer- und Verzeichnisebene geladen und analysiert werden. Jedes ASP.NET-Verzeichnis kann eine eigene Web.config-Datei enthalten. In dieser Datei sind alle Einstellungen festgelegt, anhand derer die Laufzeit entscheidet, wie der Prozess verarbeitet wird. Jede ASP.NET-Anwendung kann optional die Datei Global.asax enthalten. Diese Datei verarbeitet Ausnahmefehler, die von der Anwendung produziert werden. Die Datei Global.asax wird in ASP.NET dynamisch geparst und in einer Klasse kompiliert, die sich von der Klasse HttpApplication des Namensraumes System.Web ableitet. Wenn diese Datei nicht existiert, wird eine generische HttpApplication-Klasse erstellt. ASP.NET unterstützt neben den HttpApplication-Instanzen auch mehrere Module (HTTP-Module), die an der Anwendung beteiligt sein können. Durch die Ableitung einer Klasse von HttpModule und ihre Registrierung in Web.config können eigenen Module erzeugt werden, die zum Beispiel Filterung, Authentifizierung und einige andere Funktionen unterstützen können.
Abschließend werden bestimmte Dateitypen von HTTP-Handlern nacheinander abgearbeitet. Diese verwalteten Klassen leiten sich von HttpHandler ab und verarbeiten bestimmte Ressourcen. Wie bereits erwähnt wurde, unterstütz ASP.NET zwei Programmiermodelle: Webformulare und Web Services, die im Folgenden kurz erklärt werden.

4.6.4. Webformulare

Die Webformulare bieten einen Entwicklungsansatz für Webanwendungen, der auf Formularen basiert. Die Architektur verwendet einen seitenverarbeitenden HTTP-Handler, der verwaltete Klassen aufruft. Diese Klassen leiten sich vom Namensraum System.Web.UI.Control ab.
Die Klassen, die die Webseiten repräsentieren, werden zusammen mit dem statischen HTML der Webformulare zum Assembly der Anwendung kompiliert. Dieses Programmiermodell hat folgende Vorteile:

- Trennung der Benutzeroberfläche von der Anwendungslogik: Visual Studio.NET trennt den HTML-Code von der Klassendatei, welche die Logik enthält, in einer separaten Datei (ASPX) ab. Diese Anweisung wird als Code-Behind oder kurz CBF bezeichnet.
  Aus Entwicklersicht stellt dieses neue CBF den größten Unterschied zwischen klassischem ASP und ASP.NET dar.
- Kompilierter Code: Da das Assembly sowohl die Benutzeroberfläche als auch den Code implementiert, werden die beiden zusammen kompiliert, was zu einer deutlichen Leistungssteigerung führt.
- Browser-Unabhängigkeit: Da die Elemente der Oberfläche in Controls abstrahiert werden können, kann das erzeugte HTML je nach verwendetem Browser dynamisch angepasst werden.

4.6.5. Web Services

Die Web Services werden noch ausführlich in Kapitel 4.7 beschrieben.
Kurz zu ihrer Funktion: Web Services ermöglichen den remoten Zugriff auf Serverfunktionen. Dieses Modell basiert auf die Standardprotokollen wie HTTP und SOAP.

Die verbesserte Skalierbarkeit und Leistung von ASP.NET, sowie die neuen Web Services erreichen die Basis für eine Welt verteilter Anwendungen, die aus einer Vielzahl privater und öffentlich verfügbarer Web Services bestehen werden.

4.7. Verteilte Anwendungen

Bei genauerer Betrachtung des .NET Frameworks sollte schnell deutlich werden, dass Microsoft eine bestimmte Strategie in die Richtung von verteilten Anwendungen verfolgt. So kommt auch die Namensgebung nicht von ungefähr, insbesondere soll jetzt auch das Internet zur Verbindung dieser Anwendungen dienen.

Sind klassische Anwendungen in der Regel nur auf ein einziges System beschränkt, so können sich verteilte Anwendungen auf mehrere miteinander vernetzte Systeme verteilen. Prinzipiell ist dies auch nichts Neues. Über diverse proprietäre Schnittstellen konnten verteilte Anwendungen ihre Daten schon lange on-the-fly zwischen verschiedenen Systemen austauschen. Architekturen wie das Distributed Component Object Model (DCOM) und Remote Procedure Calls (RPC) sollen die Abkehr von diesen Individuallösungen ermöglichen. Diese Techniken erlauben das Übertragen von Parametern und das Ausführen von Anfragen auf entfernte Systeme, die von diesen Parametern beeinflusst werden. Das Ergebnis einer solchen Anfrage wird schließlich auch wieder zurückübertragen.
Einige Nachteile haben diese Architekturen allerdings auch. DCOM- und RPC-Aufrufe werden üblicherweise von Firewalls blockiert, um eine unkontrollierte Manipulation von Servern und Daten durch eben diese Aufrufe von außen zu verhindern. Andererseits müssen die beteiligten Systeme binärkompatibel zueinander sein, so ist man bei DCOM auf die Windows-Plattform beschränkt. Diese Nachteile gelten auch für Techniken, die auf diesen aufbauen, wie z.B. das zur Maschinen- und Betriebsdatenerfassung oft eingesetzte OPC (OLE for Process Control). Außerdem ist die Verwendung und Einrichtung dieser Architekturen nicht gerade trivial.

4.7.1. Web Services allgemein

Die bisher bestehenden Nachteile sollen nun mithilfe von Web Services, oft auch XML Web Services genannt, behoben werden. Durch die Verwendung von Standards werden proprietäre Schnittstellen und Protokolle umgangen. Web Services sind letztlich programmierbare Softwarekomponenten, auf die über Standard-Webprotokolle zugegriffen werden kann. Das Konzept eines Web Service ähnelt einer Art Black-Box, die ihre Funktionen nach außen hin zur Verfügung stellt. Ein fertiger Web Service kann von Softwareentwicklern wie ein Baustein in eigene Anwendungen eingebaut werden.

Zur Kommunikation zwischen dem Server, der den Web Service zur Verfügung stellt, und den Clients, die diesen Dienst konsumieren, wird das HTTP Protokoll (Port 80) verwendet. Daher ist es in der Regel nicht nötig Firewalls speziell zu konfigurieren, diese werden von den HTTP Anfragen ebenso problemlos passiert wie zwischengeschaltete Web Proxies.

4.7.2. SOAP

Der oft verwendete Ausdruck XML Web Services, deutet auf das Kommunikationsformat hin, welches im Zusammenhang mit Web Services verwendet wird. Zur Datenübertragung wird ein allgemeiner Standard in Form des Simple Object Access Protocol (SOAP) verwendet, ein anerkannter XML Dialekt. Wie HTTP ist auch SOAP plattformübergreifend verfügbar. Mittels SOAP ist es möglich beliebig komplexe Daten im XML Format über HTTP zu übertragen. Komplexe Datentypen können über festgelegte Regeln auf Standarddatentypen zurückgeführt werden. Außerdem wird bei jeder Übertragung eine Beschreibung der verwendeten Datenstruktur mitübertragen, wodurch die Informationen immer eindeutig interpretiert werden können.

4.7.3. WSDL

Mittels SOAP ist nun zwar die die Darstellung und Kodierung der Datenkommunikation zwischen Client und Web Service geregelt, um einen Web Service aber nutzen zu können benötigt man eine Beschreibung über den Dienst, über seine Methoden und die möglichen Parameter. Dies geschieht ebenfalls mit einem XML Dialekt in Form der Web Service Description Language (WSDL), die die Schnittstelle des Web Service beschreibt und dem Client zur Verfügung stellt.

Üblicherweise wird zunächst ein Web Service implementiert und anschließend eine WSDL Datei generiert, die diesen Dienst beschreibt. Dies geschieht in der Regel mithilfe von Tools, die meist mit der Entwicklungsumgebung, z.B. Visual Studio .NET, mitgeliefert werden. Der IIS ist sogar in der Lage die WSDL Datei eines gehosteten ASP.NET Web Service dynamisch auf Anfrage zu generieren.

4.7.4. ASP.NET Web Services

Sofern man sich an die vorgegebenen Standards hält, lassen sich Web Services prinzipiell auf beliebigen Plattformen entwickeln. In Microsoft Umgebungen werden sie in der Regel mithilfe von ASP.NET erstellt und von dem IIS gehosted. Die Arbeitsweise eines Web Service unterscheidet sich von der Arbeitsweise einer aktiven Webseite vor allem auch nur in dem Ausgabeformat. Während aktive ASP.NET Webseiten (*.aspx) HTML Dokumente auf Anfrage zurückgeben, geben ASP.NET Web Services (*.asmx) SOAP Nachrichten zurück, die als Funktionen konsumiert werden können. Microsoft unterstützt die Entwicklung von Web Services mit ihrem .NET Framework besonders durch Bereitstellung aller wichtigen Methoden, die das Erstellen und Konsumieren enorm vereinfachen. So braucht sich ein Entwickler niemals selbst um die SOAP Kommunikation kümmern und auch WSDL Dateien werden, wie bereits erwähnt, automatisch erstellt.

Trotz allem haben aber auch Webservices einige Nachteile. Da die Kommunikation wie bei normalen Webseiten über HTTP läuft, ist auch hier eine serverseitig initiierte Kommunikation nicht vorgesehen. Außerdem sind Web Services für performancekritische Anwendungen eher ungeeignet, da erstens HTTP nicht auf besonders schnelle Paketzustellungen optimiert ist und zweitens SOAP Nachrichten neben den eigentlichen Nutzdaten einen enormen Overhead an Daten über die Datenstruktur und Formatierungsdaten enthalten.

4.7.5. .NET Remoting

Das .NET Remoting ist eine weitere Architektur, die die Entwicklung von verteilten Anwendungen erlaubt. Im Gegensatz zu Web Services ist .NET Remoting nicht auf Plattformunabhängigkeit und dem Überwinden von Netzwerkgrenzen ausgelegt, sondern auf homogene .NET Umgebungen beschränkt. In diesen Umgebungen läuft es daher auch wesentlich performanter ab.

Das .NET Remoting ist relativ flexibel, was die Kommunikationsmöglichkeiten betrifft, so stehen als Protokolle wahlweise HTTP, SMTP oder TCP zur Verfügung. Die Daten werden entweder SOAP-formatiert oder im Binärformat übertragen. In der Regel wird jedoch das Binärformat über TCP verwendet, das die performanteste Kombination darstellt.

Ein Vorteil neben der erhöhten Geschwindigkeit ist die Möglichkeit Rückrufe und Ereignisse implementieren zu können. Sobald ein Client bei einem Server registriert ist, kann auch eine serverinitiierte Kommunikation stattfinden, was aber nur bei Verwendung von TCP möglich ist. Bei Verwendung von HTTP ist man jedoch wie bei Web Services auf clientinitiierte Kommunikation beschränkt.

Ein anderer Vorteil ist auch, dass eine Anwendung selber als Remoting Server arbeiten kann, ohne auf einem Web Server gehosted werden zu müssen, wie Web Services.

4.8. Windows-Dienst

Seit Windows NT und Windows 95 sind auch Microsofts Windows Betriebssysteme endlich multitaskingfähig, d.h. unter diesen Betriebssystemen können mehrere Tasks oder Prozesse gleichzeitig, parallel ausgeführt werden. Wurde dies anfänglich noch hauptsächlich dazu benutzt, um mehrere klassische Anwendungen gleichzeitig in verschiedenen Fenstern auszuführen, so werden jetzt zunehmend mehr Prozesse weitgehend unbemerkt vom Benutzer im Hintergrund ausgeführt um bestimmte Aufgaben zu erfüllen. Dieses sind die so genannten Windows-Dienste.

Ein Windows-Dienst ist eine Anwendung, die über keine eigene Benutzeroberfläche verfügt und unabhängig und unbemerkt vom Benutzer im Hintergrund läuft. Damit eignet er sich besonders für unbeaufsichtigte Server- und Wartungsaufgaben, die ausgeführt werden können ohne den Benutzer bei seiner eigentlichen Arbeit zu stören. Auch die Hauptaufgabe des Drucker-Inventarsystems, zu bestimmten Zeiten Netzwerkdrucker zu suchen, auszulesen und die gewonnenen Daten in einer Datenbank zu sichern, ist daher geradezu prädestiniert als Windows-Dienst entwickelt zu werden.

Eine weitere Eigenschaft von Windows-Diensten ist es, dass sie unter einem bestimmten Benutzerkonto ablaufen, das nicht mit dem des angemeldeten Benutzers übereinstimmen muss. Unter Windows 9x Betriebssystemen ist diese Eigenschaft uninteressant, da diese keine Multiuser Umgebung bereitstellen, d.h. alle Prozesse laufen im selben Sicherheitskontext ab, der systemweiten Vollzugriff erlaubt. Die auf Windows NT basierenden Betriebssysteme (also auch Windows 2000 und Windows XP) bieten jedoch eine echte Multiuser Umgebung. Ein Prozess arbeitet hier im Sicherheitskontext des jeweiligen Aufrufers. Es können auch mehrere verschiedene Prozesse unter verschiedenen Sicherheitskontexten gleichzeitig ausgeführt werden.
Systemeigene Prozesse laufen dabei unter einem besonderen Benutzerkonto, dem lokalen Systemaccount, der wie unter Windows 9x systemweiten Vollzugriff besitzt, jedoch keine Berechtigungen im lokalen Netzwerk besitzt. Für andere Benutzer können andere Rechte definiert werden unter denen dann die Benutzerprozesse von den Systemprozessen getrennt gleichzeitig ablaufen können. Dieses Vorgehen sorgt normalerweise für erhöhte Sicherheit, wenn die Benutzerkonten auch nur mit den für ihre Arbeit wirklich nötigen Rechten ausgestattet sind. Für den Drucker-Inventar-Dienst bedeutet dies, dass er unter einem Benutzerkonto ablaufen muss, das über ausreichende Rechte verfügt, um SNMP-Abfragen, .NET Remoting und Datenbankzugriffe über das lokale Netzwerk durchführen zu können. Der lokale Systemaccount kommt daher nicht in Frage.

Einen Windows-Dienst zu erstellen ist aber auch etwas komplizierter, als beispielsweise eine Konsolenanwendung. Da ein solcher Dienst unbemerkt im Hintergrund laufen soll, ist es schwierig seinen Zustand zu überwachen um mögliche Fehler aufzuspüren. Er kann leider auch nicht wie eine Konsolenanwendung im Debugmodus im Visual Studio .NET getestet werden. Des Weiteren sollte eine Installationsroutine miterstellt werden, die den Dienst installiert, beim Betriebssystem anmeldet und ihn dabei gleich einem Benutzerkonto zuweisen kann. Dies geht zwar auch per Hand mit der Kommandozeile, ist dann aber deutlich aufwendiger, wenn man den Dienst zu Testzwecken mehrmals installieren muss.

Zu Debugzwecken ist es durchaus ratsam einen Dienst zunächst als Konsolenanwendung zu implementieren, um seine fehlerfreie Funktion zu testen. Später kann dann darauf basierend ein Projekt für einen Windows-Dienst erstellt werden. Dieses geht besonders einfach, wenn man die eigentliche Funktionalität in Form einer Klassenbibliothek implementiert und diese dann im Dienst Projekt und zum Debuggen gleichzeitig auch in einem Konsolen Projekt importiert.

4.9. Multithreading

4.9.1. Nebenläufige Prozesse

Wie bereits im vorigen Abschnitt erwähnt sind moderne multitaskingfähige Betriebssysteme in der Lage mehrere Prozesse scheinbar gleichzeitig ablaufen zu lassen. Auf den gängigen Einprozessormaschinen ist dieses jedoch nur eine Quasiparallelität oder auch Nebenläufigkeit. Tatsächlich schaltet das Betriebssystem die Prozesse alle paar Millisekunden um. Dadurch wird keine wirkliche Parallelität erreicht, sondern nur durch verzahnte Bearbeitung der Prozesse simuliert. Auch auf Multiprozessormaschinen können nur so viele Prozesse wirklich parallel ablaufen, wie Prozessoren vorhanden sind, weitere Prozesse werden ebenfalls nebenläufig ausgeführt.

Der Teil des Betriebssystems, der die Prozessumschaltung übernimmt, heißt Scheduler. Auf Basis von Priorität und dem jeweiligen Steueralgorithmus weist der Scheduler jedem einzelnen Prozess einen Zeitraum zu, den so genannten *Time Slice*, den er laufen kann. Ist der Time Slice abgelaufen, wird der Prozess angehalten und eine Warteschlange gestellt, sein Zustand bzw. sein Kontext wird dabei gesichert, darauf kann ein anderer Prozess seine Arbeit (wieder-) aufnehmen. Zu beachten ist auch, dass die virtuelle Speicherverwaltung des Betriebssystems die Adressräume der einzelnen Prozesse voneinander trennt, wodurch kein Prozess den Speicherraum eines anderen Prozesses beeinflussen kann.

Auch wenn dadurch keine wirkliche Parallelisierung erreicht wird, ist diese Nebenläufigkeit von Prozessen meist von Vorteil. Ein gerne genanntes Beispiel sind Texteditoren, deren Haupttätigkeit es eigentlich ist auf Tastatureingaben des Benutzers zu warten. Ein heutiger Prozessor mit einigen Gigahertz Taktfrequenz ist dabei hoffnungslos unterfordert. Zwischen den einzelnen Tastendrücken könnte die CPU durchaus mehrere Millionen bis Milliarden Einzeloperationen durchführen. Würde der Texteditor nur als einziger Prozess ausgeführt werden können, wäre die Leistungsfähigkeit des Prozessors größtenteils vergeudet. Durch Nebenläufigkeit können andere Prozesse die CPU während der Wartezeiten des Texteditors verwenden, wodurch eine bessere Auslastung und somit auch eine höhere Gesamtleistung der vorhandenen Hardware erreicht werden kann.

4.9.2. Nebenläufige Threads

Wünschenswert ist es auch eine solche Nebenläufigkeit innerhalb eines einzigen Programms bzw. Prozesses zur Verfügung zu haben und nicht nur innerhalb des Betriebssystems. Die CLR des .NET Frameworks unterstützt dieses. Das Konzept beruht auf so genannten *Threads*[2]. Threads sind nebenläufige Aktivitäten, die sehr schnell in der Umschaltung sind. Innerhalb eines Prozesses kann es mehrere Threads geben, die alle zusammen im selben Adressraum ablaufen. Die einzelnen Threads können daher untereinander auf ihre öffentlichen Daten zugreifen. Wenn mehrere Threads wie Prozesse verzahnt ausgeführt werden, so dass sich für den Benutzer der Eindruck der Gleichzeitigkeit ergibt, spricht man von Multithreading.

4.9.3. Multithreading unter .NET

Bei der CLR wird jede Anwendung in einem Hauptthread gestartet, der mehrere untergeordnete Worker-Threads erzeugen kann, die jeweils ihre eigenen Ausnahmebehandlungen und Kontextdaten haben können. Die Unterstützung seitens des .NET Frameworks befindet sich im Namensraum System.Threading.

Die wichtigste Klasse zur Manipulation und Verwendung von Threads ist hier die Klasse Thread. Hier finden sich Methoden, mit denen man Threads starten, wieder aufnehmen, abbrechen, anhalten und auf einen anderen Thread warten lassen kann. Ebenso ist es möglich den Status eines Threads abzufragen und zu beeinflussen. Die meisten Methoden sind Objektmethoden, die nur von einer bestimmten Instanz der Thread Klasse genutzt werden können.
Um einen neuen Thread zu starten, muss noch ein Einstiegspunkt definiert werden, an dem er mit der Ausführung beginnt. Dieses ist üblicherweise eine Prozedur die weder Argumente noch Rückgabewerte haben darf. Sobald diese Prozedur abgearbeitet ist, endet in der Regel auch der Thread automatisch.

Eine andere Möglichkeit zur Threaderzeugung und -verwaltung bietet die Klasse ThreadPool. Thread-Pools werden von der CLR gesteuert und arbeiten weitgehend automatisch. Man kann ihnen zu erledigende Aufgaben als so genannte *Work Items* übergeben, die dann von Worker-Threads abgearbeitet werden. Besonders vorteilhaft ist auch, dass asynchrone Operationen und Ereignisse automatisch behandelt werden. Für wenige kurzlebige Threads können sich Thread-Pools sehr gut eignen.

[2] Engl. Thread – Faden oder Ausführungsstrang

Allerdings ist zum einen die Anzahl der Work Items auf 15 beschränkt und zum anderen sind die Möglichkeiten zur Kontrolle der laufenden Threads stark eingeschränkt. Es lässt sich nicht beeinflussen, ob die Work Items auch tatsächlich parallel von Worker-Threads abgearbeitet werden, da die CLR selbstständig versucht die Threadanzahl an die Leistungsfähigkeit des Rechners anzupassen, was nicht immer gelingt. Auch abgestürzte oder hängende Worker-Threads können nicht von außen beendet werden. Aus diesen Gründen wird wie auch beim Vorgängerprojekt auf die Verwendung von Thread-Pools verzichtet.

4.9.4. Gemeinsame Ressourcen

Am sichersten ist es, Threads zu verwenden, die auf keine gemeinsamen Ressourcen zugreifen. Diese laufen dann ähnlich wie Prozesse unbeeinflusst voneinander nebenher. Aber häufig ist gerade die Fähigkeit der Threads auf gemeinsame Ressourcen zugreifen zu können für ihre Aufgabe auch erforderlich. Hierbei ist aber darauf zu achten, dass diese Zugriffe synchronisiert sind. Geschieht dies nicht, so könnte beispielsweise ein Thread den Wert einer Variablen verändern, die gerade auch von einem anderen Thread verwendet wird, was leicht zu undefinierten Zuständen führen kann. Daher sollte für die Dauer des Zugriffes auf eine gemeinsame Ressource sichergestellt werden, dass der jeweilige Thread exklusiven Zugriff darauf hat und kein anderer Thread auf die Ressource zugreifen kann.

Die Klasse Monitor stellt hierbei ein häufig verwendetes Hilfsmittel zur Koordination dar. Ein Monitor ist in der Lage einen so genannten *kritischen Bereich* im Programmcode für ein übergebenes Objekt zu erzeugen. Solange dieser kritische Bereich nicht verlassen wird, kann nicht auf dieses Objekt zugegriffen werden, alle Zugriffsversuche werden in eine Warteschlange gestellt. Sobald der kritische Bereich verlassen werden die wartenden Threads in der Warteschlange benachrichtigt und können erneut versuchen auf das Objekt zuzugreifen.

4.10. XML Konfigurationsdatei

Bei der Planung dieses Projekts war schnell klar, dass einige konfigurierbare Werte auftreten müssen. Viele Rahmenbedingungen des Druckerinventarisierens waren zu Beginn einfach nicht bekannt oder sie müssen für den sinnvollen Einsatz des Systems gelegentlich geändert werden. Zu diesen Werten gehören z.B. der zu durchsuchende Netzwerkbereich, der Name des Datenbankservers oder die Zeitpunkte für automatische Aktualisierungen. Diese Werte sollten möglichst nicht im Quellcode festgelegt werden, da dieser sonst für jede Änderung neu kompiliert werden müsste. Da die Hauptkomponente als Windows-Dienst ablaufen sollte kommen auch keine Kommandozeilenparameter in Frage. Die nahe liegende Lösung war einfach eine Konfigurationsdatei zu verwenden, die beim Start des Dienstes automatisch eingelesen werden kann.

Als Format für eine Konfigurationsdatei bietet sich die Sprache XML (eXtensible Markup Language) geradezu an. XML ist ein weit verbreiteter, vielseitiger Standard. Microsoft verwendet XML im Rahmen des .NET Framework auch sehr häufig, so sind wie bereits erwähnt die Konfigurationsdateien von ASP.NET Webseiten in XML formuliert, und auch im Zusammenhang mit Web Services wird XML in Form von SOAP und WSDL verwendet. Auch das Client-Inventarsystem setzt auf XML in Konfigurationsdateien. Im Rahmen des .NET Frameworks stehen dem Entwickler zahlreiche Methoden zum Erstellen und Bearbeiten von XML Dokumenten und zum direkten Zugriff auf XML Elemente und Attribute zur Verfügung. Ebenso vorteilhaft ist, dass XML Dokumente als einfache Textdateien gespeichert werden und somit leicht vom Benutzer geladen und editiert werden können. Es können auch komplexe Daten einfach und für Menschliche Augen gut lesbar dargestellt werden.
Nachteilig ist der verhältnismäßig hohe Speicherverbrauch einer XML Datei, der jedoch bei den wenigen Konfigurationsdaten vernachlässigt werden kann.

Im Gegensatz zum Vorgängerprojekt wird beim Drucker-Inventar-Dienst nur eine einzelne Konfigurationsdatei „drconfig.xml“ verwendet.
Inhalt von drconfig.xml:

```
<root>
  <SNMPCommunities>
    <Name>public</Name>
    <Name>admin</Name>
  </SNMPCommunities>
  <AusleseThreads>10</AusleseThreads>
  <DruckerPraefix>
    <Praefix>vwagszp</Praefix>
    <Praefix>prtn</Praefix>
  </DruckerPraefix>
```

```
  <IPBereich>
    <Range>192.168.0.0-192.168.255.255</Range>
  </IPBereich>
  <KomplettScanZeiten>
    <Zeit>Montag,12:00</Zeit>
  </KomplettScanZeiten>
  <AktualisierungsZeiten>
    <Zeit>Immer,10:00</Zeit>
    <Zeit>Werktags,14:00</Zeit>
  </AktualisierungsZeiten>
  <Veraltet>90</Veraltet>
  <WEBServicePort>7778</WEBServicePort>
  <DBServer>Servername</DBServer>
  <DBName>Datenbankname</DBName>
</root>
```

Zu Erkennen ist, dass für einige Konfigurationswerte Listen vorgesehen sind, so können z.B. unter Aktualisierungszeiten theoretisch beliebig viele Zeitpunkte verwendet werden.

Der Drucker-Inventar-Dienst ist auch so entworfen, dass dieser beim ersten Start eine XML Datei mit Defaultwerten erstellt und in seinem Verzeichnis speichert, wenn er keine vorher erstellte Konfigurationsdatei finden kann. Anhand der so vorgegebenen Datei ist es für den Benutzer deutlich einfacher die Konfiguration anzupassen, als wenn die Datei komplett per Hand neu erstellt werden müsste. Sollten aber trotzdem beim Editieren Fehler gemacht werden, so werden die nicht erkannten Werte vom Dienst durch Defaultwerte ersetzt in die Datei korrekt zurückgeschrieben.

5. SNMP

5.1. Was ist SNMP?

Das Simple Network Management Protocol (SNMP) ist ein Standard Protokoll für die Netzwerkverwaltung. Mithilfe von SNMP können Netzwerkgeräte überwacht und verwaltet werden.

5.1.1. Historisches

In der Frühzeit des Internets bzw. des damaligen ARPANETs wurden keine großartigen Gedanken in die Verwaltung der Netze vergeudet. Es gab anfangs auch kaum verschiedene Hersteller von Netzwerkhardware und diese wenigen kamen mit der Verwaltung der eigenen Hardware mittels herstellerspezifischer Techniken noch gut zurecht.
1983 wurde schließlich das TCP/IP Protokoll vom amerikanischen Verteidigungsministerium zum Standard Internet Protokoll erklärt, was ein sprunghaftes Wachstum des Internets einleitete. Auch die Vielfalt an Hardware und Werkzeugen zu deren Verwaltung nahm zu, und es wurde bald nach übergreifenden Netzwerk-Managementmodellen geforscht.

Das früheste Modell war das High-Level Entity Mangement System (HEMS), welches über volle Funktionalität verfügte, aber letztlich nie im Internet eingesetzt wurde.

1987 stellte die OSI-Gruppe ihr Common Management Information Protocol (CMIP) der ISO vor. Darauf basierend entstand CMOT, das auch auf TCP Netzwerken funktionieren sollte.

Im selben Jahr entstand noch ein drittes Modell unter dem Namen Simple Gateway Monitoring Protocol (SGMP), welches sehr einfach aufgebaut und leicht integrierbar war. Schon zum Ende des Jahres war es auf mehreren Betriebssystemen verfügbar, wurde zunehmend verwendet und schließlich im RFC[3] 1028[4] vorgestellt.

[3] Request For Comments – offizielle Dokumentenreihe, die Vorschläge für Internettechnologien beschreibt
[4] http://velociraptor.mni.fh-giessen.de/rfc/rfc1028.txt

1988 sollte das Internet Activities Board (IAB) darüber entscheiden, welches dieser Modelle in Zukunft genutzt werden solle. Die Entscheidung fiel auf das noch nicht einsatzfähige CMOT, welches allerdings tatsächlich nie stabil eingesetzt werden konnte. Zur Überbrückung sollte ein erweitertes SGMP verwendet werden. Aus dieser Erweiterung ging das Simple Network Management Protocol hervor. Aufgrund der weiterhin bestehenden Probleme mit CMOT wurde 1989 schließlich ein neuer SNMP Entwurf unter RFC 1157[5] veröffentlicht, der 1990 schließlich zum offiziellen Standard ernannt wurde.

5.1.2. Versionen

Der einzig wirklich breit genutzte Standard ist nur die erste Version SNMPv1, jedoch gibt es Versuche diese Version zu erneuern. Um insbesondere die Sicherheitsmechanismen zu verbessern wurde an *Secure SNMP* (SNMPsec) gearbeitet, welches bald in SNMPv2 überging. Aufgrund von Uneinigkeiten und Akzeptanzproblemen spaltete sich die SNMPv2 Entwicklung in mehrere Zweige auf.

Die wichtigsten Zweige sind:

- SNMPv2u
  experimentell, mit benutzerbasierenden Sicherheitsfunktionen und Protokollfunktionen
- SNMPv2*
  experimentell, mit benutzerbasierenden Sicherheitsfunktionen und weiteren Funktionen
- SNMPv2c
  mit gemeinschaftlich basierenden Sicherheitsfunktionen und Protokollfunktionen

Alle diese Zweige arbeiten parallel und getrennt voneinander weiter. Wirklich benutzt, wenn auch nur von einer kleinen Gruppe, wird nur SNMPv2c, welches sich noch am engsten an SNMPv1 anlehnt. Es gibt aber auch Bemühungen zur Entwicklung von SNMPv3, welches die Konzepte von SNMPv2u und SNMPv2* vereinigen soll.

[5] http://velociraptor.mni.fh-giessen.de/rfc/rfc1157.txt

5.1.3. Wichtige Begriffe zu SNMP

- **Host**

  Bezeichnung für eine beliebige mit dem Netzwerk verbundene Hardwarekomponente inklusive Workstations und Server.

- **Verwaltete Objekte**

  Sowohl Host-Hardware wie auch Ressourcen-Hardware, die von einem anderen Computer im Netzwerk verwaltet werden können.

- **Agent**

  Eine Software, welche die Datenanforderung eines Managers bearbeitet. Nach der Datenaufforderung eines Managers kann der Agent die Daten eines verwalteten Objektes abrufen.

- **Manager**

  Oder auch Network Management Station (**NMS**) ist eine Software, die Daten von einem Agenten im Netzwerk anfordern kann. Üblicherweise verfügt eine NMS über ein Benutzer-Interface, über das Status und Daten eines Host angezeigt werden können.

- **MIB (Management Information Base)**

  Datei in der die Daten der verwalteten Objekte eines Host gespeichert sind. Jeder Host verfügt in der Regel über mehrere MIBs.

- **OID (Object Identifier)**

  Eindeutiger Objektbezeichner für die Objekte in einer MIB.

5.1.4. SNMP im OSI Schichtenmodell

Innerhalb des OSI Schichtenmodells befindet sich SNMP wie beispielsweise auch HTTP, FTP oder SMTP in der Schicht 7, dem Application Layer.

<table>
<tr><td rowspan="2">7</td><td rowspan="2">Application Layer</td><td>Management- und Agent-APIs</td></tr>
<tr><td>SNMP</td></tr>
<tr><td>6</td><td>Presentation Layer</td><td>ASN.1 und BER</td></tr>
<tr><td>5</td><td>Session Layer</td><td>RCP und NetBios</td></tr>
<tr><td>4</td><td>Transport Layer</td><td>TCP und UDP</td></tr>
<tr><td>3</td><td>Network Layer</td><td>IP und IPX</td></tr>
<tr><td>2</td><td>Data Link Layer</td><td rowspan="2">Ethernet, ArcNet und Token Ring</td></tr>
<tr><td>1</td><td>Physical Layer</td></tr>
</table>

5.2. Wie funktioniert SNMP?

Prinzipiell kann ein SNMP Manager bzw. eine Network Management Station Daten über verwaltete Objekte per SNMP von den SNMP Agenten der Zielhosts abrufen oder verändern. Die Daten befinden sich in einer lokalen MIB des Zielhosts, auf die der Agent nach Aufforderung zugreift. Der Agent hat dabei auch eine Art Wächterfunktion, weil alle Zugriffe des NMS auf die jeweilige MIB eines Zielhosts zwingend über ihn laufen müssen, direkte Zugriffe sind nicht möglich.

Folgende Zeichnung soll einen Eindruck über die grundlegende Arbeitsweise von SNMP vermitteln:

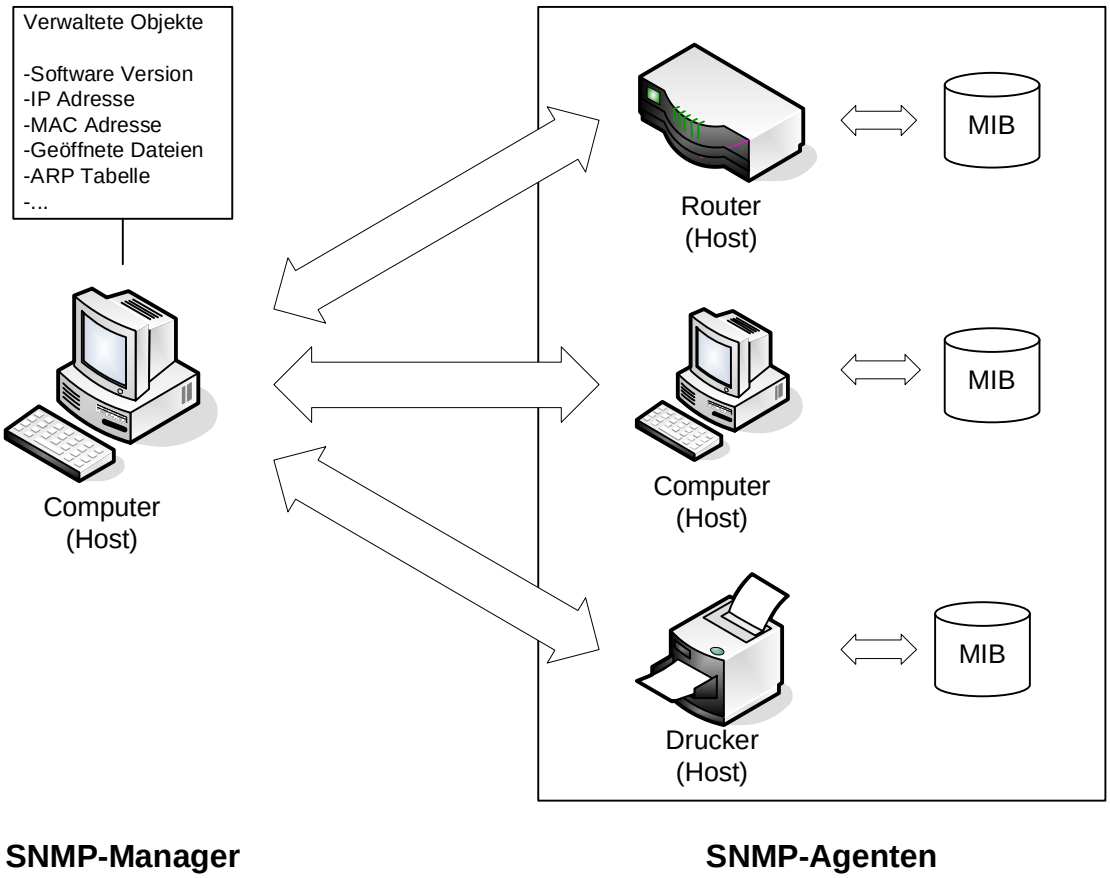

Abbildung 2.: SNMP Übersicht

5.2.1. Verfügbare Funktionen

Das SNMP Protokoll stellt 5 verschiedene Funktionen für das Ausführen von allen Netzwerk Management Funktionen zur Verfügung. 3 von ihnen werden vom Manager eingeleitet, die anderen beiden vom Agenten:

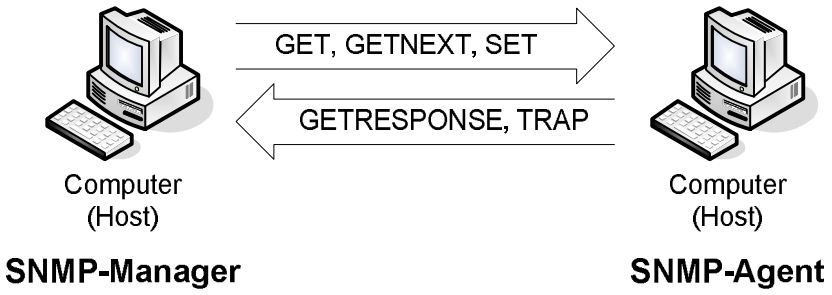

Abbildung 3.: SNMP Funktionen

5.2.1.1. GET

Die GET Anfrage ist die am häufigsten verwendete Funktion, sie fordert den Wert eines bestimmten verwalteten Objekts, das man gezielt mit seinem Objektbezeichner (OID) anspricht, an.

5.2.1.2. GETNEXT

Mit der GETNEXT Anfrage wird der Wert des nächsten verwalteten Objekts abgefragt. Wie auch schon bei der GET Anfragen wird hier mit der Anfrage ein bestimmter OID angegeben. Man erhält dann jedoch den Wert des Objekts, das an der nächsten Position nach diesem OID in der MIB steht. Diese Funktion ist zur Abfrage von Objekten gedacht, deren genauer OID zunächst nicht bekannt ist.

5.2.1.3. SET

SET dient zum ändern des Wertes eines bestimmten, gezielt angegebenen Objekts. Da die meisten Werte jedoch schreibgeschützt sind, wird diese Funktion eher selten benutzt.

5.2.1.4. GETRESPONSE

Die GETRESPONSE ist die Antwort des Agenten auf eine der obigen Anfragen. Sie enthält den genauen OID des adressierten Objekts und seinen Wert, im Falle der GETNEXT Anfrage sind das natürlich OID und Wert des nächsten Objekts.

5.2.1.5. TRAP

Die TRAP Funktion ist ein Spezialfall, in dem ein Agent unaufgefordert eine SNMP Nachricht an den Manager schickt. Dies geschieht beispielsweise, wenn eine bestimmte Fehlerbedingung auf dem Agenten erfüllt ist.

5.2.2. Protocol Data Units

Um die SNMP Nachrichten über das Netzwerk verschicken zu können, werden aus den jeweiligen Funktionen samt deren Argumenten so genannte Protocol Data Units erstellt. Dies ist letztlich eine Binärfolge, welche die jeweilige Funktion repräsentiert. Da SNMP auf dem verbindungslosen UDP basiert, wird diese PDU dann in ein UDP Paket eingebettet und schließlich über das Netzwerk verschickt. Das Erstellen einer PDU geschieht in der OSI Schicht 6, dem Presentation Layer, und es sind 3 Sprachen daran beteiligt.

- **Structure of Management Information (SMI)**
  Mit SMI wird das Format der zu verarbeitenden Objekte spezifiziert, auf die über das SNMP Protokoll zugegriffen werden soll.
- **Abstract Syntax Notation One (ASN.1)**
  Diese Sprache wird dazu genutzt, das Format von SNMP Nachrichten und MIB Modulen, die ein eindeutiges Datenbeschreibungsformat benutzen, zu definieren.
- **Basic Encoding Rules (BER)**
  Mit BER werden SNMP Nachrichten in ein geeignetes (Binär-) Format umgewandelt, damit die Daten in einem Netz übertragen werden können.

5.2.3. Ports

Standardmäßig werden folgende Ports für SNMP verwendet:

- Port 161 – SNMP Nachrichten
- Port 162 – SNMP Traps

5.2.4. Management Information Base

Jeder Host, der über SNMP verwaltet wird verfügt über eine MIB. Dieses ist eine Datenbank, in der die zu verwaltenden Objekte gespeichert werden.

Die Objekte darin werden folgendermaßen definiert:

- Beziehung zwischen Objekt, Objektname und OID
- Beschreibung des Datentyps, des Objekts bzw. seines Wertes
- Lese- Schreibzugriff für das Objekt
- Beschreibung des Objekts in Textform
- Indexierungsverfahren für das Objekt

In RFC 1213[6] ist eine spezielle MIB, die MIB-2 definiert, die einen Industriestandard darstellt. Die Mehrzahl der per SNMP verwalteten Geräte sollte diese MIB zumindest großteils implementieren. Es gibt jedoch auch herstellerspezifische MIBs, die jeder Hersteller selbst spezifizieren kann.

5.2.5. Objektbezeichner

Jedes Objekt einer MIB hat einen eindeutigen Objektbezeichner (OID). Das OID Namensschema ist in einer hierarchischen Baumstruktur aufgebaut. Jeder Zweig stellt gewissermaßen eine eigene MIB dar, einige sind bereits in den Standards festgelegt, aber es lassen sich quasi beliebig viele Zweige hinzufügen. Ein OID wird über die Namen der Zweige bezeichnet, die man von der Wurzel aus gesehen passieren muss, um das gewünschte Objekt zu erreichen. Es gibt dabei zwei Darstellungsformen, zum einen sind die Verzweigungen auf jeder Ebene durchnummeriert, zum anderen besitzen sie eine bestimmte Zeichenkette als Namen. Zur Darstellung werden dann entweder die Namen oder die Nummern durch Punkte voneinander getrennt, wie z.B. „.1.3.6.1.2.1.1.4“ bzw. „iso.org.dod.internet.mgmt.mib-2.system.sysContact“.

[6] http://velociraptor.mni.fh-giessen.de/rfc/rfc1213.txt

Die Baumstruktur stellt sich etwa wie folgt dar:

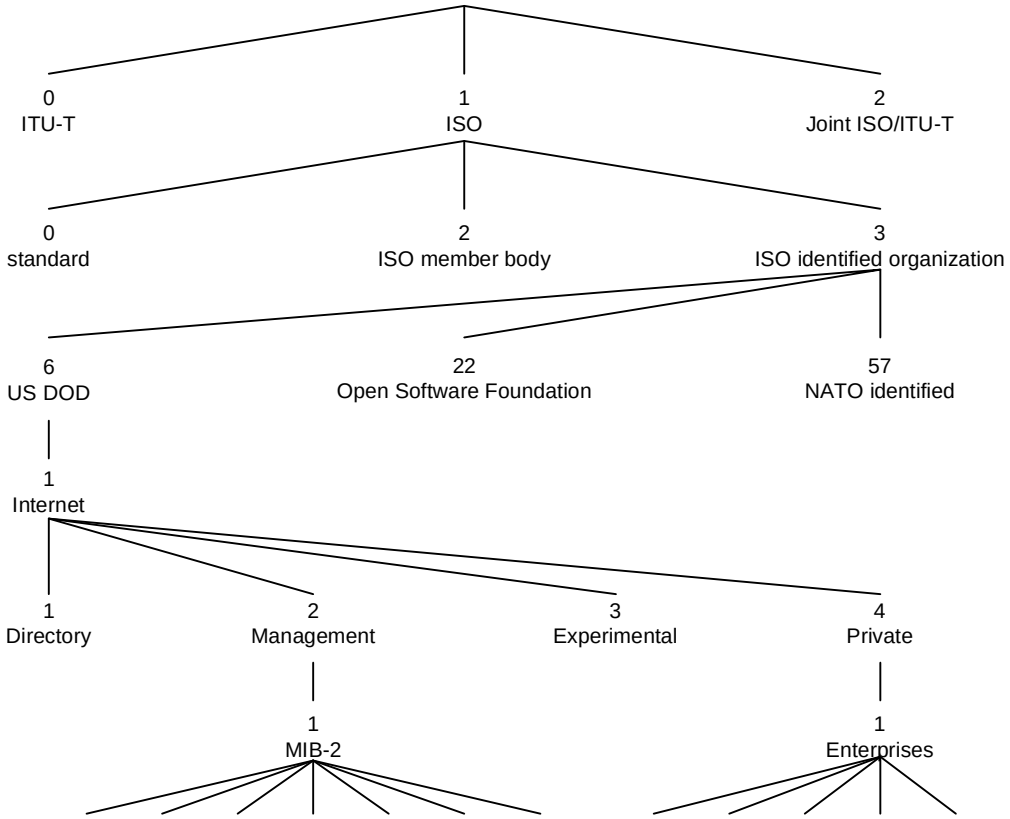

Abbildung 4.: OID Baumstruktur

5.2.6. Scalar und Columnar Variablen

Zu bemerken ist noch, dass es so genannte Scalar und Columnar Variablen gibt. Es sind Objekte möglich die genau einen (Scalar) oder auch mehrere Werte (Columnar) enthalten können. Auf diese Werte greift man zu, indem man den OID um eine Index Zahl erweitert. Scalar Variablen haben dabei immer den Index 0 und Columnar Variablen mindestens den Index 1. Zugriffe auf Columnar Variablen erfolgen in der Regel mittels GETNEXT.

5.2.7. Sicherheit

Die einzige Sicherheitsfunktion bei SNMP ist der so genannte Communityname, der bei jeder Nachricht mitgeschickt wird. Damit Manager und Agenten zusammenarbeiten können müssen sie der gleichen Community angehören, ist dies nicht der Fall, verwirft der Agent die an ihn gerichteten Anfragen oder generiert eine entsprechende Trap Nachricht. Ist der Agent jedoch keiner Community zugeordnet, so bearbeitet er alle Anfragen.

5.3. Die SNMP Bibliothek

Leider ist die Verwendung des Simple Network Mangement Protokolls gar nicht so einfach, auch wenn die Namensgebung dies vielleicht vermuten lässt. So war zum Zeitpunkt des Erstellens dieser Diplomarbeit noch keine direkte Unterstützung dieses Protokolls im .NET Framework integriert. Das Schreiben eines eigenen Moduls zum Erstellen, Verpacken und Verschicken der jeweiligen SNMP PDUs in UDP Pakete, sowie zum Empfangen und Interpretieren von Antworten der SNMP Agenten wäre zwar prinzipiell möglich gewesen, jedoch gäbe allein dieses Modul dann genug Material für eine weitere Diplomarbeit und die eigentliche Aufgabe des Drucker Inventarisierens wäre damit auch noch nicht erfüllt.

Daher wurde nach einer geeigneten API gesucht, mit deren Hilfe sich die Kommunikation per SNMP möglichst einfach (und möglichst kostenlos) aus der .NET Umgebung heraus realisieren lässt. Es wurden auch gleich eine Reihe solcher APIs gefunden, die jedoch bei näherer Betrachtung weder besonders einfach noch kostenlos waren. Außerdem basierten diese Lösungen meist entweder auf (unmanaged) C++ oder JAVA, was in jedem Fall einen Ausbruch aus der .NET Umgebung bedeutet und somit eine mögliche, zusätzliche Problemquelle dargestellt hätte.

Schließlich fand sich doch noch eine geeignete Lösung in Form einer SNMP Klassenbibliothek für das .NET Framework. Diese in C# erstellte Klassenbibliothek wurde von Malcolm Crowe von der University of Paisley[7] als freie Software inklusive der Quelltexte zur Verfügung gestellt. Obwohl in C# erstellt, lässt sich diese Klassenbibliothek weitgehend problemlos in das, ansonsten in VB.NET erstellte, Drucker Inventar Projekt integrieren und demonstriert somit beispielhaft die Sprachunabhängigkeit der .NET Plattform. Zum anderen ist es sehr hilfreich, dass auch der Quellcode vorliegt, um den Problemen, die im Detail eben doch auftreten, begegnen zu können.

Verwendet wurde die Version 0.3.3f der SNMP Bibliothek. Sie besteht aus zwei Teilen, die jeweils zu einer eigenen DLL kompiliert werden:

- Snmp.dll:
  Enthält die Kernfunktionalitäten der Bibliothek.
- Mib.dll:
  Übersetzt die ausgeschriebene Variante von MIB OIDs in die intern benutzte Variante.

[7] http://cis.paisley.ac.uk/crow-ci0/

Die Snmp.dll

Diese Klassenbibliothek enthält zwei Namensräume, X690 und Snmp.
Der Namensraum X690 enthält eine Implementierung der Basic Encoding Rules (BER) für die Abstract Syntax Notation 1 (ASN.1), wie sie im SNMP verwendet wird und durch internationale Standards spezifiziert ist. Der ISO Kode dieses Standards ist X.690. Prinzipiell sind hier die entsprechenden Datentypen und Typkonvertierungsmethoden definiert, wie z.B. der OctetString. Von besonderer Bedeutung bei fortgeschrittener Anwendung ist hier die X690.Universal Klasse, die, wie der Name schon nahe legt, eine Art universellen Datentyp darstellt und praktisch alle bei SNMP auftretenden Datenformen aufnehmen kann.
Der Namensraum Snmp implementiert die Kommunikation über das Netzwerk und die eigentlichen Funktionen des Protokolls, wie die OID Zugriffe mittels GET, GETNEXT und SET. Hier wird die numerische Form der Darstellung von MIB OIDs verwendet, wie z.B. ".1.3.6.1.2.1.1.1.0".

Die Mib.dll

In dieser Bibliothek ist nur der Namensraum RFC1157 enthalten. Hier wird die Übersetzung von MIB OIDs in deren numerischer Form, wie z.B. „.1.3.6.1.2.1.1.4.0“, in deren lesbare Form, wie z.B. „system.sysContact.0“, in beide Richtungen behandelt. Außerdem werden die Hilfsinformationen der MIB Systemdateien verwendet, dazu müssen diese allerdings auf dem System installiert sein (unter Windows normalerweise in c:\windows\system32).
Diese Funktionalität wird eigentlich nur benötigt, um in konsolenähnlichen Umgebungen auf die MIB OIDs einfach zugreifen zu können und ist für das Drucker Inventar Projekt nicht relevant. Da das Nachladen dieser MIB Systemdateien auch noch unnötige Zeit verbraucht und die Dateien nicht auf jedem System installiert sind, wurde auf die Verwendung dieses Teils im Drucker Inventar Projekt verzichtet.

5.3.1. Einfaches Anwenden der Bibliothek

Hier folgen einige Anwendungsbeispiele der SNMP Bibliothek.

5.3.1.1. Einfaches GET

Die einfachste Möglichkeit, das SNMP Protokoll zu verwenden, ist eine GET Anfrage für einen einzelnen MIB Eintrag, z.B. „system.sysDescr.0“, an einen Agenten, z.B. „localhost“, in der Community „public“.

Mithilfe der SNMP Bibliothek würde eine solche Anfrage in C# Syntax wie folgt aussehen:

```
RFC1157.Mgmt mib = new RFC1157.Mgmt();
Snmp.ManagerSession sess = new Snmp.ManagerSession("localhost", "public");
uint[] oid = mib.OID("mgmt.mib-2.system.sysDescr.0");
Snmp.ManagerItem mi = new Snmp.ManagerItem(sess, oid);
Console.WriteLine(mi.Value.ToString());
```

Zur Einführung wird dieses erste Beispiel besonders ausführlich, Schritt für Schritt erläutert:

1. Man erzeugt eine Mgmt Instanz, um OIDs übersetzen zu können:

```
new RFC1157.Mgmt();
```

Der zurückgegebene Wert wird in der RFC1157.Mgmt Variablen mib gespeichert. Dieser Konstruktor liest zunächst alle MIB Definitionen ein, die auf dem System gefunden werden.

2. Man öffnet eine ManagerSession zum Agenten in der gewählten Community:

```
new Snmp.ManagerSession("localhost", "public");
```

Der zurückgegebene Wert wird in der Snmp.ManagerSession Variablen sess gespeichert. Man hätte auch die IP-Adresse des Agenten mit „127.0.0.1“ statt mit „localhost“ angeben können.

3. Der gewünschte OID wird in ein unsigned Integer Array übersetzt:

```
uint[] oid = mib.OID("mgmt.mib-2.system.sysDescr.0");
```

Der Wert wird unter oid gespeichert. Das Präfix „mgmt.mib-2.“ gibt den gewünschten MIB Zweig an. Typischerweise sind mehrere solcher MIB Zweige auf einem System definiert, alle beginnen mit „iso.org.dod.internet.“ gefolgt von beispielsweise „mgmt.mib-2.“.

4. Man erzeugt ein ManagerItem, um das Ergebnis zu erhalten:

```
new Snmp.ManagerItem(sess, oid);
```

Die ManagerSession und der OID werden übergeben, das Ergebnis wird in der Snmp.ManagerItem Variablen mi gespeichert.

5. Das Ergebnis wird ausgegeben:

```
Console.WriteLine(mi.Value.ToString());
```

Die Snmp.ManagerItem Variable mi enthält nun unter mi.Oid den tatsächlich abgefragten OID in Form eines unsigned Integer Array, der unter Verwendung von mib.OID(mi.Oid) wieder in die lesbare Form übersetzt werden könnte, und mi.Value enthält den ausgelesenen Wert des OID in Form eines X690.Universal Objekts. Zur Ausgabe wird der Wert mittels der .ToString Methode in eine Zeichenfolge umgewandelt und auf der Konsole ausgegeben.

5.3.1.2. Einfaches SET

Ein SET könnte in C# Syntax etwa so aussehen:

```
RFC1157.Mgmt mib = new RFC1157.Mgmt();
Snmp.ManagerSession sess = new Snmp.ManagerSession("localhost", "public");
uint[] oid = mib.OID("mgmt.mib-2.system.sysDescr.0");
Snmp.ManagerItem mi = new Snmp.ManagerItem(sess, oid);
mi.Value = new X690.Universal("local agent");
```

Bis auf die letzte Zeile ist das Beispiel mit dem vorangegangenen identisch, hier wird jetzt jedoch mi.Value ein neuer Wert zugewiesen, statt ihn auszugeben. Der Wert muss ein X690.Universal Objekt sein, das hier, mit „local agent“ als Parameter, neu erzeugt wird. Hierdurch würde der MIB Eintrag „system.sysDescr.0“ des lokalen SNMP Agenten in der Community „public“ den Wert „local agent“ bekommen. Dies ist jedoch nur möglich, wenn die Snmp.ManagerItem Variable noch nicht für ein GET benutzt wurde und wenn der betreffende Agent das Setzen von MIB Einträgen erlaubt.

5.3.1.3. Unterbaum abfragen

Die Abfrage eines ganzen Unterbaums mittels GET/GETNEXT könnte etwa so aussehen:

```
RFC1157.Mgmt mib = new RFC1157.Mgmt();
Snmp.ManagerSession sess = new Snmp.ManagerSession("localhost", "public");
uint[] oid = mib.OID("mgmt.mib-2.system");
Snmp.ManagerSubTree ms = new Snmp.ManagerSubTree(sess, oid);
if (ms.Length==0)
   Console.WriteLine("SubTree is empty");
else
   foreach(Snmp.ManagerItem mi in ms){
      Console.WriteLine(mib.OID(mi.Oid) + " : " + mi.Value.ToString());
   }
```

Der Anfang ist hier wieder identisch mit den vorangegangenen Beispielen, als OID wird diesmal jedoch nur „mgmt.mib-2.system“ verwendet, der den ganzen System-Unterbaum bezeichnet. Statt einem Snmp.ManagerItem wird nun ein Snmp.ManagerSubTree erzeugt, der praktisch ein Feld von Snmp.ManagerItems darstellt. Sofern der Unterbaum nicht leer ist, wird das Feld ab der achten Zeile in einer foreach-Schleife durchlaufen, die gefunden OIDs sowie deren Werte werden analog zum ersten Beispiel in 5.3.1.1. auf der Konsole ausgegeben.

5.3.2. Fortgeschrittenere Anwendungsmöglichkeiten

Es ist auch möglich, GET, GETNEXT und SET Anfragen direkt auf eine Snmp.ManagerSession anzuwenden, was weitere Möglichkeiten eröffnet, z.B. auch mehrere OIDs in einer einzelnen GET Anfrage unterzubringen.

5.3.2.1. Mehrere GETs

In C# Syntax könnte das beispielsweise folgendermaßen aussehen:

```
RFC1157.Mgmt mib = new RFC1157.Mgmt();
Snmp.ManagerSession sess = new Snmp.ManagerSession("localhost", "public");
uint[] oid1 = mib.OID("mgmt.mib-2.system.sysDescr.0");
uint[] oid2 = mib.OID("mgmt.mib-2.system.sysLocation.0");
X690.Universal[] results = sess.Get(sess.VarBind(oid1), sess.VarBind(oid2));
foreach(X690.Universal result in results){
   uint[] oid = (uint[])result[0].Value;
   string value = result[1].Value.ToString();
   Console.WriteLine(mib.OID(oid) + " : " + value);
}
```

Zu Beginn wird wieder eine Snmp.ManagerSession wie gehabt geöffnet, diesmal werden aber die zwei OIDs oid1 und oid2 definiert. Mittels sess.VarBind() werden aus den OIDs zwei X690.Universal Objektpaare erstellt, die wiederum als Parameter für die sess.Get() Methode erforderlich sind. Das Ergebnis dieser Methode ist wieder ein Feld von X690.Universal Objektpaaren, das unter results gespeichert wird. Die einzelnen Objektpaare werden im Anschluß in der foreach-Schleife durchlaufen. Der Wert des jeweils ersten X690.Objekts enthält den abgefragten OID, der mittels des Typecast Operators (uint[]) in dem unsigned Integer Array oid gespeichert wird. Der Wert des zweiten X690.Universal Objekts enthält den jeweils ausgelesenen Wert, der hier in der String Variablen value gespeichert wird. Der OID wird schließlich wieder in die einfach lesbare Form übersetzt und mit dem ausgelesenen Wert auf der Konsole ausgegeben.

5.3.2.2. Mehrere SETs

Mehrere SET Anweisungen können beispielsweise auch so realisiert werden:

```
RFC1157.Mgmt mib = new RFC1157.Mgmt();
Snmp.ManagerSession sess = new Snmp.ManagerSession("localhost", "public");
uint[] oid1 = mib.OID("mgmt.mib-2.system.sysDescr.0");
uint[] oid2 = mib.OID("mgmt.mib-2.system.sysLocation.0");
X690.Universal val1 = new X690.Universal("local agent");
X690.Universal val2 = new X690.Universal("at home");
sess.Set(sess.VarBind(oid1, val1), sess.VarBind(oid2, val2));
```

Der Beginn ist wieder identisch zum vorigen Beispiel, im Anschluss werden aber diesmal noch die zwei X690.Universal Objekte val1 und val2 definiert, die die zu setzenden Werte enthalten. Mittels sess.VarBind() werden jetzt zwei X690.Universal Objektpaare aus oid1 und val1 bzw. aus oid2 und val2 erstellt, welche als Parameter für die sess.Set() Methode dienen. Sofern der lokale SNMP Agent das Setzen der Einträge erlaubt hat, sollten danach der MIB Eintrag „system.sysDescr.0“ den Wert „local agent“ und der MIB Eintrag „system.sysLocation.0“ den Wert „at home“ bekommen haben.

5.3.2.3. GETNEXT Anfrage

Eine direkte GETNEXT Anfrage kann etwa so aussehen:

```
RFC1157.Mgmt mib = new RFC1157.Mgmt();
Snmp.ManagerSession sess = new Snmp.ManagerSession("localhost", "public");
uint[] oid = mib.OID("mgmt.mib-2.system.sysName.0");
X690.Universal unioid = new X690.Universal(oid);
X690.Universal uni = new X690.Universal(unioid, X690.Universal.Null);
if (sess.GetNext(ref uni)){
   oid = (uint[])uni[0].Value;
   string value = uni[1].ToString();
   Console.WriteLine(mib.OID(oid) + " : " + value);
}
```

Der Anfang ist wieder wie bei den vorigen Beispielen, es wird der OID „...system.sysName.0“ erstellt. Aus diesem wird in der vierten Zeile das X690.Universal Objekt unioid erstellt. Im Anschluß wird das X690.Universal Objekt uni erstellt, das letztlich ein Objektpaar aus dem zuvor erstellten unioid und einem leeren X690.Universal Objekt (X690.Universal.Null) darstellt. Dieses Objektpaar dient schließlich als Referenzparameter für die bool’sche Methode sess.GetNext(). War die GETNEXT Anfrage erfolgreich, so gibt die Methode ein „true“ zurück und die übergebene Referenz enthält die erhaltenen Werte. In diesem Fall enthält das erste Objekt des Objektpaares uni den neuen OID, der normalerweise „...system.sysLocation.0“ entsprechen sollte, und das zweite Objekt enthält den ausgelesenen Wert. Die Ausgabe der Werte erfolgt wieder analog zum Beispiel in 5.3.2.1.

5.4. Abfragerealisierung

Die fertig kompilierte Version der SNMP Bibliothek kann einfach als Verweis dem Drucker Inventar Projekt hinzugefügt werden und ihre Klassen und Methoden können dort verwendet werden, auch wenn das Projekt ansonsten komplett auf VB.NET setzt. Lediglich die Aufrufe müssen an die andere Syntax angepasst werden. Konkret wird jedoch der Quelltext in Form eines Teilprojektes verwendet, wodurch auch der Debugger der Entwicklungsumgebung bei Problemen auf die betreffenden Stellen im Quelltext hinweisen kann, was während der Entwicklung sehr hilfreich ist.

Für die eigentliche Arbeit des Drucker-Inventar-Dienstes werden jedoch nur einige Teilfunktionalitäten der SNMP Bibliothek benötigt. Da nur Informationen ausgelesen werden sollen, werden nur GET bzw. GETNEXT Anfragen verwendet. Das Ändern der Informationen ist nicht vorgesehen, daher kann vollständig auf SET Anfragen verzichtet werden. Wie bereits in Kapitel 5.3 erwähnt, wird ebenfalls auf die Verwendung der Mib.dll verzichtet.

5.4.1. Auszulesende Informationen

Bei den per SNMP abfragbaren Informationen ist auch nur ein kleiner Teil für das Projekt wirklich von Interesse. Andererseits sind längst nicht alle Informationen der verschiedenen Standard-MIBs auf den verschiedenen Druckern verfügbar, auch die genaue Position der Informationen in den MIBs kann bei einigen Geräten variieren. Viele durchaus interessante Informationen sind oft auch in herstellerspezifischen MIBs zu finden, die jedoch kaum öffentlich dokumentiert sind. Außerdem lassen sich leider einige der Drucker auch gar nicht per SNMP auslesen.

Aus diesen Gründen wurden im Rahmen des Projektes nur einige spezielle Variablen aus den Standard MIBs zum Auslesen ausgewählt, die zum einen von besonderem Interesse bzw. Nutzen sind und zum anderen mit möglichst großer Wahrscheinlichkeit auch tatsächlich auslesbar sind. Sie sind allesamt im MIB-2-Zweig (1.3.6.1.2.1...) zu finden, auf herstellerspezifische MIBs (1.3.6.1.4.1…) wurde komplett verzichtet.

Praktisch immer, wenn ein Gerät überhaupt auf SNMP Anfragen reagiert, sind die System- (1.3.6.1.2.1.1…) und die Interface Gruppe (1.3.6.1.2.1.2…) abfragbar. Beide Gruppen sind standardmäßig im MIB-2-Zweig enthalten, der im RFC 1213[8] definiert ist. Hier sind eher allgemeine Informationen zu finden, insbesondere ein Beschreibungsstring des Geräts und auch dessen MAC-Adresse.

[8] http://velociraptor.mni.fh-giessen.de/rfc/rfc1213.txt

Der Host Resources- (1.3.6.1.2.1.25…) und besonders auch der Printer MIB-Zweig (1.3.6.1.2.1.43…) lassen sich leider etwas seltener auslesen, jedoch enthalten erst diese Zweige Statusinformationen und druckerspezifische Informationen, wie z.B. den Namen des Druckermodells, die Anzahl der bisher gedruckten Seiten, oder den Tonerfüllstand. Die Host Resources MIB ist im RFC 1514[9] und die Printer MIB im RFC 1759[10] definiert.

Konkret werden folgende OIDs in diesem Projekt zum Auslesen verwendet:

5.4.1.1. Mib-2 System Gruppe

(iso(1).org(3).dod(6).internet(1).mgmt(2).mib-2(1).system(1))

- sysDescr(1)
  Beschreibung des Gerätes, meist mit Versionsangaben.
- sysUpTime(3)
  Zeit seit der Initialisierung des Netzwerk Teils des Geräts in hundertstel Sekunden.
- sysContact(4)
  Kontaktinformationen zu einem für das Gerät zuständigen Betreuer.
- sysLocation(6)
  Standortinformationen des Gerätes.

5.4.1.2. Mib-2 Interfaces Gruppe

(iso(1).org(3).dod(6).internet(1).mgmt(2).mib-2(1).interfaces(2))

- ifPhysAdress(6)
  MAC-Adresse des Netzwerk Teils des Gerätes.

5.4.1.3. Host Resources MIB

(iso(1).org(3).dod(6).internet(1).mgmt(2).mib-2(1).host(25))

- hrSystemUptime(1.1)
  Zeit seit der Initialisierung des Hostgeräts in hundertstel Sekunden.
- hrDeviceType(3.2.1.2)
  Gibt den Typ eines Hostgeräts an, Drucker enden z.B. mit der Zahl 5.
- hrDeviceDescr(3.2.1.3)
  Beschreibung des Hostgeräts, gibt z.B. das Druckermodell an.

[9] http://velociraptor.mni.fh-giessen.de/rfc/rfc1514.txt
[10] http://velociraptor.mni.fh-giessen.de/rfc/rfc1759.txt

- hrDeviceStatus(3.2.1.5)
  Gegenwärtiger (zahlenkodierter) Betriebsstatus des Hostgeräts.
- hrPrinterStatus(3.5.1.1)
  Gegenwärtiger (zahlenkodierter) Status des Druckers.
- hrPrinterDetectedErrorState(3.5.1.2)
  Erkannte (binärkodierte) Fehlerarten des Druckers.

5.4.1.4. Printer MIB

(iso(1).org(3).dod(6).internet(1).mgmt(2).mib-2(1).printmib(43))

- prtInputMaxCapacity(8.2.1.9)
  Maximale Kapazität des Papiervorratfaches.
- prtInputCurrentLevel(8.2.1.10)
  Gegenwärtiger Füllstand des Papiervorratfaches.
- prtInputName(8.2.1.13)
  Bezeichnung des Papiervorratfaches.
- prtOutputMaxCapacity(9.2.1.4)
  Maximale Kapazität des Papierausgabefaches.
- prtOutputRemainingCapacity(9.2.1.5)
  Gegenwärtig verbleibende Restkapazität des Papierausgabefaches.
- prtOutputName(9.2.1.7)
  Bezeichnung des Papierausgabefaches.
- prtMarkerLifeCount(10.2.1.4)
  Gesamtzahl der gedruckten Seiten des Druckers.
- prtMarkerPowerOnCount(10.2.1.5)
  Zahl der gedruckten Seiten seit dem letzten Einschalten des Druckers.
- prtMarkerSuppliesType(11.1.1.5)
  Art des Farbvorrats, z.B. Toner(3) oder Tinte(5).
- prtMarkerSuppliesDescription(11.1.1.6)
  Beschreibung des Farbvorrats, z.B. Farbe und Farbpatronentyp.
- prtMarkerSuppliesMaxCapacity(11.1.1.8)
  Maximale Kapazität des Farbvorrats.
- prtMarkerSuppliesLevel(11.1.1.9)
  Gegenwärtiger Füllstand des Farbvorrats.

5.4.2. Der Auslesevorgang

Um diese ausgewählten Informationen im Projekt auszulesen wurde eine eigene VB.NET Klasse „AbfrageSession“ erstellt, die alle Zugriffe auf die SNMP Bibliothek und somit die gesamte SNMP Kommunikation übernimmt. Zum Erzeugen einer Objektinstanz müssen ein Zielhost und der zu verwendende SNMP Communityname angegeben werden. Die Zugriffe auf die gewünschten Informationen geschehen dann über die entsprechenden Objektmethoden, die jeweils für jede der ausgewählten SNMP Variablen erstellt wurden. Die Ergebnisse werden in Form von CTS kompatiblen Typen (z.B. String() oder Integer) zurückgegeben, was die problemlose Weiterverwendung im Projekt erlaubt.

Ein einfacher Zugriff mit anschließender Ausgabe auf der Konsole auf die Variable sysDescr innerhalb des Drucker Inventar Projekts könnte dann in VB.NET Syntax wie folgt aussehen:

```
Dim session As AbfrageSession = New AbfrageSession("localhost", "public")
Dim descr As String = session.GetSysDescr()
Console.WriteLine("sysDescr: " & descr)
session.Close
```

Ein solcher Zugriff ist relativ unspektakulär, trägt aber gerade deshalb sehr zur Übersichtlichkeit des Quelltextes des Projekts bei. Die eigentliche Arbeit, die zum Auslesen der SNMP Variablen nötig ist, wird durch die jeweiligen Methoden verborgen. An zwei konkreten Beispielen soll im Folgenden gezeigt werden wie diese Methoden arbeiten.

5.4.2.1. Eine einfache Abfragemethode

Als einfaches Beispiel dient hier wieder einmal die Abfrage der sysDescr Variable. Die zugehörige Methode sieht wie folgt aus:

```
'Ermittelt den SysDescr und gibt ihn zurück oder bei Problemen Nothing
Public Function GetSysDescr() As String
    Dim s As String
    Try
        mi = New ManagerItem(ms, OID.ToUInt32Array(".1.3.6.1.2.1.1.1.0"))
        s = RemoveQuotes(mi.Value.ToString())
        If s = "" Then s = Nothing
        If s = "NULL" Then s = Nothing
    Catch ex As Exception
        s = Nothing
    End Try
    Return s
End Function
```

Hierbei ist zu beachten, dass ms und mi Objektnamen für Snmp.ManagerSession bzw. Snmp.ManagerItem Objekte sind, die bereits beim Erzeugen der AbfrageSession initialisiert werden und klassenweit zugreifbar sind. Wie im Beispiel von Kapitel 5.3.1.1 geschieht hier die Abfrage mithilfe eines Snmp.ManagerItems, die Verwendung unterscheidet sich kaum. Statt auf die Mib.dll zurückzugreifen wird hier die selbst geschriebene Hilfsfunktion OID.ToUInt32Array verwendet. Der als String übergebene OID wird von ihr in den gewünschten unsigned Integer Array umgewandelt. Der Wert des Snmp.ManagerItems wird schließlich als String gespeichert und die Anführungszeichen werden entfernt. Ist der String leer, enthält er die Zeichenfolge „NULL“ oder geschieht ein Ausnahmefehler, der im Catch-Block abgefangen wird, so wird der String auf Nothing gesetzt und zurückgegeben, somit kann anhand des Rückgabewertes dieser Funktion erkannt werden, ob die sysDescr Variable korrekt ausgelesen werden konnte, oder nicht.

So einfach lassen sich aber nur die Variablen aus der System Gruppe abfragen, da diese nur einfach unter dem Index 0 vorliegen, weshalb bei den bisherigen Beispielen auch immer „.0“ an den eigentlichen OID der jeweiligen Variable angehängt wurde.

5.4.2.2. Eine fortgeschrittenere Abfragemethode

Insbesondere innerhalb der Host Resources und der Printer MIB sind häufig mehrere Werte pro SNMP Variable mit jeweils unterschiedlichen Indizes vorhanden. Da der genaue Index eines gewünschten Wertes in der Regel vorher ebenfalls nicht bekannt ist, kommt man hier nicht um die Verwendung von GETNEXT Anfragen herum. Die Methode im folgenden Beispiel soll den Index des Druckers innerhalb der Host Resources MIB ermitteln, mit dessen Hilfe man danach einige der weiteren Variablen wieder gezielt mittels GET abfragen kann.

```
'gibt Index des Druckers im Host Resources MIB zurück, oder -1 wenn keiner gefunden
Public Function GethrPrinterDeviceIndex() As Integer
    Dim i As Integer
    Dim uarray As UInt32()
    Dim oidDevType, oidTest As OID
    oidDevType = New OID(".1.3.6.1.2.1.25.3.2.1.2")
    oidTest    = New OID(".1.3.6.1.2.1.25.3.2.1.2")
    Try
        Do While oidTest.GetString().StartsWith(oidDevType.GetString())
            'auf oidTest basierte Universal Variable deklarieren,
            'die als Referenzparameter für GetNext dient
            Dim uni As Universal = New Universal(New Universal _
                                   (oidTest.GetUInt32Array), Universal.Null)

            If ms.GetNext(uni) Then
                'der mit GetNext (uni.Value(0).Value) ermittelte OID
                'wird in oidTest zurückgeschrieben
                oidTest.SetValue(DirectCast(uni.Value, Universal())(0).ToString)
```

```
                If oidTest.GetString().StartsWith(oidDevType.GetString()) AndAlso _
                String.Compare(oidTest.GetString(), oidDevType.GetString(), True) _
                <> 0 Then
                    'gefundener Wert (uni.Value(1).Value)
                    'muß ein OID bzw. UInt32() sein
                    uarray = DirectCast((DirectCast(uni.Value, _
                             Universal())(1).Value), UInt32())
                    If System.Convert.ToInt32(uarray(uarray.Length - 1)) = 5 Then
                        i = oidTest.GetLastIndex()
                        Exit Do
                    Else
                        'Eintrag ist kein Drucker
                        i = -1
                    End If
                Else
                    'hrDeviceType Eintrag nicht gefunden
                    i = -1
                    Exit Do
                End If
            Else
                'GetNext fehlgeschlagen
                i = -1
                Exit Do
            End If
        Loop
    Catch ex As Exception
        i = -1
    End Try
    Return i
End Function
```

Auch hier ist die Snmp.ManagerSession ms bereits initialisiert und es wird auf die selbstgeschriebene OID-Hilfsklasse zurückgegriffen. Untersucht wird die SNMP Variable hrDeviceType deren OID in den beiden Hilfsvariablen oidDevType und oidTest gespeichert wird, wobei letztere als Laufvariable dienen soll. Darauf wird eine Schleife durchlaufen, solange oidTest und oidDevType noch den gleichen Anfang haben, d.h. solange man sich noch innerhalb von hrDeviceType befindet. Ähnlich wie im Beispiel von Kapitel 5.3.2.3. wird nun eine GETNEXT Anfrage durchgeführt, war diese erfolgreich wird der ermittelte OID wieder in der Laufvariable gespeichert. Wenn nun oidTest und oidDevType immer noch gleich beginnen, aber nicht mehr identisch sind, muss ein Wert in Form eines unsigned Integer Array ausgelesen worden sein. Ist die letzte Zahl dieses Feldes eine 5, so steht der Wert gemäß der Definition von hrDeviceType für einen Drucker und der Index der Laufvariablen oidTest ist der gesuchte Index, daraufhin wird die Schleife abgebrochen und der Index wird zurückgegeben, ansonsten wird die Schleife erneut durchlaufen. Wenn kein Druckereintrag in hrDeviceType gefunden wird, die GETNEXT Anfrage fehlschlägt oder eine andere Ausnahme auftritt wird eine -1 zurückgegeben, wodurch leicht erkannt werden kann, ob der gesuchte Index ermittelt werden konnte, da dieser ansonsten nicht negativ sein kann.

Bei den anderen Methoden wird meist ähnlich an die Suche nach den gewünschten Informationen herangegangen, jedoch sind diese oft noch umfangreicher oder müssen zusätzliche Aufgaben bewältigen. Der binärkodierte Wert von hrPrinterDetectedErrorState wird beispielsweise zunächst interpretiert um die gefunden Fehlerinformationen in geeigneter Form als String ausgeben zu können. Bei prtMarkerSuppliesLevel sind meist mehrere Werte gesucht, wenn es sich um Farbdrucker handelt, außerdem wird gleich mithilfe des jeweils entsprechenden Wertes von prtMarkerSuppliesMaxCapacity ein prozentualer Wert für den Farbfüllstand ermittelt.

5.4.3. Probleme beim Auslesen

Wie bereits zu Beginn von Kapitel 5.3 erwähnt traten beim praktischen Umgang mit der SNMP Bibliothek einige Probleme auf. Im Folgenden werden diese Probleme und ihre Lösungen etwas genauer dargestellt.

5.4.3.1. Initialisieren von OIDs

In den vorangegangenen Beispielen konnte man bereits vermuten, dass die SNMP Bibliothek intern für die Darstellung von OIDs unsigned Integer Arrays verwendet, zumindest werden sie in dieser Form als Übergabeparameter verlangt. Leider ist unsigned Integer kein CTS kompatibler Datentyp, was an sich kein größeres Problem darstellt. Problematisch bzw. unpraktisch ist jedoch die Initialisierung eines solchen Feldes mit speziellen Startwerten außerhalb von C#. In C# würde eine solche Initialisierung einfach folgendermaßen aussehen:

```
uint[] oid = {1, 3, 6, 1, 2, 1, 1, 1, 0};
```

Die nahe liegende Umsetzung in VB.NET in Form von:

```
Dim oid As UInt32() = {1, 3, 6, 1, 2, 1, 1, 1, 0}
```

funktioniert leider nicht, da die übergebenen Werte in VB.NET als Integer gelten und nicht automatisch zu unsigned Integer konvertiert werden können. Auch die Verwendung der üblichen Konvertierung mittels CType wird nicht akzeptiert:

```
Dim oid As UInt32() = {CType(1,Uint32), CType(3,Uint32), CType(6,Uint32), _
                       CType(1,Uint32), CType(2,Uint32), CType(1,Uint32), _
                       CType(1,Uint32), CType(1,Uint32), CType(0,Uint32)}
```

Erst die Verwendung der Methode System.Convert.ToUInt32() aus der Klassenbibliothek des .NET Frameworks schafft hier Abhilfe:

```
Imports System.Convert
' […]
Dim oid As UInt32() = {ToUint32(1), ToUint32(3), ToUint32(6), _
                       ToUint32(1), ToUint32(2), ToUint32(1), _
                       ToUint32(1), ToUint32(1), ToUint32(0)}
```

Diese Form der Initialisierung ist im Vergleich zur Verwendung in C# jedoch sehr umständlich und unübersichtlich. Aus diesem Grunde wurde die bereits in den vorangegangenen Beispielen verwendete OID-Hilfsklasse erstellt, mit deren Hilfe sich OIDs einfach mit einem String initialisieren lassen und OIDs je nach Bedarf als String oder unsigned Integer Array zur Verfügung stellen lassen. Alternativ ließe sich auch die Mib.dll für diese Aufgaben verwenden, was jedoch wie bereits erwähnt andere Probleme aufwerfen kann.
Am einfachsten wäre es wohl gewesen, wenn bei der SNMP Bibliothek auf CTS kompatible Typen in den Schnittstellen gesetzt worden wäre. So zeigt sich jedenfalls, dass .NET zwar eine gewisse Sprachunabhängigkeit bietet, es aber auch Unterschiede außer der Syntax zwischen den .NET Sprachen gibt.

5.4.3.2. Negative Abfrageergebnisse

Ein weiteres Problem tritt auf wenn das Ergebnis einer SNMP Abfrage einen negativen Integer Wert darstellen soll. Negative Werte treten vor allem im Zusammenhang mit Kapazitätsangaben bei Variablen innerhalb der Printer MIB auf. Dort werden sie für spezielle Zustände verwendet, so steht meist -1 für „unbegrenzt“, -2 für „unbekannt“ und -3 für „unbekannt, aber noch nicht leer“, vor allem die -3 tritt in der Praxis häufig auf. Zwar existiert im Namensraum X690 der SNMP Bibliothek eine eigene Integerklasse, in die die Abfrageergebnisse konvertiert werden können, jedoch kann dieser offensichtlich nicht mit negativen Werten umgehen. In der Praxis wurden hier oft unrealistische und auch widersprüchliche Abfrageergebnisse beobachtet, wenn beispielsweise der Farbfüllstand größer als die maximale Kapazität war. Eine direkte Konvertierung in den CTS Integertyp ist auch nicht möglich. Glücklicherweise kann man jedoch bei X690.Integer Objekten den tatsächlichen Wert in Form eines Bytefeldes abrufen. Wenn hier das führende Byte größer als 127 ist, so handelt es sich eigentlich um einen negativen Wert, da das erste Bit für das negative Vorzeichen steht und es in diesem Fall gesetzt ist. Daraufhin wurde eine eigene Hilfsfunktion erstellt, die X690.Integer Objekte korrekt in Integer Werte umwandelt. Offensichtlich handelt es sich bei diesem Problem um einen echten Fehler in der SNMP Bibliothek.

5.4.3.3. Nicht antwortende SNMP Agenten

Da SNMP auf UDP basiert, ist nicht garantiert, dass eine Anfrage den Zielhost überhaupt erreicht, oder dass der Sender eine Fehlernachricht bekommt, wenn etwas fehlgeschlagen ist. So passiert es in der Praxis durchaus, dass ein SNMP Agent auf eine Anfrage nicht antwortet. Da das Drucker-Inventarsystem als eigenständiger Dienst implementiert wird, muss auch diese Möglichkeit berücksichtigt werden. Der Dienst sollte in dem Fall weder abstürzen oder sonst seine Arbeit einstellen. Leider ist aber bei der Verwendung der SNMP Bibliothek genau dies der Fall. Wenn eine SNMP Abfrage gesendet wurde wird der jeweilige Thread angehalten und er wartet auf eine Antwort. Kommt diese Antwort jedoch nicht wartet der Thread solange, bis er manuell abgebrochen wird. Dieses Verhalten ist für einen Dienst jedoch unannehmbar.

Der Grund für dieses Verhalten liegt an der Verwendung der UdpClient Klasse aus dem Namensraum System.Net.Sockets des .NET Frameworks. Der UdpClient ist eine sehr einfach zu verwendende Klasse, mit deren Hilfe UDP Pakete verschickt und empfangen werden können. Nachteilig ist aber genau das beschriebene Verhalten, dass die Empfangsmethode immer auf eine Antwort wartet. Idealerweise würde man für den Empfang ein Timeout setzen, eine Zeitspanne über die längstens auf eine Antwort gewartet werden würde, jedoch ist dieses bei der UdpClient Klasse nicht vorgesehen. Im selben Namensraum System.Net.Sockets steht aber auch die Klasse Socket zur Verfügung, die zwar komplizierter zu verwenden ist als die UdpClient Klasse, aber mehr Möglichkeiten bietet, unter anderem kann man hier eben ein Timeout für den Empfang setzen.

Da die SNMP Bibliothek auch im Quelltext vorliegt, wurden zur Lösung des Problems einfach alle Aufrufe der UdpClient Klasse ausfindig gemacht und durch äquivalente Aufrufe der Socket Klasse ersetzt. Zusätzlich kann man nun beim Starten einer Snmp.ManagerSession ein Timeout angeben. Standardmäßig wird eine Sekunde gewartet, was sich in der Praxis als völlig ausreichend darstellt.

Eine Alternative Lösung wäre gewesen Aufrufe der SNMP Bibliothek in separaten Threads durchzuführen, deren Laufzeit von einem weiteren Thread überwacht wird und der sie bei Bedarf beenden kann. Eine vergleichbare Lösung wurde beim Vorgängerprojekt beim Auslesen von Clients per WMI verwendet. Die Lösung mit dem Timeout vermeidet jedoch unnötiges Multithreading und den damit verbundenen erhöhten Rechen- und Zeitaufand.

6. Der Drucker-Inventar-Dienst

6.1. Allgemeines

Der Drucker-Inventar-Dienst hat zum einen die Aufgabe, das lokale Netzwerk des Standortes nach Netzwerkdruckern zu durchsuchen. Zum anderen soll er diese Drucker per SNMP auslesen und die ermittelten Daten in die Datenbank eintragen. Des Weiteren unterhält der Dienst auch eine Kommunikation mit dem Web Service, über den auch Druckernamen zum Auslesen manuell übergeben werden können. Ebenso erhält man darüber auch Zugriff auf die wesentlichen Funktionen und Konfigurationsinformationen des Dienstes.

Ähnlich wie beim Vorgängerprojekt, dem Inventar-Server, werden auch beim Drucker-Inventar-Dienst Warteschlangen für die Übergänge zwischen den Arbeitsschritten verwendet, da auch hier der Zeitaufwand der jeweiligen Schritte nicht vorhersehbar ist. Konkret sind das die Auslesewarteschlange und die Datenbankwarteschlange. Die Auslesewarteschlange nimmt entweder Hostnamen oder IP-Adressen auf, die als Drucker identifiziert und gegebenenfalls ausgelesen werden sollen. Bei Erfolg werden jeweils Druckerobjekte, die die ausgelesenen Daten enthalten, in die Datenbankwarteschlange eingetragen.
Die Einträge der Auslesewarteschlange werden von mehreren Auslesethreads bearbeitet, deren Anzahl, zur Anpassung an die zur Verfügung stehende Hardware, über die Konfiguration eingestellt wird. Da die Auslesethreads die meiste Zeit mit dem Warten auf Antworten über das Netzwerk beschäftigt sind, erreicht man hier einen enormen Performancegewinn, indem man mehrere Threads parallel warten lässt. Das Schreiben der Druckerdaten in die Datenbank wird nur über einen einzelnen Thread realisiert, da dies etwas einfacher ist und auch kein nennenswerter Performancegewinn durch Multithreading zu erwarten ist.
Der größte Unterschied zum Inventar-Server besteht in der Art und Weise, wie Einträge in die Auslesewarteschlange gelangen. Da sich die Netzwerkdrucker nicht selbstständig beim Drucker-Inventar-Dienst anmelden können und auch sonst keine Informationen zur Verfügung stehen, wie viele Drucker sich wo im Netzwerk befinden, trägt ein Scheduler[11] zu gewissen Zeiten sämtliche IP-Adressen des Standortes in die Auslesewarteschlange ein. Auch können damit, zeitlich gesteuert, die Hostnamen der Drucker aus der Datenbank erneut in die Auslesewarteschlange eingetragen werden, sofern bereits Druckereinträge in der Datenbank vorhanden sind. Der dritte Weg ist das manuelle Eintragen über das Web Interface.

[11] Begriff basiert auf Engl. Schedule - Zeitplan

Folgende Abbildung soll den groben Aufbau illustrieren:

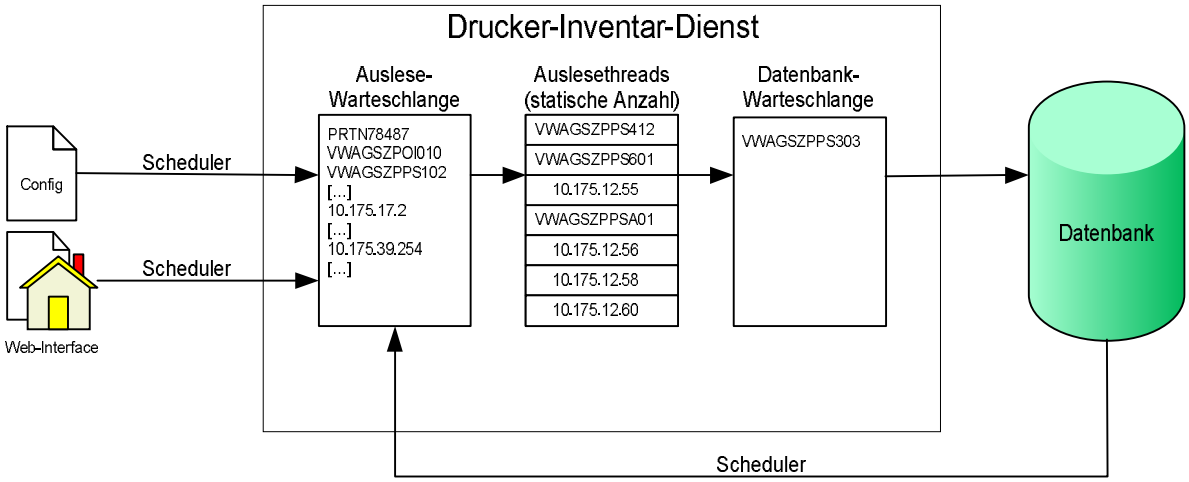

Abbildung 5.: Ablaufplan Druckerinventarisierung

Der Vorgang des Eintragens der IP-Adressen des Standortes, dessen Adressbereich konfigurierbar ist, in die Auslesewarteschlange wird im Folgenden als Suche bezeichnet. Das Eintragen der bereits bekannten Hostnamen aus der Datenbank in die Auslesewarteschlange wird als Aktualisierung bezeichnet.

Da der Ablauf der Druckerinventarisierung sich konzeptionell vom Vorgängerprojekt unterscheidet, sowie aus Wartungs- und Kompatibilitätsgründen ist der Drucker-Inventar-Dienst als eigenständiger Windows-Dienst entwickelt, der zusätzlich zum Inventar-Server, aber auch allein installiert und betrieben werden kann.
Ein Windows-Dienst ist keine interaktive Anwendung und soll weitgehend unbemerkt und wartungsfrei im Hintergrund laufen. Von daher ist es notwendig, dass der Dienst selbstständig mit allen erdenklichen Fehlern umgehen kann. Insbesondere die Kommunikation über das Netzwerk kann eine Fehlerquelle darstellen, da nicht mit Sicherheit vorhergesehen werden kann, ob oder wann ein anderes Netzwerkgerät auf eine Anfrage antwortet, so kann es möglicherweise abgeschaltet, defekt, überlastet, falsch konfiguriert oder vom Netzwerk abgetrennt sein oder auch das verwendete Protokoll nicht verstehen. Und selbst wenn eine Antwort kommt, kann sie unter Umständen nicht dem erwarteten Format entsprechen oder unvollständig sein.

6.2. Die Threads

Nach seinem Start erzeugt der Drucker-Inventar-Dienst mehrere verschiedene Threads, die die Kernfunktionen des Dienstes übernehmen. Da die Threads parallel ausgeführt werden können, wird eine besonders schnelle Abarbeitung der Aufgaben ermöglicht.

Folgende Threads werden vom Dienst erzeugt:

- Main-Thread:
  Die einzige Funktion des Main-Threads ist die Instanzierung des Schedulers, der alle folgenden Aufgaben übernimmt.
- Scheduler-Thread:
  Seine Aufgabe ist es, zu konfigurierbaren Zeitpunkten die Eintragung von neuen IP-Adressen oder Hostnamen in die Auslesewarteschlange auszulösen.
- Restart-Thread:
  Der Restart-Thread existiert nur kurzzeitig und wird erzeugt, wenn der Dienst auf seinen Startzustand zurückgesetzt werden soll.
- Datenbank-Thread:
  Dieser Thread arbeitet die Datenbankwarteschlange ab und schreibt die Druckerdaten in die Datenbank.
- Auslese-Threads:
  Es wird jeweils versucht, den Zielhost als Drucker zu identifizieren, diesen auszulesen und die gewonnenen Daten in die Datenbankwarteschlange einzutragen.

6.2.1. Der Main-Thread

Der Main-Thread ist der Einstiegspunkt des Dienstes, er wird beim Dienststart automatisch ausgeführt. Er erzeugt eine Instanz des Schedulers, der daraufhin alle weiteren Aufgaben des Dienstes übernimmt.

6.2.2. Der Scheduler-Thread

Mit dem Scheduler wird auch der Scheduler-Thread gestartet. Er ermittelt, anhand des in der Konfiguration eingestellten Zeitplans, jeweils den nächsten Zeitpunkt für eine Suche und eine Aktualisierung und wird daraufhin pausiert. Alle 60 Sekunden wird die Pausierung aufgehoben,

und der Thread überprüft, ob einer der beiden Zeitpunkte erreicht ist. Sobald dies der Fall ist, wird die entsprechende Aktion eingeleitet, der nächste Zeitpunkt für diese Aktion bestimmt und der Thread wieder pausiert.

6.2.3. Der Restart-Thread

Wenn über das Web Interface eine Änderung der Konfiguration vorgenommen wurde, wird auch kurzzeitig ein neuer Restart-Thread erzeugt. Er beendet die anderen Threads des Dienstes, leert die Warteschlangen, und startet sie anschließend neu. Da sich neben den anderen Konfigurationsdaten auch der Port für die Kommunikation mit dem Web Service geändert haben kann, muss auch der alte Port abgemeldet und das .NET Remoting neu gestartet werden. Dies ist auch der Grund, diese Aufgabe als eigenen Thread zu realisieren, der nach Beenden der Verbindung zum Web Service noch weiterarbeitet, da das Abmelden des Remoting Ports bei aktivem Remoting nicht möglich ist.

6.2.4. Der Datenbank-Thread

Dieser Thread versucht, nacheinander alle Druckerobjekte aus der Datenbankwarteschlange herauszunehmen. Ist die Warteschlange leer, so wird der Thread in den Ruhemodus versetzt, der erst wieder aufgehoben wird, wenn ein neues Druckerobjekt mittels der entsprechenden Einfügefunktion in die Datenbankwarteschlange eingefügt wurde. Wenn jedoch ein Druckerobjekt erfolgreich aus der Warteschlange herausgenommen wurde, wird dessen Datenbank-Einfügefunktion aufgerufen. Im Abschnitt über die Drucker Klasse wird noch genauer darauf eingegangen.

6.2.5. Der Auslese-Thread

Der Auslese-Thread arbeitet im Prinzip recht ähnlich wie der Datenbank-Thread, indem er sich selbstständig die zu bearbeitenden Objekte aus der entsprechenden Warteschlange holt. Ist kein Objekt mehr vorhanden, so wird der Thread in den Ruhemodus versetzt und wieder aktiviert, sobald mindestens ein neues Objekt hinzugefügt wurde.

Die Objekte können einerseits IP-Adressen sein, die für eine neue Suche über das Netzwerk in die Auslesewarteschlange eingefügt wurden, oder es sind Hostnamen von Druckern, die zwecks Aktualisierung automatisch, oder manuell über das Web Interface eingefügt wurden. Um die Reaktion des Dienstes auf manuelle Eingaben zu erhöhen, werden Hostnamen mit höherer Priorität bearbeitet, als IP-Adressen. Tatsächlich werden die Hostnamen sogar intern in einer

Stapelstruktur gespeichert, die einfach vor der Warteschlange behandelt wird, die die IP-Adressen enthält. Daher ist die Auslesewarteschlange auch keine echte Warteschlange.

Wird ein neues Objekt bearbeitet, so wird zunächst überprüft, ob eine Kommunikation mit dem entsprechenden Zielhost möglich ist. Dazu wird ein einfaches ICMP-Paket („Ping“) verschickt und anhand der Antwort bestimmt, ob der Zielhost gerade aktiv ist. Kommt innerhalb eines Timeouts (ca. 1-2 Sekunden) keine Antwort, so wird davon ausgegangen, dass das Ziel nicht aktiv ist oder gar nicht existiert, und das Objekt wird verworfen.

Ist eine Verbindung möglich, so wird jeweils der zur IP-Adresse gehörende Hostname oder umgekehrt die IP-Adresse des Zielhosts über den DNS bestimmt. Anhand des Hostnamen wird überprüft, ob der Zielhost ein Drucker sein kann, dazu muss der Name mit einem bestimmten Präfix beginnen, der dem Namensschema des Standortes entspricht. Normalerweise ist dies am Standort Salzgitter die Zeichenkette „VWAGSZP“, andere oder weitere erlaubte Präfixe können aber auch über die Konfiguration angegeben werden. Wenn der Zielhost hiernach kein Drucker sein kann, so wird er ebenfalls verworfen, ansonsten wird das Auslesen des Druckers per SNMP eingeleitet. War auch dies erfolgreich, so werden die gewonnenen Daten in Form eines Druckerobjekts in die Datenbankwarteschlange eingetragen.

In der Regel existieren unabhängig voneinander mehrere dieser Threads gleichzeitig. Anders als beim Inventar-Server ist deren Anzahl aber fest durch die Konfiguration vorgegeben, die sich jedoch auch zur Laufzeit manuell anpassen lässt.

6.3. Die Hauptklassen

Der Drucker-Inventar-Dienst ist weitestgehend wie eine Klassenbibliothek aufgebaut, um die Arbeitsweise besser zu verstehen wird hier kurz auf die wichtigsten Klassen eingegangen.

6.3.1. Der Scheduler

Die Scheduler Klasse ist von besonders zentraler Bedeutung. Um den Dienst zu starten braucht man letztlich nur einen Scheduler aus einem ansonsten leeren Dienst Projekt instanzieren. Der Scheduler erzeugt dann die Auslesewarteschlange und die Datenbankwarteschlange und er startet den Scheduler-Thread. Außerdem werden das .NET Remoting gestartet und die Ereignisbehandlungen für die Kommunikation mit dem Web-Service festgelegt. Hier ist auch die Neustart Funktion definiert, die vom Web Service beim ändern der Konfiguration eingeleitet wird. Diese Funktion wird von dem temporären Restart-Thread aufgerufen.

6.3.2. Die DBQueue

Die Klasse DBQueue enthält die Datenbankwarteschlange als privates Objekt, Zugriffe darauf können nur über die Methoden dieser Klasse durchgeführt werden. Dadurch ist sichergestellt, dass die Zugriffe synchronisiert werden und dass auch nur Drucker Objekte eingefügt werden können. Bei der Initialisierung wird auch der lokale Datenbank-Thread gestartet, der als einziger auch wieder Drucker Objekte aus der Warteschlange entfernen kann.

6.3.3. Die AusleseQueue

Diese Klasse enthält die Auslesewarteschlange, die wie bereits in Kapitel 6.2.5 erwähnt eigentlich aus einer Warteschlange für IP-Adressen und einem Stapel für Hostnamen besteht. Da auch hier Zugriffe nur über die Methoden dieser Klasse erfolgen wird auch die interne Struktur verborgen und das Entnehmen von Objekten erfolgt auch hier nur über die lokalen Auslese-Threads. Zum initialisieren muss hier die Referenz auf ein DBQueue Objekt übergeben werden, damit die Auslese-Threads die ausgelesenen Drucker daran übergeben können.

6.3.4. Der Drucker

Mithilfe der Drucker Klasse werden von den Auslese-Threads die Drucker Objekte erzeugt, die schließlich an die Datenbankwarteschlange übergeben werden. Tatsächlich findet das Auslesen auch erst beim erzeugen eines Drucker Objekts statt, dazu wird beim initialisieren auf die bereits im Kapitel 5.4.2. erwähnte AusleseSession zurückgegriffen. Zur Initialisierung muss auch entweder ein Hostname oder eine IP-Adresse übergeben werden. Hostname, IP-Adresse und alle Druckerwerte liegen hier als öffentliche Variablen vor, die zunächst mit Nullwerten gefüllt sind und nach erfolgtem Auslesen mit den entsprechenden Ausleseergebnissen überschrieben werden. Außerdem wird sofort nach dem Auslesen versucht den Druckerhersteller anhand der ausgelesenen Daten zu bestimmen und es wird aus den drei verschieden Druckerstatus Variablen nach dem in RFC 1759 beschriebenen Verfahren ein aussagekräftigerer, übergreifender Druckerstatus ermittelt. Es ist hier ebenfalls eine Methode implementiert, die den Datenbankzugriff realisiert und die Druckerinformationen an die Datenbank übergibt. Diese Methode wird von dem Datenbank-Thread aufgerufen.

6.3.5. DITransfer

Die DITransfer Klasse dient zum erzeugen eines gemeinsamen Objekts für den Drucker-Inventar-Dienst und den Web Service. Der Dienst erzeugt die einzige Instanz des Objektes und der Web Service ein Proxyobjekt. Die Funktionen werden per .NET Remoting an das reale Objekt weitergeleitet. Hier werden schließlich die Ereignisse erzeugt, auf die im Scheduler reagiert wird.

6.3.6. Die Konfiguration

Zuletzt ist noch die Config Klasse erwähnenswert. Hier sind alle konfigurierbaren Daten in öffentlichen Variablen gespeichert. Um sicherzustellen, dass nur ein Konfigurationsdatensatz existiert und dass dieser auch korrekt initialisiert wird, ist die Klasse nach dem Singleton Muster aufgebaut. Zugriffe können nur über die Instance Methode geschehen, die die einzige Instanz beim ersten Zugriff auch gleich initialisiert. Außerdem sind hier Methoden implementiert, die die XML Konfigurationsdatei lesen und auch in diese schreiben können, sowie Methoden, die die aktuellen Konfiguationsdaten über die DITransferklasse an den Web Service übertragen und auch neue Daten daher übernehmen können.

6.4. Performanceüberlegungen

Ein nicht unwichtiger Aspekt bei einer Anwendung wie dem Drucker-Inventar-Dienst ist dessen Performanz. Dieses kann durchaus entscheidend sein, ob der produktive Einsatz des Dienstes überhaupt sinnvoll ist. Leider sind konkrete Vorhersagen bei einer solch komplexen Aufgabe kaum möglich, daher sind praktische Tests hierbei umso wichtiger.

6.4.1. Beobachtungen

Bereits sehr früh beim konzipieren des Drucker-Inventar-Dienstes war klar, dass das Durchsuchen des Firmennetzwerks nach Druckern die zeitaufwändigste Teilaufgabe sein würde und genau hier Optimierungen nötig und lohnend sein würden. Ein erster, weitgehend funktionierender Prototyp des Dienstes, der aber noch auf einem einzelnen Thread basierte, hätte nach vorsichtiger Hochrechnung noch mehrere Wochen gebraucht, um das Firmennetz nur einmal komplett zu durchsuchen. Für den Produktiveinsatz war das inakzeptabel, auch wenn die Prozessorauslastung des Prototyps durchschnittlich sehr gering war. Vor allem der Einsatz von Multithreading erhöht bei dieser Aufgabe die Geschwindigkeit, aber leider auch die Prozessorauslastung, daher ist die Anzahl der Auslesethreads die wichtigste Einstellung, um den Dienst zu skalieren.

Basierend auf der für den praktischen Einsatz vorgesehenen, finalen Version des Drucker-Inventardienstes wurden einige Messungen in Abhängigkeit zur Auslesethreadzahl gemacht. Hauptaugenmerk gilt hier der benötigten Zeit für das Durchsuchen eines Netzbereichs und dem Aktualisieren der gefundenen Drucker, aber auch die Prozessorlast wurde während des Vorganges mithilfe des Windows Task-Managers beobachtet.

Ausgeführt wurde der Dienst dazu auf einem der Entwicklungssysteme, das mit einem einzelnen Pentium 4 mit 1,5 GHz und 512 MB RAM ausgestattet war. Die Datenbank lief auf einem anderen Entwicklungssystem, das mit einem Celeron mit 566 MHz und 256 MB RAM ausgestattet war. Das Webinterface wurde hierbei nicht verwendet. Beide Rechner waren per 100 MBit/s Ethernet an das Firmennetzwerk angeschlossen. Durchsucht wurde immer derselbe 32768 IP-Adressen umfassende Teilbereich des Firmennetzwerkes, das insgesamt 65536 IP-Adressen umfasst.

Ergebnisse:

Anzahl Auslesethreads	Zeit zum Aktualisieren	Zeit zum Durchsuchen	Prozessorlast Durchschnitt ca.	Prozessorlast-spitzen ca.
5	84 Sek	127:20 Min	5-15%	30-100%
10	49 Sek	64:42 Min	10-25%	30-100%
15	52 Sek	44:05 Min	20-45%	60-100%
20	45 Sek	33:52 Min	30-50%	100%
25	56 Sek	27:56 Min	35-60%	100%
50	62 Sek	23:13 Min	0-100%	100%

Gefunden wurden jeweils 357 bis 365 Drucker und der Speicherverbrauch des Dienstes pendelte sich nach anfänglich 10 MB bei etwa 34 MB ein. Die durchschnittliche Prozessorlast gibt hier an, zwischen welchen Prozentwerten diese die meiste Zeit während des Tests schwankt. Interessant ist das Verhalten bei 50 Auslesethreads, wo die Last praktisch ständig über den gesamten Bereich hin und her schwankt. Die Werte für die Prozessorlastspitzen geben die Größen an, in denen sich gelegentliche Spitzenwerte während der Ausführung bewegen, wie sie z.B. auftreten können, wenn ein gefundener Drucker gleich in die Datenbank eingetragen wird. Bei allen Messungen traten hier durchgängige Vollauslastungen auf, während bestimmte, eigentlich nicht erreichbare Teilbereiche des Netzwerkes durchsucht wurden. Auch während des Aktualisierungsvorganges traten zunächst durchgängige Vollauslastungen auf, die jedoch bald bis zum Ende des Vorganges auf Werte zwischen 10% und 20% abfielen. Eine Ausnahme bildete hier der Test mit 5 Auslesethreads, wo beim Aktualisieren maximal 80% Auslastung erreicht wurde, aber auch kein besonderer Rückgang der Werte bis zum Schluss beobachtet wurde.

Die Netzwerkauslastung wurde bei den Tests nicht so intensiv betrachtet, sie war jedoch durchgängig sehr gering. Laut dem Task-Manager befanden sich selbst kurze Spitzenwerte noch deutlich unter 5% und wurden auch nur bei den Datenbankzugriffen über das Netzwerk erreicht, insbesondere also wenn beim Aktualisierungsvorgang praktisch permanent Daten an die Datenbank gesendet werden.

6.4.2. Erläuterungen

Beim Betrachten der Ergebnisse für den Aktualisierungsvorgang wird schnell klar, dass in dieser Umgebung spätestens ab 10 Auslesethreads diese nicht mehr der begrenzende Faktor sind. Das Abflachen der Auslastung zum Ende hin bedeutet, dass am Schluss nur noch der Datenbankthread aktiv ist, der die Daten nicht so schnell schreiben konnte, wie sie geliefert wurden. Erstaunlich ist, dass Aktualisierungszeit mit zunehmender Threadzahl ab 10 scheinbar schon wieder ansteigt. So kostet hier vermutlich der Synchronisierungsaufwand z.B. beim Zugriff auf die Auslese- oder Datenbankwarteschlange zusätzliche Zeit. Insgesamt gesehen geht der Aktualisierungsvorgang der ca. 360 Drucker jedoch erstaunlich schnell vonstatten, selbst bei nur bei nur 5 Auslesethreads wird im Schnitt weniger als eine viertel Sekunde pro Drucker zum erneuten Auslesen benötigt.

Die Zeit zum Durchsuchen des Netzwerkes nimmt bis zum Test von 25 Auslesethreads fast antiproportional zur Threadanzahl ab und die durchschnittliche Prozessorauslastung nimmt ungefähr proportional zur Threadanzahl zu. So bedeutet die doppelte Threadzahl fast die halbe Zeit fürs Durchsuchen und die doppelte durchschnittliche Prozessorlast. Jedoch gelingt dies mit zunehmender Threadzahl immer weniger effizient. Spätestens der Test mit 50 Auslesethreads fällt hier völlig aus der Reihe. Die Beobachtung der Prozessorlast, die ständig nur zwischen 0% und 100% schwankt, deutet darauf hin, dass der Rechner die Threads kaum noch vernünftig verwalten bzw. synchronisieren kann.

Die Beobachtung der Vollauslastung bei dem Durchsuchen bestimmter, nicht erreichbarer Teilbereiche des Netzwerkes, lässt sich dadurch erklären, dass hier vermutlich der zugehörige Router für den Teilbereich die IP-Adressen beim anpingen sofort als inaktiv oder nicht erreichbar meldet. Dadurch warten die Auslesethreads das Timeout des Pings nicht ab und testen sofort die nächste ebenfalls inaktive Adresse, bis sie wieder in einen erreichbaren Netzbereich gelangen.

6.4.3. Empfehlungen

Beim praktischen Einsatz ist zu beachten, dass auf anderer Hardware natürlich andere Werte erzielt werden, aber das grundsätzliche Verhalten höchstwahrscheinlich sehr ähnlich ist. Man kann davon ausgehen, dass Server auf Multiprozessorbasis das Multithreading zwar effizienter beherrschen, jedoch auch keine Wunder bewirken werden. Der erste Versuch, den Drucker-

Inventar-Dienst auf einem Dual Pentium Pro Server mit je 200MHz mit 30 Auslesethreads produktiv zu betreiben, hat diesen beispielsweise hoffnungslos überfordert.
Besonders wenn der Server neben dem Ausführen des Drucker-Inventar-Dienstes noch andere Aufgaben zu erfüllen hat, sollte man daher eher nur eine einstellige Auslesethreadzahl verwenden. Das Durchsuchen des gesamten Firmennetzes sollte trotzdem noch problemlos innerhalb eines Arbeitstages möglich sein und es steht noch genügend Rechenzeit für die anderen Aufgaben zur Verfügung. Da die Aktualisierung auch mit geringer Threadzahl sehr schnell abläuft und auch bei laufender Suche angesetzt werden kann, kann diese problemlos auch mehrmals täglich angesetzt werden.

Sofern solche IP Adressbereiche bekannt sind, die vom Server aus gar nicht erreichbar sind bzw. in denen sich keine Drucker befinden können, sollte man diese Bereiche mittels der Konfiguration des Dienstes gar nicht erst zum Durchsuchen vorsehen. Dadurch würde man das Problem der Vollauslastung in diesen Bereichen vermeiden und das Durchsuchen würde insgesamt auch schneller ablaufen.

7. Die Datenbank

7.1. SQL und Server: Eine kurze Zusammenfassung

Die Abkürzung SQL steht für *Structured Query Language* und bezeichnet eine Datenbankabfragesprache, die am Anfang der siebziger Jahre von IBM entwickelt wurde. SQL wurde weiterentwickelt, sodass mit dieser Sprache nicht nur Datenbanken abgefragt, sondern auch erzeugt und administriert werden konnten.

Eine weitere Entwicklung von SQL war die Entstehung von firmenspezifischen SQL-Dialekten, die gleichzeitig eine Erweiterung des Sprachumfangs mit sich brachten. Zwei bekannte und verbreitete Dialekte sind:

- PL/SQL der Firma Oracle
- Transact-SQL von Microsoft

Der SQL Server ist ein relationales Datenbankentwicklungssystem, mit dem diese Abfragen durchgeführt und angewandt werden können. Der SQL Server 2000 stellt die momentan letzte Entwicklungsstufe der Microsoft SQL Server-Reihe dar.

7.2. Der Datenbank Server

Der Datenbank-Server, der in diesem Projekt benutzt wird, ist ein Microsoft SQL Server 2000. Es gibt verschiedene Versionen dieses Produktes, mit unterschiedlichen Leistungsmerkmalen. Die kleinste Version ist die kostenlose MSDE. Diese Abkürzung stand ursprünglich für *Microsoft Data Engine*. Sie basierte technologisch auf dem MS SQL Server in der damals aktuellen Version 7 und in Access 2000 bzw. XP Professional ist die MSDE 2000 auf der Grundlage des MS SQL 2000 enthalten.

Die MSDE verfügt über die folgenden Leistungsmerkmale:

- Sie ist unter Windows 9.x, ME, NT, 2000 und XP in allen Lieferversionen lauffähig.
- Sie unterstützt 2 CPUs und maximal 2 Gbyte RAM.
- Die maximale Datenbankgröße beträgt 2 Gbyte.

Anhand dieser Eigenschaften ist schon zu erkennen, dass die Leistung der MSDE in vielen Fällen und auch für mittelgroße Netzwerke vollständig ausreichend ist. Die MSDE bietet fast alle Möglichkeiten der Vollversion, hat jedoch nicht die grafischen Entwicklungs- bzw. Verwaltungstools wie *Enterprise Manager und Query Analyser*.
Die MSDE ist für Entwickler gedacht und vor allem unter dem Gesichtspunkt der Lizenzkosten stellt sie eine überlegenswerte Alternative dar.

Neben der MSDE bietet Microsoft sechs weitere Editionen an, von denen vier grundlegend wichtig sind.
Die Personal Edition ist schon ein vollwertiger SQL Server und als Entwicklerplattform gedacht. Diese unterstützt ebenfalls Einzel- und Multiprozessorsysteme mit bis zu 4 CPUs und 2 GB RAM. Sie ist jedoch nur für kleinere Projekte geeignet.

Diese Edition kann auf den folgenden Betriebssystemen eingesetzt werden:

- Windows 95/98/Me, Windows NT 4.0 Workstation, Windows 2000 Professional und Windows XP Home Edition.
- Alle höheren Windows-Betriebssysteme.

Eine andere Version von SQL Server 2000 ist die Standard Edition, die auf einer gleichen Anzahl von Einzel- und Multiprozessorsystemen wie die Personal Edition lauffähig ist. Die Enterprise Version ist für größere Unternehmen mit den entsprechenden Anwendungen gedacht. Sie unterstützt im Vergleich zur Standard Edition bis zu 32 CPUs und 64 GB RAM. Diese Version sowie die Standardversion können auf den folgenden Betriebssystemen eingesetzt werden:

- Windows NT Server 4.0, Windows 2000 Server, Windows XP Server und höhere Versionen.

Bei der vierten Edition handelt es sich um die Developer Version, die alle Funktionen der Enterprise Edition umfasst. Sie hat einen besonderen Lizenzvertrag, in dem jede produktive Nutzung verboten ist, d.h., sie ist eine reine Entwicklerumgebung. Aufgrund der ständigen Verfügbarkeit der Datenbank in diesem Projekt war die Nutzung der Standard Edition geeignet und ausreichend.

7.3. Transact-SQL

7.3.1. Allgemein

Transact-SQL steht für *Transactional Structured Query Language* und wurde von der Firmen Sybase und Microsoft entwickelt. Sie stellt eine Erweiterung zu Standard SQL (ANSI 92) dar. Die Erweiterungen umfassen vier hauptsächliche Bereiche mit dem Ziel, den Programmierer von zeitaufwendigen Programmieraufgaben zu befreien. Diese vier Bereiche sind:

- Erstellung von SQL Server-Objekten (z. B. Datenbanken, Tabellen usw.)
- Verarbeitung von Daten, wie Abfragen, Aktualisieren, Löschen oder Einfügen von Daten sowie der Aufruf von gespeicherten Prozeduren.
- Programmierung, die zu den Erweiterungen zählt u.a. Datentypen zur Optimierung der Datenspeicherung, Variablen in SQL-Anweisungen, Ausführung gespeicherter Prozeduren und Befehle zur Flusssteuerung durch Unterstützung von If…Else Anweisungen sowie Schleifen.
- SQL Server-Administration, bei der Benutzberechtigungen für Datenbanken oder Tabellen eingerichtet werden können.

Diese Neuerungen bringen den Vorteil, dass mehr Aufgaben als früher intern in der Datenbank gelöst werden können. Auf diese Weise wird die Kommunikation mit der Datenbank reduziert. Andere Datenbankhersteller haben Ihre Produkte auch mit zahlreichen dieser Erweiterungen bereichert. Das Pendant zu Transact-SQL bei Oracle ist PL/SQL bzw. SQL*Plus, welches eine ähnliche Leistungsfähigkeit hat, aber nicht zu Microsoft kompatibel ist.

7.4. Übersicht Drucker-Datenbank

Die Druckerdatenbank enthält die Daten der gescannten Drucker und ist das Zentrum des Inventarisierungssystems, da sie sowohl vom Inventardienst, als auch vom Web-Interface genutzt wird. Die Datenbank besteht aus vier Tabellen, die hier dargestellt werden. Die erste Tabelle umfasst die allgemeinen Daten eines Druckers, wie z.B. den DNS-Namen des Druckers, seine IP- und MAC-Adresse. Die Beschreibung der Version und der Name der Hardware, das Betriebssystem sowie die Netzwerksoftware befinden sich auch in dieser Tabelle. Andere Daten eines Druckers, die wichtig zu nennen sind, sind Druckerhersteller, der Gerätzustand, die Daten der zuständigen Person des Druckers, Druckerfehler, die Gesamtzahl der gedruckten Seiten und die Anzahl der gedruckten Seiten seit Einschalten des Druckers. Diese Daten sind ebenfalls dieser Tabelle zu entnehmen.

Die zweite Tabelle enthält Information über die Tinten bzw. Toner-Füllstände des Druckers und die entsprechende Beschreibung jedes Füllstandes. In der dritten Tabelle sind sowohl die Daten der Papierfüllstande, als auch die Beschreibung jedes Papierfüllstandes gespeichert. Die vierte und letzte Tabelle umfasst die Information der Papierausgabefächer eines Druckers und ihre entsprechende Beschreibung.

In den letzten drei Tabellen existiert die Möglichkeit, dass ein Drucker ein oder mehrere Füllstände bzw. Papierausgaben haben kann. Dieses Vorwissen ist für die Entscheidung, welcher Art der Beziehungstyp zwischen den Tabellen ist, ausschlaggebend.
So werden die Tabellen von den Füllständen und den Papierausgaben in einer 1:n-Beziehung durch einen Fremdschlüssel mit der Haupttabelle verknüpft.
Um die Daten von außen gut zu kapseln, ist es sinnvoll, die meisten Zugriffe von Triggern zu kontrollieren, oder diese Prozedur gleich durch eine gespeicherte Prozedur durchführen zu lassen. Mit diesen Programmteilen, die hinter der Datenbank liegen, kann nicht nur die Datenintegrität der Datenbank gewährleistet werden, sondern auch die Performance erhöht werden, da alle Prozesse in der Datenbank selbst laufen.

7.5. Verwendete Datenbankfunktionen

7.5.1. Primärschlüssel

Ein Primärschlüssel sollte in jeder Tabelle grundsätzlich definiert werden. Dieser Schlüssel ermöglicht, dass jeder Datensatz innerhalb der Tabelle eindeutig gekennzeichnet werden kann. Über den Schlüssel lässt sich der Datensatz eindeutig identifizieren.
Ein Primärschlüssel entspricht mindestens einer Spalte einer Tabelle, die niemals NULL sein darf, und deren Werte alle einmalig sind. Diese Eigenschaft wird auch als UNIQUE bezeichnet. Wenn eine Tabelle erstellt ist, werden alle diese Einschränkungen durch das Schlüsselwort CONSTRAINT eingeleitet.

Ein Primärschlüssel kann auch aus mehreren Spalten der Tabelle bestehen. Kein anderer Datensatz kann in dieser Tabelle denselben Schlüssel erhalten. Primärschlüsselfelder sind zudem für die Beziehungen zwischen Tabellen wichtig, weil sie sich durch die Einmaligkeit einer Tabellenspalte herstellen lassen und die Datenintegrität schützen.
In Microsoft SQL Server 2000 kann ein Primärschlüssel 16 Spalten enthalten, bei denen keine den NULL-Wert haben darf. Besteht der Primärschlüssel aus mehr als einer Spalte, so muss die Kombination der Werte eindeutig sein.

7.5.2. UNIQUE Einschränkung

In einer Datenbank können Spalten mit der Einschränkung UNIQUE erstellt werden. Diese Eigenschaft verhindert, dass die Werte der betreffenden Spalte innerhalb der Tabelle nicht mehrmals auftreten dürfen. Im Vergleich zu einem Primärschlüssel kann eine als UNIQUE definierte Zeile einer Tabellenspalte einen NULL-Wert annehmen. In Microsoft SQL Server 2000 kann eine UNIQUE Einschränkung 16 Spalten umfassen. Eine Tabelle kann bis zu 250 UNIQUE Einschränkungen enthalten.

7.5.3. Fremdschlüssel

Ein Fremdschlüssel ist eine Spalte oder Kombination aus mehreren Spalten, deren Werte mit dem Primärschlüssel (eindeutigen Schlüssel) in derselben oder einer anderen Tabelle zusammenpassen. Dieser wird auch als verweisender Schlüssel bezeichnet. Der Fremdschlüssel kann nur bereits beinhaltete Werte des Primärschlüssels annehmen.

Fremdschlüssel sind also auch wie Primärschlüssel für die Verknüpfung von Tabellen in relationalen Datenbanken unerlässlich. Es ist nicht mehr möglich, einen Datensatz des Primärschlüssels aus der Datenbank zu löschen, wenn ein Fremdschlüssel einen Wert des zugehörigen Primärschlüssels angenommen hat. Es sei denn, die Referenz wird entfernt.

Ein Fall, der schnell zu Inkonsistenzen führen kann, tritt ein, wenn ein Fremdschlüssel Spalten mit NULL-Werte enthalten darf. Ist dies der Fall, wird der Fremdschlüssel den Primärschlüssel nur referenzieren, wenn dieser keine NULL-Werte enthält. Beinhaltet also eine der Spalten den NULL-Wert, dann können die restlichen Spalten beliebige Werte annehmen. Außerdem ist ein einfacher Fremdschlüssel nicht einmalig, sondern kann die Werte des Primärschlüssels beliebig oft referenzieren. Das kann sich ändern, wenn der Schlüssel mit der UNIQUE- und Primärschlüssel-Einschränkung kombiniert wird. Der Microsoft SQL Server verfügt über 253 Fremdschlüssel-Einschränkungen für eine Tabelle.

7.5.4. Kaskadieren von Fremdschlüsseln

Microsoft SQL Server 2000 stellt zwei neuen Möglichkeiten zur Verfügung, die die Beziehung zwischen Primär- und Fremdschlüssel vereinfachen. Dies sind ON DELETE CASCADE und ON UPDATE CASCADE.

ON DELETE CASCADE dient zum Löschen der Zeile aus einer Tabelle, die mit dem gelöschten Primärschlüssel der zugeordneten Primärtabelle übereinstimmen. ON UPDATE CASCADE aktualisiert den Fremdschlüssel so, dass dieser mit dem neuen Wert für den zugeordneten Primärschlüssel übereinstimmt.

7.5.5. 1:n Beziehung

In einer 1:n Beziehung können einem Datensatz in Tabelle A mehrere Datensätze der Tabelle B zugeordnet sein, aber einem Datensatz der Tabelle B ist nie mehr als ein Datensatz von Tabelle A zugeordnet. In der Drucker-Datenbank kann ein Drucker mehrere Papierausgaben haben.
Deshalb wird hier dieser Beziehungstyp genutzt. Um eine 1:n Beziehung herstellen zu können, muss die Tabelle, die auf der 1-Seite der Beziehung steht, einen Primärschlüssel oder einen eindeutigen Index haben. Die 1:n Beziehung ist der häufigste Beziehungstyp.

Abbildung 6.: 1:n Beziehung

7.5.6. Gespeicherte Prozeduren

Die gespeicherten Prozeduren sind Funktionen, die Gruppierungen von SQL-Anweisungen enthalten, wie Funktionen in einer Programmiersprache aufgerufen und serverseitig ausgeführt werden.
In Transact-SQL ist die Übergabe von Ein- und Ausgabeparameter, While-Schleifen, If-Anweisungen und verschiedene Datenbankanweisungen möglich.

Einen großen Vorteil der gespeicherten Prozeduren liegt in ihrer Anwendung bei einem ständigen Datenaustausch mit dem Server, wenn große Mengen von SQL-Anweisungen auf einem Datenbank-Server über ein Netzwerk ausgeführt werden. Dadurch kann sich schnell eine erhebliche Netzbelastung ergeben.

7.5.7. Indizes

Die Indizes ermöglichen anhand von Schlüsselwerten den schnellen Zugriff auf Daten in den Zeilen einer Tabelle. Der SQL Server 2000 unterstützt gruppierte Indizes, die durch das Schlüsselwort CLUSTERED erstellt sind. Ein gruppierter Index bestimmt die physikalische Reihenfolge der Datensätze in der Tabelle. Diese Reihenfolge ist gleich der logischen Reihenfolge der Tabelle. In jeder Tabelle lässt sich nur ein gruppierter Index erstellen. Normalerweise wird dafür der Primärschlüssel als ein gruppierter Index erstellt, um die Ausführungsgeschwindigkeit von Abfragen zu optimieren.

7.5.8. Trigger

Triggern sind ähnlich einer gespeicherten Prozedur und sind immer an Tabellen gebunden. Diese werden ausgeführt, wenn das Ereignis eintritt, für das sie erstellt worden sind.
Diese können für Einfüge-, Lösch- und Aktualisierungsaktionen verwendet werden. Das entspricht INSERT-, DELETE-, und UPDATE-Triggern. Bei der Verwendung von Triggern sind die folgenden Einschränkungen zu beachten:

- Triggern können nicht auf temporären Tabellen erzeugt werden.
- Triggern müssen sich auf Tabellen in der aktuellen Datenbank beziehen.
- Triggern lassen sich nicht auf Sichten erzeugen.
- Beim Löschen einer Tabelle werden alle mit dieser Tabelle verbundenen Trigger automatisch gelöscht.

7.5.9. Transaktionen

Eine Transaktion ist eine Gruppe von Datenbankoperationen z. B. INSERT- und UPDATE-Anweisungen, die zu einer logischen Reihenfolge gruppiert sind. Diese Operationen überführen die Datenbank von einem konsistenten Zustand in einen eventuell veränderten Zustand, wobei das *ACID-Prinzip* befolgt werden muss.

7.5.9.1. ACID-Eigenschaften

- **Atomarität**
  Transaktionen können entweder ganz erfolgreich oder überhaupt nicht ausgeführt werden.

- **Konsistenz / Integritätserhaltung**
  Eine Datenbank erhält vom Anfang bis Ende einer Transaktion einen konsistenten Zustand.
- **Isolation**
  Der Benutzer, der mit einer Datenbank arbeitet, sollte den Eindruck haben, dass er mit dieser Datenbank alleine arbeitet.
- **Dauerhaftigkeit**
  Wenn eine Transaktion erfolgreich beendet ist, muss das Ergebnis dieser Transaktion „dauerhaft“ in der Datenbank gespeichert werden.

7.5.9.2. Transaktionen abbrechen und beenden

Wenn eine Fehlerprüfung während einer laufenden Transaktion stattfindet, kann die gesamte Operation durch die ROLLBACK-Anweisung abgebrochen werden. Diese Anweisung macht die Transaktion rückgängig und stellt den Anfangszustand wieder her. Dieser Befehl muss allerdings noch vor einem COMMIT-Befehl gegeben werden.
Mit der Nutzung von ROLLBACKS ist die Integrität einer Datenbank nach einem Datenbank-, Anwendungs- oder Systemfehler wiederherstellt.

Wenn hingegen alle Änderungen seit Beginn einer Transaktion fehlerfrei ausgeführt worden sind, werden sie in der Datenbank durch den Befehl COMMIT gespeichert. Die Ausführung dieses Befehls vor dem Beginn einer Transaktion garantiert, dass keine Fehler aufgetreten sind.

7.5.10. Isolationsstufen in Microsoft SQL Server 2000

Um die Datenintegrität zu sichern, werden auch die parallelen Zugriffe von unterschiedlichen Druckern durch Transaktionen koordiniert. Zur Erfüllung dieses Zwecks können gemeinsame und exklusive Sperren auf Zeilen oder Tabellen gelegt werden. Durch exklusive Sperren lassen sich jegliche weiteren Zugriffe auf eine Tabelle oder Zeile verbieten. In einer gemeinsamen Sperre hingegen können während der Änderung an einer Tabelle beliebig viele SELECT-Anweisungen ausgeführt werden.

Zur Verhinderung von Konflikten durch parallele Zugriffe werden die genannten Sperren in den von Microsoft SQL Server 2000 verfügbaren Transaktionsisolationsstufen automatisch angewandt. Um die Datenbankoperationen abzusichern, stehen vier Isolationsstufen zur Verfügung:

Read Uncommited ist die niedrigste Isolationsstufe. In dieser Stufe ist es möglich Daten, die geändert wurden, zu lesen. Hier sind keine gemeinsamen Sperren gegeben und keine exklusiven Sperren berücksichtigt. In diesem Fall ergeben sich die so genannten „Dirty Reads“, die schnelle aber ungenaue Ergebnisse liefern. Diese Isolationsstufe ist nicht zu empfehlen.

Read Commited ist die zweite Isolationsstufe und im Vergleich zu der ersten Stufe werden beim Datenlesen gemeinsame Sperren verwendet. Dadurch wird das Lesen von geänderten Daten verhindert. Trotzdem können die Daten vor dem Ende der Transaktion geändert werden, was zu nicht wiederholbaren Lesevorgänge oder Phantomdaten führen kann.

Die dritte Isolationsstufe **Repeatable Read** aktiviert alle gemeinsamen und exklusiven Sperren für sämtliche Daten, die in einer Abfrage verwendet werden. Damit können die Daten nicht durch andere Benutzer aktualisiert werden. Diese Isolationsstufe verhindert nicht wiederholbare Lesevorgänge. Hier existieren jedoch immer noch Phantomenzeilen.

Serializable ist die höchste Isolationsstufe. Durch die Festlegung einer Bereichsperre für das DataSet werden das Einfügen und die Aktualisierung einer Zeile in das DataSet von anderen Benutzern vor dem Abschluss der Transaktion verhindert.

7.6. Aufbau der Datenbank

Beim Entwurf einer Datenbank ist es immer wichtig festzulegen, welche Werte überhaupt als Primärschlüssel dienen können. Bei der main-Tabelle in der Druckerdatenbank soll der Primärschlüssel den kompletten Datensatz eines Druckers identifizieren können. Zuerst wurde überlegt, den Namen eines Druckers als Primärschlüssel zu nehmen. Ein Drucker hat im Netzwerk immer einen Name, mit dem er sich eindeutig identifizieren lässt.

Der Druckername ist also als eindeutig anzusehen und eignet sich zunächst einmal als Primärschlüssel. Das Problem mit diesem Wert ist jedoch, dass ein Druckername eine Zeichenkette mit variabler Länge besitzt, die, obwohl sie in der Datenbank als VARCHAR(256) deklariert wird, sehr viel Speicherplatz besetzt. Im VW-Netzwerk Salzgitter hat der Druckername 12 Zeichen und belegt damit auch 12 Byte Speicherplatz. Das kann problematisch sein, wenn man andere Tabellen mit der main-Tabelle durch diesen Schlüssel miteinander verknüpfen möchte, da sich der benötigte Speicherplatz sehr schnell erhöht. Deswegen wird der Druckername nur verwendet, um zu überprüfen, ob sich ein Drucker mit diesem Name in der Datenbank befindet. Um diese Überprüfung auszuführen, wurde die Spalte des Druckernamens als UNIQUE deklariert. Der Druckername darf hierbei nicht NULL sein.

Daher ist ein Index besser als Primärschlüssel geeignet. Ein Index belegt weniger Speicherplatz und kann aus jedem ganzzahligen Wert bestehen, wie ein Integertyp, der zwischen 8 und 64 Bit benötigt. Transact-SQL bietet den Datentyp SMALLINT an, der eine Breite von 16 Bit entspricht und über 32767 positive Indizes verfügt. Die Wahl dieses Datentyps reicht für die Datenbank, da die Anzahl der Drucker nicht größer als 32767 (tatsächlich nur wenige hundert) ist, außerdem können die Indizes gelöschter, veralteter Drucker wieder verwendet werden.
Diese Indexspalte kann vom Microsoft SQL Server 2000 in eine so genannte Identitätsspalte verändert werden, von der jede Tabelle lediglich eine besitzen kann. Eine Identitätsspalte spezifiziert eine Identitätsnummer für die erste eingefügte Zeile in einer Tabelle sowie eine Zunahme, um aufeinander folgende Identitätsnummern zu bestimmen. Wenn ein Wert in einer Tabelle mit einer Identitätsspalte eingefügt wird, wird der nächste anzunehmende Identitätswert automatisch vom Microsoft SQL Server 2000 generiert. Der Nachteil dieser Eigenschaft ist, dass der Datenbankserver nach dem Löschen eines Datensatzes den freigewordenen Index nicht wieder benutzt, sondern der neue Datensatz in der nächsten Indexspalte eingeführt wird.
Bei der Druckerdatenbank wird zuerst überprüft, ob die neuen Daten den Daten der Datenbank entsprechen. Ist dies der Fall, werden die alten Daten außer dem Datum der letzten Abfrage in der Datenbank belassen. Wenn die neuen Daten nicht gleich den existierenden Daten sind, wird der gesamte Datensatz des Druckers gelöscht und die neuen Werte eingefügt.

Dieses Löschen von kompletten Datensätzen führt aber leider zu einem großen Anstieg der Indizes, da freigewordene Indizes nicht wieder verwendet werden. Um die freigewordenen Lücken wieder zu füllen, kann die Identitätsfunktion dieser Indizes abgeschaltet werden. Dafür müsste nach jeder INSERT-Anweisung das SQL Kommando SET QUOTED_IDENTIFIER OFF ausgeführt werden, um den freien Index vor der Anweisung manuell zu ermitteln.

Anschließend muss die Identitätsfunktion mit SET QUOTED_IDENTIFIER ON wieder eingeschaltet werden, um die Indizes wieder zu erstellen und manipulieren zu können.

Der freie Index wird während des Einfügens durch einen Trigger ermittelt. Damit kann der Anstieg der Indizes verhindert werden. Eine der Erweiterung von Transact-SQL ist die Möglichkeit, Schleifen wie die WHILE-Schleife benutzen zu können. Dadurch wird der Index von null bis zum maximalen Wert durchlaufen und der Datensatz wird in dem ersten freien Index eingefügt.

Der folgende Code zeigt die Ermittlung eines freien Index:

```
/*Ermitteln einer freien ID*/
WHILE (@EmptyId < @Limit AND NOT @Test IS NULL)
  BEGIN
     SET @Test = (SELECT Stand_ID FROM druckertype WHERE Stand_ID = @EmptyId)
     IF @Test IS NULL
        BEGIN
             INSERT INTO druckertype (Stand_ID, Drucker_ID, Tinte_Toner_Fullstande,
             Beschreibung) VALUES (@EmptyId, @Drucker_ID, @Tinte_Toner_Fullstande,
             @Beschreibung)
        END
     ELSE
        BEGIN
             SET @EmptyId = @EmptyId + 1
        END
  END
```

Die Übertragung des gesamten Inhalts eines Datensatzes an die Datenbank wird ohne Index ausgeführt, sodass der Inventardienst nicht vor jeder INSERT-Anweisung vom Datenbankserver einen freien Index für jeden Datensatz zu holen braucht. Damit wird nicht nur der Datenverkehr zwischen Inventardienst und Datenbankserver reduziert, sondern es wird auch gesichert, dass der gewählte Index noch frei ist und beim Einfügen nicht gleichzeitig von einer INSERT-Anweisung genutzt wurde.

In allen Tabellen, in denen die Indizes als Primärschlüssel deklariert wurden, werden INSTEAD OF INSERT-Trigger ausgeführt, um die Tabellen mit den Druckerdaten zu füllen. Wenn ein INSTEAD OF INSERT-Trigger verwendet wird, können keine Datensätze mehr direkt in den Tabellen eingefügt werden.

Hier wird der Code eines INSTEAD OF INSERT-Triggers gezeigt:

```
/**Der Trigger übernimmt das Einfügen in die Tabelle druckertype**/
CREATE TRIGGER Insertdruckertype
ON dbo.druckertype
INSTEAD OF INSERT AS

DECLARE @EmptyId BIGINT
DECLARE @Limit BIGINT
DECLARE @Test VARCHAR (256)

DECLARE @Drucker_ID SMALLINT
DECLARE @Tinte_Toner_Fullstande INT
DECLARE @Beschreibung VARCHAR (100)
```

```
SET @Drucker_ID = (SELECT inserted.Drucker_ID FROM inserted)
SET @Tinte_Toner_Fullstande = (SELECT inserted.Tinte_Toner_Fullstande FROM
                                inserted)
SET @Beschreibung = (SELECT inserted.Beschreibung FROM inserted)

/*Variablen zum Ermitteln einer freien ID*/
SET @EmptyId = 0
SET @Limit = 9223372036854775807
SET @Test = 'START'



IF NOT EXISTS (SELECT druckertype.Drucker_ID FROM druckertype, inserted WHERE
   inserted.Drucker_ID = druckertype.Drucker_ID
   AND inserted.Tinte_Toner_Fullstande = druckertype.Tinte_Toner_Fullstande
   AND inserted.Beschreibung = druckertype.Beschreibung)


BEGIN
 /*Ermitteln einer freien ID*/
 WHILE (@EmptyId < @Limit AND NOT @Test IS NULL)
     BEGIN
        SET @Test = (SELECT Stand_ID FROM druckertype WHERE Stand_ID = @EmptyId)
        IF @Test IS NULL
           BEGIN
             INSERT INTO druckertype (Stand_ID, Drucker_ID,
             Tinte_Toner_Fullstande,Beschreibung) VALUES (@EmptyId, @Drucker_ID,
             @Tinte_Toner_Fullstande, @Beschreibung)
           END
        ELSE
           BEGIN
             SET @EmptyId = @EmptyId + 1
           END
     END
END
```

Die einzelnen Werte werden vom Trigger über SELECT-Anweisungen geholt und anschließend in lokalen Variablen gespeichert. Die Werte dieser Variablen werden durch die INSERT-Anweisung in den verschiedenen Tabellen der Druckerdatenbank eingefügt.

7.7. Einfügen eines Datensatzes in der Datenbank

Wenn die Drucker neu ausgelesen worden sind, fügt der Inventardienst sämtliche Tabellen in die Datenbank ein. Zuerst wird zum Datenbankserver ein Signal des Beginns einer Transaktion gesendet. Damit kann der teilweise eingefügte Datensatz wieder entfernt werden, sobald ein Fehler existiert.

Die main-Tabelle wird zuerst mit dem vom Inventardienst gelieferten Datensatz gefüllt. Dieses Einfügen führt der INSTEAD OF INSERT-Trigger aus. In dem Trigger wird überprüft, ob bereits ein solcher Drucker existiert. Wenn das der Fall ist, werden die neuen Daten mit denen in der Datenbank verglichen. Ist der Umfang der neuen Daten geringer, als die schon vorhandenen, so werden die neuen Daten verworfen, ansonsten wird der gesamte Datensatz gelöscht (siehe Löschen eines kompletten Datensatzes) und durch die neuen Daten ersetzt. Dies wird auch durch den Trigger bestimmt.

Dummerweise ist es nicht möglich, zeitgleich eine Löschanweisung für eine Tabelle auszuführen, wenn ein Trigger mit einer INSERT-Anweisung mit dieser Tabelle arbeitet. Die Lösung für dieses Problem ist der Aufruf einer gespeicherten Prozedur, die den Datensatz des Druckers über alle Tabellen löscht. Dafür liefert die Prozedur die zu löschende Drucker_ID über die der Drucker in allen Tabellen eindeutig identifiziert werden kann. Wenn der Druckerdatensatz gelöscht wurde, wird der Trigger einen freien Index für den Drucker ermitteln, und die main-Tabelle mit dem Datensatz gefüllt.

Beim Einfügen weiterer Informationen in die anderen Tabellen braucht der Inventardienst den vergebenen Index des gerade eingefügten Druckers, der durch eine SELECT-Anweisung über den Druckernamen ausgelesen werden kann. Auf diese Weise können nun alle Tabellen, die über 1:n Beziehungen verknüpft sind, direkt mit Daten gefüllt werden. Das ist zum Beispiel der Fall in den Tabellen „druckertype“, „papierfull“ und „papierausgabe“, in denen die Datensätze analog zur main-Tabelle eingefügt werden. Für jeden Tinten- oder Tonerfüllstand in der Tabelle „druckertype“ wird ein eindeutiger Index durch einen INSTEAD OF INSERT- Trigger gegeben. Bei den Tabellen „papierfull“ und „papierausgabe“ werden die Indizes ebenfalls für jedes Papiereingabefach und -ausgabefach vergeben. Wenn ein Fehler während des Einfügens eines Datensatzes eintritt, wird der Inventardienst die ROLLBACK-Anweisung anwenden um die Transaktion abzubrechen. Andernfalls werden alle Änderungen in der Datenbank mit dem COMMIT-Befehl gespeichert (siehe Transaktionen abbrechen und beenden).

7.8. Löschen eines kompletten Datensatzes

Um einen kompletten Datensatz zu löschen, wird eine gespeicherte Prozedur verwendet. Diese Prozedur wird sowohl automatisch vom Insert-Trigger der main-Tabelle vor dem Einfügen von neuen Daten aufgerufen, als auch vom Web-Interface, um einen kompletten Drucker aus der Datenbank zu löschen. Dieses Löschen wird durch eine Ermittlung der ID des zu löschendes Druckers ermöglicht.

7.9. Löschen und Aktualisieren von Datensätzen mit 1:n Beziehungen

In einer 1:n Beziehung können Datensätze durch einen kaskadierten Fremdschlüssel gelöscht und aktualisiert werden. Diese Funktionalität wird mit der Angabe ON DELETE CASCADE und ON UPDATE CASCADE (siehe Kaskadieren von Fremdschüsseln) erstellt.
In den Beziehungen zwischen der main-Tabelle und den Tabellen „druckertype“, „papierfull“ und „papierausgabe“ wird diese Funktionalität des Fremdschlüssels verwendet.
Diese drei Tabellen besitzen einen Primärschlüssel, der sich aus einem einmaligen Index des Tinten- oder Tonerfüllstandes, Papierfüllstandes bzw. Papierausgabe und des zugehörigen Druckers zusammensetzt. Der Index des Druckers stellt eine Referenz des Primärschlüssels der main-Tabelle dar und wird deshalb über die Kaskade gelöscht bzw. aktualisiert.

7.10. Aktualisieren der Datenbank

Wenn in einem langen Zeitraum ein Drucker nicht mehr abgefragt worden ist, soll der Drucker gelöscht werden, um die Datenbank zu bereinigen und zu aktualisieren. Dazu wird vom Inventardienst eine gespeicherte Prozedur aufgerufen. Das Löschintervall wird durch einen in der Konfigurationsdatei übergebenen Parameter bestimmt, der in Tagen angegeben wird. Als Beispiel, werden alle Drucker, die älter als neunzig Tage sind, automatisch gelöscht, wenn der Wert des Löschintervalls neunzig beträgt. Damit wird verhindert, dass die Datenbankkapazität nicht mit veralteten Daten überlastet wird.

7.11. Datenbankgröße

Um ungefähr den Speicherplatz, der in der Datenbank pro Drucker belegt wird, berechnen zu können, wurde eine detaillierte Liste aller Datenbankfelder mit ihrer maximalen Feldlängen erstellt. Ebenfalls wurde zu diesem Zeitpunkt die momentane Anzahl der belegten Zeilen pro Tabelle und die durchschnittliche Ausnutzung der maximalen Feldlänge untersucht.

Für die Berechnung der Durchschnittswerte und die maximalen Werte wurde die Datenmenge, die ein einzelner Drucker von einem üblichen Hersteller in der Datenbank belegt, errechnet. Ebenfalls wurde die Menge der Informationen, die vom ersten in die Datenbank eingetragenen Drucker herrühren, sowie der Anzahl der einmaligen Daten, die noch zum letzten Drucker gespeichert werden müssen, berechnet.

Zum Zeitpunkt der Untersuchung waren 361 Drucker in der Datenbank gespeichert. Nach der Berechnung aller Daten, erhielten wir folgende Ergebnisse:

Im schlimmsten Fall, wenn alle Informationen von einem Drucker ausgelesen werden können und alle Datenbankfelder bis zu ihrer maximal zulässigen Länge gefüllt wurden, kann der Datensatz dieses Druckers bis zu 1,5 Kilobyte umfassen. Beim ersten erfassten Drucker, der die Feldlänge nur zu den oben ermittelten Durchschnittswerte ausnutzt, belegt ca. 299 Bytes Speicherplatz in der Datenbank. Für die nachfolgenden Drucker kann die Größe des jeweiligen Datensatzes variabel sein, da einige Drucker das SNMP unterstützen können, anderen wiederum nicht. Daher belegt der letzte erfasste Drucker 332 Bytes innerhalb der Datenbank.

8. Das Web-Interface

8.1. Anforderungen

Vom Auftraggeber war erwünscht, dass die gewonnenen Druckerdaten über ein Web-Interface dargestellt werden. Bei Verwendung eines Web-Interfaces ist zum Betrachten der Auswertung lediglich ein Webbrowser nötig. Ein solcher Webbrowser ist heutzutage bei fast jedem Betriebssystem standardmäßig mitinstalliert. Bei Windows ist das der Internet Explorer, aber auch die meisten anderen Webbrowser könnten verwendet werden. Der besondere Vorteil hierbei ist, dass keine spezielle Betrachtungssoftware benötigt wird, die womöglich auf verschiedene Systeme angepasst werden müsste.

Die Bedienung des Web-Interfaces sollte möglichst einfach und intuitiv sein, und die Darstellung der Informationen sollte möglichst übersichtlich und auf häufige Fragestellungen angepasst sein. Auch ein Export dieser Information sollte möglich sein, um sie anderweitig weiterverwenden zu können.

8.2. Voraussetzungen

Die vor allem entscheidende Voraussetzung des Web-Interfaces ist das Web-Interface des Vorgängerprojektes. Hier war vom Auftraggeber eine möglichst nahtlose Integration der neuen Webseiten in die alten gewünscht. Aus diesem Grund liegt es nahe dieselbe Architektur des Vorgängers zu verwenden.

Daher wird ebenfalls der Microsoft Internet Information Server (IIS) verwendet, der mit dem aktuellen .NET Framework ausgestattet wurde, um dynamisch generierte ASP.NET Webseiten bereitstellen zu können. Zwar gibt es inzwischen auch die Möglichkeit, ASP.NET Seiten auf alternativen Webservern (z.B. Apache) mittels Mono[12] verwenden zu können, aber in diesem Fall kann nicht die integrierte Windows-Authentifizierung verwendet werden. Jedoch findet gerade diese Funktionalität ihre Anwendung im Vorgängerprojekt, und zwar um benutzerabhängige Berechtigung auf das Web-Interface zu gewähren.

Neben dem Webserver wird ebenfalls auch auf die Stilvorlagen (CSS-Datei), das Code-Archiv und auf den grundlegenden Aufbau der Webseiten des Vorgängers zurückgegriffen.

[12] http://www.go-mono.com

8.3. Abfragemöglichkeiten

Die über das Web-Interface ausführbaren Abfragen sind vor der Erstellung der Webseiten definiert worden. Dafür wurde zuerst überlegt, welche Daten für die Ausführung der Abfragen geeignet sein könnten. Durch diese Daten sollen sich die Druckerinformationen in der Datenbank schneller abfragen lassen.

Die im Web-Interface implementierten Abfragen sind:

- Eine Übersicht der vom Netzwerk ausgelesenen Drucker mit ihren Druckernamen, Druckertyp, Abfragezeit und Status. Hier lassen sich die Drucker nach einem bestimmten im Druckernamen enthaltenen String sowie nach dem Druckerhersteller abfragen. Es ist auch möglich, die Abfrage von Druckern auszuführen, die seit einer bestimmten Anzahl von Tagen nicht mehr abgefragt wurden, aber noch nicht als veraltet gelten.
- Eine Liste der erfassten Drucker mit ihren entsprechenden IP-, MAC-Adressen und Druckernamen. Die Drucker können nach diesen drei Attributen abgefragt werden.
- Eine Auflistung über alle Druckerhersteller mit der entsprechenden Anzahl der Drucker, die zu diesem Hersteller gehören, sowie die Anzahl der gedruckten Seiten von diesen Druckern. Zusätzlich wird die Anzahl der Drucker und der gedruckten Seiten in Prozent gegeben. Die gesamte Anzahl der erfassten Netzwerkdrucker sowie die von ihnen insgesamt gedruckten Seiten werden auch in dieser Abfrage angezeigt.

Die erhaltenen Druckerlisten haben einen Hyperlink, der zu einer Webseite mit allen erfassten Details zum einzelnen Drucker führt. Dieser Hyperlink ist zum Beispiel in der Option Druckerübersicht der Webseite in einer Spalte der Liste abgelegt worden. Bei der Option MAC-Adresse kann der Hyperlink durch einen Klick auf den Druckernamen abgerufen werden. Die implementierten Abfragen wurden an den verantwortlichen Projektbetreuer übergeben.

8.4. Zugriffberechtigung

Zur Einschränkung der Zugriffberechtigungen auf die Webseiten wird das Authentifizierungssystem des Vorgängerprojektes verwendet. Diese wurde in Form einer kompilierten Klassenbibliothek (DLL-Datei) im Verzeichnis der Webanwendung (BIN-Verzeichnis) hinterlegt, so kann sie sowohl von den Webseiten als auch vom Web-Service verwendet werden. Die Bibliothek stellt Funktionen wie „ist_berechtigt“, „ist_Admin“ und „darf_nur_Lesen“ zur Verfügung. Mit Hilfe dieser Funktionen kann vor der Ausführung von Aktionen im Web-Interface überprüft werden, ob die betreffende Berechtigung besteht. Besteht keine Berechtigung, so kann auf eine Webseite mit einer entsprechenden Fehlermeldung umgeleitet werden.

8.5. Code-Archiv

Das Code-Archiv enthält Funktionen, die meist von mehreren Webseiten verwendet werden. Im Gegensatz zum Authentifizierungssystem sind diese aber in einer Code-Behind-Datei als Quellcode abgelegt. Webseiten, die Funktionen hieraus benötigen, können von dem Code-Archiv erben. Die Funktionen werden unter anderem für die Sortierung der DataGrid Webcontrols (siehe Kapitel 8.7) verwendet und es wird auch beim Binden eines DataGrids dieses um einen Laufbalken erweitert. Im Rahmen dieses Projektes wurde lediglich eine einfache Funktion hinzugefügt, die als Zahlen angegebene Zeitspannen in lesbare Zeichenkette umwandelt. Dieses wird gelegentlich benötigt, um die Laufzeiten der Drucker, die eigentlich als Zahlwert ausgelesen werden, sinnvoll darstellen zu können.

8.5.1. CSV-Export

Ebenfalls im Code-Archiv übernommen sind die Funktionen zum CSV- und PDF-Export, die auch auf den Webseiten des Drucker-Inventarsystems in entsprechender Anpassung ihre Verwendung finden. Über diese Funktion lassen sich die Abfrageergebnisse des Web-Interfaces abspeichern und sich somit neben dem einfachen Betrachten auch anderweitig verwenden. Das CSV-Format ist besonders zur Weiterbearbeitung mit anderen Tools (z.B. Excel) geeignet. Prinzipiell wird hierbei eine Datentabelle als Textdatei abgespeichert, wobei die einzelnen Spalten durch Trennzeichen begrenzt werden. Dieser Funktion werden einfach das zu exportierende Dataset und eine Liste der gewünschten Spalten übergeben. Die Funktion erzeugt daraufhin eine solche CSV-Datei und liefert den Pfad zu dieser Datei als Rückgabewert zurück. Auf der Webseite kann schließlich ein Download-Link zu dieser Datei dargestellt werden.

8.5.2. PDF-Export

Als Alternative zum CSV-Export existiert auch die Möglichkeit zum PDF-Export, die zwar weniger zur Weiterverarbeitung geeignet ist, jedoch optisch anspruchsvoller ist und sich besser zum Ausdrucken (z.B. für Berichte) eignet. Da sich PDF-Dateien jedoch nicht so einfach erzeugen lassen, wird bei dieser Funktion auf ein externes Tool zurückgegriffen. Verwendet wird die kostenlose Lite Version des Produktes „DynamicPDF™ Generator.NET“ der Firma „ceTe Software“[13]. Der Aufruf dieser Funktion ist ähnlich wie der beim CSV-Export. Es müssen lediglich einige zusätzliche Formatierungsinformationen mit übergeben werden. Daraufhin wird das PDF-Dokument erzeugt und der Pfad dazu zurückgegeben.

8.6. Administrativer Bereich

Das Web-Interface stellt ebenfalls einen Verwaltungsbereich zur Verfügung. Dieser Bereich besteht aus zwei Seiten: „Dienst-Verwaltung“ und „Konfig. ändern“. Die Bedienung dieser Webseiten wurde weitgehend selbsterklärend gestaltet. Zur Unterstützung sind für alle Felder und Schaltflächen kurze Hilfstexte eingerichtet, die erscheinen, wenn man den Mauszeiger über die jeweiligen Bereiche fährt und der Browser dieses unterstützt.

Die Webseite „Dienst-Verwaltung“ stellt Informationen über den gegenwärtigen Zustand des Drucker-Inventar-Dienstes zur Verfügung. Es wird angezeigt wie viele Elemente sich derzeit jeweils in der Auslese Warteschlange und in der Datenbank Warteschlange befinden. Auf eine detaillierte Auflistung der Elemente wurde verzichtet, da die Inhalte der Warteschlangen normalerweise extrem schnell geleert werden und sie meiste Zeit leer sein sollten. Sollte dies jedoch nicht der Fall sein, so deutet dies auf ein Problem mit dem Drucker-Inventar-Dienst hin.
Außerdem werden die Zeitpunkte angezeigt, zu denen die jeweils nächste automatische Druckersuche bzw. -aktualisierung angesetzt sind. Auf Wunsch können diese aber auch über zwei Schaltflächen sofort manuell gestartet werden. Ebenfalls ist es hier möglich einzelne Drucker oder eine Liste von Druckern manuell zur Auslesewarteschlange hinzuzufügen.

Über die Webseite „Konfig. ändern“ erhält man schließlich Zugriff auf die Konfiguration des Drucker-Inventar-Dienstes und des Web Services. Alle Einstellungen und Listen können hier geändert werden. Die Änderungen werden mithilfe der Übernahmeschaltfläche gespeichert und der Dienst initialisiert sich danach selbst neu.

[13] http://www.cete.com

8.7. Aufbau der ASP-Seiten

Die Struktur einer ASP-Seite besteht aus zwei Teilen - dem Programmteil und der Benutzeroberfläche bzw. dem HTML-Teil, in dem die Informationen dargestellt werden. Für die Benutzeroberfläche können sowohl HTML-Controls als auch die von ASP.NET bereitgestellten Web-Controls benutzt werden. Die Web-Controls sind ASP.NET-Serversteuerelemente mit einem Objektmodell zur dynamischen Erzeugung und Handhabung von HTML auf der Server-Seite. Diese Web-Controls bestehen nicht nur aus Formularelementen, wie Buttons und Textfeldern, sondern auch aus Steuerelementen für spezielle Zwecke wie z.B. das DataGrid. Mit Hilfe dieses Elements können Datensätze komfortabel dargestellt werden. Beim Web-Interface des Drucker-Inventarsystems wurden solche Web-Controls verwendet.

Zum Programmteil kann jede Sprache, die von .NET unterstützt wird, verwendet werden. In den ASP.NET Seiten des Drucker-Inventarsystems wird VB.NET zur Ausführung des Programmteils eingesetzt. Der Code des Programmteils wird entweder zusammen mit dem HTML-Code direkt innerhalb der ASPX-Datei, oder in einer externen Code-Behind-Datei gespeichert. In diesem Projekt werden alle konkreten Funktionen einer Webseite direkt in einer ASPX-Datei implementiert. Die Funktionen, die von mehreren Seiten gemeinsam verwendet werden, sind in einer Code-Behind-Datei gespeichert. Das Aufbauschema des Programmteils ist in allen Webseiten dieses Projektes immer gleich. Zuerst werden Daten aus der Drucker-Datenbank durch eine Datenbindungsprozedur abgerufen. Zur Darstellung der Daten werden sie an ein DataGrid oder einen Repeater gebunden.

Beim ersten Aufruf einer Seite durch einen Client wird die Datenbindungsprozedur von einem Page_Load Ereignis aufgerufen. Dieses Ereignis ermöglicht, dass beim ersten Aufruf einer Webseite ein Suchergebnis geladen werden kann. Alle Filter und Sortierkriterien sind zunächst auf Standardwerte gesetzt. Die Datenbindungsprozedur wird ebenfalls aufgerufen, wenn vom Benutzer eine Abfrage oder ein Sortierkriterium ausgeführt wird. Daher werden entsprechende Eventhandler verwendet, die die Datenbindung erneut aufbauen können, wobei die gewünschten Abfrageeinschränkungen berücksichtig werden.

Die Struktur der Datenbindungsprozedur umfasst mehrere Teile. Der erste ist ein dynamisch zusammengesetzter String für eine SQL-Anweisung. Dieser String muss immer eine gültige Syntax aufweisen. Diese muss für alle möglichen Kombinationen von Abragen und Filtereinträgen korrekt sein, wie sie vom Benutzer gesetzt werden können.

Der erstellte SQL-String kann danach an die Datenbank gesendet werden. Mit Hilfe eines DataAdapters (siehe „die wichtigsten Klassen des .NET Framework-Datenproviders“) wird ein DataSet mit den Abfrageergebnissen gefüllt. Anschließend wird dieses DataSet zur Darstellung der Informationen im Web-Interface an ein DataGrid oder Repeater gebunden.

Das DataGrid bietet neue Möglichkeiten bei der Arbeit mit Datenbanken. Ein DataGrid ist die Visualisierung einer Tabelle in einem DataSet. Mit diesem ASP.NET Web Control lassen sich Datenbankinhalte darstellen und manipulieren. So können mit Hilfe des DataGrid verknüpfte Daten übersichtlich dargestellt, sowie Datensätze gelöscht werden. Hyperlinks zum Seitenwechsel (z.B. „nächste 40 Datensätze“) oder zu einer anderen Webseite können auch automatisch erstellt werden. Ebenfalls kann, zum Beispiel durch den Klick auf einen Spaltenkopf, eine Neusortierung der Datenmenge durchgeführt werden. In den ASP-Seiten „druckerübersicht“ und „mac_drucker“ wurde das DataGrid für die Manipulation und Darstellung den Druckerdaten verwendet.

Mit dem Repeater Web Control können Listen von Daten angezeigt und strukturiert ausgegeben werden. Dieses Control ermöglicht ein benutzerdefiniertes Layout, indem eine angegebene Vorlage auf jedes in der Liste angezeigte Element angewendet wird. Das Layout und der Inhalt von Listenelementen werden in Repeater durch Templates definiert. Ein Repeater muss zumindest ein ItemTemplate definieren. Zur Darstellung der Druckerzusammenfassung und Druckerdetails wurde in den entsprechenden ASP-Seiten das Repeater Web Control benutzt.

Wenn eine Webseite von einem Client aufgerufen wird, wird zuerst der Programmteil ausgeführt. Die Web Controls werden mit Werten dieses Programmteils gefüllt und deren Darstellung angepasst. Bevor die Seite an den Client gesendet wird, werden alle Web Controls auf Standard HTML-Tags zurückgeführt. Aus einem Repeater wird zum Beispiel eine Datenliste und aus einem DataGrid eine komplexe HTML-Tabelle.

8.8. Der Web Service

Zusätzlich zum Web-Interface wird über den IIS noch ein ASP.NET Web Service bereitgestellt. Dieser wird zur Fernsteuerung und Überwachung des Drucker-Inventar-Dienstes verwendet, der in der Regel auf einem anderen Server läuft. Da der Web Service des Client-Inventarsystems übernommen und erweitert wurde, stehen dieselben Funktionen zur Verfügung.

Es werden folgende zusätzliche Funktionen vom Web Service angeboten:

- Abfrage der Zustandsinformationen des Drucker-Inventar-Dienstes
- Abfrage der aktuellen Konfigurationsdaten des Drucker-Inventar-Dienstes
- Übertragen neuer Konfigurationsdaten an den Drucker-Inventar-Dienst
- Reinitialisierung bzw. Neustart des Drucker-Inventar-Dienstes
- Manuelles Starten der Druckersuche
- Manuelles Starten der Druckeraktualisierung
- Manuelles Einfügen eines Druckernamens in die Auslesewarteschlange
- Manuelles Einfügen einer Liste von Druckernamen in die Auslesewarteschlange

Die Funktionen des Web Service werden mittels .NET-Remoting an die entsprechenden Schnittstellen des Drucker-Inventar-Dienstes gekoppelt.

9. Abschluss

9.1. Geplante Erweiterungen

Das jetzt vorhandene und im Produktivbetrieb befindliche Inventarsystem ist schon jetzt sehr hilfreich bei den administrativen Tätigkeiten. Von Seiten der Firma sind aber weiterhin noch einige Erweiterungen für das gesamte Inventarsystem angedacht. Zwei Punkte wurden bereits im Rahmen der vorangegangenen Diplomarbeit genannt.

Das ist zum einen die Anbindung an das konzernweite Beschaffungssystem „Hani“, mit dessen Hilfe Informationen über Garantiezeit und Abschreibezeitraum der jeweiligen Geräte abgerufen werden können. Problematisch ist, dass weiterhin bei vielen Geräten die nötige Inventarnummer nicht automatisch erfasst werden kann. Ohne diese Nummer ist ein Datenabgleich mit „Hani“ kaum möglich.

Zum anderen ist ein Anschluss an das Kabelmanagement-System angedacht. Das Kabelmanagement-System ist ein System zur Verwaltung der Netzwerk-Infrastruktur, welches bei VW Salzgitter eingesetzt wird. Mithilfe der MAC-Adresse ließe sich die Netzwerkdosen-Nummer herausfinden und somit könnte auch der Standort der jeweiligen Geräte mit in der Inventardatenbank abgelegt werden.

Leider wurden die nötigen Zugriffsmöglichkeiten auf diese beiden Systeme während des Praxissemesters firmenseitig nicht gewährt, daher wurde den Erweiterungen auch nicht weiter nachgegangen.

Des Weiteren ist jetzt ein neuer Wunsch hinzugekommen, so wird über ein automatisches Reportsystem nachgedacht, das Berichte über die verschiedenen Inventardaten erstellt. Diese Funktionalität soll über die des bisherigen Web-Interfaces hinausgehen und eventuell über eine komplett neue Benutzeroberfläche zur Verfügung gestellt werden. Dieses wird möglicherweise im Rahmen einer weiteren Diplomarbeit angegangen.

9.2. Abschließende Worte (Karla)

Neben allen Fachkenntnissen, die man in der Fachhochschule lernt, spielt auch die Praxis bzw. die Anwendung dieser Kenntnisse in den verschiedenen Arbeitsbereichen der Unternehmen eine große Rolle für die Berufsausbildung der Studenten.
Im Rahmen dieser Diplomarbeit hatte ich die Möglichkeiten sowohl neue Kenntnisse, als auch neue und interessante Technologien (z.B. die Microsoft .NET Technologie) kennen lernen zu können. Besonders wichtig für mich war, dass ich viel praktische Erfahrung sammeln und meine fachlichen Kenntnisse anwenden konnte.

Auf der anderen Seite konnte ich mehrere Leute kennen gelernt. Durch den Kontakt mit diesen Leuten, konnte ich sowohl verbal als auch schriftlich meine Deutschkenntnisse verbessern. Des Weiteren hatte ich die Möglichkeit mit einem anderen Studenten zusammenarbeiten, wodurch ich die einen Einblick in Teamarbeit erhalten konnte, die zur Zeit sehr wichtig für eine bessere Zweckmäßigkeit der Firmen ist.

Anschließend möchte ich mich bei allen Leuten für die Unterstützung während der Realisierung dieser Diplomarbeit bedanken.

9.3. Abschließende Worte (Holger)

Ich kann mich weitgehend Karlas Worten anschließen. Es war sehr interessant einen Einblick in die Arbeitswelt eines großen Konzerns wie die VW AG zu bekommen.
Vor allem hat mich aber die Aufgabe des Praxissemesters, die auch Thema dieser Diplomarbeit ist, begeistert. Ich konnte hier praktische Erfahrungen mit enorm vielen Technologien und Konzepten machen, die an der Fachhochschule nicht in dieser Form oder gar nicht behandelt werden, wie eben Microsofts .NET Framework oder das SNMP Protokoll. Da wir als Entwickler in einem eigentlich administrativen Bereich eingesetzt waren, waren wir bei der Umsetzung des Projekts weitgehend auf uns selbst angewiesen, was an sich nicht zum Nachteil war. Nachteilig war daran nur, dass die Einarbeitung doch sehr viel Zeit in Anspruch nahm. Daher haben wir das Praxissemester um zwei Monate verlängert und konnten schließlich auch ein funktionierendes, produktives Produkt abliefern. Ebenfalls fand ich auch die Teamarbeit bei diesem Projekt sehr produktiv, allein hätte ich die Aufgabe kaum in der Zeit bewältigen können.

Zum Schluss danke auch ich allen Personen, die mich direkt oder indirekt bei dieser Diplomarbeit unterstützt haben.

Anlagenteil

zur Diplomarbeit

„SNMP-basiertes Drucker-Inventarsystem“

A. Programmierhinweise

A.1. Allgemeine Anmerkungen

Visual Studio –Teil

Während der Entwicklung wurde das MS Visual Studio .NET 2003 benutzt. Es wird mindestens die Professional-Version benötigt, da die Standard-Version keine Windows-Dienste unterstützt. Alle im Visual Studio erstellten Teil-Projekte sind in einer Projektmappe zusammengefasst.

Dabei handelt es sich um die Projekte:

- DIApp (Konsolenversion des Dienstes)
- DICode (Die Funktionalität des Dienstes, Kern des Gesamtprojektes)
- DIDienst (der eigentliche Windows Dienst)
- DITransfer
- Snmp (modifizierte Version der Snmp Bibliothek)
- WebService (nur Verweis, Quellcode liegt auf dem Webserver)
- WebServiceProxy
- DIDienstSetup

ASP.NET-Teil

Die Webseiten wurden mit Macromedia Dreamweaver MX 2004 (v7.01) erstellt. Da aber keine Macromedia-spezifischen Funktionen verwendet werden, kann für die Weiterentwicklung auch jede andere Entwicklungsumgebung oder ein einfacher Texteditor verwendet werden.

A.2. Übersicht Drucker-Inventardienst

Das VB.NET-Projekt „DIDienst“ stellt den eigentlichen Windows-Dienst dar, welcher das Auslesen der Clients und das Einfügen der Daten in die Datenbank übernimmt. Die Funktionalität steht aber vollständig in dem Projekt „DICode“, welches eine Klassenbibliothek darstellt, die vom Dienst importiert wird.

A.2.1. Klassenübersicht

Die wichtigsten Klassen wurden bereits im Hauptteil beschrieben, bei ihnen ist nur der Verweis auf das betreffende Kapitel angegeben.

Klasse Scheduler

Siehe Kapitel 6.3.1.

Klasse DBQueue

Siehe Kapitel 6.3.2.

Klasse AusleseQueue

Siehe Kapitel 6.3.3.

Klasse Drucker

Siehe Kapitel 6.3.4.

Klasse DITransfer

Siehe Kapitel 6.3.5.

Klasse Config

Siehe Kapitel 6.3.6.

Klasse AbfrageSession

Siehe Kapitel 5.4.2.

Klasse IPAdresse

Diese Hilfsklasse dient dem komfortableren Umgang mit IP-Adressen. Zwar stellt das .NET Framework schon die IPAdress Klasse zur Verfügung, die jedoch für den Gebrauch in diesem Projekt nicht besonders komfortabel ist. Ein IPAdresse Objekt kann einfach mit einem String, einem Byte Feld oder auch einem gültigen Hostnamen initialisiert werden. Es Befinden sich hier Methoden zum Vergleichen und Hochzählen von IP-Adressen sowie eine Ping Methode, die prüft, ob der Host an der betreffenden Adresse aktiv ist.

Klasse OID

Auch diese Hilfsklasse wurde in Kapitel 5.4.2. ff. erwähnt, sie dient dem komfortableren Umgang mit OIDs. Mit ihr lassen sich leicht OID Objekte erzeugen und in das jeweils gebrauchte Format umwandeln.

Klasse IPRange

Mithilfe dieser Klasse können IP Adressbereiche dargestellt werden, ihre Objekte enthalten ein IP-Adressenpaar das automatisch nach der Größe geordnet ist. Die Initialisierung erfolgt entweder mit einem String oder direkt mit zwei IP-Adressen.

Klasse Zeitpunkt

Diese Klasse kann Zeitpunkt Objekte zur Verfügung stellen, wie sie vom Scheduler-Thread benötigt werden. Initialisiert werden sie entweder mit einem String oder der Angabe von Wochentag, Stunde und Minute als Zahlenwerte. Wichtigste Funktion ist es den nächsten zukünftigen Zeitpunkt aus einer Liste von Zeitpunkten zu ermitteln.

Klasse Tools

Hier werden nur einige statische Hilfs-Methoden zur Verfügung gestellt. So kann hier der Pfad der Anwendung zur Laufzeit ermittelt werden und hier ist die Logbuchfunktion implementiert.

A.3. Übersicht Web-Service

Der Web Service kapselt die vom Drucker-Inventar-Dienst via Remoting bereitgestellten Funktionen, damit diese von beliebigen Plattformen aus und ohne großen Aufwand benutzt werden können.

Um die Remoting-Kommunikation mit dem Inventar-Server herstellen zu können, muss die gemeinsame Klasse „DITransfer“ importiert werden. Ein Proxy-Objekt dieser Klasse wird mit dem realen Objekt im Drucker-Inventar-Dienst verbunden.

Der Web Service ist eine Erweiterung des Web Service vom Vorgängerprojekt und hat nur einige zusätzliche Funktionen:

- `Function DI_Restart(ByVal suche As Boolean)`
  Neustart der Threads des Drucker-Inventar-Dienstes veranlassen
- `Function DI_getinfo() As Long()`
  Liefert Statusinformationen des Drucker-Inventar-Dienstes
- `Function DI_getconfig() As String()`
  Liefert die aktuelle Konfiguration des Drucker-Inventar-Dienstes
- `Function DI_setconfig(ByVal config As String())`
  Übernimmt und Schreibt die Konfiguration auf dem Drucker-Inventar-Dienst neu
- `Function DI_scan()`
  Initiiert komplette Abfrage des eingestellten IP-Bereichs
- `Function DI_aktualisieren()`
  Initiiert erneutes Auslesen aller bekannten Drucker in der Datenbank
- `Function drucker_einfuegen(ByVal druckername As String)`
  Fügt manuell einen Drucker zur Auslese-Queue hinzu
- `Function drucker_einfuegen_Liste(ByVal druckernamen As String())`
  Fügt manuell eine Liste von Druckern zur Auslese-Queue hinzu

A.4. Übersicht Webseiten

Die Webseiten können sowohl mit jedem Web-Editor, als auch mit dem WordPad bearbeitet werden.

Alle konkreten Funktionen, die direkt zu einer Webseite gehören, sind auch zusammen mit dem HTML-Code in dieser gespeichert. Eine von diesen Funktionen ist die Erstellung der dynamischen SQL-Anweisungen, die auf die Datenbank zugreifen und auf Benutzerabfragen reagieren. Die Funktionen, die von mehreren Webseiten gemeinsam verwendet werden, sind in der Code-Behind-Datei „Inventar-CodeArchiv.vb“ zusammengefasst. Webseiten, die Funktionen hieraus benötigen, können diese vom Code-Archiv erben. Im Rahmen dieses Projektes wurde auf das Code-Archiv des Vorgängers zurückgegriffen und lediglich eine einfache Funktion hinzugefügt.

Diese Funktion kann folgendermaßen aussehen:

```
Function TimeSpanString(ByVal l As Long) As String
```

Wandelt eine in Zahlen angegebene Zeitspanne in eine lesbare Zeichenkette um.

A.5. Übersicht Datenbank

A.5.1. Datenbankdiagramm

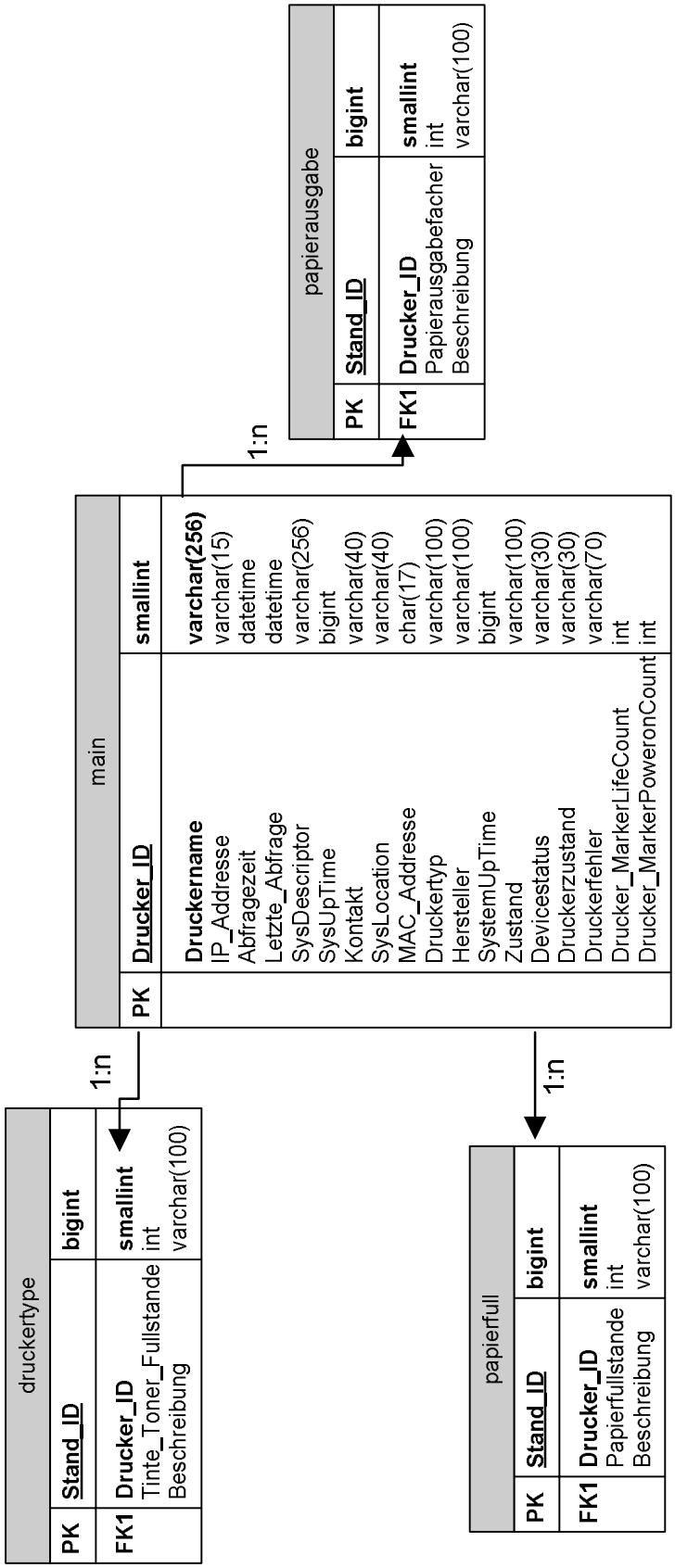

Abbildung 7.: Datenbankdiagramm

A.5.2. Beschreibung der Tabellen

Tabelle „druckertype“

Die Tabelle „druckertype“ speichert die einzelnen Toner oder Tinten-Füllstände eines Druckers mit ihren entsprechenden Beschreibungen.

In dieser Tabelle können bis zu 2^{63-1} (9223372036854775807) Datensätze gespeichert werden.

Primärschlüssel

Bei dieser Tabelle wurde die Stand_ID als Primärschlüssel definiert.

Fremdschlüssel

Die Drucker_ID wurde als Fremdschlüssel definiert. Dieser referenziert den Primärschlüssel Drucker_ID aus der Tabelle „main“.

Trigger Insertdruckertype.

Typ: INSTEAD OF TRIGGER

Der Trigger ermittelt eine freie Stand_ID und übernimmt das Einfügen der Datensätze in die Tabelle „druckertype“, wenn dieser noch nicht existiert.

Tabelle „main“

Diese Tabelle enthält den Hauptdatensatz eines Druckers. Alle Werte dieser Tabelle sind einmalig. Durch eine ID des Druckers wird diese Tabelle mit anderen Tabellen der Datenbank verknüpft. Die Tabelle „main“ kann 37.767 Datensätze speichern.

Primärschlüssel

Die Drucker_ID ist der Primärschlüssel dieser Tabelle.

Trigger Insertident

Typ: INSERT OF TRIGGER

Der Trigger übernimmt das Einfügen der Datensätze in die Tabelle „main“. In dem Trigger wird überprüft, ob bereits ein solcher Drucker existiert. Wenn dies der Fall ist, werden die neuen Daten mit denen in der Datenbank verglichen. Sind die neuen Daten nicht gleich, wird der gesamte Datensatz mit Hilfe der gespeicherten Prozedur „Druckerloeschen“ gelöscht. Danach wird eine freie Drucker_ID ermittelt und der neue Datensatz eingetragen.

Tabelle „papierausgabe“

Die Tabelle „papierausgabe“ speichert die einzelnen Papierausgaben jedes Druckers mit ihren entsprechenden Beschreibung.

In dieser Tabelle können bis 2^{63-1} (9223372036854775807) Datensätze gespeichert werden.

Primärschlüssel

In dieser Tabelle wurde die Stand_ID als Primärschlüssel definiert.

Fremdschlüssel

Bei dieser Tabelle wurde die Drucker_ID als Fremdschlüssel definiert. Dieser referenziert den Primärschlüssel Drucker_ID aus der Tabelle „main“.

Trigger Insertpapierausgabe

Typ: INSTEAD OF TRIGGER

Der Trigger ermittelt eine freie Stand_ID und fügt den Datensätze in die Tabelle „papierausgabe“ ein, wenn dieser noch nicht existiert.

Tabelle „papierfull“

In der Tabelle „papierfull“ werden die einzelnen Papierfüllstände jedes Druckers mit ihren entsprechenden Beschreibungen gespeichert.

Die Tabelle „papierfull“ kann 2^{63-1} (9223372036854775807) Datensätze speichern.

Primärschlüssel

Die Stand_ID wurde als Primärschlüssel definiert.

Fremdschlüssel

In der Tabelle papierfull wurde die Drucker_ID als Fremdschlüssel definiert. Dieser Schlüssel referenziert den Primärschlüssel Drucker_ID aus der Tabelle „main“.

Trigger Insertpapierfull

Typ: INSTEAD OF TRIGGER

Der Trigger ermittelt eine freie Stand_ID und übernimmt das Einfügen der Datensätze in die Tabelle „papierfull“, wenn dieser noch nicht existiert.

A.5.3. Beschreibung der gespeicherten Prozedur

Die gespeicherte Prozedur „altedruckerloeschen“

Diese Prozedur wird vom Inventardienst aufgerufen um die Datenbank zu bereinigen. Hier werden veraltete Drucker gelöscht. Das Löschintervall wird vom Parameter entnommen.

Eingabeparameter

Tagen

Bearbeitete Tabelle

main

Die gespeicherte Prozedur „Druckerloeschen“

Diese Prozedur wird vom Web-Interface aufgerufen und löscht einen Drucker aus der Datenbank.

Eingabeparameter

Drucker_ID

Bearbeitete Tabelle

main

A.5.4. Einfügen eines Datensatzes

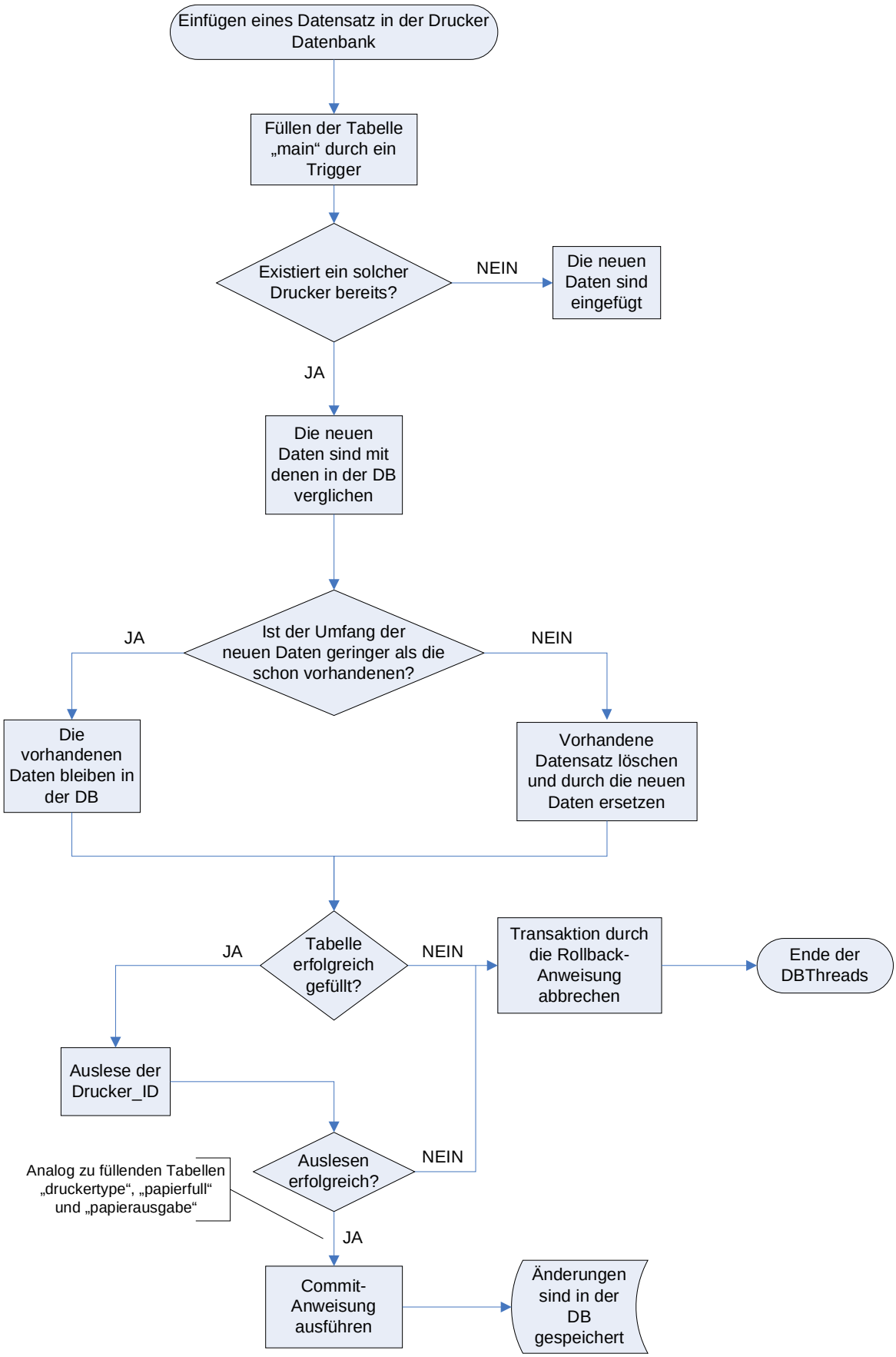

Abbildung 8.: Flussdiagramm Datensatz einfügen

A.5.5. Erwartete Datenbankgröße

Tabellenname	Feldname	Datentyp	Größe in Byte	Max. Feldgröße in Byte	Durchschnittliche Feldgröße in Byte	Max. Feldgröße im Einsatz in Bytes	Anzahl Zeilen in Tabelle	durchschnittl. Anzahl Zeilen pro Drucker	Datenmenge pro Drucker im WorstCase	Einträge erster Drucker	Einträge letzter Drucker	erster Drucker Bytes	letzter Drucker Bytes
main	Drucker_ID	smallint	2	2	2	2	360	1,000	2,000	1	1	2	2
	Druckername	varchar	256	256	23	23	360	1,000	256,000	1	1	23	23
	IP_Addresse	varchar	15	15	12	13	360	1,000	15,000	1	1	12	12
	Abfragezeit	datetime	8	8	8	8	360	1,000	8,000	1	1	8	8
	Letzte_Abfrage	datetime	8	8	8	8	360	1,000	8,000	1	1	8	8
	SysDescriptor	varchar	256	256	89	122	360	1,000	256,000	1	1	89	89
	SysUpTime	bigint	8	8	8	8	360	1,000	8,000	1	1	8	8
	Kontakt	varchar	40	40	22	40	360	1,000	40,000	1	0	22	0
	SysLocation	varchar	40	40	17	40	360	1,000	40,000	1	0	17	0
	MAC_Addresse	char	17	17	17	17	360	1,000	17,000	1	1	17	17
	Druckertyp	varchar	100	100	16	54	360	1,000	100,000	1	1	16	16
	Hersteller	varchar	100	100	2	9	360	1,000	100,000	1	1	2	2
	SystemUpTime	bigint	8	8	8	8	360	1,000	8,000	1	1	8	8
	Zustand	varchar	100	100	13	21	360	1,000	100,000	1	1	13	13
	Devicestatus	varchar	30	30	10	10	360	1,000	30,000	1	1	10	10
	Druckerzustand	varchar	30	30	7	11	360	1,000	30,000	1	1	7	7
	Druckerfehler	varchar	70	70	29	41	360	1,000	70,000	1	0	29	0
	Drucker_MarkerLifeCount	int	4	4	4	4	360	1,000	4,000	1	1	4	4
	Drucker_MarkerPoweronCount	int	4	4	4	4	360	1,000	4,000	1	1	4	4
druckertype	Stand_ID	bigint	8	8	8	8	368	1,022	8,178	0	2	0	16
	Drucker_ID	smallint	2	2	2	2	368	1,022	2,044	0	2	0	4
	Tinte_Toner_Fullstande	int	4	4	4	4	368	1,022	4,089	0	2	0	8
	Beschreibung	varchar	100	100	15	33	368	1,022	102,222	0	2	0	30
papierfull	Stand_ID	bigint	8	8	8	8	441	1,225	9,800	0	1	0	8
	Drucker_ID	smallint	2	2	2	2	441	1,225	2,450	0	1	0	2
	Papierfullstande	int	4	4	4	4	441	1,225	4,900	0	1	0	4
	Beschreibung	varchar	100	100	6	17	441	1,225	122,500	0	0	0	0
papierausgabe	Stand_ID	bigint	8	8	8	8	280	0,778	6,222	0	1	0	8
	Drucker_ID	smallint	2	2	2	2	280	0,778	1,556	0	1	0	2
	Papierausgabefacher	int	4	4	4	4	280	0,778	3,111	0	1	0	4
	Beschreibung	varchar	100	100	15	15	280	0,778	77,778	0	1	0	15
									Summe in Bytes			**Summe in Bytes**	**Summe in Bytes**
									1440,850			**299**	**332**

Abbildung 9.: Erwartete Datenbankgröße

B. Installation

B.1. Server-Installation

Voraussetzungen:

Auf dem Inventar-Server wird ein Microsoft Betriebssystem ab Windows 2000 Professional benötigt. Ein Server-Betriebssystem ist nicht erforderlich.

Weitere Voraussetzungen sind ein vorhandenes .NET-Framework ab Version 1.1, sowie das Vorhanden sein der „Microsoft Data Access Components“ (MDAC). Beides steht auf der Microsoft Homepage zum kostenlosen Download zur Verfügung.

Installation:

Die Installation des Drucker-Inventar-Dienstes geschieht durch das Ausführen des Setup-Programms auf der gewünschten Maschine. Dieses kopiert alle benötigten Dateien und registriert den Windows-Dienst im System. Während der Installation wird der Anwender zur Eingabe eines Windows-Kontos aufgefordert, unter welchem der Inventar-Dienst arbeiten soll. Hier muss ein Account angegeben werden, der über ausreichende Rechte für die Netzwerkkommunikation mit der Datenbank und dem Web-Interface verfügt. Der installierte Dienst startet nach Abschluss der Installation automatisch. Ein Neustart des Servers ist nicht notwendig.

Konfiguration:

Beim ersten Start des Drucker-Inventar-Dienstes erstellt dieser eine XML Konfigurationsdatei mit Defaultwerten im Installationsverzeichnis und verwendet diese zunächst. Die Einstellungen können entweder per Hand in dieser Datei angepasst werden oder auch mit dem Web-Interface verändert werden, sofern dieses mit den Defaultwerten zusammenarbeitet.

B.2. Konfiguration der Datenbank

Für die Drucker-Datenbank wird ein MS SQL Server 2000 als Server benötigt. Die Drucker-Datenbank kann auf dem Server durch ein SQL-Skript erstellt werden. Wenn die Erstellung der Datenbank erfolgt, kann im SQL-Skript zunächst der gewünschte physikalische Speicherort für die Datenbank angepasst werden.

Das liegt an Zeile Nr. 6 des Skripts:

```
CREATE DATABASE [drucker]ON (NAME = N'drucker_Data', FILENAME =
N'C:\Programme\Microsoft SQL Server\MSSQL\Data\drucker.mdf' , SIZE = 10, FILEGROWTH
= 10%)
LOG ON (NAME = N'drucker_Log', FILENAME = N'C:\Programme\Microsoft SQL
Server\MSSQL\Data\drucker_log.ldf' , SIZE = 1, FILEGROWTH = 10%)
```

Das Skript legt den Datenbank-Benutzer „CinvUser“ an. Über diesen Benutzer können die Webseiten auf die Datenbank zugreifen. Der Benutzer hat Lese- und Schreibzugriff auf alle Tabellen der Drucker-Datenbank, sowie Ausführberechtigung für die gespeicherten Prozeduren. Um sich bei dem Account einzuloggen, muss auf dem SQL Server die SQL-Authentifizierung aktiviert sein. Das Passwort des Benutzers „CinvUser“ wird vom Installationsskript auf „`HJsdf84/U34jas73s`“ gesetzt. Dieses sollte von dem Administrator später geändert werden.

B.3. Konfiguration des Web-Interfaces

Beim Web-Interface des Drucker-Inventarsystems werden die gleichen Anforderungen wie beim Web-Interface des Vorgängerprojektes für die Konfiguration benötigt. Diese sind: ein Microsoft Internet Information Server (IIS) ab Version 6 und das .NET-Framework Version 1.1, das auf dem Server installiert sein muss. Außerdem muss der Webserver für das Bereitstellen von ASP.NET-Anwendungen konfiguriert sein.

Da dieses Projekt eine Erweiterung des Vorgängerprojektes darstellt, werden die Dateien des Web-Interfaces in das vorhandene virtuellen Verzeichnis des Client-Inventarsystems kopiert.

B.3.1. Datenbankzugriff

Bei den Zugangsdaten für den Datenbankzugriff wird in der vorhandenen Datei „web.config“ die folgende Zeile am Ende hinzugefügt:

```
<add key="ConStr2"
value="server=vwagszcoia08;database=drucker;uid=CinvUser;pwd=HJsdf84/U34jas73s;" />
```

Hier sind die Zugangsdaten für den Datenbank-User „CinvUser“ einzutragen, welcher beim Erstellen der Datenbank automatisch vom SQL-Skript angelegt wurde. Aus Sicherheitsgründen kann diese Datei nicht über einen Web-Browser editiert werden. Daher ist diese Datei mit einem Texteditor zu öffnen um die Zugangsdaten anpassen zu können.

B.3.2. Überwachung und Steuerung des Drucker-Inventar-Dienstes

Um den Drucker-Inventar-Dienst über das Web-Interface überwachen zu können, muss die Konfiguration des .NET Remoting und des Web Service angepasst werden. Dies kann über einen Browser in den Verwaltungsseiten erledigt werden, oder man editiert die XML Datei „web.config“ direkt.

Dazu müssen die Werte in den folgenden Zeilen am Ende der Datei angepasst werden:

```
<add key="DIServername" value="vwagszcoia06" />
<add key="Port_DIDienst" value="7778" />
<add key="DIWebServiceURL" value="http://localhost/inventar/webservice.asmx" />
```

Hinter „DIServername“ muss der Name des Servers angegeben werden, auf dem der Dienst installiert ist. „Port_DIDienst“ gibt den Kommunikationsport an, der mit dem im Dienst eingestellten Port übereinstimmen muss. Bei „DIWebServiceURL“ muss schließlich der URL zum Web Service angegeben werden.

B.3.3. NTFS Berechtigungen

Wenn Berichte (CSV, PDF) von der Webseite aus erstellt werden sollen, so erfolgt der schreibende Zugriff auf das Berichtverzeichnis „csv_output“ mit der Identität des Webseiten-Benutzers. Es muss also sichergestellt sein, dass die Benutzer die notwendigen NTFS Berechtigungen in diesem Verzeichnis besitzen.

C. Wartung

Das Drucker-Inventar-System arbeitet weitestgehend selbständig und benötigt im Regelfall keine Überwachung durch einen Administrator. Die Datenbank wird bei den regelmäßigen Aktualisierungen durch den Drucker-Inventar-Dienst von veralteten Druckereinträgen befreit, wenn deren konfigurierter Aufbewahrungszeitraum überschritten ist. Der Zustand des Dienstes kann über das Web-Interface kontrolliert werden.

Der Drucker-Inventar-Dienst protokolliert seine Arbeit in der Datei „LOG.TXT“, welche im Installationsverzeichnis des Programms angelegt wird. Überschreitet die Logbuchgröße die Grenze von 1 MB, so wird eine neue Logbuch-Datei angelegt, und die bisherige Datei archiviert. Die archivierten Logbücher erhalten jeweils einen Dateinamen der Form „LOG_<Datum>_<Uhrzeit>.TXT“. Es werden die letzten 10 Logbücher vorgehalten, ältere werden automatisch gelöscht.

Die vom Webserver angelegten Berichte (CSV, PDF) werden im Unterverzeichnis „csv_output“ der Webanwendung gespeichert. Jedes mal bevor ein neuer Bericht erzeugt wird, werden alle vorhandenen Berichte, die älter sind als 15 Minuten, gelöscht.

D. Bedienungshandbuch der Webseiten

Das Web-Interface des Drucker-Inventarsystems verfügt über ein Menü, das sich auf der linken Seite die verschiedenen Berichtseiten anzeigen lässt.
Die zur Verfügung stehenden Menüoptionen sind:

D.1. Allgemeine Abfragen

Diese können von jedem Benutzer betrachtet werden. Die im Web-Interface bereitgestellten allgemeinen Abfragen sind „Druckerübersicht“, „MAC-Suche“, „Drucker-Details“ und „Zusammenfassung“.

Druckerübersicht

Auf dieser Seite werden alle vom Netzwerk erfassten Drucker aufgelistet. Im oberen Teil der Druckerübersicht steht der Link „Zusammenfassung“. Die enthaltenen Informationen dieses Links werden zu einem späteren Zeitpunkt erläutert.
Auf der linke Seite befinden sich die Textboxen „Druckername enthält“ und „Letzte Abfrage älter als Tagen“. Über diese Textboxen können Filter eingegeben werden, die die Abfrage weiter einschränken.

In der Textbox „Druckername enthält“ kann ein Teilstring eingegeben werden. Nach diesem String soll der Druckername gesucht werden. Hier kann ein String mit Groß- und Kleinschreibung eingetragen werden. Beispiele dieser Option sind:

- Suche nach dem Teilstring: „SZ“ in der Textbox „Druckername enthält“ zeigt alle Druckernamen, die die Zeichenfolge „SZ“ beinhalten.
- Suche nach mehrere Teilstrings: Wenn ein Druckername sowohl „SZ“ als auch „PS“ enthalten soll, können die beiden Strings mit dem Prozentzeichen „%“ aneinander gesetzt werden. Zum Beispiel: SZ % PS
- Suche mittels Platzhalter: In dem Suchstring kann auch ein Platzhalter für einen oder mehrere Buchstaben gestellt werden. Dieser Platzhalter ist der Unterstrich „_“. Zum Beispiel: „VWAGSZPOI_01“

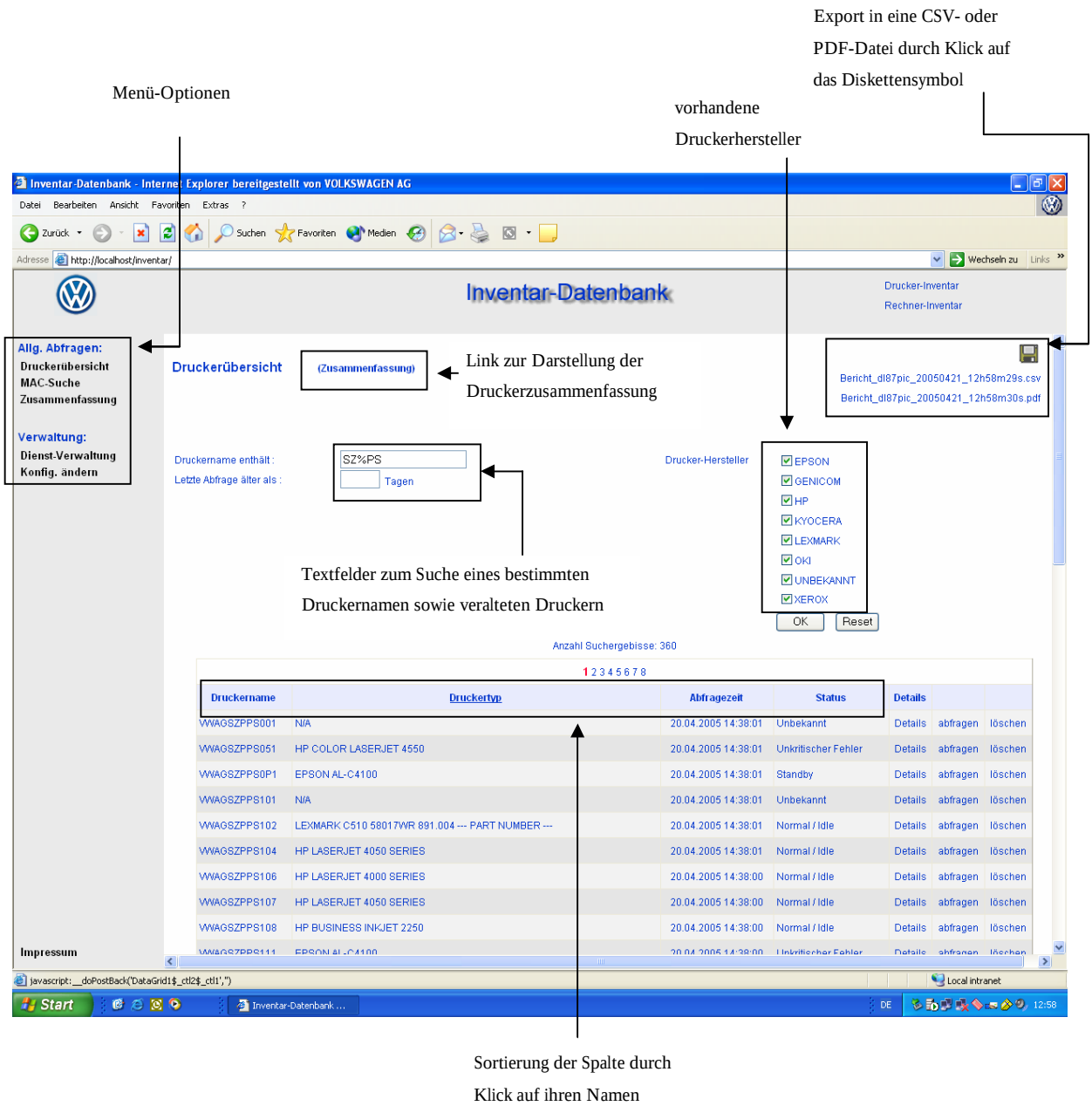

Abbildung 10.: Webseite „Druckerübersicht“

Bei der Textbox „Letzte Abfrage älter als Tagen“ können Werte zwischen 0 und 32000 eingetragen werden. Zum Beispiel, wenn der Wert 90 eingegeben wird, werden alle Drucker, die vor min. neunzig Tagen abgefragt wurden aufgelistet. Bei jeder anderen Eingabe wird eine Fehlermeldung ausgegeben.

Auf der rechten Seite können Drucker nach dem Hersteller über die Checkboxen ausgewählt werden.

Die erfassten Daten der Druckerübersicht werden in einer Tabelle dargestellt, in der sich Links befinden. Der Detaillink zeigt allen ausgelesenen Details zum ausgewählten Drucker (wie IP-, MAC-Adresse, Druckertyp, etc.) an.

Der Abfragelink fügt den Drucker in die Warteschlange des Inventarsystems ein. Der Drucker wird nun vom Netzwerk ausgelesen. Der Link „Löschen“ entfernt einen Drucker aus der Datenbank. Alle Spalten der Tabelle können durch Klicken ihres Namens sortiert werden. Auf dieser Seite können erzeugte CSV- oder PDF-Dateien gespeichert werden. Beim Klicken des Diskettensymbols werden beiden Dateien erzeugt und ein entsprechender Downloadlink eingeblendet. Diese Berichte liegen mindestens 15 Minuten auf dem Server und werden dann bei der nächsten Erzeugung neuer Berichte überschrieben.

MAC-Suche

Hier kann einen Drucker über seine MAC-Adresse oder einen Teil davon abgefragt werden. Abfragen über die IP-Adresse sowie dem Druckernamen können auch hier durchgeführt werden. Die Daten werden wie in der Option „Druckerübersicht“ in einer Tabelle dargestellt. Die Details eines Druckers können durch einen Klick auf den Druckernamen angezeigt werden.

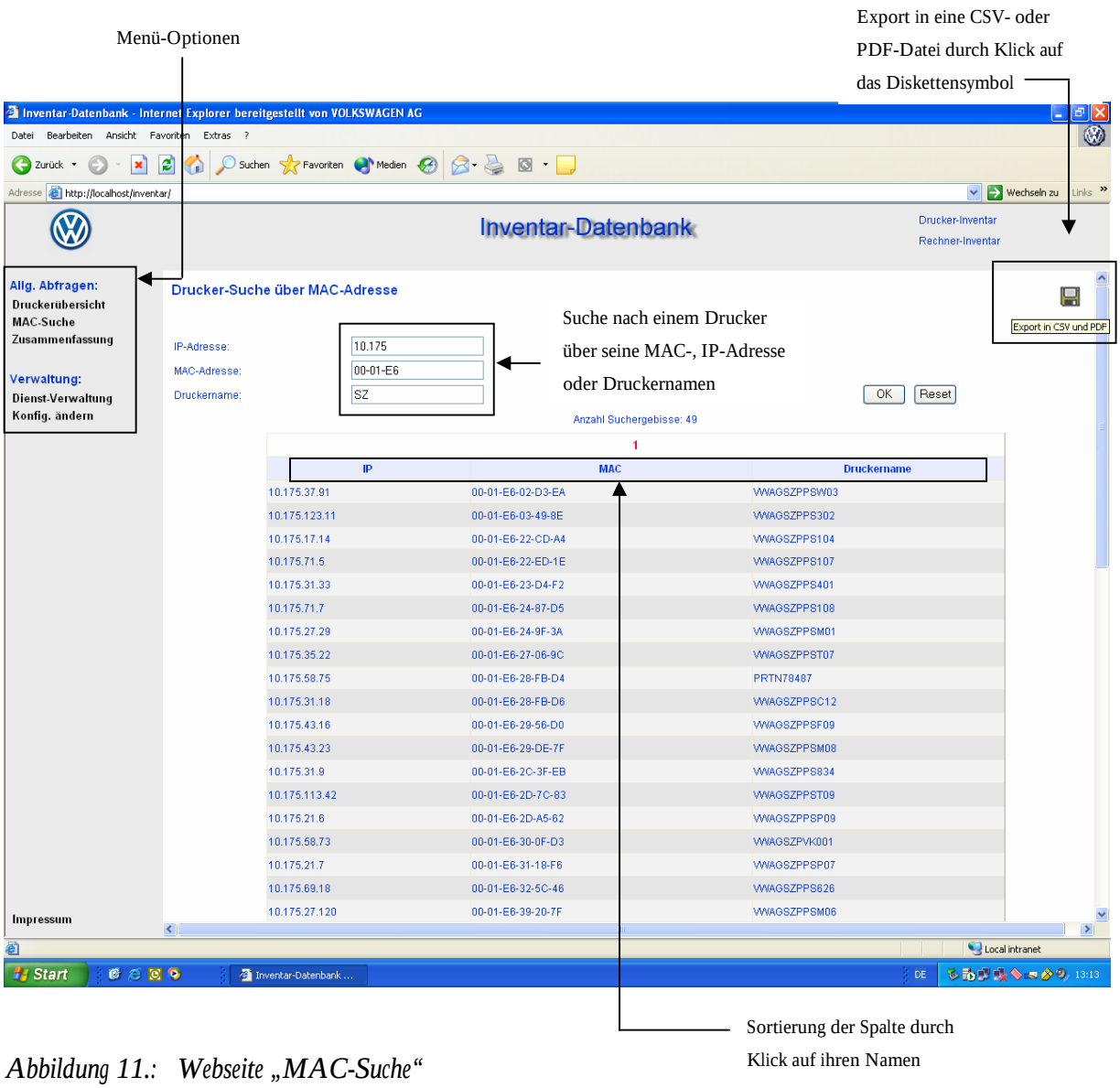

Abbildung 11.: Webseite „MAC-Suche“

Die Spalten der Informationstabelle lassen sich hier ebenfalls durch Klicken ihres Namens sortieren. Die Erzeugung und Speicherung von Berichten ist durch einen Klick auf das Diskettensymbol ebenfalls möglich.

Drucker-Details

Die Seite Drucker-Details bietet eine Darstellung der allgemeinen Details eines Druckers sowie Informationen über seine Tinten-, Toner- oder Papierfüllstände und Papierausgabenfächer an.

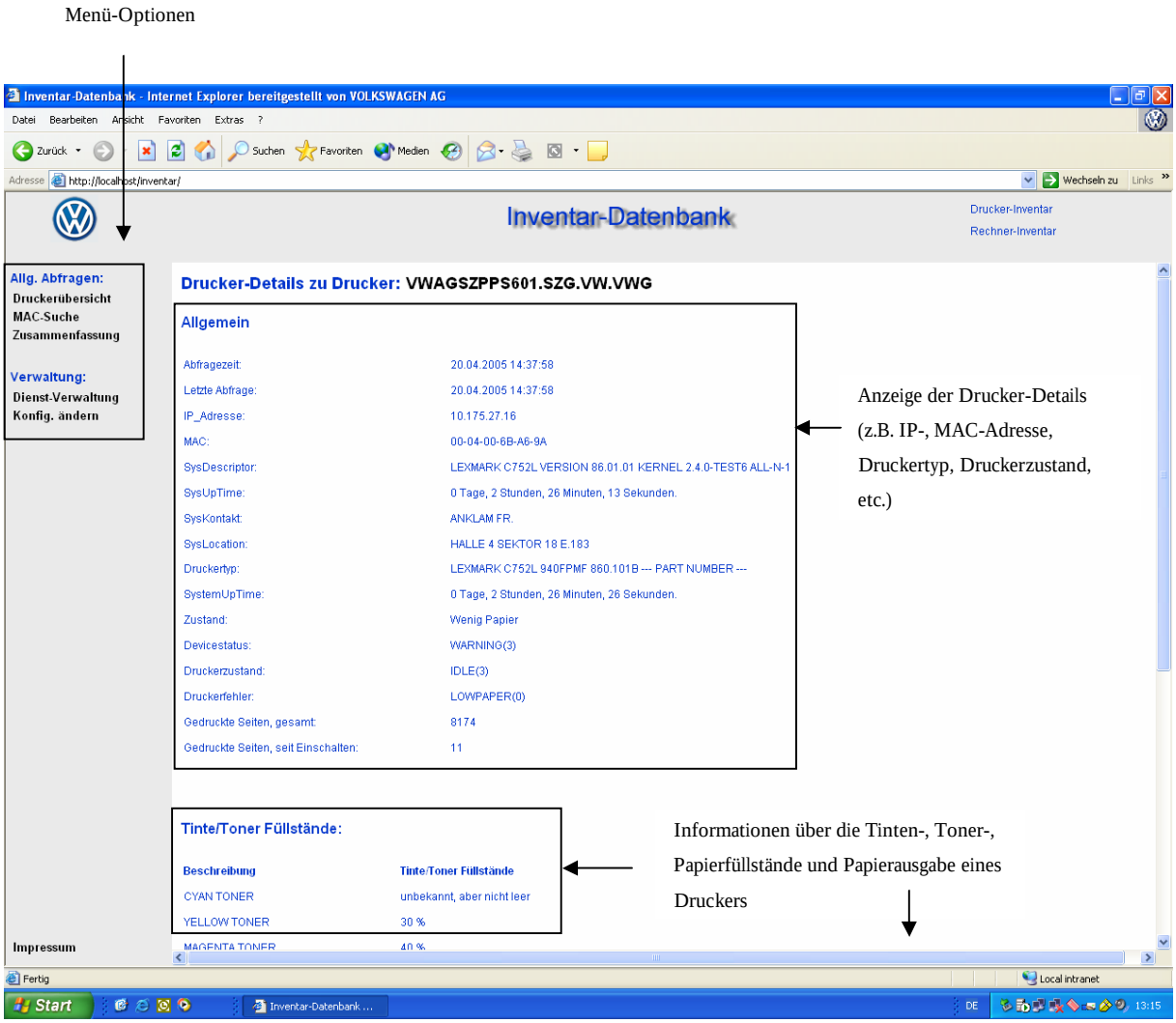

Abbildung 12.: Webseite „Drucker-Details“

Die in dieser Seite enthaltenen Informationen sind:

Allgemein

- Abfragezeit

  Zeitpunkt, an dem der Drucker abgefragt wurde.

- Letzte Abfrage

  Letzter Zeitpunkt, an dem der Drucker abgefragt wurde.
- IP-Adresse

  IP-Adresse des Druckers
- MAC

  MAC-Adresse des Druckers
- SysDescriptor

  Eine Beschreibung der Version, des Namens der Hardware, des Betriebssystems und der Netzwerksoftware.
- SysUpTime

  Zeitraum der vergangen ist, seitdem sich der SNMP-Prozess neu initialisiert hat.
- SysKontakt

  Name der Person, der Gruppe oder der Organisation die für die Wartung, Pflege usw. des Druckers verantwortlich ist.
- SysLocation

  Beschreibung, wo der Drucker sich befindet.
- Druckertyp
- SystemUpTime

  Zeitangabe, wann dieser Drucker zuletzt initialisiert wurde.
- Zustand
- Devicestatus

  Zustand des Gerätes
- Druckerzustand
- Druckerfehler
- Gedruckte Seiten (gesamt)
- Gedruckte Seiten (seit Einschalten)

Tinte/Toner Füllstände

- Beschreibung

  Eine Beschreibung der Tintepatronen bzw. Toner eines Druckers.

- Tinte/Toner Füllstände

  Prozentsatz oder Beschreibung der Tinte bzw. Toner Füllstände.

Papierfüllstände

- Beschreibung

  Beschreibung der verfügbaren Papiereingabefächer des Druckers.

- Papierfüllstände

  Prozentsatz oder Beschreibung der Papierfüllstände.

Papierausgabefächer

- Beschreibung

  Beschreibung der Papierausgabefächer eines Druckers.

- Papierausgabefächer

  Prozentsatz oder Beschreibung der Papierausgabefächer.

Zusammenfassung

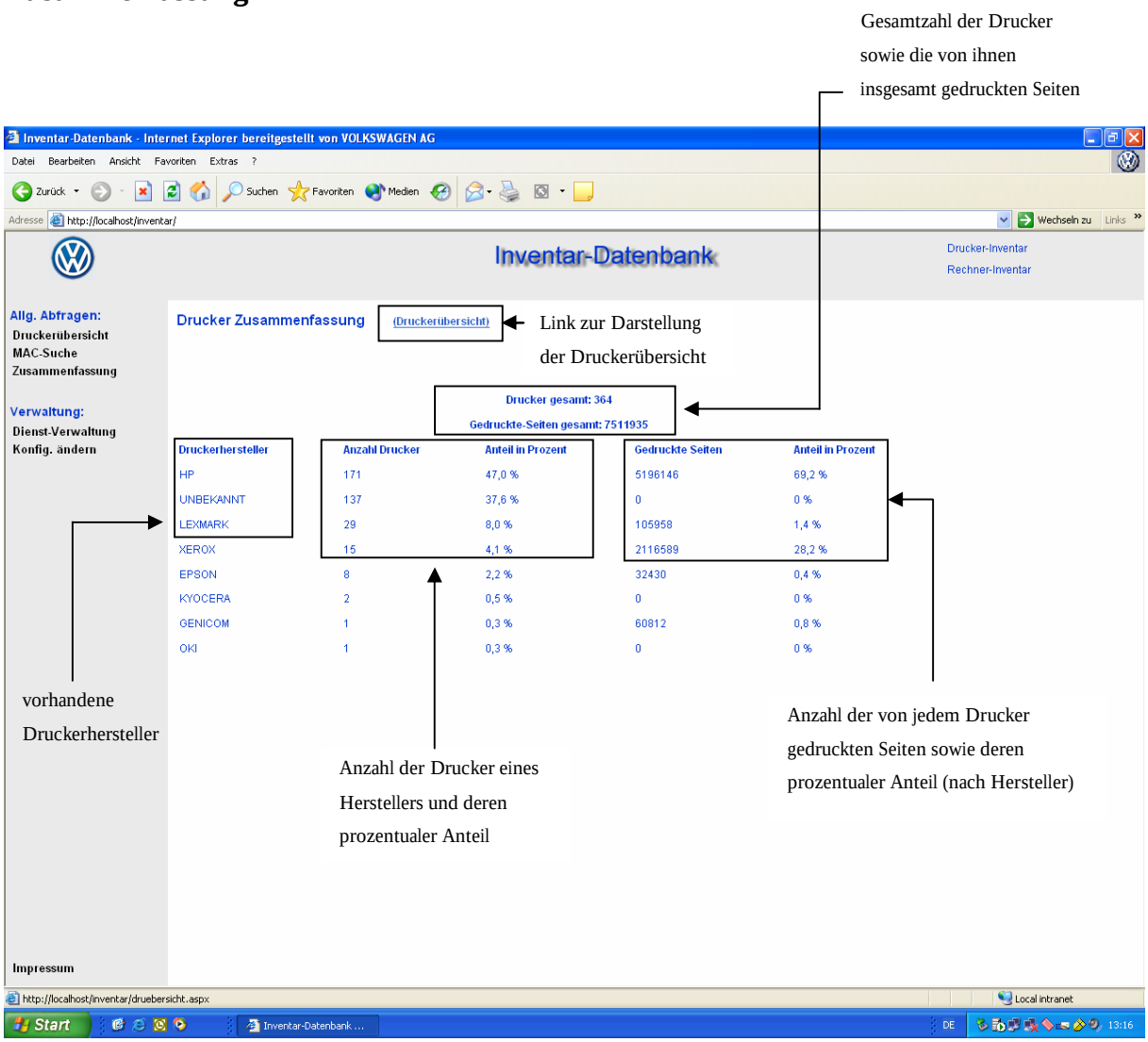

Abbildung 13.: Webseite „Zusammenfassung“

In dieser Seite werden alle Druckerhersteller mit der entsprechenden Anzahl der Drucker, die zu diesem Hersteller gehören, sowie die Anzahl der gedruckten Seiten von diesen Druckern zusammengefasst. Eine Darstellung der Anzahl der Drucker und der gedruckten Seiten in Prozent ist ebenfalls hier ablesbar.
Die gesamte Anzahl der erfassten Netzwerkdrucker, sowie die von ihnen gesamt gedruckten Seiten werden auch auf dieser Seite angezeigt.

Im oberen Teil der Zusammenfassung steht der Link „Druckerübersicht“, der komfortabel zu der entsprechenden Webseite dieses Links führt.

D.2. Verwaltung

Die Verwaltung steht nur DOI-Administratoren zur Verfügung. Die Bedienung der beiden Verwaltungsseiten ist weitgehend selbsterklärend aufgebaut. Für jedes Informationsfeld, jedes Eingabefeld und jede Schaltfläche sind Hilfstexte definiert, die beim überfahren mit dem Mauszeiger angezeigt werden.

Dienst-Verwaltung

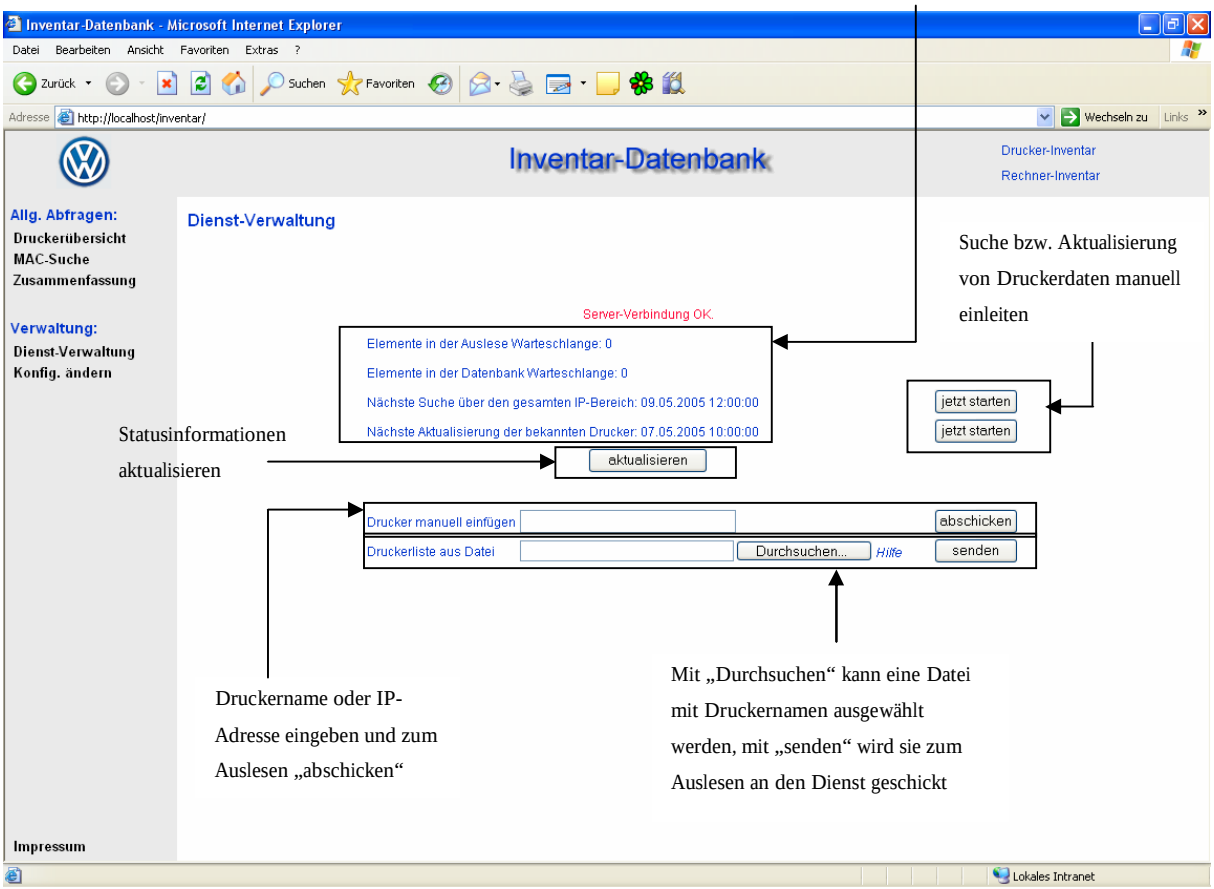

Abbildung 14.: Webseite Dienst-Verwaltung

Konfiguration ändern

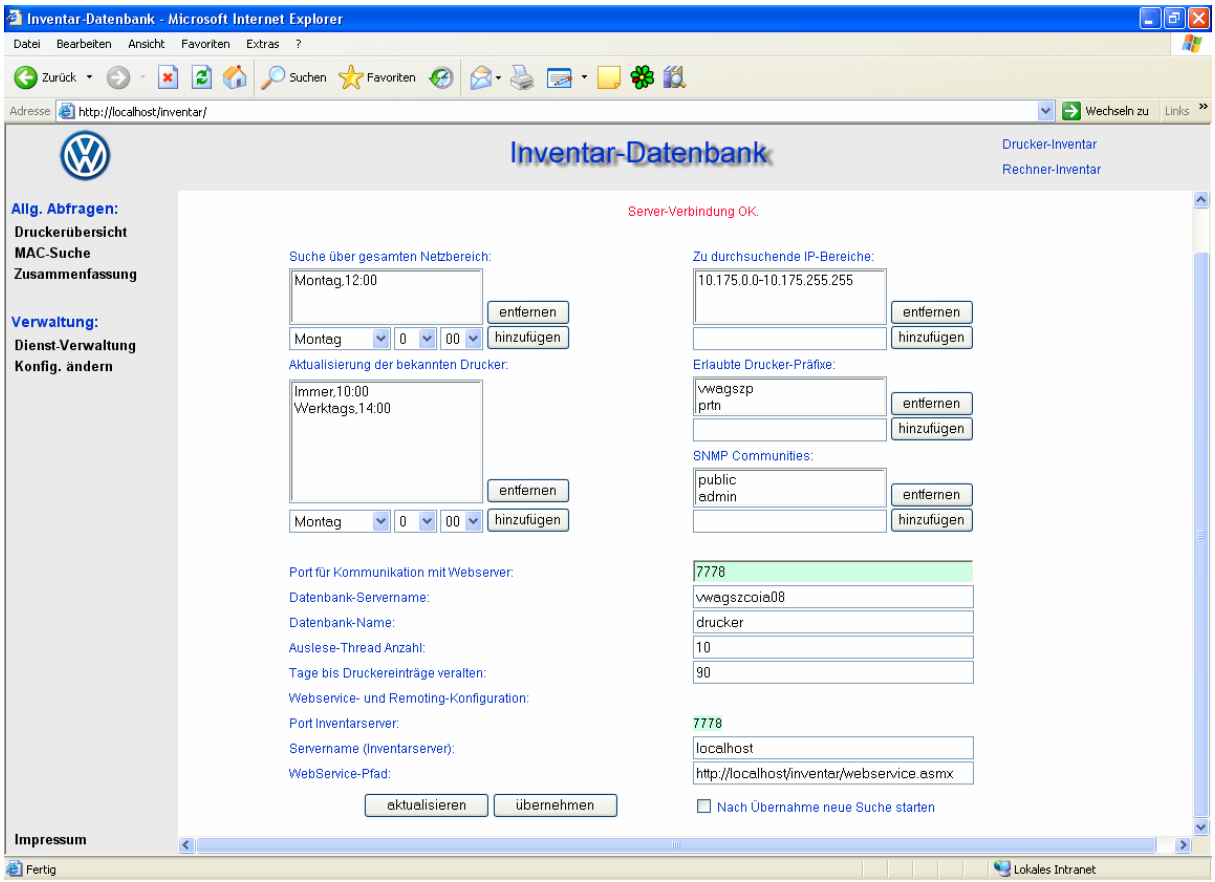

Abbildung 15.: Webseite Konfig. ändern

Abbildungsverzeichnis

Abbildung 1.: ADO.NET-Architektur 16
Abbildung 2.: SNMP Übersicht 42
Abbildung 3.: SNMP Funktionen 43
Abbildung 4.: OID Baumstruktur 46
Abbildung 5.: Ablaufplan Druckerinventarisierung 63
Abbildung 6.: 1:n Beziehung 78
Abbildung 7.: Datenbankdiagramm 102
Abbildung 8.: Flussdiagramm Datensatz einfügen 106
Abbildung 9.: Erwartete Datenbankgröße 107
Abbildung 10.: Webseite „Druckerübersicht“ 112
Abbildung 11.: Webseite „MAC-Suche“ 113
Abbildung 12.: Webseite „Drucker-Details“ 114
Abbildung 13.: Webseite „Zusammenfassung“ 116
Abbildung 14.: Webseite Dienst-Verwaltung 117
Abbildung 15.: Webseite Konfig. ändern 118

Abkürzungsverzeichnis

ADO.NET	Active Data Object .NET
ANSI	American National Standards Institute
API	Application Programming Interface
ARPANET	Advanced Research Projects Agency Network
ASN.1	Abstract Syntax Notation 1
ASP.NET	Active Server Page .NET
BER	Basic Encoding Rules
CBF	Code-Behind
CGI	Common Gateway Interface
CLR	Common Language Runtime
CLS	Common Language Specification
CMIP	Common Management Information Protocol
CMOT	CMIP over TCP
CSS	Cascading Style Sheets
CSV	Comma Separated Values
CTS	Common Type System
DCOM	Distributed Component Object Model
DLL	Dynamic Link Library
DNS	Domain Name Service
FTP	File Transfer Protocol
HANI	Hardware- und Netz-Information
HEMS	High-Level Entity Mangement System
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
IAB	Internet Activities Board
IBM	International Business Machines
ICMP	Internet Control Message Protocol
IIS	Internet Information Server
IP-Adresse	Internet Protocol-Adresse
ISAPI	Internet Server Application Programming Interface
ISO	International Organization for Standardization
JScript	Java Script
MAC-Adresse	Media Access Control-Adresse
MIB	Management Information Base

MMS	Multimedia Message Service
MSDE	Microsoft Data Engine
ODBC	Open Database Connectivity
OID	Object Identifier
OLE DB	Object Linking and Embedding Database
OPC	OLE for Process Control
OSI	Open Systems Interconnection
PDA	Personal Digital Assistent
PDF	Portable Document Format
PDU	Protocol Data Unit
PHP	PHP: Hypertext Preprocessor
RAM	Random Access Memory
RFC	Request for Comments
RPC	Remote Procedure Calls
SGMP	Simple Gateway Monitoring Protocol
SMI	Structure of Management Information
SMS	Short Message Service
SMTP	Simple Message Transfer Protocol
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protokoll
SQL	Structured Query Language
TCP	Transmission Control Protocol
Transact-SQL	Transactional Structured Query Language
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VB.NET	Visual Basic .NET
VBScript	Visual Basic Script
VW-Netzwerk	Volkswagen-Netzwerk
WAP	Wireless Application Protocol
WMI	Windows Management Instrumentation
WSDL	Web Service Definition Language
XML	Extended Markup Language

Literaturverzeichnis

[1] Günter Born
Visual Basic .NET
Markt und Technik, 2002
ISBN: 3-8272-6449-9

[2] Jürgen Bayer
Visual Basic .NET - Nitty Gritty
Addison-Wesley, 2002
ISBN: 3-8273-1857-2

[3] Dan Fox
Verteilte Anwendungen mit Visual Basic .NET
Markt und Technik, 2002
ISBN: 3-8272-6319-0

[4] Christian Wenz, Andreas Kordwig, Christian Trennhaus
ASP .NET – Nitty Gritty
Addison-Wesley, 2003
ISBN: 3-8273-2048-8

[5] Ryan K. Stephens u.a.
SQL in 21 Tagen
Markt und Technik, 1998
ISBN: 3-8272-2020-3

[6] Klaus Giesen
SQL Server 2000, Das bhv Taschenbuch
bhv, 2002
ISBN: 3-8266-8117-7

Weitere Quellen:

[7] Robin Schulz, Carsten Grünewald
Diplomarbeit und Anlagenband zum Thema
„Client-Inventory in Microsoft-basierten Netzwerken“
Sommer 2004

[8] Sebastian Eichner
Dynamische Inhalte in Webseiten
Sommersemester 2001
http://www.sebastian-eichner.de/study/sem_dynamic_content/pdf/Seminar_Dynamic_Content.pdf

[9] Jürgen Klein
SNMP Techniker-Projekt
http://www.jklein.de/technikerseite.htm

[10] MSDN Library
http://msdn.microsoft.com/library/spa/ (Spanisch)
http://msdn.microsoft.com/library/deu/ (Deutsch)

[11] Google: http://www.google.de