

Universität Hannover
Institut für Rechnergestützte Wissensverarbeitung
Prof. Dr. techn. W. Nejd1

Studienarbeit

**Ein agentenorientiertes
Konfigurationsmanagementsystem
in Java**

Verfasser: Enno-Edzard Steen
Betreuer: Dipl.-Inform. P. Fröhlich
Erstprüfer: Prof. Dr. techn. W. Nejd1
Zweitprüfer: Prof. Dr.-Ing. C. Müller-Schloer

Datum: 30. April 1998

Inhaltsverzeichnis

1. Einleitung	4
2. Allgemeine Informationen zu den Aglets und Beschreibung der einzelnen Agenten sowie der Funktionsweise der agentenbasierten Versionskontrolle AVC	6
2.1 Was sind Aglets ?	6
2.2 Beschreibung der einzelnen Agenten der agentenbasierten Versionskontrolle	7
2.2.1 Die Realisierung als Client/Server-Architektur	7
2.2.2 Die stationären Agenten auf dem Server	8
2.2.3 Die stationären Agenten auf dem Client	9
2.2.4 Der mobile Agent auf dem Client	9
2.2.5 Die mobilen Agenten auf dem Server	9
2.2.6 Mobile Agenten zwischen dem Server und den Clients	10
2.3 Wie funktioniert die agentenbasierte Versionskontrolle ?	10
2.3.1 Erzeugung des Verwaltungsagenten und der serverseitigen Benutzer-Agenten	10
2.3.2 Bereitstellung der Benutzeroberfläche und Erzeugung und Beseitigung des clientseitigen Benutzeragenten	12
2.3.3 Regelmäßige Überprüfung der überwachten Dateien	14
2.3.4 Überprüfung einer einzelnen, vom Benutzer ausgewählten Datei	16
2.3.5 Benachrichtigung des Benutzers über die Unterschiede zu anderen Benutzern	18
3. Installation des Programmes	20
3.1 Installation auf dem Server	20
3.2 Installation auf dem Rechner des Benutzers	21

4. Benutzeroberfläche	23
4.1 Erzeugung und Beseitigung des Clientagenten	23
4.2 Benutzerangaben festlegen	23
4.3 Auswählen der regelmäßig überprüften Dateien	24
4.4 Sofortige Überprüfung einer ausgewählten Datei	25
4.5 Benachrichtigung des Benutzers im Falle eines CVS-Konfliktes	26
4.6 Ankunft einer neuen Mail	26
4.7 Anzeigen und Löschen der gespeicherten Mails	27
5. Beschreibung der Klassen und Textdateien	29
5.1 Die Benutzeroberfläche	29
5.1.1 AuswahlmenuAnzeigen	29
5.1.2 FilemenuAnzeigen	31
5.1.3 HauptmenuAnzeigen	34
5.1.4 KonfliktAlarm	36
5.1.5 MailAlarm	37
5.1.6 MailAnzeigen	37
5.1.7 MailmenuAnzeigen	39
5.1.8 VerwaltungsmenuAnzeigen	41
5.2 Die Server-Organisation	43
5.2.1 ServerDir	43
5.2.2 ServerURL	43
5.3 Die Agenten	44
5.3.1 Clientagent	44
5.3.2 ClientPause	47
5.3.3 ClientServerAnmeldung	48

5.3.4 ClientStart	49
5.3.5 DiffAgent	50
5.3.6 FileAgent	51
5.3.7 FilenamenAgent	52
5.3.8 FilePruefAgent	53
5.3.9 MailAgent	53
5.3.10 Serveragent	54
5.3.11 ServeragentStart	60
5.3.12 ServerAnmeldung	61
5.3.13 Serverstart	62
5.3.14 Verwaltungsagent	62
5.4 Die Textdateien	64
5.4.1 Benutzerliste.txt	64
5.4.2 FileListe.txt	65
5.4.3 NeuerBenutzer.txt	65
5.4.4 User.txt	66
5.4.5 Userdaten.txt	66
6. Diskussion und Ausblick	67
7. Verwendete Materialien	68

1. Einleitung

Ohne Einsatz einer Versionskontrolle kann immer nur die aktuellste Version einer von einem oder von mehreren Benutzern bearbeiteten Datei gespeichert bzw. geladen werden. Die früheren Versionen dieser Datei sind hierbei nicht mehr zugänglich, da sie gelöscht worden sind und somit nicht mehr reproduzierbar sind. Um diesen Nachteil zu beheben, können die Benutzer das CVS (Concurrent Versions System) verwenden. CVS ist nämlich ein Versionskontroll-System, in dem alle jemals erzeugten Versionen einer durch das CVS verwalteten Datei zugänglich sind. Um dieses zu erreichen, werden allerdings nicht die einzelnen Versionen komplett gespeichert, sondern nur die Unterschiede zwischen den einzelnen Versionen einer Datei. Jede einzelne Version einer Datei besitzt hierbei eine Versionsnummer, die zur Unterscheidung zu den entsprechenden anderen Versionen dient.

Arbeiten mehrere Benutzer an denselben CVS-Dateien, so kann es dabei vorkommen, daß sie gleichzeitig inkompatible Änderungen an einer CVS-Datei ausführen, ohne es zu bemerken. Das kann dazu führen, daß die Ausführung der bearbeiteten und gemergten Datei nicht möglich ist, wenn die Änderungen des einen Benutzers inhaltlich im Widerspruch zu den Änderungen eines anderen Benutzers stehen (z.B.: unterschiedliche Variablenbenennungen, unterschiedliche Variableninitialisierungen), oder daß zwei Dateiversionen nicht gemergt werden können, weil sich die Bereiche der Datei, die der eine Benutzer modifiziert, mit den Bereichen, die der andere Anwender bearbeitet, überschneiden.

Aufgabe meiner hier vorliegenden Studienarbeit war es, eine bessere Koordination zwischen den einzelnen Benutzern zu erzielen. Um dieses Ziel zu erreichen, habe ich die agentenbasierte Versionskontrolle AVC entworfen und programmiert. Dieses Programm stellt die zwischen den einzelnen Benutzerversionen derselben CVS-Datei vorkommenden Unterschiede fest und teilt sie den betroffenen Benutzern mit, wobei jeder Benutzer dieses Programmes durch seinen vollständigen Namen und durch eine Nummer identifiziert wird. Es wird ferner das Client/Server-Konzept verwendet, bei dem die einzelnen Benutzer den Clients zuzuordnen sind. Die Überprüfung auf Unterschiede zwischen den entsprechenden Dateien, die unabhängig von der Anwesenheit der Benutzer ist, wird hierbei, soweit sie aufgrund der einzelnen Versionsnummern zulässig ist, auf dem Server durchgeführt, auf dem sich auch die überwachten Dateien befinden.

Die Unabhängigkeit von der Anwesenheit der Benutzer und die Selbständigkeit bei der Erkennung der Unterschiede wird durch ein Agentensystem realisiert. Bei der Implementierung dieses Konzeptes werden sowohl stationäre Agenten für Verwaltungsaufgaben auf Client und Server eingesetzt als auch mobile Agenten für die Kommunikation zwischen den einzelnen stationären Agenten. Alle diese Agenten verwenden die von IBM entwickelte, in Java geschriebene Aglet Workbench AWB, wobei die Agenten als Aglets bezeichnet werden.

In Abbildung 1.1 auf der kommenden Seite ist ein Beispiel für das Konzept meiner Studienarbeit skizziert. In diesem Beispiel bearbeiten die drei Benutzer A, B und C die Datei xy. Somit existieren die drei Benutzerversionen Datei xy_A, Datei xy_B und Datei xy_C, die im Falle des Mergens die Datei xy ergeben. Die agentenbasierte Versionskontrolle überprüft in diesem Beispiel, ob zwischen den drei Benutzerversionen Unterschiede bestehen, und informiert daraufhin bei Feststellung von Unterschieden die betroffenen Benutzer. Die eben erwähnte Untersuchung wird auch bei Abwesenheit der Benutzer durchgeführt, was in Abbildung 1.1 durch die Abwesenheit des Benutzers B zum Ausdruck kommt.

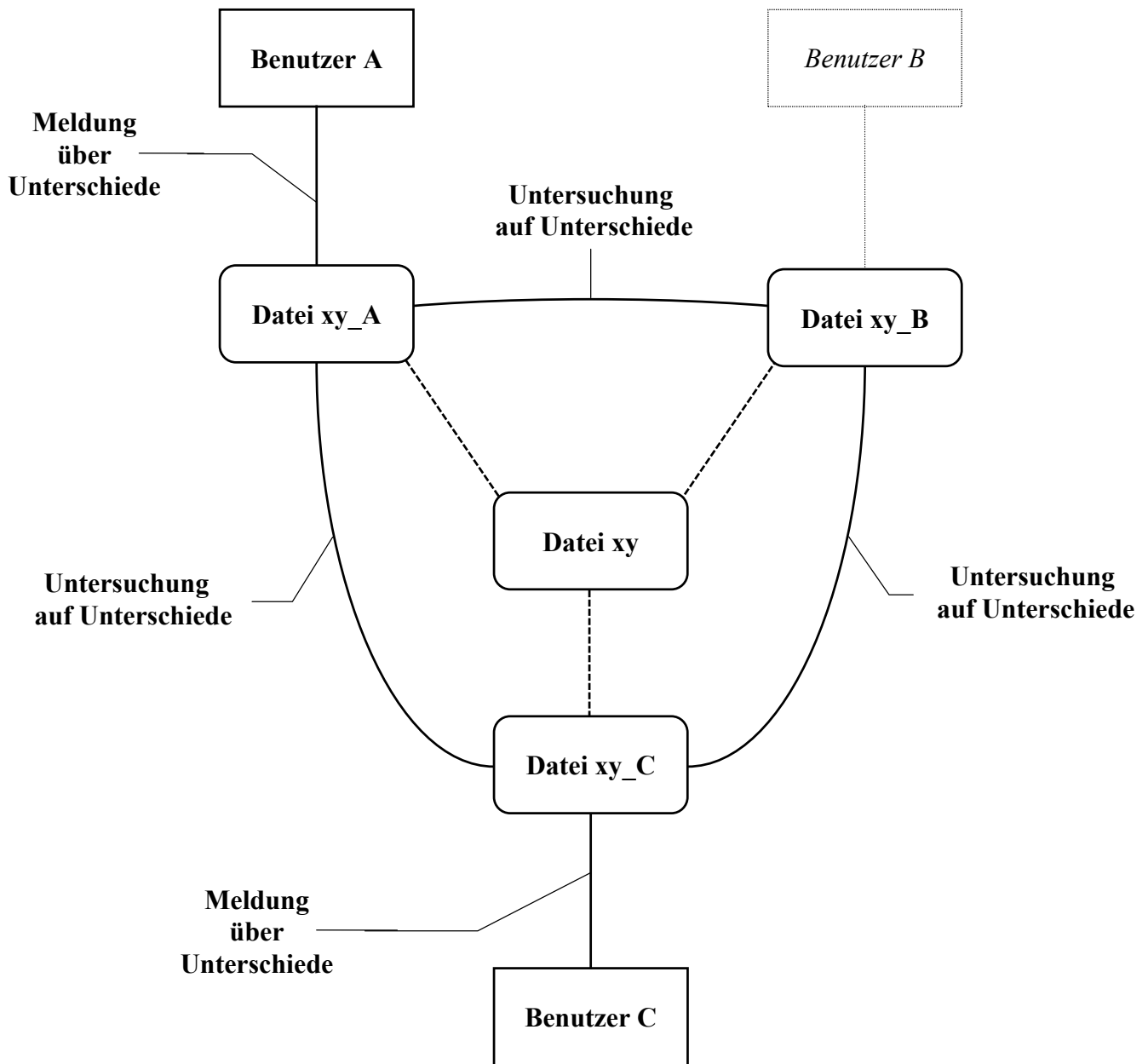


Abb. 1.1: Beispiel für das Konzept der agentenbasierten Versionskontrolle

2. Allgemeine Informationen zu den Aglets und Beschreibung der einzelnen Agenten sowie der Funktionsweise der agentenbasierten Versionskontrolle AVC

2.1 Was sind Aglets ?

Ein Aglet ist ein auf Java basierender selbständiger Software-Agent. Es handelt sich hierbei um einen mobilen Agenten, der innerhalb eines Computernetzwerkes alle aglet-fähigen Hosts besuchen kann. Dabei kann er seine Ausführung auf dem einen Host anhalten und zu einem anderen Host gelangen, um dort seine Aufgaben zu erfüllen. Die Autonomie dieses Agenten wird dadurch erreicht, daß er in der Umgebung seiner Erzeugung oder des besuchten Hosts einen eigenen Ausführungsthread besitzt, wobei ein Thread als ein einzelner sequentieller Kontrollfluß innerhalb eines Programmes definiert ist. Aufgrund des Multithreadings können also mehrere Aglets gleichzeitig auf einem Host ausgeführt werden. Ein Aglet kann auch als stationärer Agent seine Aufgaben erledigen, wobei dieser stationäre Agent als Spezialfall der mobilen Agenten angesehen wird. Außerdem besitzt ein Aglet die Fähigkeit, auf an ihn gesandte Nachrichten zu reagieren.

Der Aufenthaltsort bzw. Arbeitsplatz eines Aglets wird als Context bezeichnet. Dieser Context ist ein stationäres Objekt auf einem Rechner. In dieser als Context benannten Umgebung befinden sich die einzelnen Aglets, die entweder diesen Context besuchen oder in ihm erzeugt worden sind, und werden in ihr ausgeführt. Es ist dabei selbstverständlich, daß ein Aglet sich zu einer bestimmten Zeit nur in einem Context aufhalten kann. Auf einem Rechner können mehrere Contexts existieren, die sich voneinander durch ihre Portnummer unterscheiden. Somit kann ein Aglet seinen derzeitigen Context verlassen, um in einen anderen Context zu gelangen, der sich entweder auf dem gleichen oder auf einem anderen Rechner befindet.

Jedes Aglet besitzt einen Proxy. Dieser Proxy fungiert dabei als Vertreter dieses Aglets, das er vor dem direkten Zugriff auf dessen public-Methoden beschützt. Weiterhin sorgt ein Proxy für die Ortstransparenz seines Aglets, da er dessen tatsächlichen Aufenthaltsort verbergen kann, denn die Kommunikation der Aglets untereinander wird über die zugehörigen Proxies abgewickelt. Die gerade erwähnte Kommunikation wird unter Verwendung von Nachrichten, auch messages genannt, bewerkstelligt, wobei der Austausch dieser Nachrichten zwischen den betroffenen Aglets, genauer gesagt zwischen den Proxies dieser Aglets, sowohl synchron als auch asynchron sein kann. Durch die schon genannte Mobilität sind die Aglets in der Lage, mehrere Contexts nacheinander zu besuchen. Sie erhalten dafür einen Reiseplan, genannt itinerary, beim Beginn ihrer Reise übergeben. Ein weiteres Merkmal eines Aglets ist sein Identifikationswort (Aglet-Identifizierer AID), das global einzigartig und zur Lebenszeit des Aglets unveränderlich ist.

Es können während der Lebenszeit des Aglets die folgenden Ereignisse vorkommen: Creation, Cloning, Dispatching, Retraction, Deactivation, Activation, Disposal, Messaging. Zuerst muß das Aglet natürlich erzeugt werden. Es bekommt dabei sein Identifikationswort zugeteilt, wird in den entsprechenden Context eingesetzt und initialisiert. Danach beginnt das Aglet mit der Ausführung seiner Aufgaben. Ein neues Aglet muß allerdings nicht zwangsläufig neu erzeugt worden sein (Creation), sondern es kann auch durch Klonen eines bereits vorhandenen Aglets kreierte worden sein (Cloning). Das geklonte Aglet erhält hierbei ein neues Identifikationswort und befindet sich in dem gleichen Context wie das Original. Der

Ausführungsthread wird aber nicht geklont, so daß das neue Aglet die Ausführung seiner Aufgaben vom Anfang an beginnt.

Weiterhin kann ein Aglet seinen derzeitigen Context verlassen und in einen anderen Context gelangen, wo es mit der Ausführung seiner Aufgaben von neuem beginnt (Dispatching). Man kann nun darauf warten, daß dieses Aglet in seinen alten Context zurückkehrt, oder man kann dessen frühzeitige Rückkehr erzwingen (Retraction). Soll ein Aglet von einem Context in einen anderen gelangen (Dispatching bzw. Retraction), so wird dabei ein Transfer-Protokoll verwendet, um das Aglet (Byte-Code und Zustandsinformation) sicher zum anderen Context zu bringen. Die Zustandsinformation des Aglets wird in diesem Fall unter Verwendung der 'Object Serialization' in einem Byte-Strom konserviert und durch Umkehrung dieser Technik in dem Ziel-Context zurückerhalten.

Eine weitere Möglichkeit ist die Deaktivierung eines Aglets (Deactivation). Hierbei wird das Aglet für eine vorgegebene Zeit aus dem derzeitigen Context entfernt und schlafen gelegt. Dieses wird durch Speicherung, z.B. auf der Festplatte, erreicht. Ist die festgelegte Zeit um, so kehrt das Aglet in den Context zurück und setzt seine Ausführung fort. Es besteht auch die Möglichkeit, vor Ablauf der vorgegebenen Zeit das Aglet durch Aktivierung aufzuwecken (Activation), damit es nach seiner Rückkehr in den Context seine Aufgaben verrichtet.

Soll die Lebenszeit des Aglets zu Ende gehen, so wird es beseitigt, wobei die laufende Ausführung gestoppt und das Aglet aus seinem Context entfernt wird (Disposal). Dies ist wichtig, denn das Aglet beansprucht verschiedene Ressourcen, solange es sich in seinem Context aufhält. Die Aglets sollen deshalb möglichst wenig Ressourcen in Anspruch nehmen. Wird ein Aglet beseitigt, so fordert das System die von diesem verwendeten Ressourcen zurück. Der betroffene Aglet-Context gibt zu diesem Zweck die entsprechenden Ressourcen frei.

Ein anderer Punkt ist das Schicken, Empfangen und Behandeln von Nachrichten, die zwischen den Aglets ausgetauscht werden (Messaging). Eine Nachricht wird dabei durch Angabe ihrer Art charakterisiert und unterscheidet sich hierdurch von anderen Nachrichten. Wie schon oben erwähnt, erfolgt dieser Nachrichtenaustausch zwischen den Aglets mit Hilfe von Proxies, die den zugehörigen Aglets als Schutzschilder dienen und die für die bereits genannte Ortstransparenz der Aglets verantwortlich sind.

Zu den oben genannten Ereignisse im Leben eines Aglets muß noch ergänzend hinzugefügt werden, daß, wenn ein Aglet darüber informiert wird, daß es dupliziert, in einen anderen Context geschickt, aus einem anderen Context zurückgeholt, deaktiviert oder beseitigt werden soll, es sich darauf vorbereiten oder dieses ablehnen kann.

2.2 Beschreibung der einzelnen Agenten der agentenbasierten Versionskontrolle

2.2.1 Die Realisierung als Client/Server-Architektur

Die agentenbasierte Versionskontrolle verwendet das Client/Server-Prinzip, wobei die einzelnen Benutzer den Clients zuzuordnen sind. Jeder Client ist mit dem Server verbunden, der seinerseits für die Überprüfung der Dateien auf Unterschiede zuständig ist. Auf dem Server und auf den Clients existieren stationäre Agenten, die sich unterteilen lassen in einen mit der

serverseitigen Verwaltung beauftragten Agenten und in die server- und clientseitigen Benutzeragenten. Diese stationären Agenten kommunizieren unter Verwendung von mobilen Agenten. Ein mobiler Agent wird dabei von dem beauftragenden stationären Agenten erzeugt, wobei dieser stationäre Agent als Master des von ihm erzeugten mobilen Agenten bezeichnet wird. Dieser mobile Agent ist in diesem Zusammenhang der Slave seines Master. Hat der mobile Agent seine Aufgaben erfüllt, so kehrt er zu seinem Master zurück und informiert diesen über das Resultat seines Auftrages, bevor er beseitigt wird. Der clientseitige Benutzeragent gebraucht außerdem für die Erfüllung seiner Aufgaben einen mobilen Agenten, wobei auch hier das Master/Slave-Prinzip angewendet wird.

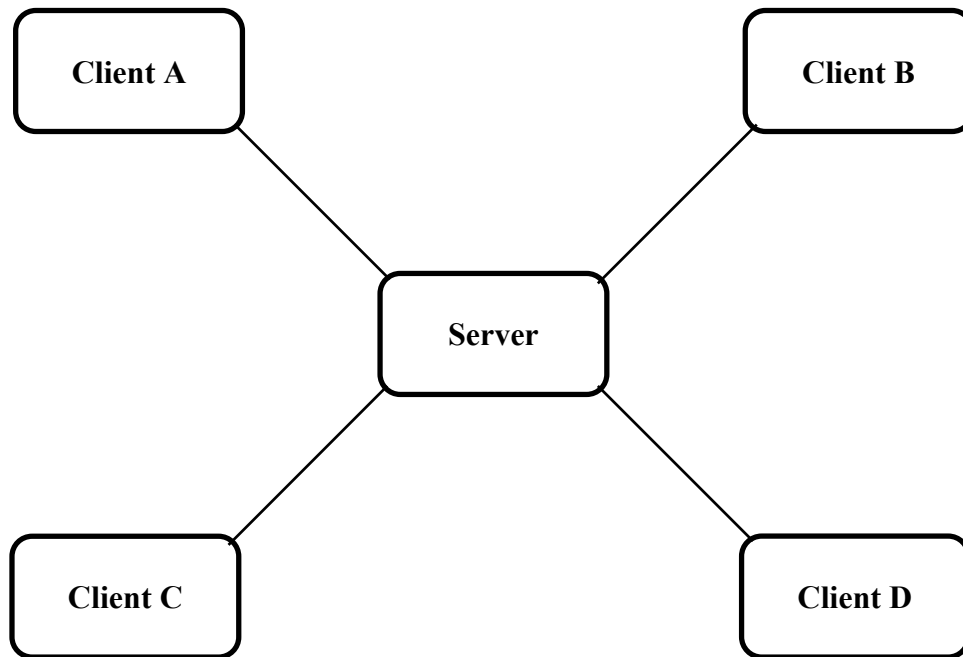


Abb. 2.1: Die Client/Server-Architektur

2.2.2 Die stationären Agenten auf dem Server

Serveragent:

Der Serveragent ist der serverseitige Benutzeragent. Jeder Anwender, der mit diesem Programm arbeitet, besitzt also seinen eigenen Serveragenten. Dieser Agent ist auf dem Server verantwortlich für die Untersuchung der Dateien seines Benutzers auf Unterschiede zu den entsprechenden überwachten Dateiversionen der anderen Benutzer und, falls er dabei Unterschiede feststellt, für die Benachrichtigung seines Benutzers darüber.

Verwaltungsagent:

Ein weiterer wichtiger Agent ist der Verwaltungsagent. Er verwaltet zum einen eine Liste mit den einzelnen Serveragenten und zum anderen eine Liste pro überwachten Dateinamen, in der die Serveragenten aufgeführt sind, die die zugehörige Datei beobachten lassen.

Serverstart und ServeragentStart:

Wird die agentenbasierte Versionskontrolle auf dem Server gestartet, so müssen hierbei der

Verwaltungsagent und die einzelnen Serveragenten generiert werden. Die Erzeugung dieser Agenten wird vom Serverstart-Agenten durchgeführt, dessen Lebenszeit dabei auf die Verrichtung dieser Aufgabe beschränkt ist. Möchte ein neuer Benutzer mit diesem Programm arbeiten und existieren der Verwaltungsagenten und die Serveragenten der anderen Anwender bereits, so wird der Serveragent dieses neuen Benutzers durch den ServeragentStart-Agenten nachträglich erzeugt. Danach beseitigt sich der ServeragentStart-Agent.

2.2.3 Die stationären Agenten auf dem Client

Clientagent und Clientstart:

Der clientseitige Benutzeragent wird als Clientagent bezeichnet. Er legt fest, welche Dateien im Rahmen der nächsten Überprüfung bei seinem Serveragenten untersucht werden sollen, und startet danach diese Überprüfung. Seine Lebenszeit bestimmt der Benutzer innerhalb der Benutzeroberfläche. Die Bereitstellung dieser Benutzeroberfläche und die Erzeugung bzw. Beseitigung des Clientagenten sind Aufgaben des Clientstart-Agenten. Dieser Agent beseitigt sich, wenn der Benutzer ihn innerhalb der Benutzeroberfläche damit beauftragt.

2.2.4 Der mobile Agent auf dem Client

ClientPause:

Nach Ablauf einer vom Benutzer festgelegten Zeitspanne startet der Clientagent eine Überprüfung der von ihm vorher bestimmten Dateien, wobei diese Überprüfung auch als regelmäßige Untersuchung bezeichnet wird. Um diese Aufgabe erfüllen zu können, benutzt der Clientagent den ClientPause-Agenten, indem er diesen ClientPause-Slave erzeugt und ihm den Zeitabstand zwischen zwei regelmäßigen Überprüfungen übergibt. Dieser ClientPause-Agent deaktiviert sich dabei für den übergeben bekommenen Zeitabstand. Nach Ablauf dieser Zeitspanne aktiviert sich der ClientPause-Agent wieder und kehrt zu seinem Master zurück, um dem Clientagenten darüber in Kenntnis zu setzen, daß der Zeitpunkt für eine erneute regelmäßige Überprüfung gekommen ist. Danach ist die Lebenszeit dieses ClientPause-Slaves beendet.

2.2.5 Die mobilen Agenten auf dem Server

DiffAgent:

Soll der Serveragent eine Dateiversion seines Benutzers auf Unterschiede zu den Dateiversionen der anderen Benutzer, die diese Datei überwachen lassen, untersuchen, so erzeugt dieser Serveragent einen DiffAgenten. Dieser DiffAgent sammelt dann für seinen Master die Dateiversionen der anderen Benutzer ein und übergibt sie seinem Master, nachdem er zu diesem zurückgekehrt ist. Anschließend geht die Lebenszeit dieses Slaves zu Ende.

FilennamenAgent:

Hat der Benutzer die Liste der in seinem Auftrag überwachten Dateien verändert, so erzeugt der Serveragent dieses Benutzers den FilennamenAgenten. Dieser FilennamenAgent informiert daraufhin den Verwaltungsagenten über die vom Benutzer gemachten Veränderungen. Danach kehrt dieser Slave zu seinem Master zurück und teilt diesem mit, daß der Verwaltungsagent bezüglich der von ihm überwachten Dateien auf den neusten Stand gebracht worden ist. Im Anschluß daran wird der FilennamenAgent wieder beseitigt.

ServerAnmeldung:

Möchte sich der Serveragent beim Verwaltungsagenten an- bzw. abmelden, so generiert er einen ServerAnmeldung-Slave. Dieser Slave informiert daraufhin den Verwaltungsagenten über den Grund seiner Erzeugung. Sollte er seinen Master anmelden, so kehrt er vor seiner Beseitigung zum Serveragenten zurück und benachrichtigt diesen über dessen erfolgreiche oder nicht erfolgreiche Anmeldung beim Verwaltungsagenten. Bestand die Aufgabe dieses ServerAnmeldung-Agenten in der Abmeldung seines Masters, dann wird dieser Slave beseitigt, sobald er den Verwaltungsagenten benachrichtigt hat.

2.2.6 Mobile Agenten zwischen dem Server und den Clients

ClientServerAnmeldung:

Der ClientServerAnmeldung-Agent wird vom Clientagenten erzeugt, wenn dieser sich beim zugehörigen Serveragenten an- oder abmelden möchte. Dieser Slave informiert diesen Serveragenten dabei über den Grund seiner Erzeugung und gelangt danach wieder in den Context seines Masters. Sollte der ClientServerAnmeldung-Agent den Clientagenten anmelden, so benachrichtigt er diesen über das Resultat seines Auftrages. Danach ist die Lebenszeit dieses Slaves beendet. Im Fall der Abmeldung des Clientagenten wird der ClientServerAnmeldung-Agent sofort nach der Rückkehr in den Context seines Masters beseitigt.

FileAgent:

Initiiert der Benutzer die Überprüfung einer bestimmten Datei, so erzeugt der für die Benutzeroberfläche verantwortliche Clientstart-Agent einen FileAgenten und schickt ihn mit der zu überprüfenden Datei zum Serveragenten des Benutzers. Nachdem der FileAgent diese Datei beim Serveragenten abgegeben hat, kehrt er zu seinem Master zurück und informiert diesen über die erfolgte Überbringung der Datei. Danach wird dieser Slave beseitigt.

FilePruefAgent:

Nachdem der Clientagent im Rahmen der regelmäßigen Überprüfung die Dateien bestimmt hat, die von seinem Serveragenten untersucht werden sollen, erzeugt er den FilePruefAgenten und schickt diesen mit den zu überprüfenden Dateien zu seinem Serveragenten. Sobald dieser Slave die Dateien beim Serveragenten abgegeben hat, kehrt er zu seinem Master zurück und überbringt ihm das Resultat seines Auftrages. Danach geht die Lebenszeit des FilePruefAgenten zu Ende.

MailAgent:

Hat der Serveragent bei der Untersuchung einer Datei Unterschiede zu einer Dateiversion eines anderen Benutzer festgestellt und ist sein Clientagent vorhanden, so erzeugt er einen MailAgenten und schickt diesen mit samt der erkannten Unterschiede zu seinem Clientagenten. Nachdem dieser MailAgent die aus den Unterschieden entstandene Mail abgeliefert hat, kehrt er zu seinem Master zurück, um diesen über das Ergebnis seiner Aufgabe zu informieren. Anschließend wird dieser MailAgent wieder beseitigt.

2.3 Wie funktioniert die agentenbasierte Versionskontrolle ?

2.3.1 Erzeugung des Verwaltungsagenten und der serverseitigen Benutzeragenten

Dieses Programm benötigt für den korrekten Ablauf einen Server-Context, in dem ein Ver-

waltungsagent und pro Benutzer ein Serveragent existieren und der immer die Portnummer 9010 besitzt. Der Verwaltungsagent und die Serveragenten werden dabei beim Hochfahren des Servers erzeugt und sind solange vorhanden, wie der Server anwesend ist. Die Erzeugung dieser Agenten wird von einem anderen Agenten durchgeführt, nämlich durch den Serverstart-Agenten. Dieser Serverstart-Agent wird durch den Administrator im Server-Context erzeugt und erledigt dann die oben angegebenen Aufgaben, wobei er aus der Datei *Benutzerliste.txt* die Namen und Nummern der einzelnen Benutzer einliest. Ist allerdings der Verwaltungsagent bereits vorhanden, so kreiert dieser Agent keine Agenten. In beiden Fällen wird der Serverstart-Agent daraufhin beseitigt.

Es besteht auch die Möglichkeit, daß ein neuer Benutzer, der noch keinen Serveragenten besitzt, während des Programmablaufes unter Verwendung des ServeragentStart-Agenten in das Programm eingefügt wird. Dabei liest der ServeragentStart-Agent aus der Datei *NeuerBenutzer.txt* die Daten des neu hinzuzufügenden Benutzers ein und erzeugt den entsprechenden Serveragenten, bevor er beseitigt wird.

Jeder Serveragent, der auf einer dieser beiden oben erwähnten Möglichkeiten erzeugt worden ist, meldet sich zuerst beim Verwaltungsagenten an, indem er den ServerAnmeldung-Slave erzeugt und zum Verwaltungsagenten schickt. Nachdem der Slave seinen Master angemeldet

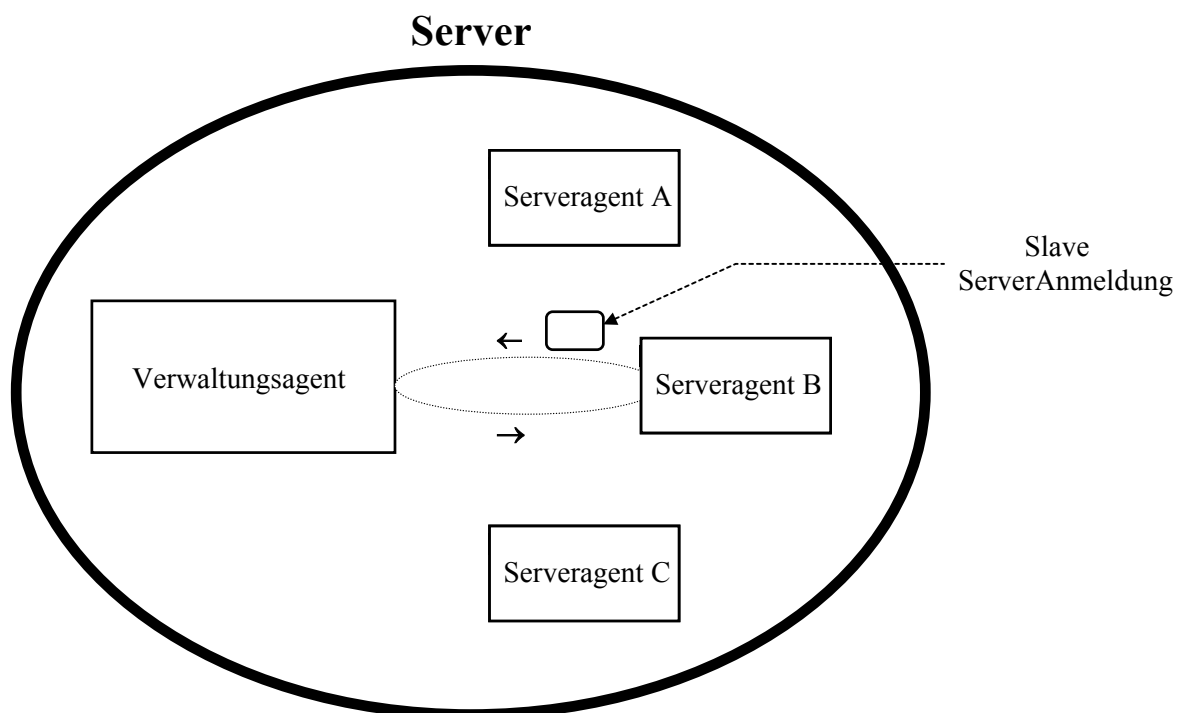


Abb. 2.2: Der Serveragent B meldet sich beim Verwaltungsagenten an

hat, kehrt er zu seinem Master zurück und informiert diesen über die erfolgte Anmeldung. Danach wird dieser Slave entfernt. Tritt allerdings der Fall ein, daß dieser Slave den Verwaltungsagenten nicht finden kann, so berichtet er seinem Serveragenten darüber, bevor er beseitigt wird. In diesem Fall beseitigt sich auch der Serveragent. Im Rahmen der Eliminierung des Serveragenten meldet sich dieser beim Verwaltungsagenten unter Verwendung des ServerAnmeldung-Slave ab. In diesem Fall kehrt der ServerAnmeldung-Slave vor seiner

Entfernung nicht zu seinem Master zurück. Wenn der Verwaltungsagent eliminiert wird, beseitigt dieser die bei ihm angemeldeten Serveragenten.

2.3.2 Bereitstellung der Benutzeroberfläche und Erzeugung und Beseitigung des clientseitigen Benutzeragenten

Möchte ein Benutzer mit diesem Programm arbeiten, so muß er zuerst die Benutzeroberfläche zur Verfügung gestellt bekommen. Deshalb wird, sobald der Aglet-Context dieses Benutzers vorhanden ist, in ihm der Clientstart-Agent erzeugt, der seinerseits die Benutzeroberfläche bereitstellt. Der clientseitige Benutzeragent, nämlich der Clientagent, wird nun, sofern er noch nicht existiert, durch Anklicken des Buttons *Create* im AVC - Main-Menu erzeugt, wo-bei der Clientstart-Agent den Namen und die Nummer des Benutzers aus der Datei *User.txt* einliest.

Dieser Clientagent meldet sich daraufhin unter Verwendung des von ihm generierten Client-ServerAnmeldung-Slaves bei seinem Serveragenten an. Dieser Slave gelangt dabei in den Server-Context und befragt den Verwaltungsagenten bezüglich der AgletID seines Serveragenten. Nachdem er diese erhalten hat, überbringt dieser Slave dem serverseitigen Benutzeragenten die Anmeldung des Clientagenten und kehrt danach zu seinem Erzeuger zurück, um diesem über die erfolgreiche Anmeldung zu informieren.

Konnte der ClientServerAnmeldung-Slave allerdings den Verwaltungsagenten oder den Serveragenten nicht finden, so setzt er seinen Master darüber in Kenntnis, der in diesem Fall seinerseits den Clientstart-Agenten und sich selbst beseitigt. Auf jeden Fall wird der Slave, nachdem er das Resultat seines Auftrages seinem Clientagenten mitgeteilt hat, entfernt. Ist der Clientagent erfolgreich bei seinem Serveragenten angemeldet worden, so startet dieser eine regelmäßige Überprüfung der von ihm überwachten Dateien. Außerdem erzeugt der Clientagent zu Beginn seiner Lebenszeit den ClientPause-Slave, der sich daraufhin für den aus der Datei *Userdaten.txt* eingelesene Zeit zwischen zwei regelmäßigen Überprüfungen deaktiviert.

Die Anmeldung des Clientagenten bewirkt beim zugehörigen Serveragenten, daß dieser überprüft, ob er noch Mails besitzt, die er seinem Clientagenten noch nicht erfolgreich zuschicken konnte. Falls dieses der Fall sein sollte, so erzeugt er pro zu überbringender Mail einen Mail-Agenten, übergibt diesem die Mail und schickt ihn zum entsprechenden Clientagenten.

Konnte im Rahmen der Anmeldung beim Serveragenten der Verwaltungsagent oder der zugehörige Serveragent nicht gefunden werden oder hat der Benutzer im AVC - Main-Menu den Button *Dispose* angeklickt, so beseitigt sich der Clientagent. Dabei entfernt er seinen Client-Pause-Slave und meldet sich durch Erzeugung und Aussendung eines ClientServerAnmeldung-Slaves bei seinem Serveragenten ab. Dieser Slave verhält sich hierbei genauso wie bei der Anmeldung mit dem einen Unterschied, daß er seinem Master nicht über das Ergebnis seines Auftrages berichtet.

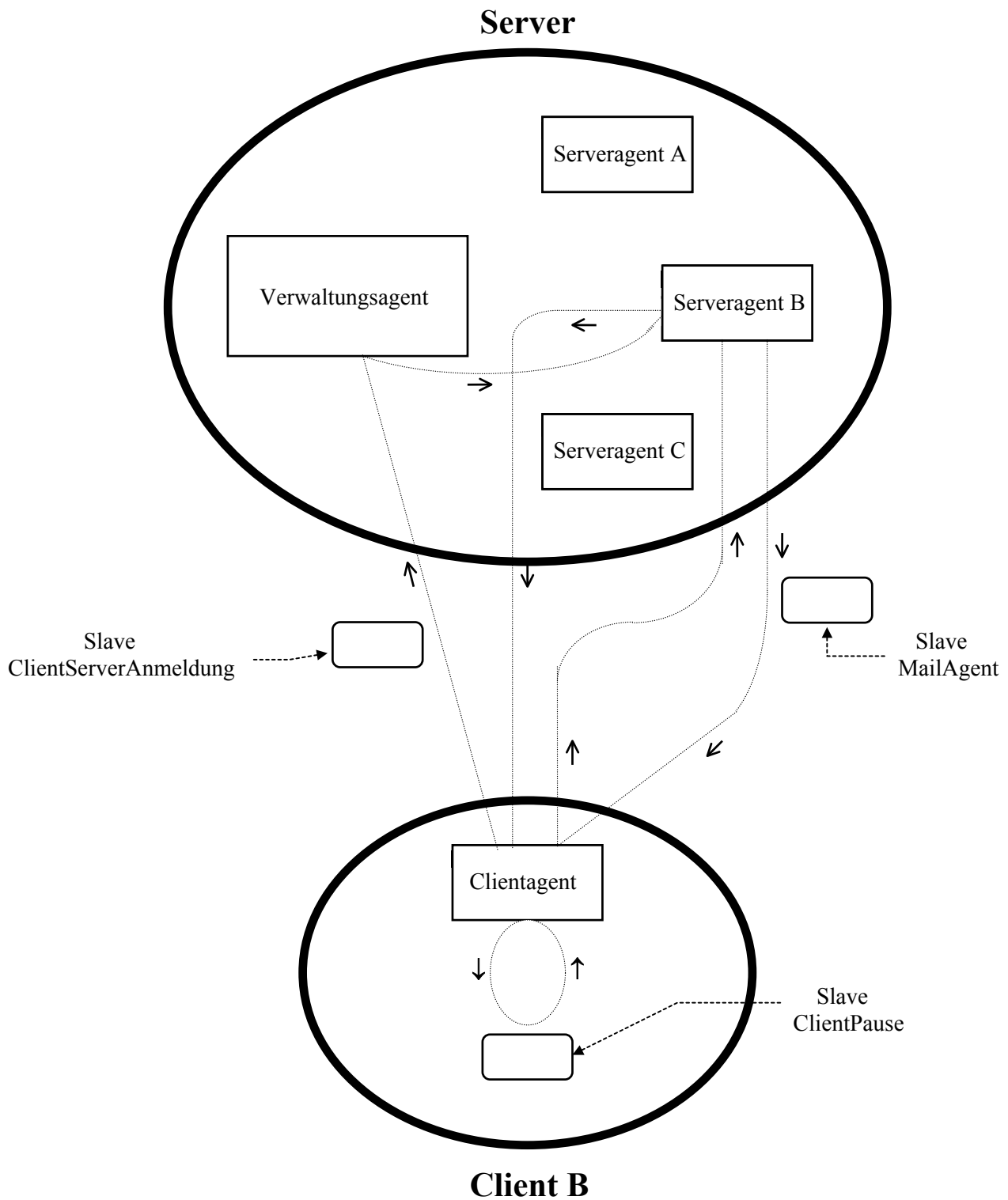


Abb. 2.3: Der Clientagent des Benutzers B wird erzeugt und meldet sich bei seinem Serveragenten an

2.3.3 Regelmäßige Überprüfung der überwachten Dateien

Aufgabe dieser regelmäßigen Überprüfung ist es, die im Auftrag des einzelnen Benutzers überwachten Dateien auf Unterschiede zu den überwachten Dateiversionen der anderen Benutzer zu untersuchen. Jeder Benutzer kann hierfür im AVC - Choice-Menü die Dateien auswählen, die überwacht werden sollen. Die Überprüfung wird vom Clientagenten initiiert, wenn dieser sich erfolgreich bei seinem Serveragenten angemeldet hat, oder wenn der ClientPause-Slave zum Clientagenten zurückgekehrt ist. Der zeitliche Abstand zwischen der Aussendung und der Rückkehr des ClientPause-Slaves entspricht dabei genau dem Zeitabstand zwischen zwei regelmäßigen Überprüfungen, der in der Benutzerdatei *Userdaten.txt* gespeichert ist. Allerdings kann es auch vorkommen, daß der ClientPause-Slave schon früher den Clientagenten mit der Initiierung der regelmäßigen Überprüfung beauftragt, nämlich dann, wenn der Benutzer den Zeitabstand zwischen den regelmäßigen Überprüfungen im AVC - Management verändert hat und danach das AVC - Management - Fenster verlassen hat.

Ist die regelmäßige Überprüfung durch die Rückkehr des ClientPause-Slaves in Auftrag gegeben worden, so wird dieser Slave beseitigt und ein neuer ClientPause-Slave erzeugt. Dieser Slave erhält dann den aktuellen Zeitabstand zwischen den Überprüfungen übergeben, der wieder aus der Datei *Userdaten.txt* eingelesen wird, um nach Ablauf dieser Zeitspanne zu seinem Master zurückzukehren.

Der Clientagent prüft zu Beginn dieser Überprüfung, ob seit der letzten Überprüfung die Liste der zu überwachenden Dateien verändert worden ist oder der Benutzer bezüglich der überwachten Dateien den CVS-Befehl checkout oder update aufgerufen hat. Sollte dies der Fall sein, so werden alle momentan überwachten Dateien in die zu überbringende Dateiliste eingefügt. Andernfalls werden nur die Dateien in die eben genannte Liste eingetragen, die sich seit der letzten regelmäßigen Überprüfung verändert haben. Ist aber bei einer dieser überwachten Dateien ein CVS-Konflikt aufgetreten, so wird diese Datei aus der Liste der beobachteten Dateien entfernt, wobei diese Liste der Inhalt der Datei *FileListe.txt* ist. Außerdem wird zur Benachrichtigung des Benutzers dieser durch das AVC - Conflict-Alarm - Fenster darüber informiert.

Hat der Clientagent die Liste der auf Unterschiede zu untersuchenden Dateien bestimmt und ist diese nicht leer, so wird der FilePruefAgent erzeugt und mit dieser Liste zum zugehörigen Serveragenten geschickt. Dabei gelangt er zuerst zum Verwaltungsagenten und erhält von diesem die AgletID des gesuchten Serveragenten. Danach kommt er zu seinem Serveragenten und übergibt diesem die Liste der zu untersuchenden Dateien. Daraufhin kehrt der FilePruefAgent zu seinem Master zurück und informiert diesen über die erfolgreiche Ablieferung der Dateien. Konnte dieser Slave jedoch den Verwaltungsagenten oder den entsprechenden Serveragenten nicht finden, so wird der Clientagent ebenfalls darüber informiert, der seinerseits den Clientstart-Agenten und sich selbst beseitigt. Berichtet der FilePruefAgent allerdings dem Clientagenten, daß der Serveragent neu erzeugt worden ist, so meldet sich der Clientagent erneut bei seinem Serveragenten an. Nachdem der FilePruefAgent dem Clientagenten das Resultat seines Auftrages mitgeteilt hat, wird er beseitigt.

Die Überprüfung auf mögliche Unterschiede wird nun von dem Serveragenten erledigt. Falls sich die Liste der überwachten Dateien beim Clientagenten verändert hat oder falls der Benutzer bezüglich der überwachten Dateien den CVS-Befehl checkout oder update aufgerufen hat, so wird zuerst der FilenamensAgent erzeugt und mit der Liste der überwachten Dateien zum Verwaltungsagenten geschickt. Nachdem er diese Liste dort abgegeben hat, kehrt dieser

FilenamenAgent zu seinem Erzeuger zurück und benachrichtigt diesen darüber. Danach wird dieser FilenamenAgent beseitigt. Der Verwaltungsagent seinerseits bringt die Dateilisten, in denen die Serveragenten angegeben sind, die die zugehörige Datei jeder dieser Listen überwachen, auf den neusten Stand.

Auf jeden Fall erzeugt der Serveragent pro Datei, die überprüft werden soll, einen DiffAgenten. Jeder dieser DiffAgenten wird dann zum Verwaltungsagenten gesendet, um dort die Liste der Serveragenten zu bekommen, die auch diese Datei überwachen. Nachdem der DiffAgent diese Liste erhalten hat, besucht er jeden auf der Liste stehenden Serveragenten, um die entsprechende Dateiversion zu erhalten. Dieser DiffAgent kehrt, sobald er alle diese Serveragenten aufgesucht und deren Dateiversionen bekommen hat, zu seinem Master zurück und übergibt ihm die Liste der Dateiversionen der anderen Benutzer, bevor er beseitigt wird.

Der Serveragent vergleicht nun die eigene Dateiversion mit jeder Dateiversion der anderen Benutzer, sofern dieser Vergleich zulässig ist. Dieses wird aufgrund der Versionsnummern der einzelnen Dateiversionen festgestellt, wobei die Dateien nur miteinander verglichen werden dürfen, wenn deren Versionsnummern unter Vernachlässigung der letzten Zahl identisch sind, oder wenn die eine Versionsnummer vollständig mit dem Anfang der anderen Versionsnummer übereinstimmt. Sollte unter Berücksichtigung dieser Einschränkung eine Untersuchung auf Unterschiede erlaubt sein, so wird diese durch Verwendung des Befehles `diff` durchgeführt.

Werden dabei Unterschiede bemerkt, so wird eine Mail für den eigenen Clientagenten dieses Serveragenten erzeugt, in der diese Unterschiede einschließlich der Daten des anderen Benutzers, dessen Dateiversion mit der eigenen verglichen worden ist, aufgelistet sind. Ist der Clientagent vorhanden, so wird ihm diese Mail durch den MailAgenten überbracht. Andernfalls wird die Mail im serverseitigen Verzeichnis des Benutzers gespeichert. Hat der Serveragent alle zu untersuchenden Dateien des Benutzers mit den entsprechenden zulässigen Versionen der anderen Benutzer verglichen, so ist die regelmäßige Überprüfung beendet.

2.3.4 Überprüfung einer einzelnen, vom Benutzer ausgewählten Datei

Möchte der Benutzer eine bestimmte überwachte Datei schon vor der nächsten regelmäßigen Überprüfung oder eine Datei, die nicht überwacht wird, einmalig auf Unterschiede zu den entsprechenden beobachteten Dateiversionen der anderen Benutzer untersuchen lassen, so kann er diese im AVC - File-Menu auswählen und durch Anklicken des Buttons *Test* überprüfen lassen. Eine Datei, bei der durch den Befehl `checkout` oder `update` ein CVS-Konflikt aufgetreten ist, kann allerdings nicht selektiert werden.

Das Drücken des *Test*-Buttons bewirkt, daß der Clientstart-Agent einen FileAgenten erzeugt und diesen mit der vorher markierten Datei zum Serveragenten schickt. Dieser FileAgent gelangt dabei zuerst zum Verwaltungsagenten und befragt diesen bezüglich der AgletID des gesuchten Serveragenten. Nachdem er diese erhalten hat und daraufhin beim benutzereigenen Serveragenten angekommen ist, übergibt er diesem die zu überprüfende Datei. Danach kehrt er zu seinem Master, dem Clientstart-Agenten, zurück und informiert diesen über die erfolgreiche Überbringung der zu untersuchenden Datei. Konnte dieser Slave jedoch den Verwaltungsagenten oder den Serveragenten nicht finden, so wird auch dieses dem Clientstart-Agenten nach der Rückkehr bekanntgemacht. Nachdem der FileAgent dem Clientstart-Agenten das Ergebnis seiner Aufgabe mitgeteilt hat, wird er beseitigt.

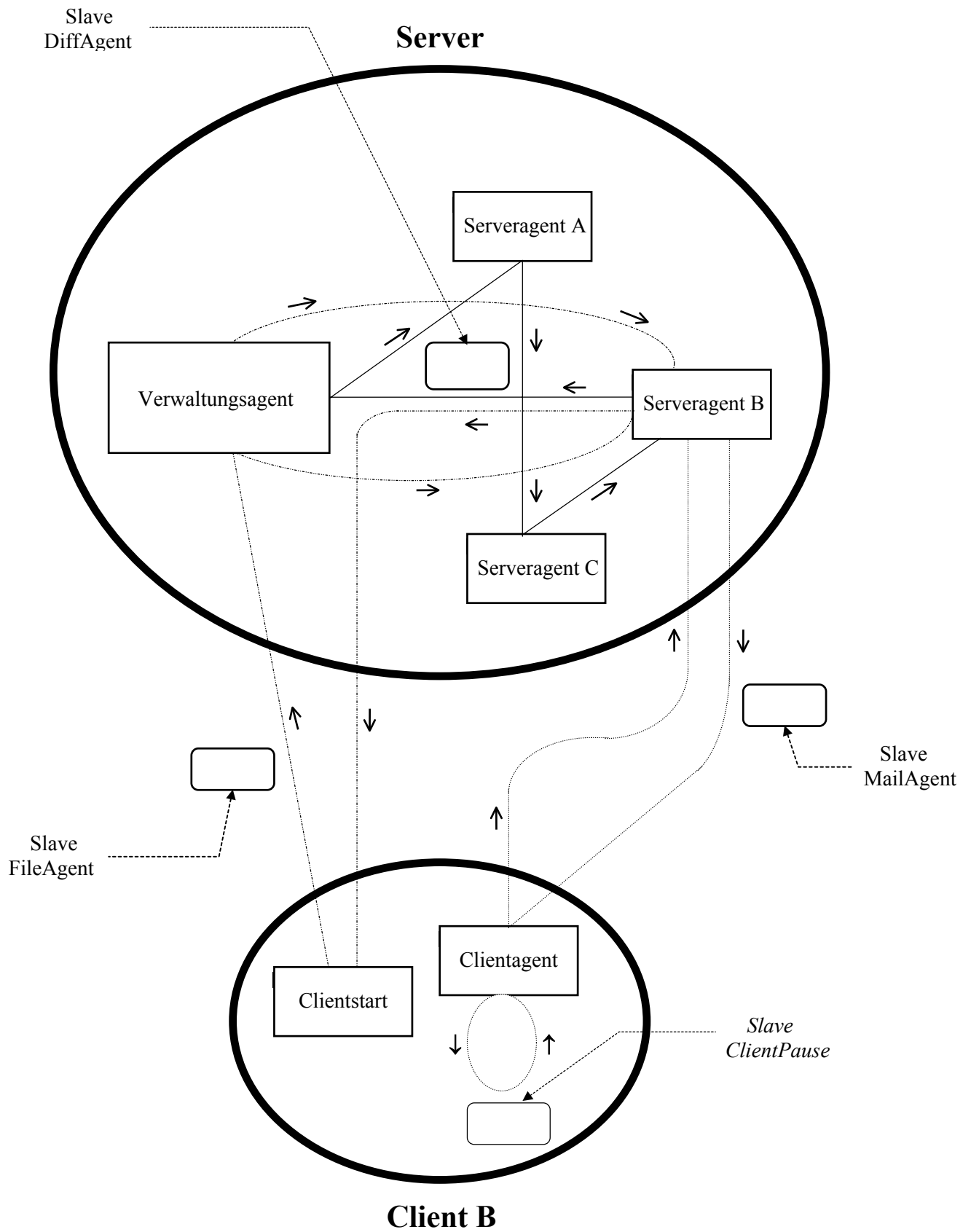


Abb. 2.5: Der Benutzer läßt eine einzelne, von ihm ausgewählte Datei überprüfen

Der Serveragent, der von seinem Benutzer den Auftrag erhalten hat, die ausgewählte Datei auf Unterschiede zu anderen Dateiversionen zu überprüfen, erzeugt einen DiffAgenten und schickt diesen zum Verwaltungsagenten. Vom Verwaltungsagenten erhält der DiffAgent daraufhin die AgletIDs der Serveragenten, die auch diese Datei überwachen. Der DiffAgent besucht nun die vorher genannten Serveragenten und bekommt von diesen die eigenen überwachten Dateiversionen übergeben. Danach kehrt dieser DiffAgent zu seinem Master zurück, überreicht ihm die Liste der Dateiversionen der anderen Benutzer und wird im Anschluß daran beseitigt.

Der Serveragent, der diese Liste der anderen Dateiversionen erhalten hat und für dessen Benutzer die Untersuchung durchgeführt werden soll, vergleicht jetzt die Dateiversion des eigenen Benutzers mit jeder überbrachten Dateiversion der anderen Benutzer auf Unterschiede zwischen ihnen, sofern diese Überprüfung unter Berücksichtigung der einzelnen Versionsnummern zulässig ist. Diese Untersuchung ist nämlich nur erlaubt, wenn für die Versionsnummern der beiden zu vergleichenden Dateiversionen gilt, daß sie unter Vernachlässigung jeweils der letzten Zahl identisch sind oder daß die eine Versionsnummer vollständig mit dem Anfangsstück der anderen Versionsnummer übereinstimmt. Sollte unter Berücksichtigung dieser Einschränkung eine Untersuchung auf Unterschiede erlaubt sein, so wird diese durch Verwendung des Befehles diff durchgeführt.

Werden dabei Unterschiede bemerkt, so wird eine Mail für den eigenen Clientagenten dieses Serveragenten erzeugt, in der diese Unterschiede einschließlich der Daten des anderen Benutzers, dessen Dateiversion mit der eigenen verglichen worden ist, aufgelistet sind. Ist der Clientagent vorhanden, so wird ihm diese Mail durch den MailAgenten überbracht. Andernfalls wird die Mail im serverseitigen Verzeichnis des Benutzers gespeichert. Hat der Serveragent die eigene Datei mit allen überbrachten Dateiversionen der anderen Benutzer verglichen, so ist die Überprüfung beendet.

2.3.5 Benachrichtigung des Benutzers über die Unterschiede zu anderen Benutzern

Hat der Serveragent bei der Überprüfung der Dateiversion des eigenen Benutzers auf Unterschiede zu einer Version derselben Datei eines anderen Benutzers festgestellt, daß zwischen diesen beiden Versionen Differenzen bestehen, so werden diese Differenzen in einer Mail gespeichert. Außerdem beinhaltet diese Mail noch den Namen und die Nummer des Benutzers, mit dessen Dateiversion die Unterschiede erkannt worden sind, und den Zeitpunkt dieser Überprüfung.

Die so entstandene Mail wird dann im serverseitigen Verzeichnis des Benutzers gespeichert. Ist der Clientagent dieses Benutzers vorhanden, so erzeugt der Serveragent einen MailAgenten und schickt diesen mit samt der Mail zum eben genannten Clientagenten. Nachdem dieser MailAgent die Mail abgeliefert hat, kehrt er zu seinem Erzeuger zurück und teilt ihm mit, daß er die Mail dem Clientagenten überbracht hat. Konnte der MailAgent allerdings den Clientagenten nicht erreichen, so informiert er seinen Serveragenten darüber. Sobald der MailAgent dem Serveragenten das Resultat seines Auftrages berichtet hat, ist seine Lebenszeit beendet.

Konnte die Mail erfolgreich übergeben werden, so löscht der Serveragent sie aus dem serverseitigen Verzeichnis des Benutzers. Andernfalls bleibt sie in dem Verzeichnis gespeichert. Meldet sich dann der Clientagent neu bei seinem Serveragenten an, so wird jede Mail, die

sich noch beim Serveragenten befindet, durch einen MailAgenten zu diesem Clientagenten geschickt.

Wenn ein Clientagent von einem MailAgenten eine Mail übergeben bekommt, speichert er diese in dem entsprechenden Benutzerverzeichnis und schickt den MailAgenten zum Serveragenten zurück. Dann schaut er in der Datei *Userdaten.txt* nach, ob der Benutzer über die Ankunft einer neuen Mail informiert werden möchte. Sollte dies der Fall sein, so wird der Benutzer durch das AVC - Mail-Alarm - Fenster darüber in Kenntnis gesetzt. Der Benutzer besitzt die Möglichkeit, sich jede Mail, die in seinem clientseitigen Verzeichnis gespeichert ist, im AVC - Mail - Fenster anzeigen zu lassen, nachdem er die gewünschte Mail im AVC - Mail-Menu ausgewählt hat.

3. Installation des Programmes

3.1 Installation auf dem Server

Zuerst müssen die einzelnen Klassen dieses Programmes auf dem Server gespeichert werden. Dazu wird im Verzeichnis *public*, das sich im *Aglets*-Verzeichnis der AWB befindet, das Verzeichnis *AVC* angelegt. In dieses zuletzt genannte Verzeichnis müssen daraufhin die einzelnen Klassen dieses Programmes gespeichert werden. Hieran anschließend müssen die Umgebungsvariablen *CLASSPATH*, *AGLET_PATH* und *AGLET_EXPORT_PATH* durch das oben erwähnte Verzeichnis *public* erweitert werden.

Im nächsten Schritt muß in einem beliebigen Verzeichnis das Programmverzeichnis *AVC* erzeugt werden, in dem wiederum das Verzeichnis *server* generiert werden muß. In diesem Verzeichnis *server* werden dann die einzelnen serverseitigen Benutzerverzeichnisse installiert. Jedes dieser Benutzerverzeichnisse trägt dabei den Namen, der aus dem Namen und der Nummer des jeweiligen Benutzers gebildet wird: *Name_Nummer*, wobei auch die Leerzeichen, die im Benutzernamen auftauchen, durch ‘_’ ersetzt werden. Innerhalb jedes dieser eben genannten Benutzerverzeichnisse müssen im Anschluß daran jeweils zwei Unterverzeichnisse angelegt werden, nämlich die Verzeichnisse *FileListe* und *savedMails*.

Weiterhin muß die Datei *Benutzerliste.txt* erzeugt werden. Dabei werden in ihr alle Anwender dieses Programmes eingetragen, indem in jeder Zeile dieser Datei sowohl der Name als auch die Nummer des einzelnen Benutzers aufgeführt werden, wobei zwischen dem Namen und der Nummer zur Unterscheidung dieser beiden Angaben ein Tabulator-Schritt eingefügt wird. Diese Datei *Benutzerliste.txt* wird im oben genannten Verzeichnis *server* gespeichert.

Ferner müssen die Klassen *ServerDir* und *ServerURL* angepaßt werden. Die zugehörigen Java-Dateien werden zu diesem Zweck umgeschrieben:

1. In der Java-Datei *ServerDir.java* muß die folgende Zeile den tatsächlich vorhandenen Gegebenheiten angeglichen werden:

```
serverDir = absoluter Pfad des AVC_Oberverzeichnisses ;
```

Das AVC-Oberverzeichnis ist hierbei das Verzeichnis, in dem sich das im zweiten Absatz genannte Programmverzeichnis *AVC* als Unterverzeichnis befindet.

2. In der Java-Datei *ServerURL.java* muß die folgende Zeile verändert werden:

```
serverUrl = new URL("atp://Internet-Adresse des Servers:9010");
```

Diese beiden Java-Dateien müssen hiernach kompiliert werden. Die daraus entstehenden Klassen werden dann im oben genannten Klassenverzeichnis *AVC* gespeichert.

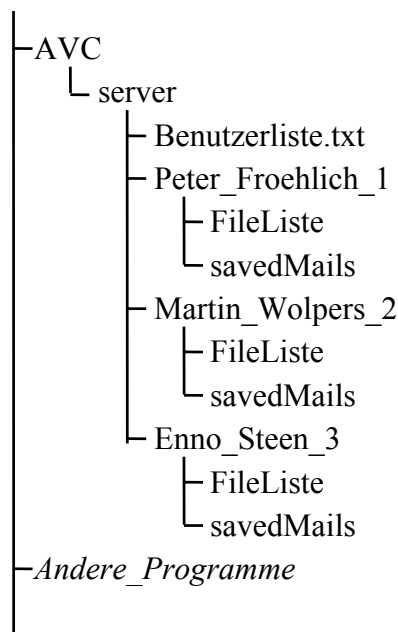
Beispielhaft seien nun die folgenden Benutzer verwendet:

Name: Peter Froehlich	Nummer: 1
Name: Martin Wolpers	Nummer: 2
Name: Enno Steen	Nummer: 3

Der Inhalt der Datei *Benutzerliste.txt* sieht dann folgendermaßen aus:

Peter Froehlich	1
Martin Wolpers	2
Enno Steen	3

Die Verzeichnisstruktur auf dem Server besitzt dann das folgende Aussehen:



Es besteht auch die Möglichkeit, daß ein neuer Benutzer, der noch keinen Eintrag in der Datei *Benutzerliste.txt* besitzt, während des Programmablaufes unter Verwendung des Serveragent-Start-Agenten in das Programm eingefügt wird. Dafür muß, wie oben bereits beschrieben, die Verzeichnisstruktur entsprechend erweitert werden. Danach wird im Verzeichnis *server* die Datei *NeuerBenutzer.txt* gespeichert, deren Inhalt aus einer Zeile besteht. Diese Zeile setzt sich aus dem Namen und der Nummer des neu hinzuzufügenden Benutzers zusammen, wobei zwischen diesen beiden Angaben ein Tabulator-Schritt eingefügt werden muß.

3.2 Installation auf dem Rechner des Benutzers

Zunächst müssen die einzelnen Klassen dieses Programmes auf dem Rechner zugänglich sein. Zu diesem Zweck wird mit diesen Klassen und den Umgebungsvariablen *CLASSPATH*, *AGLET_PATH* und *AGLET_EXPORT_PATH*, wie bereits im Kapitel 3.1 beschrieben, verfahren. Außerdem müssen die Klassen *ServerDir* und *ServerURL* angepaßt werden, was dadurch erreicht wird, daß sie die gleichen Angaben beinhalten wie die entsprechenden Klassen auf dem Server. Die Verfahrensweise bezüglich dieser beiden Klassen ist dabei zu der in Kapitel 3.1 erwähnten identisch.

Im nächsten Schritt muß in einem beliebigen Verzeichnis das Programmverzeichnis *AVC* erzeugt werden. In diesem Verzeichnis *AVC* wird dann das Benutzerverzeichnis angelegt, wobei der Name dieses Benutzerverzeichnisses aus dem Namen und der Nummer des Anwenders besteht. Dabei werden die Leerstellen innerhalb des Namens jeweils durch das Zeichen ‘_’ ersetzt. Dieses Zeichen befindet sich ebenfalls an der Verbindungsstelle zwischen dem Namen und der Nummer. In diesem Benutzerverzeichnis müssen daraufhin die beiden Unterverzeichnisse *FileListe* und *Mails* generiert werden.

Danach müssen dann noch drei Dateien in die vorher erzeugte Verzeichnisstruktur eingefügt werden. Das ist als erstes die leere Datei *FileListe.txt*, die im Benutzerverzeichnis gespeichert wird. Die Datei *User.txt* wird im Programmverzeichnis *AVC* abgelegt. Sie besitzt dabei den folgenden Aufbau:

```
## Benutzerdaten
Name : Name des Benutzers
Nummer : Nummer des Benutzers
```

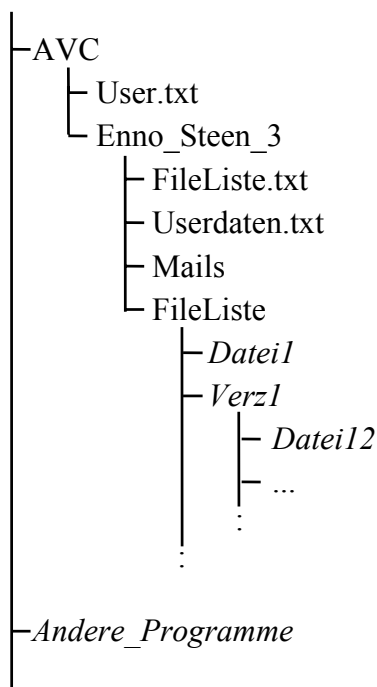
Als letztes wird die Datei *Userdaten.txt* im Benutzerverzeichnis gespeichert. Diese Datei verfügt zum Zeitpunkt dieser Installation über das folgende Aussehen:

```
## Benutzerdaten
Name : Name des Benutzers
Nummer : Nummer des Benutzers
Ueberpruefung : 5
Alarm : off
Letzte Pruefung : 0
```

Die Dateien einschließlich der zugehörigen Verzeichnisse, die durch die agentenbasierte Versionskontrolle verwaltet werden sollen, werden nun aus dem CVS ausgecheckt und im vorher erzeugten Verzeichnis *FileListe* gespeichert.

Beispielhaft sei nun die Verzeichnisstruktur des folgenden Benutzers skizziert:

Name: Enno Steen Nummer: 3



4. Benutzeroberfläche

4.1 Erzeugung und Beseitigung des Clientagenten

Die Erzeugung bzw. Beseitigung des Clientagenten kann durch den Benutzer im *AVC - Main-Menu* - Fenster durchgeführt werden. Dieses Menü wird auf dem Bildschirm angezeigt, wenn der Clientstart-Agent erzeugt oder der Button *Back* im *AVC - Management* - Fenster oder im *AVC - Mail-Menu* - Fenster angeklickt worden ist. Ist dabei der Clientagent noch nicht vorhanden, so ist der Button *Create* aktiviert und der Button *Dispose* deaktiviert. Der Benutzer ist nun in der Lage, durch Anklicken des *Create*-Buttons den Clientagenten zu generieren, wodurch die eben genannten Buttons ihren Status bezüglich der Aktivierungsmöglichkeit wechseln. Soll ein existierender Clientagent beseitigt werden, so kann das Vorhandensein dieses Agenten durch den aktivierten *Dispose*-Button bzw. durch den deaktivierten *Create*-Button erkannt werden. Die Beseitigung des Clientagenten wird in diesem Fall durch Drücken des Buttons *Dispose* initiiert, was außerdem zur Folge hat, daß die beiden vorher erwähnten Buttons ihren Aktivierungszustand austauschen.

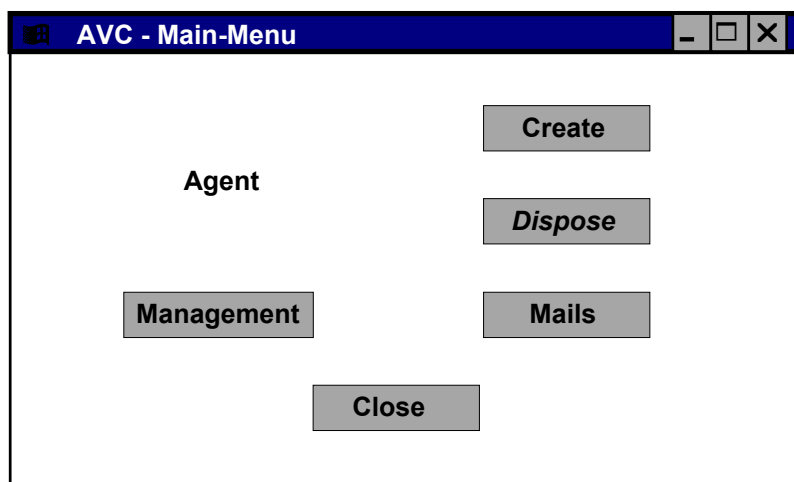


Abb. 4.1: Beispiel für ein *AVC - Main-Menu* - Fenster
(Der Clientagent existiert nicht)

4.2 Benutzerangaben festlegen

Der Benutzer besitzt bei diesem Programm die Möglichkeit anzugeben, in welchen Zeitabständen die regelmäßige Überprüfung der überwachten Dateien durchgeführt werden soll, und ob er über die Ankunft einer neuen Mail informiert werden möchte. Diese Angaben werden dabei im *AVC - Management* - Fenster verwaltet. Zu diesem Fenster gelangt der Benutzer, wenn er im *AVC - Main-Menu* - Fenster den Button *Management* oder den *Back*-Button im *AVC - Choice-Menu* - Fenster oder im *AVC - File-Menu* - Fenster anklickt. In der Zeile *Test : ...* erkennt der Benutzer den derzeitigen Zeitabstand zwischen zwei regelmäßigen Überprüfungen in Minuten, und er kann nun diesen Zeitabstand durch Eingabe eines neuen Wertes verändern. Dabei sind nur Zeitabstände zwischen 5 und 100 Minuten zulässig. In der folgenden Zeile, nämlich der *Alarm : ...* - Zeile, wird dem Benutzer angezeigt, ob er über die Ankunft einer neuen Mail informiert wird. Dies wird dadurch erreicht, daß die entsprechende Angabe markiert ist. Möchte der Benutzer den momentanen Zustand bezüglich der Alarmierung im Rahmen der Mailankunft verändern, so geschieht dieses durch Markierung des ge-

wünschten Kontrollkästchens. Beim Verlassen des *AVC - Management* - Fensters werden die zu diesem Zeitpunkt gültigen Werte hinsichtlich dieser beiden Benutzerangaben dem Programm bekanntgemacht. Ist hierbei bezüglich der regelmäßigen Überprüfung ein unzulässiger Zeitabstand eingegeben worden, so wird dieser durch einen Zeitabstand von 5 Minuten ersetzt.

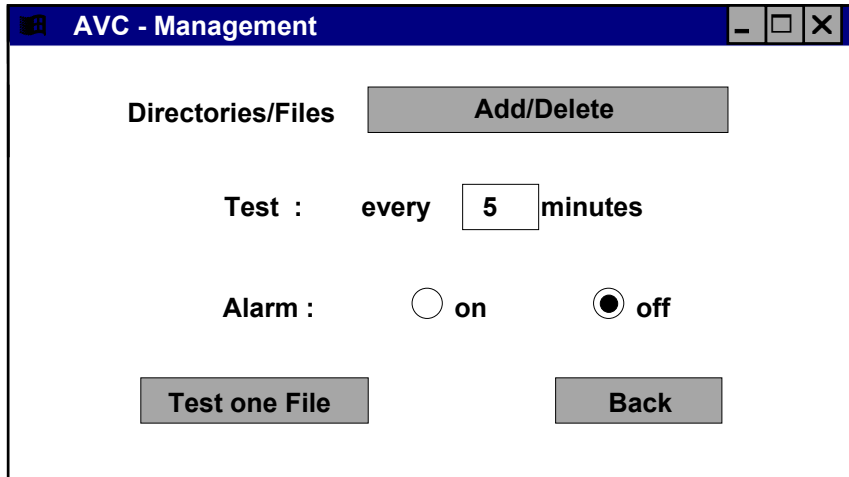


Abb. 4.2: Beispiel für ein *AVC - Management* - Fenster

4.3 Auswählen der regelmäßig überprüften Dateien

Der Benutzer kann im *AVC - Choice-Menu* - Fenster festlegen, welche Dateien im Rahmen der regelmäßigen Überprüfung untersucht werden sollen. Dieses *AVC - Choice-Menu* - Fenster wird auf dem Bildschirm angezeigt, wenn der Clientagent existiert und der Benutzer im *AVC - Management* - Fenster den Button *Add/Delete* anklickt. In diesem Fenster befinden sich alle Dateien und Verzeichnisse, die im Benutzerverzeichnis *FileListe* und in den entsprechenden Unterverzeichnissen gespeichert sind, wobei jedem Verzeichnisnamen das Zeichen '+' und jedem Dateinamen das Zeichen '-' vorangestellt ist. Alle zu einem Verzeichnis gehörenden Unterverzeichnisse und Dateien sind hierbei eingerückt zu diesem aufgeführt. Ist bei einer in diesem Fenster vorhandenen Datei ein CVS-Konflikt aufgetreten, so ist der Name

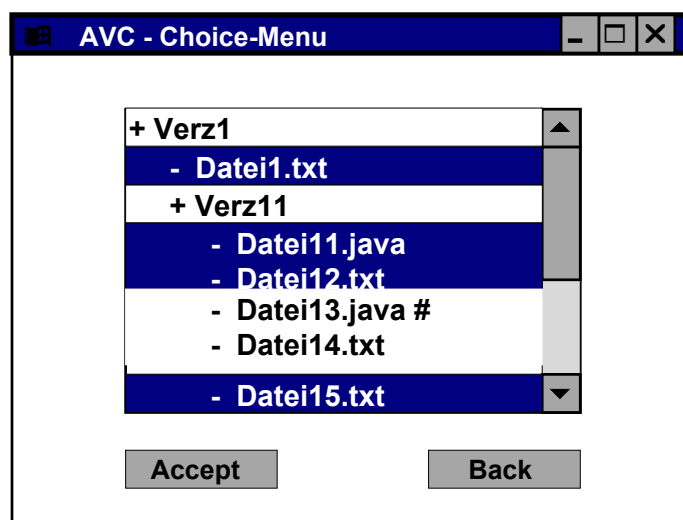


Abb. 4.3: Beispiel für ein *AVC - Choice-Menu* - Fenster

dieser Datei durch das Zeichen ‘#’ ergänzt. Beim Aufbau dieses Fensters werden die Dateien markiert angezeigt, die derzeit durch die agentenbasierte Versionskontrolle überwacht werden. Der Benutzer ist nun in der Lage, die Liste der überwachten Dateien zu verändern, indem er entweder die Markierung aufhebt, falls die entsprechende Datei nicht mehr überprüft werden soll, oder eine Datei neu markiert, die ab dem Zeitpunkt überwacht werden soll. Die Markierung eines Verzeichnisses oder einer Datei, bei der ein CVS-Konflikt vorgekommen ist, ist dabei nicht erlaubt und wird deshalb wieder aufgehoben. Hat der Benutzer alle Veränderungen an der Liste der zu überwachenden Dateien vorgenommen, so erlangt die aktualisierte Liste durch Drücken des Buttons *Accept* Gültigkeit.

4.4 Sofortige Überprüfung einer ausgewählten Datei

Die agentenbasierte Versionskontrolle AVC bietet dem Benutzer die Möglichkeit, eine einzelne vorher ausgewählte Datei augenblicklich überprüfen zu lassen. Diese Untersuchung ist dann zweckmäßig, wenn der Benutzer eine Datei, die nicht im Rahmen der regelmäßigen Überwachung berücksichtigt wird, einmalig auf Unterschiede zu anderen Versionen überprüfen lassen möchte, oder wenn er eine überwachte Datei schon vor der nächsten Untersuchung bezüglich der Unterschiede zu anderen Versionen inspizieren lassen möchte. Die auf diese Art zu überprüfende Datei wird im *AVC - File-Menu* - Fenster bestimmt. Zu diesem Menü gelangt der Benutzer, sofern der Clientagent vorhanden ist, wenn er im *AVC - Management* - Fenster den Button *Test one File* anklickt.

Das *AVC - File-Menu* - Fenster enthält alle Dateien und Verzeichnisse, die im Benutzerverzeichnis *FileListe* und in den entsprechenden Unterverzeichnissen gespeichert sind, wobei jedem Verzeichnisnamen das Zeichen ‘+’ und jedem Dateinamen das Zeichen ‘-’ vorangestellt ist. Alle zu einem Verzeichnis gehörenden Unterverzeichnisse und Dateien sind hierbei eingerückt zu diesem aufgeführt. Ist bei einer in diesem Fenster vorhandenen Datei ein CVS-Konflikt aufgetreten, so ist der Name dieser Datei durch das Zeichen ‘#’ erweitert.

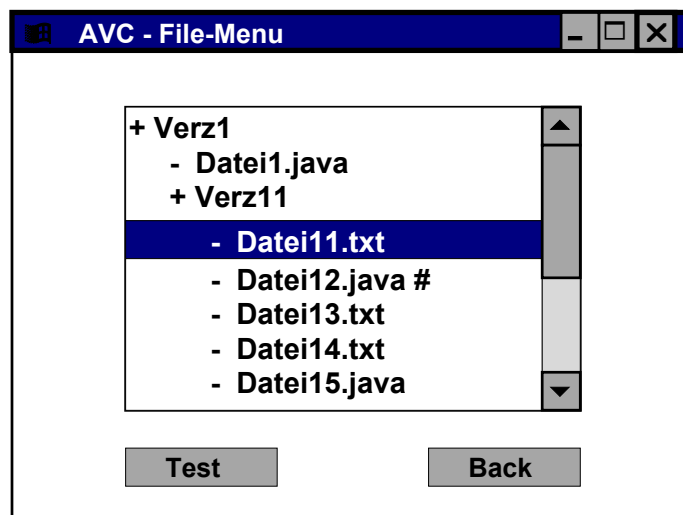


Abb. 4.4: Beispiel für ein *AVC - File-Menu* - Fenster

Hat der Benutzer eine Datei markiert, so ist der Button *Test* aktiviert. Durch das Drücken dieses Buttons initiiert der Benutzer die Überprüfung der vorher markierten Datei. Die Markierung eines Verzeichnisses oder einer Datei, bei der ein CVS-Konflikt vorgekommen ist,

ist dabei nicht erlaubt und wird deshalb wieder aufgehoben. Der Button *Test* ist dabei deaktiviert, solange keine Datei selektiert ist.

4.5 Benachrichtigung des Benutzers im Falle eines CVS-Konfliktes

Ist bei einer Datei, die durch die regelmäßige Überprüfung überwacht wird, ein durch den CVS-Befehl checkout oder update hervorgerufener CVS-Konflikt erkannt worden, so wird diese Datei aus der Liste der überwachten Dateien entfernt. Außerdem wird der Benutzer darüber durch das *AVC - Conflict-Alarm* - Fenster informiert. In diesem Fenster ist hierfür die entsprechende Konfliktdatei einschließlich ihres zum Verzeichnis *FileListe* relativen Pfades aufgeführt. Durch das Anklicken des sich ebenfalls in diesem Fenster befindlichen Buttons *Okay* bescheinigt der Benutzer, daß er den CVS-Konflikt bezüglich der erwähnten Datei zur Kenntnis genommen hat.



Abb. 4.5: Beispiel für ein *AVC - Conflict-Alarm* - Fenster

4.6 Ankunft einer neuen Mail

Sind bei der regelmäßigen bzw. bei der einmaligen Überprüfung einer Datei Unterschiede zu einer Dateiversion eines anderen Benutzers festgestellt worden, so werden diese in einer Mail gespeichert und dem anwesenden Benutzer überbracht. Sind während der Abwesenheit des Benutzers neue Mails entstanden, so erhält dieser jene im Verlauf der nächsten Clientagent-Erzeugung zugesendet. Hat der Anwender dieses Programmes im *AVC - Management* - Fenster angegeben, daß er über die Ankunft einer derartigen Mail informiert werden möchte, so wird im Fall des Eintreffens einer Mail das *AVC - Mail-Alarm* - Fenster auf dem Bildschirm angezeigt. Durch das Anklicken des im Fenster vorhandenen Buttons *Okay* dokumentiert der Benutzer, daß er die Ankunft dieser neuen Mail registriert hat.



Abb. 4.6: Das *AVC - Mail-Alarm* - Fenster

4.7 Anzeigen und Löschen der gespeicherten Mails

Alle Mails, die beim Benutzer angekommen sind, werden im benutzereigenen *Mails*-Verzeichnis gespeichert. Möchte sich der Benutzer über die Liste der gespeicherten Mails informieren, so ist dieses im *AVC - Mail-Menu* - Fenster möglich. Zu diesem Fenster gelangt der Anwender, indem er im *AVC - Main-Menu* - Fenster den Buttons *Mails* anklickt. Innerhalb dieses Fensters sind die Mails aufgelistet, die sich im oben genannten Verzeichnis befinden, wobei nur die pro Datei und Überprüfungspartner aktuellste Mail gespeichert ist. Der Name einer Mail setzt sich dabei aus dem zum *FileListe*-Verzeichnis relativen Pfad der Datei, bei der die Unterschiede festgestellt worden sind, und aus dem Namen und der Nummer des Überprüfungspartners, mit dessen Dateiversion diese Unterschiede erkannt worden sind, zusammen.

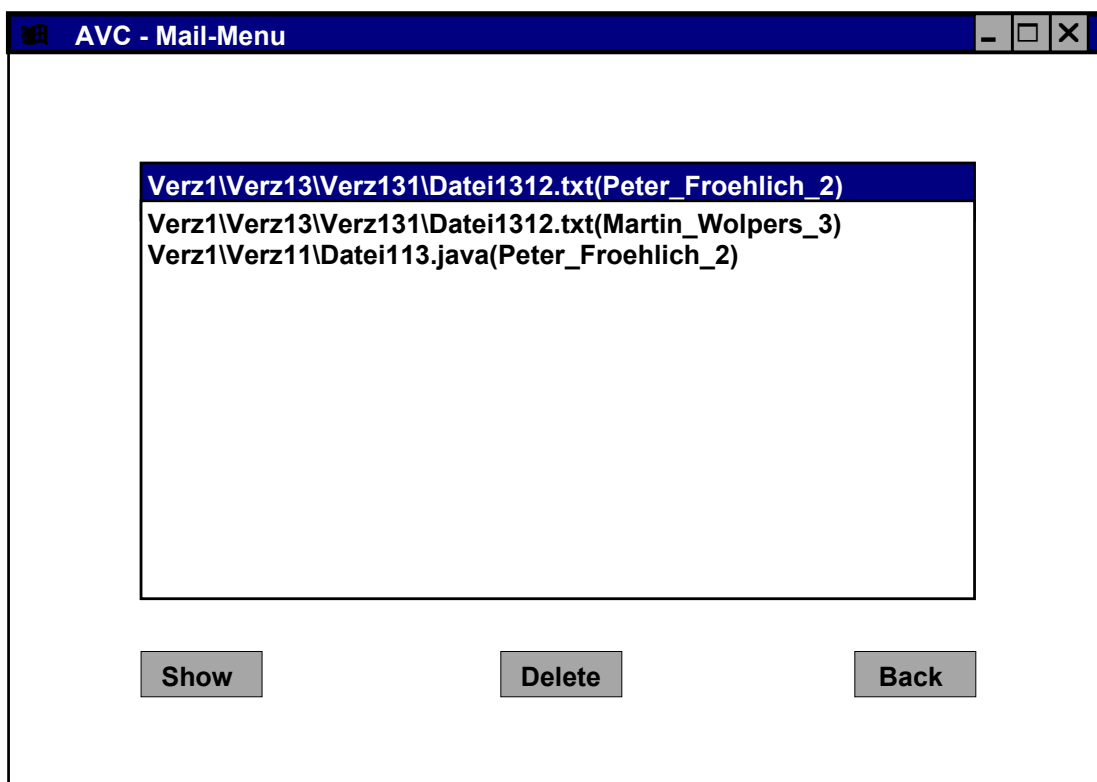


Abb. 4.7: Beispiel für ein *AVC - Mail-Menu* - Fenster

Der Anwender hat nun die Möglichkeit, eine Mail im *AVC - Mail-Menu* - Fenster zu markieren. Danach kann durch Drücken des Buttons *Delete* diese markierte Mail gelöscht werden, wodurch sie sowohl aus der im Fenster vorhandenen Liste als auch aus dem Verzeichnis *Mails* entfernt wird. Wird jedoch der Button *Show* angeklickt, so wird dieses Fenster beseitigt und das *AVC - Mail* - Fenster auf dem Bildschirm angezeigt. In diesem *AVC - Mail* - Fenster sind nun die einzelnen Angaben, die in der vorher markierten Mail enthalten sind, aufgeführt. Diese Angaben beziehen sich dabei auf den relativen Pfad der Datei, bei der die Unterschiede entdeckt worden sind, auf den Zeitpunkt der Untersuchung, auf den Namen und die Nummer des Überprüfungspartners, mit dessen Dateiversion die Unterschiede aufgetreten sind, und natürlich auf die Unterschiede an sich. Durch Anklicken des im Fenster existenten Buttons *Back* gelangt der Benutzer wieder zum *AVC - Mail-Menu* - Fenster. Beinhaltet die Liste im *AVC - Mail-Menu* - Fenster keine Mails, so sind die Buttons *Show* und *Delete* deaktiviert.

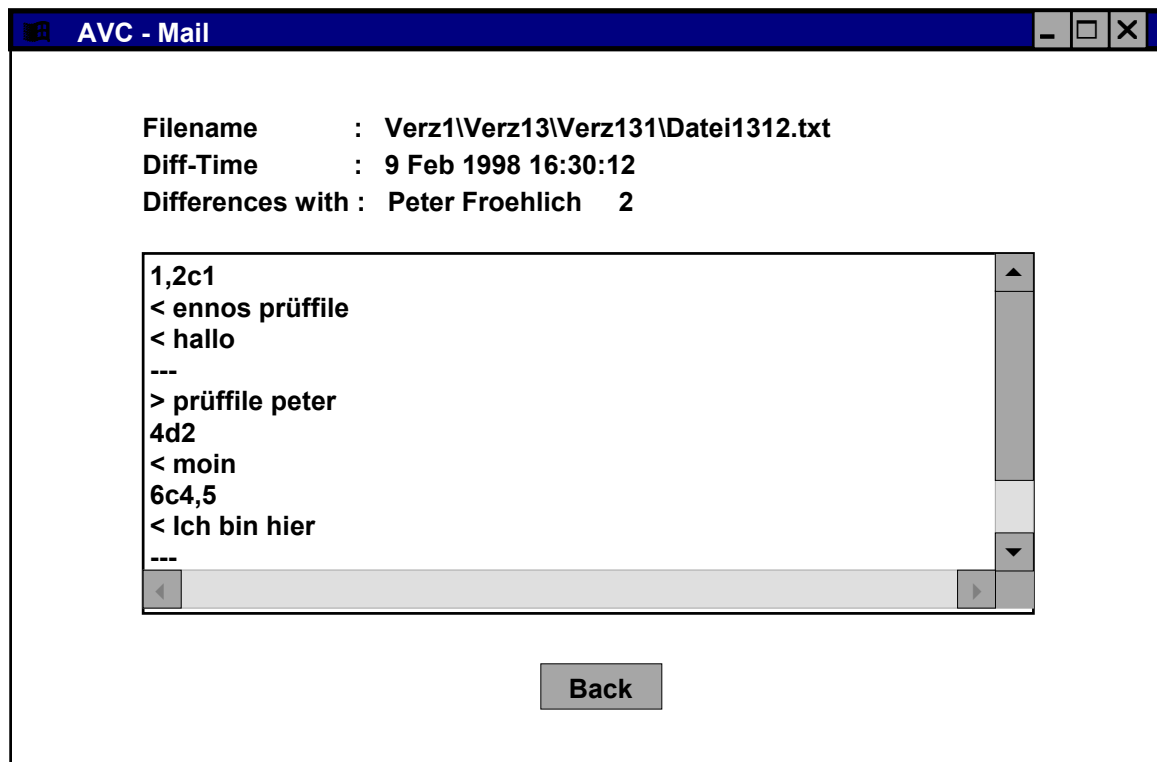


Abb. 4.8: Beispiel für ein *AVC - Mail* - Fenster

5. Beschreibung der Klassen und Textdateien

5.1 Die Benutzeroberfläche

5.1.1 AuswahlmenuAnzeigen

Diese Klasse ist ein Teil der Benutzeroberfläche. Ihre Superklasse ist die Klasse *Frame*. Wird ein Objekt dieser Klasse erzeugt, so wird das *AVC - Choice-Menu* - Fenster auf dem Bildschirm angezeigt. Dieses Fenster besitzt den Fenstertitel *AVC - Choice-Menu*. In diesem Fenster befinden sich die Liste der Dateien, die im Verzeichnis *FileListe* des Benutzers einschließlich ihrer relativen Pfade gespeichert sind, und die beiden Buttons *Accept* und *Back*. Die für die einzelnen Verzeichnisse zuständigen *CVS* - Verzeichnisse werden nicht in der Liste aufgeführt. Ist bei einer Datei ein Konflikt durch die *CVS* - Befehle *checkout* oder *update* aufgetreten, so wird die mit *.'* beginnende Datei ebenfalls nicht in der Liste angezeigt. Jede Zeile dieser Liste besteht entweder aus einem Verzeichnis oder aus einer Datei. Dabei beginnen die Zeilen der Verzeichnisse mit *'+'* und die Zeilen der Dateien mit *'-'*. Ist bei einer Datei ein *CVS* - Konflikt aufgetreten, so wird die Konfliktdatei, die zu der vorher genannten, mit *.'* beginnenden Datei gehört, folgendermaßen eingetragen: *'- Konfliktdatei #'*. Alle Verzeichnisse und Dateien, die in einem in der Liste vorkommenden Verzeichnis stehen, werden eingerückt zu diesem in die Liste eingetragen.

Die Dateien, die dieses Programm überwacht, werden markiert angezeigt, wobei die Liste der überwachten Dateien aus der Datei *FileListe.txt* des Benutzers eingelesen wird. Um die Liste der überwachten Dateien zu verändern, kann der Benutzer die neu zu überwachende Datei markieren bzw. bei der nicht mehr zu überwachenden Datei die Markierung aufheben. Versucht der Benutzer, eine Konfliktdatei oder ein Verzeichnis zu markieren, so wird die Markierung sofort wieder aufgehoben. Sollen die gemachten Veränderungen Gültigkeit erlangen, so muß der *Accept* - Button angeklickt werden, um die neue Liste der überwachten Dateien in der Datei *FileListe.txt* des Benutzers zu speichern und um in der Datei *Userdaten.txt* die letzte Überprüfungszeit auf *'0'* zu setzen, damit bei der nächsten Überprüfung alle zu überwachenden Dateien zum *Serveragenten* geschickt werden und dieser *Serveragent* und der *Verwaltungsagent* bezüglich der überwachten Dateien auf den neusten Stand gebracht werden.

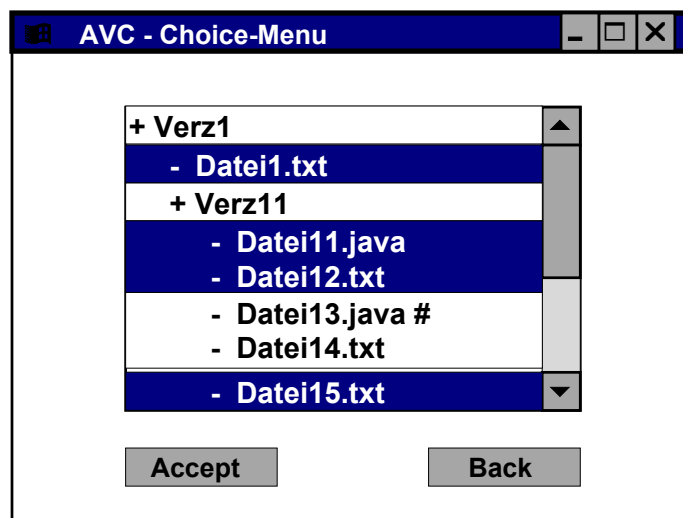


Abb. 5.1: Beispiel für ein *AVC - Choice-Menu* - Fenster

Dieses Fenster wird auf dem Bildschirm angezeigt, wenn der Benutzer im *AVC - Management* - Fenster den Button *Add/Delete* anklickt. Durch Drücken des *Back* - Buttons in diesem *AVC - Choice-Menu* - Fenster wird dieses Fenster wieder beseitigt und das *AVC - Management* - Fenster an seiner Stelle dargestellt.

Diese Klasse beinhaltet die folgenden Methoden:

1. *AuswahlmenuAnzeigen(booleen vorhanden , Aglet bspPart3 , String username , String usernumber)*

Diese spezielle Methode ist der Konstruktor dieser Klasse. In ihr wird das Fenster aufgebaut. Alle Verzeichnisse und Dateien im Verzeichnis *FileListe* werden in eine Liste, die in das Fenster eingefügt worden ist, eingetragen, sofern sie nicht das zuständige *CVS* - Verzeichnis oder eine mit *'#'* beginnende Konfliktdatei sind. Danach werden die überwachten Dateien markiert. Außerdem werden die Buttons *Accept* und *Back* in das Fenster eingefügt. Übergeben bekommt diese Methode, ob der *Clientagent vorhanden* ist, *bspPart3* als *Clientstart* - Agent und den Namen(*username*) und die Nummer(*usernumber*) des Benutzers.

2. *void makelabel(String name , GridBagLayout gridbag , GridBagConstraints f)*

Diese Methode wird aufgerufen, um in das Fenster den übergebenen String *name* einzufügen.

Übergeben bekommt sie den einzufügenden String *name* und die dafür notwendigen Argumente *gridbag* und *f*.

3. *boolean handleEvent(Event e)*

Wird ein Eintrag in der Liste markiert oder wird die Markierung aufgehoben, so wird diese Methode aufgerufen. Wurde dabei ein Verzeichnis oder eine Konfliktdatei markiert, so wird diese Markierung beseitigt. Diese Methode wird ebenfalls aufgerufen, wenn der *Accept* - oder der *Back* - Button angeklickt wurde. Im ersten Fall werden die Methoden *speichere_markierte_Listeneinträge()* und *löschePrüfungszeit()* aufgerufen und im zweiten Fall wird dieses Fenster beseitigt und das *AVC - Management* - Fenster auf dem Bildschirm angezeigt.

Übergeben bekommt sie, was sich ereignet hat(*Event e*). Zurückgegeben wird, ob das aufgetretene Ereignis *e* behandelt worden ist oder nicht.

4. *int subdirectories(String abstandsanzeige , String dirname, int m , Vector liste_markiert)*

Diese Methode wird aufgerufen, wenn der in die Liste im Fenster eingefügte Eintrag ein Verzeichnis ist. Alle Verzeichnisse und Dateien im Verzeichnis *dirname* werden in die Liste eingerückt(*abstandsanzeige*) ab der Stelle *m* eingetragen, sofern sie nicht das zuständige *CVS* - Verzeichnis oder eine mit *'#'* beginnende Konfliktdatei sind. Wurde ein Verzeichnis eingetragen, so ruft sich diese Methode selbst auf. Danach werden die überwachten Dateien markiert.

Übergeben werden folgende Angaben:

1. Der String *abstandsanzeige* gibt an, wie weit die Elemente dieses eingetragenen Verzeichnisses eingerückt auf dem Bildschirm dargestellt werden sollen.
2. Der String *dirname* beinhaltet den Namen des eingetragenen Verzeichnisses.
3. *m* gibt an, an welcher Stelle in der Liste der nächste Eintrag einzufügen ist.
4. Die Liste *liste_markiert* stellt eine Aufzählung aller regelmäßig überprüften Dateien zur Verfügung.

Zurückgegeben wird die Stelle in der Liste, an der zuletzt ein Eintrag erfolgt ist.

5. *Vector leseFileListe()*

Diese Methode gibt eine Aufzählung zurück, in der alle Dateien einschließlich ihrer relativen Pfade aufgelistet sind, die regelmäßig überprüft werden. Diese Liste wird aus der Textdatei *FileListe.txt* eingelesen.

6. *boolean ist_markiert(String name , Vector liste_markiert)*

Die Methode überprüft, ob eine bestimmte Datei in der Liste der regelmäßig überprüften Dateien enthalten ist. Der Methode übergeben werden der absolute Pfad der zu untersuchenden Datei (*name*) sowie die Liste aller Dateien einschließlich ihrer relativen Pfade (*liste_markiert*), die regelmäßig überprüft werden. Rückgabewert ist 'true', falls die Datei in der Liste enthalten ist, andernfalls 'false'.

7. *void speichere_markierte_Listeneinträge()*

Diese Methode wird aufgerufen, wenn im Fenster der *Accept* - Button angeklickt wird. Ihre Aufgabe besteht darin, alle markierten Dateien in der Liste im Fenster festzustellen und deren relativen Pfade und Dateinamen in der Datei *FileListe.txt* zu speichern.

8. *int subdirectories(String abstandsanzeige , String dirname , int m)*

Diese Methode wird aufgerufen, wenn beim Untersuchen der markierten Listeneinträge unter 7. oder bei dieser Methode ein Verzeichnis überprüft wird.

Übergeben werden folgende Angaben:

1. Der String *abstandsanzeige* gibt an, wie weit die Elemente dieses Verzeichnisses eingerückt auf dem Bildschirm dargestellt werden.
2. Der String *dirname* beinhaltet den Namen des untersuchten Verzeichnisses.
3. *m* gibt an, an welcher Stelle in der Liste sich der nächste Eintrag befindet.

Zurückgegeben wird die Stelle in der Liste, an der zuletzt ein Eintrag untersucht worden ist.

9. *void löschePrüfungszeit()*

Diese Methode wird aufgerufen, wenn der *Accept* - Button angeklickt worden ist. Sie besitzt die Aufgabe, den in der Datei *Userdaten.txt* gespeicherten Zeitpunkt der letzten Überprüfung der Dateien durch '0' zu ersetzen, damit bei der nächsten Inspektion alle zu überprüfenden Dateien berücksichtigt werden. Dies ist notwendig, damit der *Serveragent* und der *Verwaltungsagent* über die Veränderungen in der Liste der zu überprüfenden Dateien informiert und auf den neusten Stand gebracht werden.

5.1.2 FilemenuAnzeigen

Diese Klasse ist ein Teil der Benutzeroberfläche. Ihre Superklasse ist die Klasse *Frame*. Wird ein Objekt dieser Klasse erzeugt, so wird das *AVC - File-Menu* - Fenster auf dem Bildschirm angezeigt. Dieses Fenster besitzt den Fenstertitel *AVC - File-Menu*. In diesem Fenster befinden sich die Liste der Dateien, die im Verzeichnis *FileListe* des Benutzers einschließlich ihrer relativen Pfade gespeichert sind, und die beiden Buttons *Test* und *Back*. Die für die einzelnen Verzeichnisse zuständigen *CVS* - Verzeichnisse werden nicht in der Liste aufgeführt. Ist bei einer Datei ein Konflikt durch die *CVS* - Befehle *checkout* oder *update* aufgetreten, so wird die mit '#' beginnende Datei ebenfalls nicht in der Liste angezeigt. Jede Zeile dieser Liste besteht entweder aus einem Verzeichnis oder aus einer Datei. Dabei beginnen die Zeilen der Verzeichnisse mit '+' und die Zeilen der Dateien mit '-'. Ist bei einer Datei ein *CVS* - Konflikt aufgetreten, so wird die Konfliktdatei, die zu der vorher erwähnten, mit '#' beginnenden

Datei gehört, folgendermaßen eingetragen: ‘- *Konfliktdatei* #’. Alle Verzeichnisse und Dateien, die in einem in der Liste vorkommenen Verzeichnis stehen, werden eingerückt zu diesem in die Liste eingetragen.

Solange keine Datei markiert ist, ist der *Test* - Button deaktiviert. Der Benutzer kann nun maximal eine Datei in dieser Liste markieren, was dazu führt, daß der *Test* - Button aktiviert wird. Durch Anklicken dieses Buttons wird der *FileAgent* erzeugt und mit der markierten Datei zum eigenen *Serveragenten* geschickt. Hier wird die markierte Datei auf Unterschiede zu den Versionen der anderen Benutzer, die auch an dieser Datei arbeiten, untersucht. Es kann somit eine Datei überprüft werden, die nicht im Rahmen der regelmäßigen Überprüfung überwacht wird bzw. die sofort und nicht erst bei der regelmäßigen Überprüfung untersucht wird.

Der *FileAgent* überbringt dem *Serveragenten* die markierte Datei nicht als Datei, sondern als Vektor, in den diese Datei zeilenweise eingefügt worden ist. Zusätzlich wird auch der Name und die Nummer des Benutzers, der relative Pfad plus Dateiname und die Versionsnummer dieser Datei überbracht. Versucht der Benutzer, eine Konfliktdatei oder ein Verzeichnis zu markieren, so wird die Markierung sofort wieder aufgehoben. Der *Test* - Button ist hiernach deaktiviert.

Dieses Fenster wird auf dem Bildschirm angezeigt, wenn der Benutzer im *AVC - Management* - Fenster den *Test one File* - Button anklickt. Dieses Fenster wird durch Drücken des *Back* - Buttons in diesem *AVC - File-Menu* - Fenster wieder beseitigt und das *AVC - Management* - Fenster wird daraufhin an seiner Stelle dargestellt.

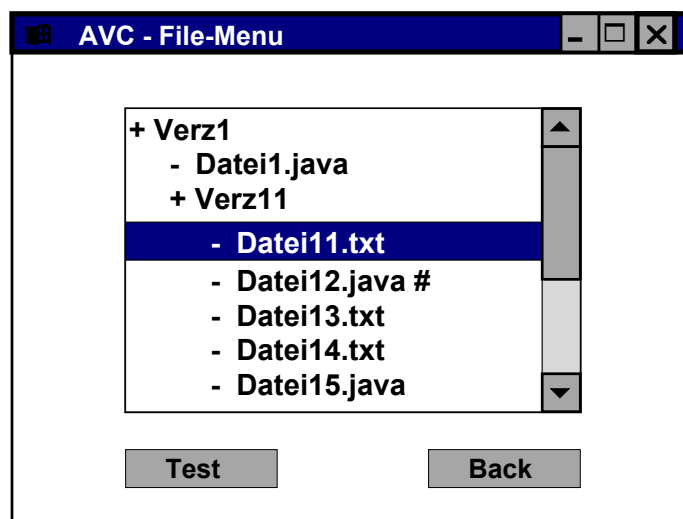


Abb. 5.2: Beispiel für ein *AVC - File-Menu* - Fenster

Diese Klasse beinhaltet die folgenden Methoden:

1. *FilemenuAnzeigen*(boolean vorhanden , Aglet bspPart3 , String username , String usernumber)

Diese spezielle Methode ist der Konstruktor dieser Klasse. In ihr wird das Fenster aufgebaut. Alle Verzeichnisse und Dateien im Verzeichnis *FileListe* werden in eine Liste, die in das Fenster eingefügt worden ist, eingetragen, sofern sie nicht das zuständige *CVS* - Verzeichnis oder eine mit ‘.#’ beginnende Konfliktdatei sind. Außerdem werden die Buttons *Test* und *Back* in das Fenster eingefügt, wobei der *Test* - Button deaktiviert ist.

Übergeben bekommt diese Methode, ob der *Clientagent* vorhanden ist, *bspPart3* als *Clientstart* - Agent und den Namen(*username*) und die Nummer(*usernumber*) des Benutzers.

2. *void makelabel(String name , GridBagLayout gridbag , GridBagConstraints g)*
Diese Methode wird aufgerufen, um in das Fenster den übergebenen String *name* einzufügen.
Übergeben bekommt sie den einzufügenden String *name* und die dafür notwendigen Argumente *gridbag* und *g*.
3. *boolean handleEvent(Event e)*
Wird ein Eintrag in der Liste markiert, so wird diese Methode aufgerufen. Wurde dabei ein Verzeichnis oder eine Konfliktdatei markiert, so wird diese Markierung beseitigt und der *Test* - Button bleibt bzw. wird deaktiviert. Andernfalls bleibt bzw. wird der *Test* - Button aktiviert. Diese Methode wird ebenfalls aufgerufen, wenn der *Test* - oder der *Back* - Button angeklickt wurde. Im ersten Fall wird der *Test* - Button deaktiviert und die Methode *starteÜberprüfung()* aufgerufen und im zweiten Fall wird dieses Fenster beseitigt und das *AVC - Management* - Fenster auf dem Bildschirm angezeigt.
Übergeben bekommt sie, was sich ereignet hat(*Event e*). Zurückgegeben wird, ob das aufgetretene Ereignis *e* behandelt worden ist oder nicht.
4. *int subdirectories(String abstandsanzeige , String dirname, int m)*
Diese Methode wird aufgerufen, wenn der in die Liste im Fenster eingefügte Eintrag ein Verzeichnis ist. Alle Verzeichnisse und Dateien im Verzeichnis *dirname* werden in die Liste eingerückt(*abstandsanzeige*) ab der Stelle *m* eingetragen, sofern sie nicht das zuständige *CVS* - Verzeichnis oder eine mit '#' beginnende Konfliktdatei sind. Wurde ein Verzeichnis eingetragen, so ruft sich diese Methode selbst auf.
Übergeben werden folgende Angaben:
 1. Der String *abstandsanzeige* gibt an, wieweit die Elemente dieses eingetragenen Verzeichnisses eingerückt auf dem Bildschirm dargestellt werden sollen.
 2. Der String *dirname* beinhaltet den Namen des eingetragenen Verzeichnisses.
 3. *m* gibt an, an welcher Stelle in der Liste der nächste Eintrag einzufügen ist.Zurückgegeben wird die Stelle in der Liste, an der zuletzt ein Eintrag erfolgt ist.
5. *void starteÜberprüfung()*
Diese Methode wird aufgerufen, wenn der Benutzer den *Test* - Button angeklickt hat. Sie besitzt die Aufgabe, die markierte Datei zu finden. Dazu verwendet sie auch die Methode *subdirectories(...)* unter 6., falls sie ein Verzeichnis untersucht. Hat sie selbst die markierte Datei gefunden, so liest sie diese Datei zeilenweise in einen Vektor ein und erhält durch Aufruf der Methode *cvsInfo(...)* von dieser die Versionsnummer dieser Datei übergeben. Sonst wird der Vektor und die Versionsnummer von der Methode unter 6. bestimmt. Nachdem die Versionsnummer und der Datei-Vektor bestimmt worden sind, erzeugt diese Methode den *FileAgenten* und schickt diesen zum *Serveragenten*, nachdem sie dem *FileAgenten* den Namen und die Nummer des Benutzers, den relativen Pfad plus Dateinamen, den Datei-Vektor und die Versionsnummer als Nachricht übergeben hat.
6. *int subdirectories(String abstandsanzeige , String dirname , int m , int egal)*
Diese Methode wird aufgerufen, wenn beim Suchen nach dem markierten Listeneintrag ein Verzeichnis überprüft wird. Hat sie die markierte Datei gefunden, so liest sie diese Datei zeilenweise in einen Vektor ein und erhält durch Aufruf der Methode *cvsInfo(...)* von dieser

die Versionsnummer dieser Datei übergeben. Nach Beendigung dieser Methode arbeitet die Methode weiter, die diese Methode aufgerufen hat.

Übergeben werden folgende Angaben:

1. Der String *abstandsanzeige* gibt an, wie weit die Elemente dieses eingetragenen Verzeichnisses eingerückt auf dem Bildschirm dargestellt werden sollen.
2. Der String *dirname* beinhaltet den Namen des eingetragenen Verzeichnisses.
3. *m* gibt an, an welcher Stelle in der Liste sich der nächste Eintrag befindet.
4. *egal* dient zur Unterscheidung zu der oben aufgeführten, gleichnamigen Methode.

Zurückgegeben wird die Stelle in der Liste, an der der letzte Eintrag untersucht worden ist.

7. *String cvsInfo(File cvsFile)*

Diese Methode besitzt die Aufgabe, die Versionsnummer der übergebenen Datei *cvsFile* zurückzugeben. Sie liest dafür die entsprechende Zeile aus der für das Verzeichnis, in dem die Datei steht, zuständigen CVS - Datei *Entries* ein und bestimmt aus dieser Zeile die Versionsnummer der Datei.

5.1.3 HauptmenuAnzeigen

Diese Klasse ist ein Teil der Benutzeroberfläche. Ihre Superklasse ist die Klasse *Frame*. Ein Objekt dieser Klasse wird erzeugt, wenn der *Clientstart* - Agent ins Leben gerufen wird oder wenn im *AVC - Management* - Fenster oder im *AVC - Mail-Menu* - Fenster der *Back* - Button angeklickt wird. Bei dieser Erzeugung wird das *AVC - Main-Menu* - Fenster auf dem Bildschirm angezeigt. Dieses Fenster besitzt den Fenstertitel *AVC - Main-Menu*. Innerhalb des Fensters befinden sich fünf Buttons: *Create*, *Dispose*, *Management*, *Mails* und *Close*.

Die ersten beiden Buttons beziehen sich auf den *Clientagenten*. Ist der *Clientagent* bereits vorhanden, so ist der Button *Create* deaktiviert und der Button *Dispose* aktiviert. Durch Anklicken des *Dispose* - Buttons kann der Benutzer den *Clientagenten* beseitigen. Dabei wird der *Dispose* - Button deaktiviert und der *Create* - Button aktiviert. Ist der *Clientagent* dagegen nicht vorhanden, so ist der Button *Create* aktiviert und der *Dispose* - Button deaktiviert. Der Benutzer hat dann die Möglichkeit, durch Anklicken des *Create* - Buttons den *Clientagenten* zu erzeugen. In diesem Fall wird der *Create* - Button deaktiviert und der Button *Dispose* aktiviert.

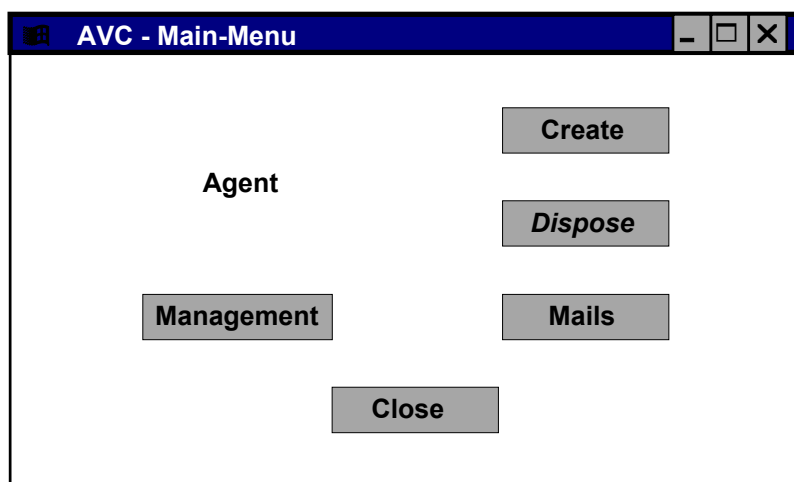


Abb. 5.3: Beispiel für ein *AVC - Main-Menu* - Fenster
(Der *Clientagent* existiert nicht)

Die restlichen drei Buttons sind jederzeit aktiviert. Durch Drücken des Buttons *Management* wird dieses Fenster entfernt und der Benutzer gelangt zum *AVC - Management* - Fenster. Wird dagegen der *Mails* - Button angeklickt, so wird das *AVC - Mail-Menu* - Fenster auf dem Bildschirm angezeigt, nachdem dieses Fenster beseitigt worden ist. Sollte der Benutzer den Button *Close* anklicken, so werden dieses Fenster und der *Clientstart* - Agent beseitigt.

Diese Klasse beinhaltet die folgenden Methoden:

1. *HauptmenuAnzeigen(boolean vorhand , Aglet bspPart3 , String username , String usernumber)*

Diese spezielle Methode ist der Konstruktor dieser Klasse. In ihr wird das Fenster aufgebaut. Dafür werden der Schriftzug *Agent* und die fünf Buttons *Create*, *Dispose*, *Management*, *Mails* und *Close* in dieses Fenster eingefügt. Ist der *Clientagent* bereits vorhanden, so wird der *Create* - Button deaktiviert und der *Dispose* - Button aktiviert. Existiert dagegen der *Clientagent* nicht, so wird der *Create* - Button aktiviert und der *Dispose* - Button deaktiviert. Die restlichen drei Buttons sind immer aktiviert. Übergeben bekommt diese Methode, ob der *Clientagent vorhanden* ist, den Namen(*username*) und die Nummer (*usernnumber*) des Benutzers und *bspPart3* als *Clientstart* - Agent.

2. *void makelabel(String name , GridBagLayout gridbag , GridBagConstraints c)*

Diese Methode wird aufgerufen, um in das Fenster den übergebenen String *name* einzufügen.

Übergeben bekommt sie den einzufügenden String *name* und die dafür notwendigen Argumente *gridbag* und *c*.

3. *boolean action(Event e , Object arg)*

Diese Methode reagiert auf das Anklicken der im Fenster vorhandenen, aktivierten Buttons. Wird der *Create* - Button angeklickt, so wird dieser Button deaktiviert und der *Dispose* - Button aktiviert. Ferner wird die Variable *vorhanden* auf 'true' gesetzt. Außerdem wird der *Clientagent* in dem *Context* erzeugt, in dem sich auch der *Clientstart* - Agent befindet. Dem *Clientagenten* übergeben werden der Benutzername, die Benutzernummer und die Portnummer des *Contextes*. Wird dagegen der *Dispose* - Button gedrückt, so wird dieser *Button* deaktiviert und der *Create* - Button aktiviert. Außerdem wird die Variable *vorhanden* auf 'false' gesetzt. Weiterhin wird der *Clientagent* beseitigt, indem seinem *Proxy* die Nachricht übergeben wird, daß er sich beseitigen soll. Diesen *Proxy* erhält diese Methode von der Methode *getProx(...)* übergeben.

Wird der *Management* - Button angeklickt, so wird dieses *AVC - Main-Menu* - Fenster beseitigt und das *AVC - Management* - Fenster auf dem Bildschirm angezeigt. Durch Drücken des Buttons *Mails* wird das *AVC - Main-Menu* - Fenster geschlossen und das *AVC - Mail-Menu* - Fenster auf dem Bildschirm dargestellt. Wird der *Close* - Button angeklickt, so wird dieses *AVC - Main-Menu* - Fenster geschlossen und der *Clientstart* - Agent mit Hilfe seines *Proxy* beseitigt. Diesen *Proxy* erhält diese Methode von der Methode *getProx(...)* übergeben.

Zurückgegeben wird, ob das aufgetretene Ereignis *e* behandelt worden ist.

4. *AgletProxy getProx(Aglet contextAglet , String agletName)*

Diese Methode wird aufgerufen, um den *Proxy* des Agenten *agletName* zu erhalten. Übergeben werden der *Clientstart* - Agent als *contextAglet*, um auf den *Context* zugreifen zu können, und der Name des gesuchten Agenten . Es wird eine Aufzählung aller in diesem *Context* vorhandenen *Aglets* erzeugt, die dann darauf untersucht wird, ob der gesuchte

Agent in ihr enthalten ist. Falls ja, so wird dessen *Proxy* zurückgegeben. Andernfalls wird *null* zurückgegeben.

5.1.4 KonfliktAlarm



Abb. 5.4: Beispiel für ein *AVC - Conflict-Alarm* - Fenster

Diese Klasse ist ein Teil der Benutzeroberfläche, wobei ihre Superklasse die Klasse *Frame* ist. Wird ein Objekt dieser Klasse erzeugt, so wird das *AVC - Conflict-Alarm* - Fenster auf dem Bildschirm angezeigt, dessen Fenstertitel *AVC - Conflict-Alarm* lautet. Der Inhalt dieses Fensters besteht aus dem Namen einer Konfliktdatei einschließlich ihres relativen Pfades zum *FileListe* - Verzeichnis und der Meldung, daß diese Datei aufgrund von *CVS* - Konflikten nicht überprüft werden kann. Außerdem beinhaltet das Fenster auch noch den Button *Okay*, durch dessen Anklicken dieses Fenster wieder beseitigt wird. Ein Objekt dieser Klasse wird erzeugt, wenn der *Clientagent* versucht, eine Datei überprüfen zu lassen, bei der durch den *CVS* - Befehl *update* bzw. *checkout* ein Konflikt aufgetreten ist. Den Namen dieser Konfliktdatei bekommt dieses Objekt vom *Clientagenten* bei seiner Erzeugung übergeben.

Diese Klasse beinhaltet die folgenden Methoden:

1. *KonfliktAlarm(String filename)*

Diese spezielle Methode ist der Konstruktor dieser Klasse. In ihr wird das Fenster aufgebaut. Zuerst wird der Name der Konfliktdatei einschließlich ihres zum Verzeichnis *FileListe* relativen Pfades in das Fenster eingefügt. Darauf wird der Schriftzug *can't be tested because of conflicts !* eingetragen. Der Button *Okay* wird ebenfalls in dieses Fenster eingebaut.

Übergeben bekommt diese Methode den Namen einschließlich des relativen Pfades der Konfliktdatei.

2. *void makelabel(String name , GridBagLayout gridbag , GridBagConstraints h)*

Diese Methode wird aufgerufen, um in das Fenster den übergebenen String *name* einzufügen.

Übergeben bekommt sie den einzufügenden String *name* und die dafür notwendigen Argumente *gridbag* und *h*.

3. *boolean action(Event e , Object arg)*

Diese Methode reagiert auf das Anklicken des im Fenster vorhandenen Buttons *Okay*. Es wird dabei die Variable *gedrueckt* auf 'true' gesetzt, was zur Folge hat, daß der *Clientagent* mit seinen Aufgaben fortfahren kann. Außerdem wird dadurch dieses Fenster beseitigt. Zurückgegeben wird, ob das aufgetretene Ereignis *e* behandelt worden ist.

5.1.5 MailAlarm

Diese Klasse ist ein Teil der Benutzeroberfläche, wobei ihre Superklasse die Klasse *Frame* ist. Wird ein Objekt dieser Klasse erzeugt, so wird das *AVC - Mail-Alarm* - Fenster auf dem Bildschirm angezeigt, dessen Fenstertitel *AVC - Mail-Alarm* lautet. Der Inhalt dieses Fensters besteht aus der Mitteilung, daß eine neue Mail vom *MailAgenten* überbracht worden ist. Außerdem beinhaltet das Fenster auch noch den Button *Okay*, durch dessen Anklicken dieses Fenster wieder beseitigt wird. Ein Objekt dieser Klasse wird erzeugt, wenn dem *Clientagenten* durch den *MailAgenten* eine neue Mail übergeben worden ist und der Benutzer im *AVC - Management* - Fenster angegeben hat, daß er über die Ankunft einer solchen Mail informiert werden möchte.



Abb. 5.5: Das *AVC - Mail-Alarm* - Fenster

Diese Klasse beinhaltet die folgenden Methoden:

1. *MailAlarm()*

Diese spezielle Methode ist der Konstruktor dieser Klasse. In ihr wird das Fenster aufgebaut. Zuerst wird der Schriftzug *A new mail has arrived !* in das Fenster eingefügt. Danach wird der Button *Okay* eingebaut.

2. *void makelabel(String name , GridBagLayout gridbag , GridBagConstraints h)*

Diese Methode wird aufgerufen, um in das Fenster den übergebenen String *name* einzufügen.

Übergeben bekommt sie den einzufügenden String *name* und die dafür notwendigen Argumente *gridbag* und *h*.

3. *boolean action(Event e , Object arg)*

Diese Methode reagiert auf das Anklicken des im Fenster vorhandenen Buttons *Okay*, was zur Folge hat, daß dieses Fenster wieder beseitigt wird.

Zurückgegeben wird, ob das aufgetretene Ereignis *e* behandelt worden ist.

5.1.6 MailAnzeigen

Diese Klasse ist ein Teil der Benutzeroberfläche. Ihre Superklasse ist die Klasse *Frame*. Wird ein Objekt dieser Klasse erzeugt, so wird das *AVC - Mail* - Fenster mit dem gleichnamigen Fenstertitel auf dem Bildschirm angezeigt. In diesem Fenster befinden sich der Name einschließlich des zum Verzeichnis *FileListe* relativen Pfades der Datei, bei der Unterschiede zu einer Version eines anderen Benutzers aufgetreten sind, der Zeitpunkt der Überprüfung, bei der diese Unterschiede festgestellt worden sind, und der Name und die Nummer dieses anderen Benutzers. Außerdem beinhaltet dieses Fenster noch einen Textbereich, in dem die Unterschiede aufgeführt sind.

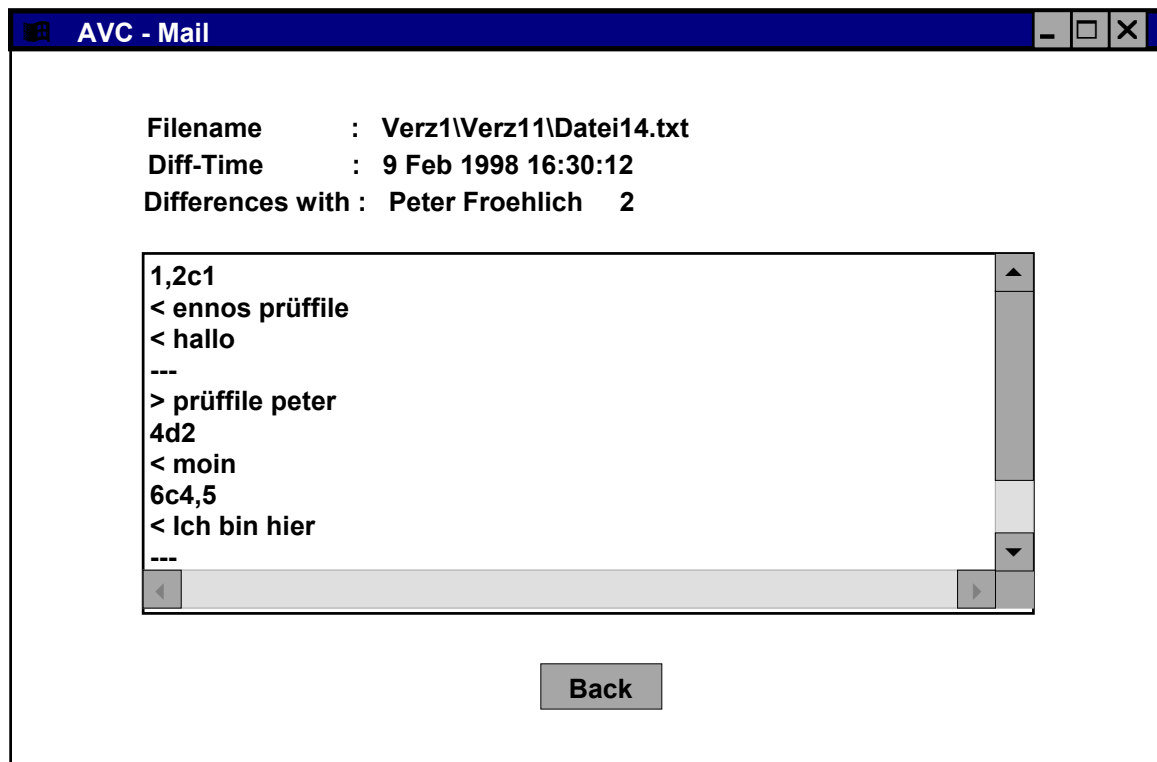


Abb. 5.6: Beispiel für ein *AVC - Mail* - Fenster

Ein Objekt dieser Klasse wird erzeugt, wenn im *AVC - Mail-Menu* - Fenster der Button *Show* angeklickt wird. Der Benutzer kann dann in diesem Fenster den Inhalt der vorher im *AVC - Mail-Menu* - Fenster markierten Mail einsehen. Durch das Drücken des ebenfalls im *AVC - Mail* - Fenster vorhandenen *Back* - Buttons wird dieses Fenster beseitigt und das *AVC - Mail-Menu* - Fenster auf dem Bildschirm angezeigt.

Diese Klasse beinhaltet die folgenden Methoden:

1. *MailAnzeigen(booleen vorhanden , Aglet bspPart3 , String username , String usernumber , File file)*

Diese spezielle Methode ist der Konstruktor dieser Klasse. In ihr wird das Fenster aufgebaut. Zuerst werden aus dem übergebenen Argument *file* folgende Angaben gewonnen: Der Name einschließlich des zum Verzeichnis *FileListe* relativen Pfades der überprüften Datei, der Name und die Nummer des Prüfungspartners, der Zeitpunkt der Überprüfung, bei der die Unterschiede aufgetreten sind, und die Unterschiede an sich. Diese Angaben werden dabei folgendermaßen erhalten: Die Mail *file* ist im Verzeichnis *Mails* des Benutzers gespeichert. Der Dateiname dieser Mail besteht dabei aus dem Namen einschließlich des relativen Pfades der untersuchten Datei zuzüglich des Namens und der Nummer, die in Klammern eingeschlossen sind. Der Name einschließlich des relativen Pfades und der Name und die Nummer des anderen Benutzers können also aus dem Namen der Mail gewonnen werden. Der Zeitpunkt der Überprüfung, bei der die Unterschiede bemerkt worden sind, ist in der ersten Zeile der Mail gespeichert. Die Unterschiede sind dann in den folgenden Zeilen dieser Mail zu finden.

Nun wird der Name und der zugehörige relative Pfad in das Fenster eingetragen:

Filename : 'Prüfdatei'.

Darauf wird der Zeitpunkt eingefügt: *Diff-Time* : 'Prüfzeit'.

Danach folgt in der dritten Zeile der Name und die Nummer des anderen Benutzers:
Differences with : 'Name' 'Nummer'.

Die Unterschiede werden nun in den Textbereich eingetragen, der vorher in das Fenster eingefügt worden ist. Unterhalb dieses Textbereiches wird dann der *Back* - Button eingebaut.

Übergeben bekommt diese Methode, ob der *Clientagent vorhanden* ist, *bspPart3* als *Clientstart* - Agent und den Namen(*username*) und die Nummer(*usernumber*) des Benutzers.

2. *void makelabel(String name , GridBagLayout gridbag , GridBagConstraints g)*

Diese Methode wird aufgerufen, um in das Fenster den übergebenen String *name* einzufügen.

Übergeben bekommt sie den einzufügenden String *name* und die dafür notwendigen Argumente *gridbag* und *g*.

3. *boolean handleEvent(Event e)*

Diese Methode reagiert auf das Anklicken des im Fenster vorhandenen *Back* - Buttons.

Dieses Fenster wird dabei beseitigt und das *AVC - Mail-Menu* - Fenster auf dem Bildschirm angezeigt.

5.1.7 MailmenuAnzeigen

Diese Klasse ist ein Teil der Benutzeroberfläche, wobei ihre Superklasse die Klasse *Frame* ist. Wird ein Objekt dieser Klasse erzeugt, so wird das *AVC - Mail-Menu* - Fenster mit dem gleichnamigen Fenstertitel auf dem Bildschirm angezeigt. In diesem Fenster befinden sich die Liste der Mails, die im Verzeichnis *Mails* des Benutzers gespeichert sind, und die drei Buttons *Show*, *Delete* und *Back*. Der in diesem Fenster angezeigte Dateiname der Mail besteht dabei aus dem Namen einschließlich des zum Verzeichnis *FileListe* relativen Pfades der Datei, bei der die Differenzen während der Überprüfung festgestellt worden sind, und dem Namen und der Nummer des Prüfungspartners, die eingeschlossen in Klammern dem vorher erwähnten Dateinamen angefügt sind. Erreicht eine neue Mails durch den *MailAgenten* den *Clientagenten*, so wird diese Mail im Verzeichnis *Mails* gespeichert und die alte Mail, die sich auf die gleiche Datei und auf denselben Prüfungspartner bezieht, dadurch gelöscht.

Hat der Benutzer eine Mail markiert, so kann er sich diese durch Anklicken des *Show* -Buttons im *AVC - Mail* - Fenster anschauen, das in diesem Fall auf dem Bildschirm angezeigt wird, nachdem dieses *AVC - Mail-Menu* - Fenster beseitigt worden ist. Außerdem besitzt der Benutzer die Möglichkeit, durch Drücken des Buttons *Delete* eine vorher markierte Mail zu löschen. Sind keine Mails in diesem Fenster aufgeführt, so sind die Buttons *Show* und *Delete* deaktiviert. Ein Objekt dieser Klasse wird erzeugt, wenn der Benutzer im *AVC - Main-Menu* - Fenster den *Mails* - Button oder im *AVC - Mail* - Fenster den *Back* - Button anklickt. Drückt der Benutzer in diesem Fenster den *Back* - Button, so wird dieses Fenster beseitigt und das *AVC - Main-Menu* - Fenster auf dem Bildschirm angezeigt.

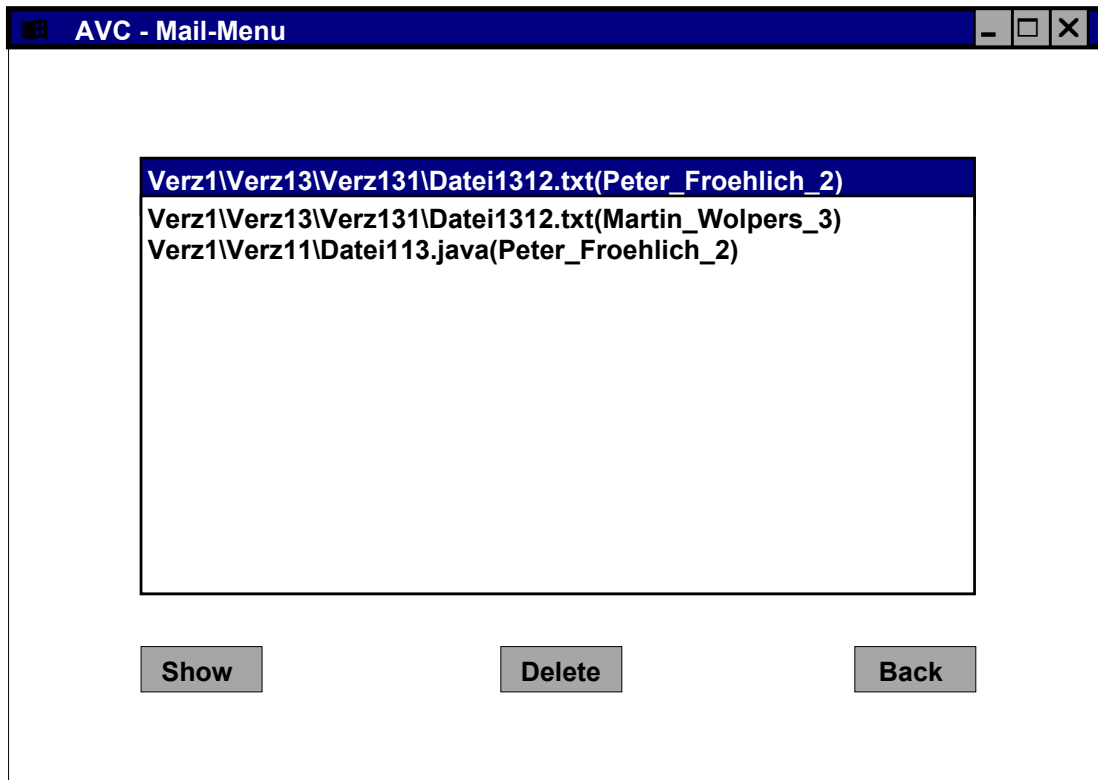


Abb. 5.7: Beispiel für ein *AVC - Mail-Menu* - Fenster

Diese Klasse beinhaltet die folgenden Methoden:

1. *MailmenuAnzeigen(Aglet bspPart3 , boolean vorh , String username , String usernumber)*

Diese spezielle Methode ist der Konstruktor dieser Klasse. In ihr wird das Fenster aufgebaut. Alle Mails, die im Verzeichnis *Mails* des Benutzers gespeichert sind, werden in die Liste, die vorher in das Fenster eingefügt worden ist, eingetragen. Danach wird die erste aufgeführte Mail dieser Liste markiert, sofern in dieser Liste Mails vorhanden sind. Außerdem werden die Buttons *Show*, *Delete* und *Back* in das Fenster eingefügt. Ist keine Mail in der Liste eingetragen, so sind die Buttons *Show* und *Delete* deaktiviert. Existiert dagegen mindestens eine Mail, so sind die gerade erwähnten Buttons aktiviert. Der Button *Back* ist immer aktiviert.

Übergeben bekommt diese Methode, ob der *Clientagent* vorhanden ist, *bspPart3* als *Clientstart* - Agent sowie den Namen(*username*) und die Nummer(*usernumber*) des Benutzers.

2. *void makelabel(String name , GridBagLayout gridbag , GridBagConstraints h)*

Diese Methode wird aufgerufen, um in das Fenster den übergebenen String *name* einzufügen.

Übergeben bekommt sie den einzufügenden String *name* und die dafür notwendigen Argumente *gridbag* und *h*.

3. *boolean action(Event e , Object arg)*

Diese Methode reagiert auf das Anklicken der im Fenster vorhandenen, aktivierten Buttons. Wird der *Show* - Button gedrückt, so wird dieses Fenster beseitigt und das *AVC - Mail* - Fenster mit der vorher markierten Mail auf dem Bildschirm angezeigt. Das Anklicken des *Delete* - Buttons bewirkt das Löschen der vorher markierten Mail. Nach diesem Löschen wird wieder die erste Mail markiert, sofern in der Liste noch eine vorhanden ist. Ist keine Mail mehr in der Liste existent, so werden die Buttons *Show* und *Delete* deaktiviert. Wird

der Button *Back* gedrückt, so wird dieses Fenster beseitigt und das *AVC - Main-Menu - Fenster* auf dem Bildschirm angezeigt.
Zurückgegeben wird, ob das aufgetretene Ereignis *e* behandelt worden ist.

5.1.8 VerwaltungsmenuAnzeigen

Diese Klasse ist ein Teil der Benutzeroberfläche. Ihre Superklasse ist die Klasse *Frame*. Ein Objekt dieser Klasse wird erzeugt, wenn der Benutzer den *Management - Button* im *AVC - Main-Menu - Fenster* oder den *Back - Button* im *AVC - Choice-Menu - Fenster* bzw. im *AVC - File-Menu - Fenster* anklickt. Es wird das *AVC - Management - Fenster* bei dieser Erzeugung auf dem Bildschirm angezeigt, das den gleichnamigen Fenstertitel besitzt. Innerhalb des Fensters befindet sich der *Add/Delete - Button*. Dieser Button ist aktiviert, wenn der *Clientagent* vorhanden ist. Andernfalls ist er deaktiviert. Wird dieser Button angeklickt, so wird dieses Fenster beseitigt und das *AVC - Choice-Menu - Fenster* angezeigt.

Außerdem beinhaltet dieses Fenster ein Panel. Innerhalb dieses Panels existiert ein Textfeld, in dem der Zeitabstand zwischen den regelmäßigen Überprüfungen in Minuten aufgeführt ist. Dieser Zeitabstand wird bei der Erzeugung dieses Fensters aus der Datei *Userdaten.txt* des Benutzers eingelesen. Der Benutzer besitzt dann die Möglichkeit, durch Eingabe eines neuen Zeitabstandes den alten zu überschreiben. Zudem sind in dem Fenster auch zwei alternierende Kontrollkästchen vorhanden, die angeben, ob der Benutzer über die Ankunft einer neuen Mail informiert werden möchte. Bei der Erzeugung dieses Fensters wird diese Angabe ebenfalls aus der Benutzerdatei *Userdaten.txt* bestimmt. Der Benutzer hat auch hier die Fähigkeit, durch Markierung des vorher nichtmarkierten Kontrollkästchens diese Angabe zu verändern.

Weiterhin befinden sich in dem Fenster noch die beiden Buttons *Test one File* und *Back*. Der erste Button davon ist deaktiviert, wenn der *Clientagent* nicht vorhanden ist. Andernfalls ist er aktiviert. Wird dieser Button angeklickt, so wird dieses Fenster beseitigt und das *AVC - File-Menu - Fenster* an seiner Stelle dargestellt. Der *Back - Button* dagegen ist bis zu seinem Anklicken aktiviert. Durch Drücken dieses Buttons wird dieses Fenster beseitigt und das *AVC - Main-Menu - Fenster* auf dem Bildschirm angezeigt. Außerdem werden die Angaben bezüglich des Alarmes infolge der Ankunft einer neuen Mail und bezüglich des Zeitabstandes zwischen zwei regelmäßigen Überprüfungen, die zum Zeitpunkt der Beseitigung aktuell sind, in der Benutzerdatei *Userdaten.txt* gespeichert.

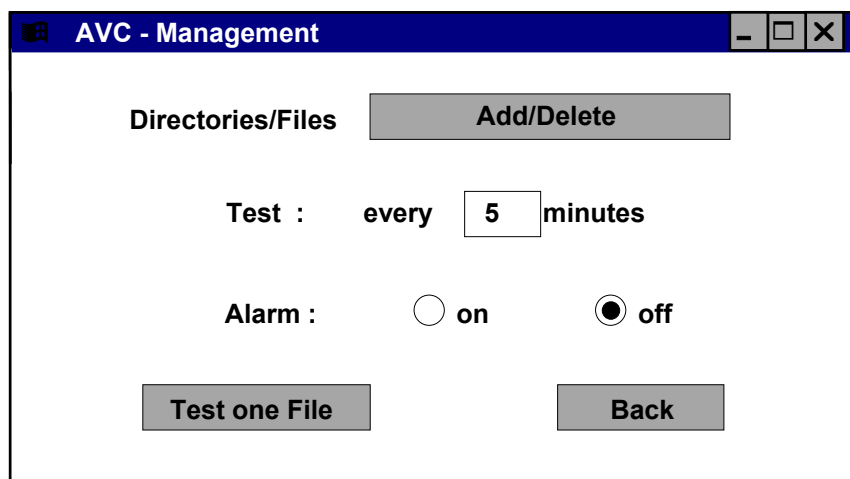


Abb. 5.8: Beispiel für ein *AVC - Management - Fenster*

Ist der Zeitabstand zwischen den Überprüfungen hierbei kleiner als 5 Minuten oder größer als 100 Minuten, so wird ein Zeitabstand von 5 Minuten in der Datei festgehalten. Hat sich dieser Zeitabstand im Vergleich zum vorher gespeicherten Wert verändert, so wird zusätzlich der *ClientPause - Slave* aktiviert.

Diese Klasse enthält die folgenden Methoden:

1. *VerwaltungsmenuAnzeigen(boolean vorhanden , Aglet bspPart3 , String username , String usernumber)*

Diese spezielle Methode ist der Konstruktor dieser Klasse. In ihr wird das Fenster aufgebaut. Dafür werden zuerst der Schriftzug *Directories/Files* und der Button *Add/Delete* eingefügt. Ist der *Clientagent* nicht vorhanden, so wird der Button deaktiviert. Andernfalls ist er aktiviert. Danach wird der Schriftzug *Test:* eingebaut, dem dann ein Panel folgt, in dem sich der Schriftzug *every* und ein Textfeld befinden. Abgeschlossen wird diese Zeile durch die Textzeile *minutes*. Den in das vorher genannte Textfeld einzutragenen Wert erhält diese Methode durch Aufruf der Methode *infoDaten(...)* übergeben. Weiterhin werden in das Fenster zwei alternierende Kontrollkästchen eingefügt, die mit *on* und *off* beschriftet sind. Die Textzeile *Alarm:* steht dabei vor diesen beiden Kontrollkästchen. Welches dieser beiden Kontrollkästchen markiert ist, bekommt diese Methode durch Aufruf der Methode *infoDaten(...)* übergeben. Zuletzt werden noch die beiden Buttons *Test one File* und *Back* in das Fenster eingefügt. Der *Test one File* - Button ist dabei nur aktiviert, wenn der *Client-agent* vorhanden ist. Andernfalls ist er deaktiviert.

Übergeben bekommt diese Methode, ob der *Clientagent vorhanden* ist, *bspPart3* als *Clientstart* - Agent sowie den Namen(*username*) und die Nummer(*usernnumber*) des Benutzers.

2. *void makelabel(String name , GridBagLayout gridbag , GridBagConstraints d)*

Diese Methode wird aufgerufen, um in das Fenster den übergebenen String *name* einzufügen.

Übergeben bekommt sie den einzufügenden String *name* und die dafür notwendigen Argumente *gridbag* und *d*.

3. *boolean action(Event e , Object arg)*

Diese Methode reagiert auf das Anklicken der im Fenster vorhandenen, aktivierten Buttons. Wird einer der aktivierten Buttons durch den Benutzer gedrückt, so werden die Informationen über den Mail - Alarm und über den Überprüfungszeitabstand an die Methode *schreibeInfoDaten(...)* übergeben, die ihrerseits zurückgibt, ob der *ClientPause - Slave* aktiviert werden soll. Falls ja, so wird die Methode *clientPauseAktivieren()* aufgerufen. War es der *Add/Delete* - Button, der angeklickt wurde, so wird im Anschluß zu dem vorher genannten das *AVC - Choice-Menu* - Fenster auf dem Bildschirm angezeigt. War es dagegen der *Test one File* - Button, so wird das *AVC - File-Menu* - Fenster dargestellt. War allerdings der *Back* - Button der Grund für den Aufruf dieser Methode, so wird, nachdem die oben genannten Aktionen durchgeführt worden sind, das *AVC - Main-Menu* - Fenster angezeigt. In allen möglichen drei Fällen wird dieses *AVC - Management* - Fenster beseitigt.

Zurückgegeben wird, ob das aufgetretene Ereignis *e* behandelt worden ist.

4. *String infoDaten(String was)*

Diese Methode wird aufgerufen, wenn das *AVC - Management* - Fenster aufgebaut wird. Sie liest die benötigte Information *was* aus der Benutzerdatei *Userdaten.txt* ein und gibt diese an die aufrufende Methode zurück.

Übergeben bekommt sie, welche Information *was* gebraucht wird. Zurückgegeben wird die gesuchte Information.

5. *boolean schreibeInfoDaten(String alarm_an , String pruefungszahl)*

Diese Methode wird aufgerufen, wenn dieses *AVC - Management* - Fenster beseitigt wird. Sie speichert die Informationen bezüglich des Mail - Alarmes und des Zeitabstandes zwischen den regelmäßigen Überprüfungen in der Datei *Userdaten.txt*. Beträgt dabei der Zeitabstand weniger als 5 Minuten oder mehr als 100 Minuten, so wird ein Zeitabstand von 5 Minuten in der Datei gespeichert.

Übergeben bekommt sie, ob der Benutzer über die Ankunft einer neuen Mail informiert werden möchte(*alarm_an*), und wie groß der Zeitabstand zwischen zwei regelmäßigen Überprüfungen sein soll(*pruefungszahl*). Sie gibt zurück, ob sich der Zeitabstand im Vergleich zur letzten Speicherung dieses Wertes verändert hat.

6. *void clientPauseAktivieren()*

Der Aufruf dieser Methode bewirkt, daß der *ClientPause - Slave* aktiviert wird. Um dieses zu erreichen, wird in der Menge aller im *Context* des *Clientstart* - Agenten vorhandenen *Proxies* der *Proxy* des *ClientPause - Slaves* gesucht. Wird dieser *Proxy* dabei gefunden, so wird mit dessen Hilfe der *ClientPause - Slave* aktiviert.

5.2 Die Server-Organisation

5.2.1 ServerDir

Diese Klasse beinhaltet das Verzeichnis auf dem Server, in dem das *AVC* - Verzeichnis zu finden ist. Objekte dieser Klasse werden von den *Serveragenten*, von dem *Serverstart* - Agenten und von dem *ServeragentStart* - Agenten verwendet. Ruft einer dieser Agenten die Methode *getServerDir()* dieser Klasse auf, so erhält der aufrufende Agent das Verzeichnis als String zurück. Um bei der Installation dieses Programmes das Verzeichnis auf dem Server festzulegen, in dem sich das *AVC* - Verzeichnis befindet, muß die folgende Zeile in der Methode *getServerDir()* angepaßt werden:

```
serverdir = gewünschtesVerzeichnis ;
```

5.2.2 ServerURL

Diese Klasse beinhaltet die Internet-Adresse sowie die Portnummer des *Contextes* auf dem Server, in dem sich die *Serveragenten* und der *Verwaltungsagent* befinden. Als Protokoll wird das *Agent Transfer Protocol (ATP)* verwendet. Jeder *Clientagent* erzeugt und verwendet ein Objekt dieser Klasse. Um die Internet-Adresse sowie die Portnummer als *URL* - Objekt zu erhalten, ruft der *Clientagent* die Methode *getServerUrl()* dieser Klasse auf. Um bei der Installation dieses Programmes die Internet-Adresse sowie die Portnummer des *Contextes* auf dem Server festzulegen, in dem sich die *Serveragenten* und der *Verwaltungsagent* befinden, muß die folgende Zeile in der Methode *getServerUrl()* angepaßt werden:

```
serverUrl = new URL("atp://Internet-Adresse:Portnummer");
```

5.3 Die Agenten

5.3.1 Clientagent

Diese Klasse repräsentiert einen stationären Agenten, wobei ihre Superklasse die Klasse *Aglet* ist. Er ist der clientseitige Benutzeragent. Pro Benutzer kann ein *Clientagent* existieren. Er wird durch das Anklicken des *Create* - Buttons im *AVC - Main-Menu* - Fenster erzeugt und durch Anklicken des Buttons *Dispose* in demselben Fenster wieder beseitigt. Bei seiner Erzeugung meldet er sich bei seinem *Serveragenten* an und erzeugt den *ClientPause - Slave*. Dieser *Slave* kehrt nach Ablauf der Zeit, die durch den Abstand zwischen zwei regelmäßigen Überprüfungen festgesetzt ist, zu diesem *Clientagenten* zurück, um die regelmäßige Überprüfung der überwachten Dateien und die Erzeugung eines neuen *ClientPause - Slaves* zu bewirken. Kehrt der mit der Anmeldung beauftragte *ClientServerAnmeldung - Slave* mit der Nachricht der erfolgreichen Anmeldung zurück, so wird ebenfalls eine regelmäßige Überprüfung gestartet.

Bei der regelmäßigen Überprüfung wird aus der Datei *Userdaten.txt* der Zeitpunkt der letzten Überprüfung eingelesen. Wird dabei eine '0' gefunden, so bedeutet das, daß die Liste der zu überwachenden Dateien sich verändert hat. Danach wird aus der Datei *FileListe.txt* die Liste der überwachten Dateien eingelesen. Für jede dieser Dateien wird überprüft, ob sich die für diese Datei zuständige *CVS - Datei Entries* seit der letzten Überprüfung verändert hat, was durch den *CVS - Befehl checkout* oder *update* geschieht. Ist dies bei mindestens einer dieser Dateien der Fall oder hat sich die Liste der überwachten Dateien verändert, so werden alle überwachte Dateien durch den *FilePruefAgenten* zum eigenen *Serveragenten* zur Überprüfung gebracht. Haben sich weder die Liste der überwachten Dateien noch die entsprechenden *CVS - Entries - Dateien* seit der letzten Überprüfung verändert, so wird jede überwachte Datei daraufhin untersucht, ob sie sich seit der letzten Überprüfung verändert hat. Alle Dateien, die sich verändert haben, werden in eine Liste eingetragen, die dann, falls sie nicht leer ist, durch den *FilePruefAgenten* zum eigenen *Serveragenten* zur Überprüfung gebracht wird.

Ist bei einer zu überwachenden Datei ein *CVS - Konflikt* aufgetreten, so wird diese Datei nicht zum *Serveragenten* gebracht, sondern aus der Datei *FileListe.txt* gelöscht, und das *AVC - Conflict-Alarm* - Fenster auf dem Bildschirm angezeigt. Kehrt der *FilePruefAgent*, nachdem er die zu überprüfenden Dateien beim eigenen *Serveragenten* abgegeben hat, zu diesem *Clientagenten* zurück, so wird die Überprüfungszeit, die beim Beginn der Auswahl der zu überbringenden Dateien bestimmt wurde, in der Datei *Userdaten.txt* als 'Letzte Pruefung' gespeichert.

Übergibt der *MailAgent* diesem *Clientagenten* die Mail seines *Serveragenten*, so wird überprüft, ob der Benutzer über die Ankunft einer Mail informiert werden möchte. Sollte dies der Fall sein, so wird das *AVC - Mail-Alarm* - Fenster auf dem Bildschirm angezeigt. Auf jeden Fall wird die Mail im Verzeichnis *Mails* des Benutzers gespeichert.

Kommt der *ClientServerAnmeldung - Slave* oder der *FilePruefAgent* zurück und übergibt die Nachricht, daß entweder der *Verwaltungsagent* oder der *Serveragent* nicht gefunden werden konnte, so werden der *Clientstart - Agent*, falls er vorhanden ist, und dieser *Clientagent* beseitigt. Ist seit Erzeugung dieses Agenten der zugehörige *Serveragent* neu ins Leben gerufen worden, so wird dieses durch den *FilePruefAgenten* festgestellt, was zur Folge hat, daß sich dieser *Clientagent* neu bei seinem *Serveragenten* anmeldet.

Soll dieser *Clientagent* beseitigt werden, so aktiviert er vorher seinen *ClientPause - Slave* und meldet sich durch den *ClientServerAnmeldung - Slave* bei seinem *Serveragenten* ab.

Diese Klasse beinhaltet die folgenden Methoden:

1. *void onCreate(Object o)*

Diese Methode wird bei der Initialisierung dieses Agenten aufgerufen. In ihr wird zuerst aus dem übergebenen Argument der Name und die Nummer des Benutzers sowie die Portnummer bestimmt. Außerdem wird dieser Agent bei seinem *Serveragenten* durch Aufruf der Methode *clientBeimServerAnmelden(name, number, true)* angemeldet. Von der Methode *read_time_for_check()* erhält sie den Zeitabstand zwischen zwei regelmäßigen Überprüfungen. Danach wird der *ClientPause - Slave* erzeugt, der den vorher genannten Zeitabstand als Argument übergeben bekommt.

2. *boolean handleMessage(Message msg)*

Diese Methode wird aufgerufen, wenn ein *Slave* eines anderen Agenten oder ein eigener *Slave* eine Nachricht (*msg*) an diesen *Clientagenten* übergibt. Kehrt zum Beispiel der *ClientServerAnmeldung - Slave* zurück und überbringt die Nachricht der erfolgreichen Anmeldung bei dem *Serveragenten* dieses *Clientagenten*, so wird die regelmäßige Überprüfung durch Aufruf der Methode *starteÜberprüfung(name, number)* gestartet. Kommt dagegen der *ClientServerAnmeldung - Slave* oder der *FilePruefAgent* zurück und übergibt die Nachricht, daß entweder der *Verwaltungsagent* oder der *Serveragent* nicht gefunden werden konnte, so wird der *Proxy* des *Clientstart - Agenten* gesucht. Wird dieser *Proxy* gefunden, so wird durch ihn sein Agent beseitigt. Egal, ob der *Proxy* gefunden wurde oder nicht, dieser *Clientagent* wird danach beseitigt.

Weiterhin möglich ist die Rückkehr des *ClientPause - Slaves*, durch die die regelmäßige Überprüfung durch Aufruf der Methode *starteÜberprüfung(name, number)* gestartet wird. Außerdem wird der derzeit gültige Zeitabstand zwischen zwei regelmäßigen Überprüfungen durch die Methode *read_time_for_check()* erhalten und der *ClientPause - Slave* erzeugt, der diesen Zeitabstand als Argument übergeben bekommt.

Kehrt der *FilePruefAgent* zurück und übergibt die Nachricht, daß der *Serveragent* später erzeugt worden ist als sich dieser *Clientagent* angemeldet hat, so wird, um diesen *Clientagenten* neu anzumelden, die Methode *clientBeimServerAnmelden(name, number, true)* aufgerufen. Überbringt dieser *Slave* allerdings die Mitteilung, daß er die zu überprüfenden Dateien dem *Serveragenten* übergeben hat, so wird der Zeitpunkt, an dem die letzte Überprüfung der überwachten Dateien bei diesem *Clientagenten* die oben genannten, übergebenen Dateien bestimmt hat, in der Datei *Userdaten.txt* gespeichert.

Überbringt der *MailAgent* eine Mail vom *Serveragenten*, so wird, falls der Benutzer über die Ankunft einer Mail informiert werden möchte, was durch den Aufruf der Methode *ist_Alarm_an()* bestimmt wird, das *AVC - Mail-Alarm* - Fenster auf dem Bildschirm angezeigt. Als Antwort bekommt der *Slave* übergeben, daß die Mail übernommen worden ist. Außerdem beinhaltet diese Antwort den Namen und die Nummer des Benutzers, mit dem die Differenzen festgestellt worden sind, den Namen samt dem relativen Pfad der Datei, bei der die Unterschiede aufgetreten sind, und den Zeitpunkt der Überprüfung. Die letztgenannten Angaben werden dabei aus der übergebenen Nachricht gewonnen. Danach wird die überbrachte Mail im *Mails* - Verzeichnis des Benutzers gespeichert, wobei der Dateiname dieser Mail aus dem Namen der betroffenen Datei und dem Namen und der Nummer des anderen Benutzers besteht. Der Zeitpunkt der Überprüfung wird in der ersten Zeile der

Mail gespeichert. Bekommt jedoch dieser *Clientagent* die Nachricht, daß er sich beseitigen soll, so führt er dieses durch.

Diese Methode gibt zurück, ob die übergebene Nachricht behandelt worden ist.

3. *int read_time_for_check()*

Diese Methode wird aufgerufen, wenn der *ClientPause - Slave* erzeugt werden soll.

Sie liest aus der Datei *Userdaten.txt* die Anzahl an Minuten ein, für die sich der *ClientPause - Slave* deaktivieren soll. Diese Zeit entspricht dem Zeitabstand zwischen zwei regelmäßigen Überprüfungen. Zurückgegeben wird die eingelesene Minutenanzahl.

4. *void onDisposing()*

Wenn dieser *Clientagent* beseitigt werden soll, so wird diese Methode aufgerufen. In ihr wird die durch Methode *clientBeimServerAnmelden(name, number, false)* dieser Agent bei seinem *Serveragenten* abgemeldet. Außerdem wird der *ClientPause - Slave* aktiviert.

5. *boolean ist_Alarm_an()*

Diese Methode wird aufgerufen, wenn der *MailAgent* des *Serveragenten* eine Mail übergeben hat. Sie liest aus der Datei *Userdaten.txt* ein, ob der Benutzer über die Ankunft einer neuen Mail informiert werden möchte. Sollte dies der Fall sein, so gibt sie 'true' zurück. Andernfalls ist der Rückgabewert 'false'.

6. *void clientBeimServerAnmelden(String username, String usernumber, boolean anmelden)*

Diese Methode wird aufgerufen, wenn sich dieser *Clientagent* bei seinem *Serveragenten* an- bzw. abmelden möchte. Sie bekommt den Namen und die Nummer des Benutzers sowie die Information, ob dieser Agent sich an- oder abmelden möchte, übergeben. Sie erzeugt den *ClientServerAnmeldung - Slave* und schickt diesen zu ihrem *Serveragenten*, wobei sie diesem den Namen und die Nummer des Benutzers, die Adresse und die AgentID dieses *Clientagenten* und den Grund für diese Nachricht mitteilt.

7. *void starteÜberprüfung(String username, String usernumber)*

Diese Methode wird aufgerufen, wenn die Dateien bestimmt werden sollen, die zur regelmäßigen Überprüfung von dem *FilePruefAgenten* zum *Serveragenten* gebracht werden müssen. Zuerst wird die Prüfzeit der letzten Überprüfung aus der Datei *Userdaten.txt* eingelesen. Dann wird die Liste der überwachten Dateien aus der Datei *FileListe.txt* bestimmt. Es werden alle überwachten Dateien zum *Serveragenten* gebracht, wenn sich die Liste der überwachten Dateien seit der letzten Überprüfung verändert hat, was dadurch erkannt wird, daß die in der Datei *Userdaten.txt* gespeicherte Prüfzeit gleich '0' ist, oder wenn durch einen *CVS* - Befehl die Datei *Entries* mindestens einer der überwachten Dateien seit der letzten Überprüfung bearbeitet worden ist, was durch Betrachtung der letzten Änderung dieser Datei festgestellt wird. Treffen diese beiden Fälle nicht zu, so werden alle die überwachten Dateien zum *Serveragenten* überbracht, die seit der letzten Überprüfung bearbeitet worden sind. Diese zu überbringenden Dateien werden durch Vergleich des Zeitpunktes ihrer letzten Speicherung mit dem Zeitpunkt der letzten Überprüfung herausgefunden.

Ist bei einer der überwachten Dateien, die zum *Serveragenten* geschickt werden sollen, ein durch den *CVS* - Befehl *update* bzw. *checkout* hervorgerufener Konflikt aufgetreten, so wird diese Datei nicht zum *Serveragenten* gebracht, sondern der Name dieser Datei wird durch Aufruf der Methode *loescheKonflikt()* aus der Liste der überwachten Dateien gelöscht

und das *AVC - Conflict-Alarm* - Fenster wird auf dem Bildschirm angezeigt. Jede Datei, die zur Überprüfung zum *Serveragenten* überbracht werden soll, wird zeilenweise in einen *Vector* eingelesen. Außerdem wird die Versionsnummer dieser Datei bestimmt, indem die Methode *cvsInfo(zupruefendesFile)* aufgerufen wird. Die in einen *Vector* eingelesene Datei samt Versionsnummer und Dateiname einschließlich des relativen Pfades wird in die zu überbringende Liste eingetragen.

Nachdem diese Liste der zu überbringenden Dateien bestimmt worden ist, wird, sofern diese Liste nicht leer ist, der *FilePruefAgent* erzeugt und mit dieser Liste zum *Serveragenten* geschickt. Zusätzlich zu dieser Liste erhält dieser *Slave* auch den Namen und die Nummer des Benutzers, die Prüfzeit, den Zeitpunkt der Erzeugung dieses *Clientagenten* und, ob die Liste der zu überwachenden Dateien sich verändert hat, übergeben.

8. *String cvsInfo(File cvsFile)*

Diese Methode wird aufgerufen, um die *CVS - Versionsnummer* der übergebenen Datei *cvsFile* zu bestimmen und zurückzugeben. Um diese Versionsnummer zu erhalten, wird in der für die übergebene Datei zuständigen *CVS - Datei Entries* nachgeschaut. Dabei steht diese *CVS - Datei* im Verzeichnis *CVS*, das sich in demselben Verzeichnis befindet wie die übergebene Datei.

9. *void loescheKonflikt(String konfliktFile)*

Ein Aufruf dieser Methode erfolgt immer dann, wenn eine Datei zur regelmäßigen Überprüfung zum *Serveragenten* geschickt werden soll, bei der durch die *CVS - Befehle update* bzw. *checkout* ein Konflikt aufgetreten ist. Es werden die überwachten Dateinamen aus der Datei *FileListe.txt* in eine Liste eingelesen. In dieser Liste wird der übergebene Konfliktdateiname gelöscht, indem der letzte Eintrag dieser Liste an dem Platz eingetragen wird, an dem sich der Name der Konfliktdatei befindet. Danach wird die Liste um den letzten Eintrag verkürzt. Zuletzt wird diese Liste als neue Datei *FileListe.txt* gespeichert.

5.3.2 ClientPause

Diese Klasse repräsentiert einen *Slave*, wobei ihre Superklasse die Klasse *Slave* ist. Er wird vom *Clientagenten* erzeugt, der somit sein *Master* ist. Die Aufgabe dieses *Slaves* besteht darin, sich, nachdem er in den *Context* dieses *Clientagenten* losgeschickt worden ist, für eine bestimmte Zeit zu deaktivieren. Dieses Zeitintervall, das den Zeitabstand zwischen zwei Überprüfungen beim *Clientagenten* darstellt, erhält er von seinem *Master* als *Argument* bei seiner Erzeugung übergeben. Nach Ablauf dieser Zeitspanne aktiviert sich der *Slave* wieder und kehrt zu seinem *Master* zurück. Soll der *Clientagent* beseitigt werden oder hat der Benutzer den Zeitabstand zwischen zwei Überprüfungen im *AVC - Management* - Fenster verändert, so wird er schon vor Ablauf des Zeitintervalles aktiviert, um zu seinem *Master* zurückzukehren. Dieser *Clientagent* bekommt dann von ihm nach seiner Rückkehr als *Result* übergeben, daß der Zeitpunkt für eine erneute Überprüfung der überwachten Dateien gekommen ist. Danach beseitigt sich dieser *Slave*.

Diese Klasse enthält die folgenden Methoden:

1. *void initializeJob()*

Die gleichnamige Methode der Superklasse ist *abstract* und muß deshalb überschrieben werden.

2. *void doJob()*

Die gleichnamige Methode der Superklasse ist *abstract* und muß deshalb überschrieben werden.

3. *void onCreate(Object obj)*

Diese Methode wird bei der Initialisierung dieses *Slaves* aufgerufen. Aus dem übergebenen Argument *obj* erhält sie die *AgletID* des *Masters* und den Zeitabstand zwischen zwei regelmäßigen Überprüfungen. Ihre Aufgabe besteht darin, sich für diesen Zeitabstand (in Minuten) zu deaktivieren.

4. *void run()*

Diese Methode wird aufgerufen, wenn dieser *Slave* aktiviert wird. Nach der Aktivierung teilt er seinem *Master* mit, daß der Zeitpunkt für eine erneute regelmäßige Überprüfung der überwachten Dateien gekommen ist. Danach beseitigt sich dieser *Slave*.

5.3.3 ClientServerAnmeldung

Diese Klasse repräsentiert einen *Slave*. Ihre Superklasse ist die Klasse *Slave*. Dieser *Slave* wird vom *Clientagenten* erzeugt, der somit sein *Master* ist, und zum *Server* geschickt, wo sich der *Verwaltungsagent* und die *Serveragenten* befinden. Die Aufgabe dieses *Slaves* besteht darin, seinen *Master* bei dessen *Serveragenten* entweder an- oder abzumelden. Diesem *Serveragenten* übergibt er zu diesem Zweck die Nachricht, die er von dem *Clientagenten* als Argument bei seiner Erzeugung erhalten hat. Sollte er seinen *Master* anmelden, so kehrt an nach Erhalt der Antwort des *Serveragenten* zu diesem zurück und übergibt ihm die vorher erwähnte Antwort. Danach wird er beseitigt. Sollte dieser *Slave* seinen *Master* dagegen abmelden, so wird er beseitigt, sobald er die Antwort von dem *Serveragenten* erhalten hat und in den *Context* seines *Masters* zurückgekehrt ist. Falls dieser *Slave* den *Verwaltungsagenten* oder den *Serveragenten* nicht finden kann, so kehrt er in den *Context* seines *Masters* zurück. Hatte er die Aufgabe, seinen Erzeuger anzumelden, so benachrichtigt er diesen über das aufgetretene Problem, bevor er beseitigt wird. Bestand die Aufgabe in der Abmeldung, so wird er nach seiner Rückkehr sofort beseitigt.

Folgende Methode sind in dieser Klasse enthalten:

1. *void initializeJob()*

Die gleichnamige Methode der Superklasse ist *abstract* und muß deshalb überschrieben werden.

2. *void doJob()*

Die gleichnamige Methode der Superklasse ist *abstract* und muß deshalb überschrieben werden.

3. *boolean handleMessage(Message msge)*

Diese Methode wird aufgerufen, wenn dieser *Slave* die Nachricht (*msge*) bekommt, daß er im *Context* des *Verwaltungsagenten* und der *Serveragenten* oder nach seiner Rückkehr im *Context* seines *Masters* angekommen ist. Erhält er als erstes die Nachricht *doJob*, so ist er in dem *Context* des *Verwaltungsagenten* angekommen und sucht dort nach dem *Proxy* des *Verwaltungsagenten*. Wird dieser gefunden, so erhält der *Verwaltungsagent* mit dessen Hilfe die Nachricht des *Clientagenten*. Die Antwort des *Verwaltungsagenten* enthält entweder die *AgletID* des gesuchten *Serveragenten* oder, falls der *Serveragent* nicht

vorhanden ist, die *AgletID* des *Verwaltungsagenten*. Ist dabei der *Serveragent* nicht existent, so enthält die Antwort für den *Master* dieses aufgetretene Problem. Ist dagegen der gesuchte *Serveragent* präsent, so wird ihm mit Hilfe seines *Proxy* die Nachricht des *Clientagenten* überbracht. Die Antwort dieses *Serveragenten* darauf ist die Nachricht für den *Master* dieses *Slaves*. Ist allerdings der Fall eingetreten, daß der *Proxy* des *Verwaltungsagenten* nicht gefunden werden konnte, so beinhaltet die Antwort dieses *Slaves* für seinen *Master* dieses Problem. Erhält dagegen dieser *Slave* die Nachricht *onReturn*, so ist er in den *Context* seines *Masters* zurückgekehrt. Sollte sein *Master* beim *Serveragenten* angemeldet werden, so übergibt er seinem *Master* durch dessen *Proxy* die Nachricht, daß die Anmeldung erfolgt ist oder daß er den *Verwaltungsagenten* oder den *Serveragenten* nicht gefunden hat. Danach wird er beseitigt. Bestand die Aufgabe in der Abmeldung seines *Clientagenten*, so wird dieser *Slave* ungeachtet der Antwort beseitigt.

5.3.4 ClientStart

Diese Klasse repräsentiert einen stationären Agenten, wobei ihre Superklasse die Klasse *Aglet* ist. Dieser Agent stellt dem Benutzer die Benutzeroberfläche zur Verfügung, indem er nach seiner Erzeugung durch den Benutzer das *AVC - Main-Menu* - Fenster auf dem Bildschirm anzeigt. Außerdem liest dieser Agent den Namen und die Nummer des Benutzers aus der Datei *User.txt* ein. Wird im *AVC - File-Menu* - Fenster der *Test* - Button angeklickt und der *FileAgent* erzeugt, so ist dieser Agent der *Master* für diesen *FileAgenten*. Nach dessen Rückkehr erhält dieser Agent das Ergebnis dessen Auftrages als Antwort übergeben. Dieser Agent wird durch das Anklicken des *Close* - Buttons im *AVC - Main-Menu* - Fenster beseitigt. Er wird außerdem durch seinen *Clientagenten* beseitigt, wenn ein von diesem *Clientagenten* erzeugter *Slave* den *Verwaltungsagenten* oder den *Serveragenten* nicht finden kann.

Diese Klasse besitzt die folgenden Methoden:

1. *void onCreate(Object o)*

Diese Methode wird bei der Initialisierung dieses Agenten aufgerufen. Zuerst wird überprüft, ob der *Clientagent* in diesem *Context* vorhanden ist. Dann wird durch zweimaliges Aufrufen der Methode *userDaten(...)* der Name und die Nummer des Benutzers bestimmt. Im Anschluß daran wird das *AVC - Main-Menu* - Fenster auf dem Bildschirm angezeigt, indem ein Objekt der Klasse *HauptmenuAnzeigen* erzeugt wird. Dem Objekt wird dabei der Name und die Nummer des Benutzers, dieser *Clientstart* - Agent und die Information, ob der *Clientagent* vorhanden ist oder nicht, übergeben.

2. *String userDaten(String was)*

Diese Methode liest abhängig von dem übergebenen String *was* entweder den Namen oder die Nummer des Benutzers aus der Datei *User.txt* ein und gibt diesen bzw. diese zurück. Tritt beim Einlesen der gesuchten Angabe aus der gerade genannten Datei ein Fehler auf, so wird dieser *Clientstart* - Agent beseitigt.

3. *void onDisposing()*

Diese Methode wird aufgerufen, wenn dieser Agent beseitigt werden soll. Ihre Aufgabe besteht darin, das *AVC - Main-Menu* - Fenster zu beseitigen.

4. *boolean handleMessage(Message msg)*

Diese Methode wird aufgerufen, wenn der *FileAgent* in den *Context* seines *Masters* (dieser

Agent) zurückkehrt und diesem Agenten die Nachricht des *Serveragenten* oder die Mitteilung, daß der *Verwaltungsagent* oder der entsprechende *Serveragent* nicht gefunden wurde, übergibt. Diese Nachricht wird dann auf dem Bildschirm angezeigt.

5.3.5 DiffAgent

Diese Klasse stellt einen *Slave* dar. Ihre Superklasse ist dabei die Klasse *Slave*. Dieser *Slave* wird vom *Serveragenten* erzeugt, der somit sein *Master* ist, und zum *Verwaltungsagenten* geschickt. Diese Erzeugung geschieht jedesmal, wenn sein *Master* eine bestimmte Datei überprüfen soll und hierfür die Dateiversionen der anderen *Serveragenten* benötigt, die auch diese Datei überwachen. Die Aufgabe dieses *Slaves* besteht nun darin, zum *Verwaltungsagenten* zu gehen und von diesem die *AgletIDs* aller *Serveragenten* zu erhalten, die auch diese Datei überwachen. Nachdem die *AgletIDs* vom *Verwaltungsagenten* als Antwort zurückgegeben worden sind, werden der Reihe nach die *Proxies* zu diesen *AgletIDs* bestimmt und damit beauftragt, dem zugehörigen *Serveragenten* die Nachricht des *Masters* dieses *Slaves* zu überbringen, um die entsprechenden Dateiversionen samt einiger weiterer Informationen der benachrichtigten *Serveragenten* zu erhalten. Hiervon wird natürlich der *Proxy* des *Masters* ausgenommen. Jede Antwort dieser *Serveragenten* wird, sofern sie nicht leer ist, in die Liste der Prüfdateien dieses *Slaves* eingefügt.

Danach kehrt dieser *DiffAgent* zu seinem *Master* zurück und übergibt ihm die Liste der Prüfdateien, sofern diese nicht leer ist. Sind dagegen keine Prüfdateien der anderen *Serveragenten* in dieser Liste vorhanden, so wird dieses dem *Master* als Antwort mitgeteilt. Ist der *Verwaltungsagent* nicht vorhanden oder gibt er keine *AgletIDs* zurück, so informiert dieser *Slave* seinen Erzeuger im Rahmen der zu überbringenden Antwort darüber.

Diese Klasse enthält folgende Methoden:

1. *void initializeJob()*

Die gleichnamige Methode der Superklasse ist *abstract* und muß deshalb überschrieben werden.

2. *void doJob()*

Die gleichnamige Methode der Superklasse ist *abstract* und muß deshalb überschrieben werden.

3. *boolean handleMessage(Message msge)*

Diese Methode wird aufgerufen, wenn dieser *Slave* die Nachricht (*msge*) bekommt, daß er im *Context* des *Verwaltungsagenten* oder nach seiner Rückkehr im *Context* seines *Masters* angekommen ist. Erhält er als erstes die Nachricht *doJob*, so befindet er sich in dem *Context* des *Verwaltungsagenten*. Er sucht dann dort nach dem *Proxy* des *Verwaltungsagenten*. Wird dabei dieser *Proxy* nicht gefunden, so enthält die Antwort dieses *Slaves* für seinen *Master* dieses Problem. Wird dagegen der gesuchte *Proxy* gefunden, so bekommt der *Verwaltungsagent* mit Hilfe dieses *Proxy* die Nachricht des *Serveragenten* übergeben. Als Antwort des *Verwaltungsagenten* erhält dieser *Slave* die Liste aller *AgletIDs* der *Serveragenten*, die auch diese Datei überwachen. Tritt hierbei der Fall ein, daß keine *AgletIDs* zurückgegeben werden, so wird der *Master* darüber im Rahmen der Antwort dieses *Slaves* bei dessen Rückkehr informiert. Sind allerdings *AgletIDs* in der Antwort des *Verwaltungsagenten* vorhanden, so wird für jede dieser *AgletIDs*, außer der des *Masters*, der zugehörige *Proxy* bestimmt. Dem *Serveragenten* jedes vorher bestimmten *Proxy* wird dann die Nach-

richt des *Masters* überbracht. Falls die Antwort dieses *Serveragenten* nicht leer ist, so wird diese in die Liste der Antworten der *Serveragenten* eingefügt, wobei es sich bei dieser Antwort um die zu überprüfende Datei (einschließlich der nötigen Zusatzinformationen) dieses *Serveragenten* handelt.

Nachdem alle befragten *Serveragenten* geantwortet haben, besteht die Antwort für den *Master* entweder aus der Mitteilung, daß es, außer der eigenen, keine weiteren Dateiversionen anderer *Serveragenten* gibt, was daran erkannt wird, daß die Liste der Antworten leer ist, oder aus der Liste der Dateiversionen samt der Zusatzinformationen der anderen *Serveragenten*. Im letzten Fall enthält die Antwort für den *Master* zusätzlich noch die eigene Dateiversion, wenn dieser *DiffAgent* aufgrund der durch das Anklicken des Buttons *Test* im *AVC - File-Menu* - Fenster gestarteten Überprüfung erzeugt worden ist.

Wenn dieser *Slave* die Nachricht *onReturn* übergeben bekommt, so ist er in den *Context* seines *Masters* zurückgekehrt. Er übergibt seinem *Master* durch dessen *Proxy* die vorher bestimmte Antwort, die also entweder aus der Liste der Dateiversionen samt Zusatzinformationen oder aus der Mitteilung besteht, daß er den *Verwaltungsagenten* nicht finden konnte oder daß, außer der eigenen, keine Dateiversionen anderer *Serveragenten* existieren. Danach wird er beseitigt.

5.3.6 FileAgent

Diese Klasse repräsentiert einen *Slave*. Ihre Superklasse ist die Klasse *Slave*. Dieser *Slave* wird durch Anklicken des *Test* - Buttons im *AVC - File-Menu* - Fenster erzeugt. Sein *Master* ist der *Clientstart* - Agent. Seine Aufgabe besteht darin, die im *AVC - File-Menu* - Fenster markierte Datei zum *Serveragenten* zur Überprüfung zu bringen. Diese Datei einschließlich weiterer Informationen bekommt er als Argument bei seiner Erzeugung übergeben. Nachdem er die Antwort dieses *Serveragenten* erhalten hat, kehrt er zu seinem *Master* zurück, um ihm diese Antwort zu übergeben, bevor er beseitigt wird.

Falls dieser *Slave* den *Verwaltungsagenten* oder den *Serveragenten* nicht finden kann, so kehrt er zu seinem *Master* zurück und informiert diesen vor seiner Beseitigung darüber.

Diese Klasse enthält die folgenden Methoden:

1. *void initializeJob()*

Die gleichnamige Methode der Superklasse ist *abstract* und muß deshalb überschrieben werden.

2. *void doJob()*

Die gleichnamige Methode der Superklasse ist *abstract* und muß deshalb überschrieben werden.

3. *boolean handleMessage(Message msge)*

Diese Methode wird aufgerufen, wenn dieser *Slave* die Nachricht (*msge*) bekommt, daß er im *Context* des *Verwaltungsagenten* und der *Serveragenten* oder nach seiner Rückkehr im *Context* seines *Masters* angekommen ist. Erhält er als erstes die Nachricht *doJob*, so ist er in dem *Context* des *Verwaltungsagenten* angekommen und sucht dort nach dem *Proxy* des *Verwaltungsagenten*. Wird dieser gefunden, so erhält der *Verwaltungsagent* mit dessen Hilfe die Nachricht des *Clientstart* - Agenten. Die Antwort des *Verwaltungsagenten* enthält entweder die *AgletID* des gesuchten *Serveragenten* oder, falls der *Serveragent*

nicht vorhanden ist, die *AgletID* des *Verwaltungsagenten*. Ist dabei der *Serveragent* nicht existent, so enthält die Antwort für den *Master* dieses aufgetretene Problem. Ist dagegen der gesuchte *Serveragent* präsent, so wird ihm mit Hilfe seines *Proxy* die Nachricht des *Clientstart* - Agenten überbracht. Die Antwort dieses *Serveragenten* darauf ist die Nachricht für den *Master* dieses *Slaves*. Ist allerdings der Fall eingetreten, daß der *Proxy* des *Verwaltungsagenten* nicht gefunden werden konnte, so beinhaltet die Antwort dieses *Slaves* für seinen *Master* dieses Problem. Erhält dieser *Slave* die Nachricht *onReturn*, so ist er in den *Context* seines *Masters* zurückgekehrt und übergibt seinem *Master* durch dessen *Proxy* die Nachricht, daß er die Datei überbracht hat oder daß er den *Verwaltungsagenten* oder den *Serveragenten* nicht gefunden hat. Danach wird er beseitigt.

5.3.7 FilenamenAgent

Diese Klasse repräsentiert einen *Slave*, wobei die Klasse *Slave* ihre Superklasse ist. Er wird vom *Serveragenten* erzeugt, der somit sein *Master* ist, und zum *Verwaltungsagenten* geschickt. Dies geschieht immer dann, wenn der Benutzer die Liste der zu überprüfenden Dateien im *AVC - Choice-Menu* - Fenster oder durch einen *CVS* - Befehl verändert und der *FilePruefAgent* den *Serveragenten* im Rahmen der regelmäßigen Überprüfung darüber informiert hat. Die Aufgabe dieses *Slaves* besteht darin, dem *Verwaltungsagenten* die *AgletID* seines *Serveragenten* und die Liste aller Dateien zu übergeben, die der Benutzer überwachen lassen möchte. Diese *AgletID* und diese Liste bekommt der *Slave* von seinem *Master* bei seiner Erzeugung als Argument übergeben. Nach Erhalt der Antwort von dem *Verwaltungsagenten* kehrt er zu seinem *Master* zurück und übergibt diesem diese Antwort. Danach wird er beseitigt. Falls dieser *Slave* den *Verwaltungsagenten* nicht finden kann, so kehrt er zu seinem Erzeuger zurück und informiert diesen darüber, bevor er beseitigt wird.

Diese Klasse beinhaltet die folgenden Methoden:

1. *void initializeJob()*

Die gleichnamige Methode der Superklasse ist *abstract* und muß deshalb überschrieben werden.

2. *void doJob()*

Die gleichnamige Methode der Superklasse ist *abstract* und muß deshalb überschrieben werden.

3. *boolean handleMessage(Message msge)*

Diese Methode wird aufgerufen, wenn dieser *Slave* die Nachricht (*msge*) bekommt, daß er im *Context* des *Verwaltungsagenten* oder nach seiner Rückkehr im *Context* seines *Masters* angekommen ist. Bekommt er als erstes die Nachricht *doJob*, so ist er in den *Context* des *Verwaltungsagenten* angekommen und sucht dort nach dem *Proxy* des *Verwaltungsagenten*. Wird dieser gefunden, so erhält der *Verwaltungsagent* mit dessen Hilfe die Nachricht des *Serveragenten*. Die Antwort des *Verwaltungsagenten* darauf ist die Nachricht für den *Master* dieses *Slaves*. Ist allerdings der Fall eingetreten, daß der *Proxy* des *Verwaltungsagenten* nicht gefunden werden konnte, so beinhaltet die Antwort dieses *Slaves* für seinen *Master* das aufgetretene Problem. Erhält dieser *Slave* die Nachricht *onReturn*, so ist er in dem *Context* seines *Masters* zurückgekehrt und übergibt seinem *Master* durch dessen *Proxy* die Nachricht, daß der *Verwaltungsagent* auf den neusten Stand bezüglich der zu überwachenden Dateien gebracht worden ist oder daß er den *Verwaltungsagenten* nicht gefunden hat. Danach wird er beseitigt.

5.3.8 FilePruefAgent

Diese Klasse repräsentiert einen *Slave*. Ihre Superklasse ist die Klasse *Slave*. Dieser *Slave* wird vom *Clientagenten* erzeugt, der somit sein *Master* ist, und zum *Server* geschickt, wo sich der *Verwaltungsagent* und die *Serveragenten* befinden. Seine Aufgabe besteht darin, die Dateien, die er als Argument einschließlich weiterer Informationen bei seiner Erzeugung übergeben bekommen hat, zu dem *Serveragenten* seines *Masters* zu bringen, damit diese dort überprüft werden können. Nachdem er die Antwort dieses *Serveragenten* erhalten hat, kehrt er zu seinem *Master* zurück, um ihm diese Antwort zu übergeben, bevor er beseitigt wird. Falls dieser *Slave* den *Verwaltungsagenten* oder den *Serveragenten* nicht finden kann, so kehrt er zu seinem *Master* zurück und informiert diesen vor seiner Beseitigung darüber.

Diese Klasse enthält die folgenden Methoden:

1. *void initializeJob()*

Die gleichnamige Methode der Superklasse ist *abstract* und muß deshalb überschrieben werden.

2. *void doJob()*

Die gleichnamige Methode der Superklasse ist *abstract* und muß deshalb überschrieben werden.

3. *boolean handleMessage(Message msge)*

Diese Methode wird aufgerufen, wenn dieser *Slave* die Nachricht (*msge*) bekommt, daß er im *Context* des *Verwaltungsagenten* und der *Serveragenten* oder nach seiner Rückkehr im *Context* seines *Masters* angekommen ist. Erhält er als erstes die Nachricht *doJob*, so ist er in dem *Context* des *Verwaltungsagenten* angekommen und sucht dort nach dem *Proxy* des *Verwaltungsagenten*. Wird dieser gefunden, so erhält der *Verwaltungsagent* mit dessen Hilfe die Nachricht des *Clientagenten*. Die Antwort des *Verwaltungsagenten* enthält entweder die *AgletID* des gesuchten *Serveragenten* oder, falls der *Serveragent* nicht vorhanden ist, die *AgletID* des *Verwaltungsagenten*. Ist dabei der *Serveragent* nicht existent, so enthält die Antwort für den *Master* dieses aufgetretene Problem. Ist dagegen der gesuchte *Serveragent* präsent, so wird ihm mit Hilfe seines *Proxy* die Nachricht des *Clientagenten* überbracht. Die Antwort dieses *Serveragenten* darauf ist die Nachricht für den *Master* dieses *Slaves*. Ist allerdings der Fall eingetreten, daß der *Proxy* des *Verwaltungsagenten* nicht gefunden werden konnte, so beinhaltet die Antwort dieses *Slaves* für seinen *Master* dieses Problem. Erhält dieser *Slave* die Nachricht *onReturn*, so ist er in den *Context* seines *Masters* zurückgekehrt und übergibt seinem *Master* durch dessen *Proxy* die Nachricht des *Serveragenten* oder, daß er den *Verwaltungsagenten* oder den *Serveragenten* nicht finden konnte. Danach wird er beseitigt.

5.3.9 MailAgent

Diese Klasse, die als Superklasse die Klasse *Slave* besitzt, repräsentiert einen *Slave*. Er wird vom *Serveragenten* erzeugt, der somit sein *Master* ist, und zum zugehörigen *Clientagenten* dieses *Serveragenten* geschickt. Seine Aufgabe besteht darin, die Mail, die er als Argument bei seiner Erzeugung übergeben bekommen hat, zu dem vorher genannten *Clientagenten* zu bringen. Nachdem er die Antwort dieses *Clientagenten* erhalten hat, kehrt er zu seinem *Master* zurück, um ihm diese Antwort zu übergeben, bevor er beseitigt wird. Falls dieser *Slave* den *Clientagenten* nicht findet, so kehrt er zu seinem *Master* zurück und informiert

diesen vor seiner Beseitigung darüber.

Als Methoden enthält diese Klasse folgende:

1. *void initializeJob()*

Die gleichnamige Methode der Superklasse ist *abstract* und muß deshalb überschrieben werden.

2. *void doJob()*

Die gleichnamige Methode der Superklasse ist *abstract* und muß deshalb überschrieben werden.

3. *boolean handleMessage(Message msge)*

Diese Methode wird aufgerufen, wenn dieser *Slave* die Nachricht (*msge*) bekommt, daß er im *Context* des *Clientagenten* oder nach seiner Rückkehr im *Context* seines *Masters* angekommen ist. Erhält er als erstes die Nachricht *doJob*, so ist er in dem *Context* des *Clientagenten* angekommen und sucht dort nach dem *Proxy* des *Clientagenten*. Wird dieser gefunden, so erhält der *Clientagent* mit dessen Hilfe die Nachricht seines *Serveragenten*. Die Antwort dieses *Clientagenten* darauf ist die Nachricht für den *Master* dieses *Slaves*. Ist allerdings der Fall eingetreten, daß der *Proxy* des *Clientagenten* nicht gefunden werden konnte, so beinhaltet die Antwort dieses *Slaves* für seinen *Master* das Nichtvorhandensein des *Clientagenten*. Bekommt dieser *Slave* dagegen die Nachricht *onReturn*, so ist er in den *Context* seines *Masters* zurückgekehrt und übergibt seinem *Master* durch dessen *Proxy* die Antwort des *Clientagenten* oder die Nachricht, daß er den *Clientagenten* nicht finden konnte. Danach wird er beseitigt.

5.3.10 Serveragent

Diese Klasse repräsentiert einen stationären Agenten. Ihre Superklasse ist die Klasse *Aglet*. Dieser Agent ist der serverseitige Benutzeragent. Pro Benutzer existiert ein solcher *Serveragent*. Er wird entweder durch den *Serverstart* - Agenten während der Erzeugung des *Verwaltungsagenten* und der anderen *Serveragenten* oder, wenn ein neuer Benutzer während des Programmablaufes hinzugefügt werden soll, durch den *ServeragentStart* - Agenten erzeugt. Seine Lebenszeit endet, wenn der *Verwaltungsagent* beseitigt wird. Nach seiner Erzeugung meldet sich dieser Agent durch Aussendung des *ServerAnmeldung* - *Slaves* beim *Verwaltungsagenten* an. Tritt hierbei der Fall ein, daß dieser *Slave* den *Verwaltungsagenten* nicht finden kann, so teilt er das diesem Agenten mit, der sich daraufhin beseitigt. Während der Beseitigung meldet er sich ebenfalls durch Aussendung des *ServerAnmeldung* - *Slaves* beim *Verwaltungsagenten* ab.

Bekommt er von dem *ClientServer-Anmeldung* - *Slave* die Nachricht, daß sein *Clientagent* vorhanden ist, so speichert er dessen Adresse und *AgletID*. Außerdem schickt er die Mails zu seinem *Clientagenten*, die er noch nicht zu diesem schicken konnte und deshalb gespeichert hatte. Erhält er dagegen von dem vorher erwähnten *Slave* die Nachricht, daß sein *Clientagent* nicht mehr vorhanden ist, so löscht er dessen Adresse und *AgletID*.

Wenn er vom *FileAgenten* die Nachricht bekommt, eine durch Anklicken des *Test* - Buttons im *AVC* - *File-Menu* - Fenster ausgewählte Datei zu überprüfen, so erzeugt er den *DiffAgenten*, der von den anderen *Serveragenten*, die auch diese Datei überwachen, deren Versionen dieser Datei holt. Nachdem der *DiffAgent* alle Versionen dieser Datei erhalten hat, kehrt er zu diesem *Serveragenten* zurück, der dann die Überprüfung anhand der übergebenen Dateien

durchführt, soweit sie aufgrund der einzelnen Versionsnummern zulässig ist. Dabei wird die eigene Version mit jeder Version der anderen *Serveragenten* verglichen. Sollten bei dieser Überprüfung Unterschiede zwischen der eigenen Datei und der Datei eines anderen Benutzers festgestellt werden, so werden diese in einer Mail gespeichert. Bei Vorhandensein des *Clientagenten* wird jede so entstandene Mail durch jeweils einen *MailAgenten* überbracht. Ist der *Clientagent* nicht vorhanden, so werden diese Mails auf dem *Server* im benutzereigenen Verzeichnis *savedMails* gespeichert.

Wenn er vom *FilePruefAgenten* die Nachricht bekommt, die überbrachten Dateien zu überprüfen, so schickt er pro Datei einen *DiffAgenten* los, der die Versionen dieser Datei von den anderen *Serveragenten* holt. Nach der Rückkehr eines *DiffAgenten* wird die eigene Datei mit den Versionen der anderen *Serveragenten* verglichen, sofern diese Überprüfung im Hinblick auf die einzelnen Versionsnummern erlaubt ist. Im Falle von Unterschieden wird die dadurch bedingte Mail entweder dem *Clientagenten* durch den *MailAgenten* überbracht, falls der *Clientagent* existiert, oder bei diesem *Serveragenten* im Verzeichnis *savedMails* gespeichert. Hat sich die Liste der überwachten Dateien beim *Client* verändert, so erzeugt der *Serveragent* den *FilenamenAgenten*, der dann beim *Verwaltungsagenten* für einen Abgleich sorgt. Erhält dieser Agent die Nachricht von einem *DiffAgenten* eines anderen *Serveragenten*, daß er die Version einer bestimmten Datei benötigt, so überreicht er diesem die angeforderte Datei.

Diese Klasse beinhaltet die folgenden Methoden:

1. *void onCreate(Object o)*

Diese Methode wird bei der Initialisierung dieses Agenten aufgerufen. In ihr wird aus dem übergebenen Argument *o* der Name und die Nummer des Benutzers bestimmt. Außerdem soll durch Aufruf der Methode *serveragentAnmelden(name, number, true)* dieser Agent beim *Verwaltungsagenten* angemeldet werden. Schlägt diese Anmeldung fehl, so beseitigt sich dieser *Serveragent*. Zuletzt werden die Adresse und die *AgletID* des *Clientagenten* mit 'null' belegt.

2. *boolean handleMessage(Message msg)*

Übergibt ein *Slave* eines anderen Agenten oder ein eigener *Slave* eine Nachricht an diesen *Serveragenten*, so wird diese Methode aufgerufen. Soll zum Beispiel sein *Clientagent* angemeldet werden, so überbringt dessen *ClientServerAnmeldung - Slave* die entsprechende Nachricht. Aus der Nachricht werden die *URL* - Adresse und die *AgletID* des *Clientagenten* erhalten und gespeichert. Dem *Slave* wird als Rückmeldung übergeben, daß der *Clientagent* angemeldet ist. Dann wird die Methode *sendSavedMails()* aufgerufen, um die eventuell gespeicherten Mails an den *Clientagenten* zu schicken. Überbringt dagegen der *ClientServerAnmeldung - Slave* die Nachricht, daß der *Clientagent* dieses *Serveragenten* abgemeldet werden soll, so werden die *URL* - Adresse und die *AgletID* des *Clientagenten* gelöscht. Dem *Slave* wird als Rückmeldung übergeben, daß der *Clientagent* abgemeldet ist.

Wird allerdings im *AVC - File-Menu* - Fenster der Button *Test* angeklickt, so wird ein *FileAgent* zu diesem *Serveragenten* geschickt. Diesem wird dann als Nachricht überbracht, daß die ebenfalls in der Botschaft enthaltene Datei überprüft werden soll. Als Rückmeldung wird übergeben, daß diese Überprüfung durchgeführt wird. Dann wird die Methode *holeFilesVonServeragenten(...)* aufgerufen, der als Parameter die folgenden Angaben, die aus der überbrachten Nachricht gewonnen werden, übergeben werden: der Pfadname der Datei, die *CVS* - Versionsnummer der Datei und die Datei selbst.

Wenn statt dessen bei der regelmäßigen Überprüfung beim *Clientagenten* festgestellt worden ist, daß mindestens eine Datei seit der letzten regelmäßigen Überprüfung bearbeitet worden ist oder daß sich die Liste der überwachten Dateien verändert hat, so wird der *FilePruefAgent* zu diesem *Serveragenten* geschickt. Die überbrachte Nachricht besteht aus dem Zeitpunkt der *Clientagent* - Erzeugung, aus der Prüfzeit, aus der Information, ob sich die Liste der überwachten Dateien verändert hat, aus der Liste der zu überprüfenden Dateien einschließlich des Dateipfades und der *CVS* - Dateiversionsnummer. Zuerst wird der Zeitpunkt der Erzeugung des *Clientagenten* mit demjenigen dieses *Serveragenten* verglichen. Ist der *Clientagent* vor dem *Serveragenten* erzeugt worden, so erhält der *Slave* als Antwort, daß sein *Master* sich neu anmelden muß, nachdem die vorher gespeicherte Informationen bezüglich der *URL* - Adresse und der *AgletID* gelöscht worden sind. Ist der *Serveragent* vor dem *Clientagenten* erzeugt worden, so erhält der *Slave* als Antwort die Prüfzeit und, daß die Dateien überprüft werden.

In beiden Fällen werden dann, wenn sich die Liste der überwachten Dateien verändert hat, alle bei diesem *Serveragenten* gespeicherten Dateien des Benutzers durch Aufruf der Methode *loescheFiles()* gelöscht und es wird der *FilenamenAgent* erzeugt und zum *Verwaltungsagenten* geschickt, um die Liste der neu überwachten Dateien und die *AgletID* dieses *Serveragenten* zu überbringen, um einen Abgleich zwischen *Verwaltungsagent* und *Serveragent* zu erreichen. Danach werden unabhängig davon, ob sich die Liste der überwachten Dateien verändert hat oder nicht, die vom *FilePruefAgenten* überbrachten Dateien samt Zusatzinformationen bei diesem Agenten durch pro Datei einmaliges Aufrufen der Methode *speichereFile(filename , fileInfo , pruefFile)* gespeichert. Zuletzt wird die Methode *holeFilesVonServeragenten(...)* für jede zu überprüfende Datei aufgerufen, um die Versionen von den anderen, auch die gleiche Datei überwachenden *Serveragenten* zu bekommen. Ihr wird dabei jeweils der Pfadname der Datei übergeben.

Benötigt der *DiffAgent* eines anderen *Serveragenten* die hier gespeicherte Version einer Datei, so wird durch Aufruf der Methode *holeFile(fileName)* die gesuchte Datei einschließlich der *CVS* - Versionsnummer erhalten. Dem *DiffAgenten* wird dann als Antwort diese Datei samt dem Dateinamen und der *CVS* - Versionsnummer und der Name und die Nummer dieses Benutzers übergeben.

Keht ein *Slave*, der von diesem *Serveragenten* erzeugt worden ist, zurück und überbringt die Nachricht, daß der *Verwaltungsagent* nicht gefunden werden konnte, so beseitigt sich dieser *Serveragent*. Kommt dagegen der *ServerAnmeldung - Slave* dieses Agenten zurück und teilt seinem *Master* mit, daß er beim *Verwaltungsagenten* angemeldet ist, so wird diese Nachricht auf dem Bildschirm angezeigt. Kehrt in einem anderen Fall der *FilenamenAgent* zu diesem *Serveragenten* zurück, der als dessen *Master* fungiert, und übergibt die Botschaft, daß das in Auftrag gegebene Update beim *Verwaltungsagenten* durchgeführt worden ist, so wird diese Botschaft ebenfalls auf dem Bildschirm angezeigt. Diese Methode wird auch aufgerufen, wenn der vorher erzeugte und losgeschickte *MailAgent* zurückkehrt und mitteilt, daß er den *Clientagenten* nicht gefunden hat. In diesem Fall werden dann die *AgletID* und die *URL* - Adresse des *Clientagenten* gelöscht.

Es besteht auch die Möglichkeit, daß der *DiffAgent* zu seinem *Master*, nämlich zu diesem *Serveragenten*, zurückkommt und ihn darüber informiert, daß, außer der eigenen, keine Versionen einer zu überprüfenden Datei anderer *Serveragenten* existieren. Diese Information wird dann auf dem Bildschirm angezeigt. Kehrt der eigene *DiffAgent* zurück und übergibt die Dateiversionen der anderen *Serveragenten*, so wird die eigene Version mit

jeder Version der anderen *Serveragenten* verglichen.

Dazu wird die eigene Version als *File1.txt* gespeichert. Die eigene Datei samt Zusatzinformationen wird dabei entweder bei der durch Anklicken des *Test* - Buttons im *AVC - File-Menu* - Fenster gestarteten Überprüfung aus der übergebenen Nachricht bestimmt oder bei der regelmäßigen Überprüfung durch Aufruf der *holeFile(filename)* - Methode erhalten. Danach wird jede der anderen Versionen erst als *File2.txt* gespeichert, sofern nach Vergleich der Versionsnummer dieser Datei mit derjenigen der eigenen Datei eine Überprüfung erlaubt ist, und dann mit der eigenen Datei *File1.txt* verglichen, bevor sie wieder gelöscht wird. Der Vergleich der Versionsnummern auf Überprüfbarkeit der Dateien wird durch Aufruf der Methode *sindDiffbar(...)* untersucht. Die *sindDiffbar(...)* - Methode benötigt, um ihre Aufgabe zu erfüllen, vier Argumente: die Versionsnummer der eigenen Dateiversion und die der Dateiversion, mit der die eigene verglichen werden soll, und die Anzahl an Punkten sowohl in der einen als in der anderen Versionsnummer, was durch zweimaliges (pro Versionsnummer einmal) Aufrufen der *getAnzahlPunkte(...)* - Methode erreicht wird.

Sind bei der oben erwähnten Untersuchung Unterschiede festgestellt worden, so werden diese in einer Mail im Verzeichnis *savedMails* des Benutzers gespeichert. Der Dateiname dieser Mail besteht dabei aus dem Dateinamen der untersuchten Datei gefolgt von in runden Klammern eingeschlossenen Namen und Nummer des Benutzers, mit dessen Dateiversion diese Unterschiede entdeckt worden sind. Die Mail beinhaltet als erste Zeile den Zeitpunkt dieser Überprüfung und als Rest die Unterschiede an sich. Ist der *Clientagent* vorhanden, so wird ihm diese Mail durch Aufruf der Methode *schickeMailZumClient(...)* überbracht. Am Ende wird die Datei *File1.txt* wieder gelöscht.

Diese Methode *handleMessage(...)* wird ebenfalls aufgerufen, wenn der *MailAgent* dieses *Serveragenten* zu seinem *Master* zurückkommt und die Nachricht übergibt, daß er die Mail beim *Clientagenten* abgegeben hat. Dieselbe Mail wird dann im *savedMails* - Verzeichnis dieses *Serveragenten* gelöscht. Die gespeicherte Mail wird allerdings nur gelöscht, wenn der Zeitstempel der gespeicherten Mail entweder mit dem Zeitstempel der überbrachten Mail übereinstimmt oder vor diesem liegt. Andernfalls sind die gespeicherten Unterschiede später als die überbrachten Unterschiede aufgetreten, weshalb dann durch Aufruf der Methode *schickeMailZumClient(...)* die gespeicherte Mail zum *Clientagenten* gebracht wird.

Diese Methode *handleMessage(...)* gibt zurück, ob die übergeben bekommene Nachricht behandelt worden ist.

3. *boolean severagentAnmelden(String username , String usernumber , boolean anmelden)*
Diese Methode wird aufgerufen, wenn dieser *Serveragent* beim *Verwaltungsagenten* entweder an- oder abgemeldet werden soll. Übergeben bekommt diese Methode den Benutzernamen, die Benutzernummer und die Information, ob dieser *Serveragent* an- oder abgemeldet werden soll. Sowohl bei der An- als auch bei der Abmeldung wird der *ServerAnmeldung - Slave* erzeugt und zum *Verwaltungsagenten* geschickt. Bei der Anmeldung überbringt dieser *Slave* den Benutzernamen, die Benutzernummer, die *AgletID* dieses *Serveragenten* und die Liste der Dateinamen der überwachten Dateien, die diese Methode durch Aufruf der Methode *getFileListe()* erhalten hat. Bei der Abmeldung dagegen wird dem *Verwaltungsagenten* lediglich der Benutzername, die Benutzernummer und die *AgletID* dieses *Serveragenten* übergeben.

4. *void onDisposing()*

Soll dieser *Serveragent* beseitigt werden, so meldet er sich durch Aufruf der Methode *serveragentAnmelden(...)* beim *Verwaltungsagenten* ab.

5. *void sendSavedMails()*

Diese Methode wird aufgerufen, wenn sich der *Clientagent* bei diesem *Serveragenten* angemeldet hat, um dem *Clientagenten* die in seiner Abwesenheit gespeicherten Mails zu überbringen. Zu diesem Zweck wird bei jeder zu überbringenden Mail aus dem Dateinamen dieser gespeicherten Mail der Name und die Nummer des Prüfungspartners festgestellt. Dann wird aus der Mail die Prüfzeit eingelesen, die in der ersten Zeile gespeichert ist. Danach wird der Rest eingelesen, der die Unterschiede an sich beschreibt. Durch Aufruf der Methode *schickeMailZumClient(...)* wird die vorher eingelesene Mail samt der Zusatzinformationen zum *Clientagenten* überbracht.

6. *void speichereFile(String filename , String fileInfo , Vector file)*

Diese Methode wird aufgerufen, wenn der *Clientagent* diesen *Serveragenten* mit der regelmäßigen Überprüfung der überbrachten Dateien beauftragt hat. Dabei wird diese Methode für jede überbrachte Datei einmal aufgerufen, um sie in dem *FileListe* - Verzeichnis zu speichern. Dabei besteht der Dateiname der nun gespeicherten Datei aus dem ursprünglichen Dateinamen sowie aus der Versionsnummer der Datei. Zwischen dem ursprünglichen Dateinamen und der Versionsnummer sind zur Unterscheidung dieser beiden Angaben zwei Doppelkreuze '##' eingefügt. Übergeben wird dieser Methode der relative Dateiname, die *CVS* - Versionsnummer dieser Datei und die Datei selbst.

7. *Vector getFileListe()*

Ein Aufruf dieser Methode erfolgt, wenn dieser *Serveragent* beim *Verwaltungsagenten* angemeldet werden soll. Diese Methode besitzt die Aufgabe, alle von diesem *Serveragenten* überwachten Dateinamen einschließlich ihrer relativen Pfade als Vektor zurückzugeben. Um dieses zu erreichen, wird für jeden Eintrag in dem *FileListe* - Verzeichnis überprüft, ob er ein Verzeichnis oder eine Datei ist. Handelt es sich dabei um ein Verzeichnis, so wird die Methode *subdirectories(...)* unter 8. aufgerufen, die die in ihr überwachten Dateien zurückgibt. Handelt es sich dagegen um eine Datei, so wird der relative Pfad und der Dateiname in die Liste der überwachten Dateinamen eingefügt.

8. *Vector subdirectories(int laenge , String dirname , Vector liste)*

Diese Methode wird aufgerufen, wenn beim Einlesen der Liste der von diesem *Serveragenten* überwachten Dateinamen ein Verzeichnis gefunden wurde. Diese Methode besitzt die Aufgabe, alle von diesem *Serveragenten* in dem gefundenen Verzeichnis überwachten Dateinamen einschließlich der relativen Pfade zurückzugeben. Übergeben bekommt diese Methode die Zeichenanzahl bei vollständigen Dateipfaden bis zum Separatorzeichen hinter dem *FileListe* - Verzeichnisnamen, den Pfad des gefundenen Verzeichnisses und die Liste der bereits eingetragenen überwachten Dateien. Um die oben genannte Aufgabe zu erfüllen, wird für jeden Eintrag in dem *dirname* - Verzeichnis überprüft, ob er ein Verzeichnis oder eine Datei ist. Handelt es sich dabei um ein Verzeichnis, so wird diese Methode *subdirectories(...)* erneut aufgerufen, die die in ihr überwachten Dateien zurückgibt. Handelt es sich dagegen um eine Datei, so wird der relative Pfad und der Dateiname in der Liste der überwachten Dateinamen eingefügt.

9. *void loescheFiles()*

Ein Aufruf dieser Methode erfolgt, wenn sich die Liste der überwachten Dateien verän-

dert hat. Ihre Aufgabe besteht darin, alle gespeicherten, überwachten Dateien im Verzeichnis *FileListe* und in den Unterverzeichnissen zu löschen, damit danach die Dateien gespeichert werden können, die nach der vorher genannten Veränderung überwacht werden sollen. Um dieses Löschen zu erreichen, wird für jeden Eintrag in dem Verzeichnis *FileListe* untersucht, ob er ein Verzeichnis oder eine Datei ist. Handelt es sich dabei um ein Verzeichnis, so wird die Methode *subdirectories(...)* unter 10. aufgerufen, damit zuerst alle Verzeichnisse und Dateien in dem gefundenen Verzeichnis gelöscht werden. Wurde dagegen eine Datei gefunden oder hat die Methode *subdirectories(...)* ihre Aufgabe bezüglich des vorher erwähnten Verzeichnisses erfüllt, so wird der untersuchte Eintrag gelöscht.

10. *void subdirectories(String dirname)*

Diese Methode wird aufgerufen, wenn beim Löschen der von diesem *Serveragenten* überwachten Dateien ein Verzeichnis gefunden wurde. Diese Methode besitzt die Aufgabe, alle von diesem *Serveragenten* in dem gefundenen Verzeichnis überwachten Dateien und Verzeichnisse zu löschen. Übergeben bekommt sie dabei den Pfad des gefundenen Verzeichnisses. Um ihre Aufgabe zu erfüllen, wird für jeden Eintrag in dem *dirname* - Verzeichnis überprüft, ob er ein Verzeichnis oder eine Datei ist. Handelt es sich hierbei um ein Verzeichnis, so wird die Methode *subdirectories(...)* unter 10. erneut aufgerufen, damit zuerst alle Verzeichnisse und Dateien in dem gefundenen Verzeichnis gelöscht werden. Wurde dagegen eine Datei gefunden oder hat die Methode *subdirectories(...)* ihre Aufgabe bezüglich des vorher erwähnten Verzeichnisses erfüllt, so wird der untersuchte Eintrag gelöscht.

11. *void holeFilesVonServeragenten(String fileName)*

Diese Methode wird aufgerufen, wenn bei der regelmäßigen Überprüfung einer Datei die Dateiversionen der anderen *Serveragenten* benötigt werden. Um die Dateiversionen von den anderen *Serveragenten* zu holen, wird der *DiffAgent* erzeugt und losgeschickt. Der *DiffAgent* erhält dabei den Dateinamen und die *AgletID* dieses *Serveragenten*. Übergeben bekommt diese Methode den Dateinamen samt dem relativen Pfad der zu überprüfenden Datei.

12. *Hashtable holeFile(String fileName)*

Aufgerufen wird diese Methode, wenn dieser *Serveragent* bei der regelmäßigen Überprüfung einer Datei die eigene Version benötigt oder wenn ein *DiffAgent* eines anderen *Serveragenten* die Dateiversion dieses Agenten für seinen *Master* gebraucht. Diese Methode bekommt den Dateinamen einschließlich des relativen Pfades übergeben und gibt die gesuchte Datei und die *CVS* - Versionsnummer zurück. Diese Versionsnummer wird dabei aus dem Dateinamen der gespeicherten Datei gewonnen, indem der Dateiname aufgesplittet wird in den Teil vor den Doppelkreuzen und in den Teil dahinter. Der Teil hinter den Doppelkreuzen entspricht hierbei der gesuchten Versionsnummer. Die gesuchte Datei wird zeilenweise in einen Vektor eingelesen und, wie oben bereits erwähnt, mit der *CVS* - Versionsnummer zurückgegeben.

13. *int getAnzahlPunkte(String info)*

Um die Anzahl der Punkte in der *CVS* - Versionsnummer zu bestimmen, wird diese Methode aufgerufen. Übergeben bekommt sie dabei die Versionsnummer und gibt die Anzahl der Punkte zurück.

14. *boolean sindDiffbar(String eigeneInfo , int eigeneAnzahl ,
String andereInfo , int andereAnzahl)*

Diese Methode wird aufgerufen, um festzustellen, ob die eigene Dateiversion mit der Dateiversion eines anderen *Serveragenten* auf Unterschiede untersucht werden darf. Dies wird anhand der beiden *CVS* - Versionsnummern überprüft. Besitzen beide Versionsnummern die gleiche Anzahl an Punkten, so sind die Dateien miteinander vergleichbar, wenn bei Vernachlässigung jeweils der letzten Zahl beide Nummern identisch sind. Besitzt eine der Versionsnummern mehr Punkte als die andere, so sind die Dateien miteinander vergleichbar, wenn die Versionsnummer mit der geringeren Anzahl an Punkten vollständig identisch ist mit dem Anfang der Versionsnummer mit der größeren Punkteanzahl. In allen anderen Fällen sind die beiden Dateien nicht miteinander vergleichbar. Diese Methode bekommt die *CVS* - Versionsnummern und die zugehörigen Punkteanzahlen übergeben und gibt 'true' zurück, falls die Dateien miteinander vergleichbar sind, andernfalls 'false'.

15. *void holeFilesVonServeragenten(String fileName , String file_info ,
Vector dasPruefFile)*

Ein Aufruf dieser Methode erfolgt immer dann, wenn bei der durch Anklicken des Buttons *Test* im *AVC - File-Menu* - Fenster gestarteten Überprüfung einer Datei die Dateiversionen der anderen *Serveragenten* benötigt werden. Um die Dateiversionen von den anderen *Serveragenten* zu holen, wird der *DiffAgent* erzeugt und losgeschickt. Der *DiffAgent* bekommt bei seiner Erzeugung den Dateinamen, die Versionsnummer der Datei und die Datei selbst übergeben. Diese Methode erhält beim Aufruf den Dateinamen samt dem relativen Pfad, die *CVS* - Versionsnummer und die zu überprüfende Datei selbst.

16. *void schickeMailZumClient(String filename , String andererName ,
String andereNummer , String diffZeit , Vector ergebnis)*

Diese Methode wird aufgerufen, wenn eine Mail zum *Clientagenten* geschickt werden soll. Übergeben bekommt diese Methode dabei den Dateinamen einschließlich des relativen Pfades, den Namen und die Nummer des Prüfungspartners, die Prüfzeit und die Unterschiede selbst. Alle diese Daten bilden die Mail, die nach Erzeugung des *MailAgenten* von diesem zum *Clientagenten* gebracht wird. Zusätzlich zu der zu überbringenden Mail erhält dieser *Slave* auch noch die *AgletID* des *Clientagenten*.

5.3.11 ServeragentStart

Diese Klasse repräsentiert einen Agenten, wobei ihre Superklasse die Klasse *Aglet* ist. Dieser Agent wird auf dem *Server* im *Context* des *Verwaltungsagenten* und der *Serveragenten* durch den Benutzer erzeugt. Seine Aufgabe besteht darin, falls der *Verwaltungsagent* und die *Serveragenten* der anderen Benutzer bereits existieren, für einen neuen Benutzer einen *Serveragenten* zu starten. Der Name und die Nummer des neuen Benutzers werden dabei aus der Datei *NeuerBenutzer.txt* eingelesen. Nachdem dieser Agent diesen *Serveragenten* erzeugt hat, ist seine Lebenszeit zu Ende und er beseitigt sich.

Diese Klasse enthält die folgende Methode:

1. *void onCreation(Object o)*

Diese Methode wird bei der Initialisierung dieses Agenten aufgerufen. In ihr wird zuerst das Verzeichnis bestimmt, in dem die Datei *NeuerBenutzer.txt* zu finden ist. Dann wird aus der eben genannten Datei der Name und die Nummer des neuen Benutzers eingelesen. An-

schließlich wird unter Berücksichtigung der vorher erhaltenen Benutzerangaben der *Serveragent* im *Context* der anderen *Serveragenten* erzeugt. Nachdem dieser Agent diese Aufgabe erfüllt hat, beseitigt er sich.

5.3.12 ServerAnmeldung

Diese Klasse repräsentiert einen *Slave*. Die Superklasse ist die Klasse *Slave*. Dieser *Slave* wird vom *Serveragenten* erzeugt, der somit sein *Master* ist, und zum *Verwaltungsagenten* geschickt. Die Aufgabe dieses *Slaves* besteht darin, seinen *Master* bei dem *Verwaltungsagenten* entweder an- oder abzumelden. Diesem *Verwaltungsagenten* übergibt er zu diesem Zweck die Nachricht, die er von seinem Erzeuger als Argument bei seiner Erzeugung erhalten hat. Nachdem der *Verwaltungsagent* ihm die Mitteilung über die erfolgte An- bzw. Abmeldung übergeben hat, kehrt er in den *Context* seines *Masters* zurück. Sollte er dabei den *Serveragenten* anmelden, so übergibt er diesem die Nachricht über die erfolgreiche Ausführung seiner Aufgabe und wird danach beseitigt. Sollte er ihn dagegen abmelden, so existiert sein *Master* nicht mehr, weshalb er ohne Übergabe einer Nachricht beseitigt wird. Konnte der *Verwaltungsagent* im Rahmen der Anmeldung nicht gefunden werden, so übergibt er nach seiner Rückkehr seinem *Serveragenten* die entsprechende Nachricht darüber, bevor er entfernt wird. Ist dieser Fall bei der Abmeldung aufgetreten, so wird er nach seiner Rückkehr in den *Context* seines *Masters* sofort beseitigt.

Die Klasse beinhaltet die folgenden Methoden:

1. *void initializeJob()*

Die gleichnamige Methode der Superklasse ist *abstract* und muß deshalb überschrieben werden.

2. *void doJob()*

Die gleichnamige Methode der Superklasse ist *abstract* und muß deshalb überschrieben werden.

3. *boolean handleMessage(Message msge)*

Diese Methode wird aufgerufen, wenn dieser *Slave* die Nachricht (*msge*) bekommt, daß er im *Context* des *Verwaltungsagenten* oder nach seiner Rückkehr im *Context* seines *Masters* angekommen ist. Bekommt er als erstes die Nachricht *doJob*, so ist er in den *Context* des *Verwaltungsagenten* angekommen und sucht dort nach dem *Proxy* des *Verwaltungsagenten*. Wird dieser gefunden, so erhält der *Verwaltungsagent* mit dessen Hilfe die Nachricht des *Serveragenten*. Die Antwort des *Verwaltungsagenten* darauf ist die Nachricht für den *Master* dieses *Slaves*. Ist allerdings der Fall eingetreten, daß der *Proxy* des *Verwaltungsagenten* nicht gefunden werden konnte, so beinhaltet die Antwort dieses *Slaves* für seinen *Master* das aufgetretene Problem.

Erhält dieser *Slave* die Nachricht *onReturn*, so ist er in den *Context* seines *Masters* zurückgekehrt. Hatte dieser *Slave* die Aufgabe zu erfüllen, seinen Erzeuger bei dem *Verwaltungsagenten* anzumelden, so übergibt er seinem *Master* durch dessen *Proxy* die Mitteilung, daß die Anmeldung erfolgt ist oder daß er den *Verwaltungsagenten* nicht gefunden hat. Danach wird er beseitigt. Sollte dagegen sein *Master* abgemeldet werden, so wird er sofort nach der Rückkehr in dessen *Context* entfernt.

5.3.13 Serverstart

Diese Klasse, die als Superklasse die Klasse *Aglet* besitzt, stellt einen Agenten dar. Dieser Agent wird auf dem *Server* durch den Administrator erzeugt. Seine Aufgabe besteht darin, den *Verwaltungsagenten* zu erzeugen, falls dieser noch nicht in dem *Server - Context* vorhanden ist. Außerdem liest er in diesem Fall aus der Datei *Benutzerliste.txt* alle Benutzer ein, für die er einen *Serveragenten* zu generieren hat. Nachdem er den *Verwaltungsagenten* und die *Serveragenten* erzeugt hat, ist seine Lebenszeit zu Ende und er wird beseitigt. Tritt allerdings der Fall ein, daß der *Verwaltungsagent* bereits vorhanden ist, so wird dieser Agent beseitigt, ohne irgendeinen Agenten generiert zu haben.

Die folgende Methode ist in dieser Klasse enthalten:

1. *void onCreate(Object o)*

Diese Methode wird bei der Initialisierung dieses Agenten aufgerufen. In ihr wird zuerst das Verzeichnis bestimmt, in dem die Datei *Benutzerliste.txt* zu finden ist. Dann wird in dem *Context* untersucht, ob bereits ein *Verwaltungsagent* existiert, was dadurch erreicht wird, daß nach dem *Proxy* eines derartigen Agenten gesucht wird. Ist schon ein *Verwaltungsagent* vorhanden, so beseitigt sich dieser *Serverstart - Agent* sofort, ohne irgendeinen Agenten generiert zu haben. Existiert dagegen noch kein *Verwaltungsagent*, so wird einer erzeugt. Weiterhin werden die *Serveragenten*, pro Benutzer einer, generiert, nachdem der Name und die Nummer eines jeden Benutzers aus der Datei *Benutzerliste.txt* eingelesen worden sind. Wenn dieser Agent diese Aufgaben ausgeführt hat, beseitigt er sich.

5.3.14 Verwaltungsagent

Diese Klasse repräsentiert einen stationären Agenten. Sie besitzt dabei die Superklasse *Aglet*. Nach seiner Erzeugung durch den *Serverstart - Agenten* befindet sich der *Verwaltungsagent* im *Server - Context*. Er verwaltet dort die Liste aller *Serveragenten*, indem er deren *AgletIDs* speichert, und die Listen der Dateinamen, die derzeit von den vorher genannten *Serveragenten* überwacht werden. Letzteres wird dadurch erreicht, daß pro überwachter Datei eine Liste existiert, in der die *Serveragenten* aufgeführt sind, die diese Datei observieren. Soll ein *Serveragent* an- bzw. abgemeldet werden, so wird die ihm entsprechende *AgletID* in der oben erwähnten Liste gespeichert bzw. gelöscht. Bei der Anmeldung wird außerdem noch die *AgletID* des *Serveragenten* in die Listen der Dateinamen eingetragen, deren entsprechende Dateien dieser *Serveragent* überwacht.

Benötigt ein von einem *Clientagenten* losgeschickter *Slave* die *AgletID* des zugehörigen *Serveragenten*, so wird ihm diese übergeben, sofern der *Serveragent* angemeldet ist. Andernfalls wird die *AgletID* dieses *Verwaltungsagenten* zurückgegeben. Beauftragt ein *Serveragent* seinen *DiffAgenten* mit der Beschaffung aller Versionen einer zu überprüfenden Datei, so erhält dieser *DiffAgent* eine Liste aller *AgletIDs* derjenigen *Serveragenten*, die auch diese Datei überwachen. Sendet dagegen ein *Serveragent* einen *FilenamenAgenten* zu diesem *Verwaltungsagenten*, so wird die *AgletID* dieses *Serveragenten* aus den Listen der Dateinamen gelöscht. Außerdem wird aus der Nachricht, die der *FilenamenAgent* übergeben hat, die Liste der Dateinamen bestimmt, die der *Serveragent* derzeit überwacht. Danach wird dessen *AgletID* neu in die entsprechenden Listen der Dateinamen eingetragen. Wird dieser Agent beseitigt, so werden auch die *Serveragenten* entfernt.

Diese Klasse beinhaltet die folgenden Methoden:

1. *void onCreate(Object o)*

Diese Methode wird bei der Initialisierung dieses Agenten aufgerufen. In ihr werden die Objekte *agletAdresse* und *listeFilenamem* erzeugt, deren Superklasse die Klasse *Hashtable* ist.

2. *boolean handleMessage(Message msg)*

Ein Aufruf dieser Methode erfolgt, wenn einer der folgenden *Slaves* diesem Agenten eine Nachricht übergibt: *ServerAnmeldung*, *ClientServerAnmeldung*, *FilePruefAgent*, *FileAgent*, *FilenamenAgent* und *DiffAgent*. Wird die zum Beispiel Nachricht übergeben, daß ein *Serveragent* angemeldet werden soll, so werden die folgenden Methoden aufgerufen: *agletEintragen(...)* und *fileListeEintragen(...)*. Dem *Slave ServerAnmeldung* wird die Antwort zurückgegeben, daß die Eintragung vorgenommen worden ist. Wenn allerdings die Botschaft überbracht wird, daß ein *Serveragent* gelöscht werden soll, so werden die Methoden *agletDelete(...)* und *fileListeLoeschen(...)* aufgerufen. Der *Slave ServerAnmeldung* erhält als Antwort, daß das Löschen erfolgt ist.

Soll dagegen eine Datei entweder aufgrund der regelmäßigen Überprüfung oder aufgrund des Anklickens des *Test* - Buttons im *AVC - File-Menu* - Fenster überprüft werden, so benötigt der *DiffAgent* alle *AgletIDs* der *Serveragenten*, die auch diese Datei überwachen. Bei Erhalt der entsprechenden Nachricht liefert dieser Agent als Antwort die Liste aller betroffenen *AgletIDs*.

Tritt nun aber der Fall ein, daß ein *Clientagent* bei seinem *Serveragenten* an- bzw. abgemeldet werden soll oder daß ein *Clientagent* seinen *Serveragenten* mit der Überprüfung einer ausgewählten Datei oder mit der regelmäßigen Überprüfung mehrerer Dateien beauftragen möchte, so gebraucht der vom *Clientagenten* losgeschickte *Slave* die *AgletID* seines *Serveragenten*. Bekommt dieser Agent die entsprechende Mitteilung, so ruft er die Methode *agletId(...)* auf. Von dieser Methode erhält dieser Agent entweder die *AgletID* des gesuchten *Serveragenten*, falls dieser existiert, oder seine eigene *AgletID* übergeben. Diese *AgletID* ist die Antwort für den beauftragenden *Slave*.

Hat sich die Liste der überwachten Dateien bei einem Benutzer verändert, so schickt der *Serveragent* den *FilenamenAgenten* zu diesem Agenten, um die Liste der Dateien mit den sie überwachenden *AgletIDs* der *Serveragenten* auf den neusten Stand zu bringen. Dazu werden die Methoden *fileListeLoeschen(...)* und *fileListeEintragen(...)* aufgerufen. Als Antwort wird zurückgegeben, daß dieses Update durchgeführt worden ist. Diese Methode gibt zurück, ob die übergeben bekommene Nachricht behandelt worden ist.

3. *void agletEintragen(Message mess)*

Diese Methode wird aufgerufen, wenn sich ein *Serveragent* durch den *Slave ServerAnmeldung* beim *Verwaltungsagenten* anmelden möchte. Es wird zu diesem Zweck die *AgletID* des *Serveragenten* in der *Hashtable agletAdresse* gespeichert, die die Liste aller *Serveragenten* darstellt, wobei als Schlüssel der Benutzername und die Benutzernummer verwendet werden. Ist dabei unter diesem Schlüssel noch eine alte *AgletID* gespeichert, so wird diese vor der Speicherung der neuen *AgletID* gelöscht. Die *AgletID* und der Name und die Nummer des Benutzers werden aus dem übergeben bekommenen Argument *mess* gewonnen.

4. *void agletDelete(Message mess)*

Ein Aufruf dieser Methode erfolgt, wenn sich ein *Serveragent* durch den *Slave ServerAnmeldung* bei diesem Agenten abmelden möchte. Es wird dabei die *AgletID* dieses *Serveragenten* aus der Liste aller *Serveragenten* gelöscht, wobei als Schlüssel der Name und die Nummer des Benutzers dieses *Serveragenten* dienen, die aus dem übergebenen Argument *mess* bestimmt werden.

5. *AgletID agletId(Message message)*

Diese Methode wird aufgerufen, wenn die *AgletID* eines bestimmten *Serveragenten* gesucht wird. Dazu wird über den Schlüssel Benutzername und Benutzernummer in der Liste aller *Serveragenten* nachgeschaut. Ist dieser Schlüssel vorhanden, so wird die *AgletID*, auf die durch diesen Schlüssel zugegriffen werden kann, zurückgegeben. Ist der Schlüssel dagegen nicht vorhanden, so wird die *AgletID* dieses *Verwaltungsagenten* zurückgegeben. Aus dem übergeben bekommenen Argument *message* werden der Name und die Nummer des Benutzers gewonnen.

6. *void fileListeEintragen(Message mess)*

Ein Aufruf dieser Methode erfolgt, wenn ein neuer *Serveragent* eingetragen werden soll oder wenn sich die Liste der Dateinamen, die von einem *Serveragenten* überwacht werden, verändert hat. Übergeben bekommt diese Methode das Argument *mess*, aus dem die *AgletID* des *Serveragenten* und die Liste der Dateinamen, die der *Serveragent* derzeit überwacht, ermittelt werden. Für jeden dieser Dateinamen wird überprüft, ob er bereits in der Liste der überwachten Dateinamen enthalten ist. Wenn ja, dann wird die übergebene *AgletID* in die Liste der *AgletIDs* dieses Dateinames eingefügt. Wenn nein, dann wird für diesen Dateinamen eine neue Liste für *AgletIDs* eingetragen und die übergebene *AgletID* eingefügt.

7. *void fileListeLoeschen(Message mess)*

Diese Methode wird aufgerufen, wenn ein *Serveragent* gelöscht werden soll oder wenn sich die Liste der Dateien, die von einem *Serveragenten* überwacht werden, verändert hat. Übergeben bekommt sie das Argument *mess*, aus dem die *AgletID* des zu löschenden *Serveragenten* bestimmt wird. Es wird eine Aufzählung aller überwachten Dateinamen erzeugt. Für jeden dieser Dateinamen wird dann überprüft, ob in seiner Liste der *AgletIDs*, auf die über diesen Dateinamen als Schlüssel zugegriffen werden kann, die vorher bestimmte *AgletID* enthalten ist. Sollte dieses der Fall sein, so wird diese *AgletID* aus der betroffenen Liste gelöscht. Ist nach diesem Löschen die Liste der *AgletIDs* für diesen Dateinamen leer, so wird dieser Dateiname aus der Liste der überwachten Dateinamen gelöscht.

8. *void onDisposing()*

Ein Aufruf dieser Methode erfolgt, wenn dieser *Verwaltungsagent* beseitigt werden soll. Ihre Aufgabe besteht darin, alle vorhandenen *Serveragenten* zu beseitigen.

5.4 Die Textdateien

5.4.1 Benutzerliste.txt

Diese Textdatei befindet sich auf dem Server in dem Verzeichnis *server*, das im Verzeichnis *AVC* steht, und wird vom *Serverstart* - Agenten verwendet. Sie beinhaltet zeilenweise die Namen und die zugehörigen Nummern der Benutzer, für die beim Start des *Verwaltungsagenten*

ten jeweils ein *Serveragent* erzeugt werden soll. Soll ein Benutzer, der bisher noch keinen *Serveragenten* besaß, beim nächsten Start des *Serverstart* - Agenten einen eigenen *Serveragenten* bekommen, so wird der Name und die Nummer dieses Benutzers als Zeile in diese Textdatei eingetragen, wobei zwischen dem Namen und der Nummer ein Tabulator-Schritt eingefügt wird. Soll dagegen ein Benutzer keinen *Serveragenten* mehr erhalten, so wird einfach die entsprechende Zeile aus dieser Datei entfernt.

Beispiel für eine Datei 'Benutzerliste.txt':

```
Peter Müller  11
Eva Obst     23
Enno Steen   17
```

5.4.2 FileListe.txt

Jeder Benutzer, der mit diesem Programm arbeitet, besitzt eine Textdatei dieser Art. In ihr sind Dateien zeilenweise durch ihren relativen Pfad und Dateinamen eingetragen. Der relative Pfad bezieht sich dabei immer auf das Verzeichnis *FileListe* des Benutzers. Die Datei *FileListe.txt* ist im Benutzerverzeichnis auf dem Computer gespeichert, auf dem sich der Benutzer eingeloggt hat. Das gerade erwähnte Benutzerverzeichnis besteht hierbei aus dem Namen und aus der Nummer des Benutzers und ist im Verzeichnis *AVC* auf dem Client gespeichert. Diese Datei wird von dem *Clientagenten* aufgerufen, um aus ihr die Liste der Dateien einzulesen, die durch die regelmäßige Überprüfung überwacht werden. Soll eine überwachte Datei überprüft werden, bei der durch den *CVS* - Befehl *update* bzw. *checkout* ein Konflikt aufgetreten ist, so wird die Zeile mit dem Dateinamen und mit dem relativen Pfad durch den *Clientagenten* aus der Datei *FileListe.txt* gelöscht.

Wird auf dem Bildschirm das *AVC - Choice-Menu* - Fenster angezeigt, so werden aus dieser Datei die Dateinamen samt den relativen Pfaden eingelesen, die durch dieses Programm überwacht werden, um sie in dem angezeigten Fenster zu markieren. Wird in dem Fenster der *Accept* - Button angeklickt, so werden die zu dem Zeitpunkt markierten Dateinamen samt den relativen Pfaden in der Datei *FileListe.txt* eingetragen, wobei die vorherige Version dieser Datei überschrieben wird.

Beispiel für eine Datei 'FileListe.txt':

```
Test_4\Test_41\Test_411\Datei1.txt
Test_4\Test_41\Test_411\Datei2.java
Test_4\Test_41\Test_411\Datei3.txt
Test_1\Datei4.doc
```

5.4.3 NeuerBenutzer.txt

Diese Datei wird benötigt, wenn ein Benutzer, der noch nicht in der Datei *Benutzerliste.txt* aufgeführt ist, nachdem der *Verwaltungsagent* und die *Serveragenten* der anderen Benutzer bereits erzeugt worden sind, ebenfalls mit diesem Programm arbeiten möchte und deshalb einen eigenen *Serveragenten* gebraucht. Dazu wird in diese Datei der Name und die Nummer des Benutzers in die erste Zeile eingetragen, wobei zwischen dem Namen und der Nummer ein Tabulator-Schritt eingefügt wird. Daraufhin wird der *ServeragentStart* - Agent erzeugt,

der dann auf diese Datei zugreift, die sich auf dem Server in dem Verzeichnis *server* befindet, das wiederum im Verzeichnis *AVC* steht.

Beispiel für eine Datei 'NeuerBenutzer.txt':

Peter Fröhlich 1

5.4.4 User.txt

Jeder Benutzer dieses Programmes besitzt eine solche Datei. Sie ist auf dem Rechner gespeichert, auf dem sich der Benutzer eingeloggt hat, und steht dort im Verzeichnis *AVC*. Der *Clientstart* - Agent benötigt diese Datei, um aus der zweiten Zeile den Namen und aus der dritten die Nummer des Benutzers zu erhalten. Dieser Agent gebraucht den Namen und die Nummer für die Benutzeroberfläche und bei der möglichen Erzeugung des *Clientagenten*.

Beispiel für eine Datei 'User.txt':

```
## Benutzerdaten  
Name : Enno Steen  
Nummer : 17
```

5.4.5 Userdaten.txt

Jeder Benutzer dieses Programmes besitzt eine solche Datei. Sie ist im Benutzerverzeichnis auf dem Rechner gespeichert, auf dem sich der Benutzer eingeloggt hat, wobei dieses Benutzerverzeichnis im *AVC* - Verzeichnis zu finden ist. Der *Clientagent* des Benutzers liest aus dieser Datei folgende Informationen ein: Wie groß ist der zeitliche Abstand zwischen zwei Überprüfungen ? Möchte der Benutzer über die Ankunft einer Mail informiert werden ? Wann war die letzte regelmäßige Überprüfung ? Der Zeitpunkt der letzten Überprüfung wird durch den *Clientagenten* in dieser Datei auf den neusten Stand gebracht, sobald die zu überprüfenden Dateien dem *Serveragenten* überbracht worden sind.

Soll das *AVC - Management* - Fenster auf dem Bildschirm angezeigt werden, so wird aus dieser Datei eingelesen, wie groß der zeitliche Abstand zwischen zwei Überprüfungen ist und ob der Benutzer über die Ankunft einer Mail informiert werden möchte. Wird das Fenster daraufhin wieder verlassen, so werden die dann gültigen Einträge bezüglich dieser beiden Benutzerinformationen(Alarm, Überprüfungsabstand) in dieser Datei gespeichert.

Beispiel für eine Datei 'Userdaten.txt':

```
## Benutzerdaten  
Name : Enno Steen  
Nummer : 17  
Ueberpruefung : 5  
Alarm : off  
Letzte Pruefung : 17 Dec 1997 18:33:04 GMT
```

6. Diskussion und Ausblick

Die derzeit vorliegenden Version der agentenbasierten Versionskontrolle untersucht die zur Überprüfung bestimmte Dateiversion des einen Benutzers auf Unterschiede zu den Dateiversionen der anderen Benutzer, die mein Programm mit der Überwachung dieser Dateiversionen beauftragt haben. Hierfür wird der Befehl *diff* verwendet. Werden bei dieser Untersuchung Unterschiede festgestellt, so wird der Benutzer, der diese Untersuchung in Auftrag gegeben hat, über diese Unterschiede informiert. Eine zukünftige Version dieser agentenbasierten Versionskontrolle könnte dahin gehend verändert werden, daß sie die einzelnen Dateiversionen nicht auf Unterschiede, sondern auf mögliche CVS-Konflikte überprüft, die beim Mergen dieser Dateiversionen entstehen würden. Der entsprechende Befehl ist *merge*. Die Entdeckung eines potentiellen CVS-Konfliktes würde dann eine Benachrichtigung der betroffenen Benutzer nach sich ziehen. Um mein Programm in diese Richtung verändern zu können, muß allerdings noch genauer untersucht werden, wie das Mergen zu realisieren ist, wenn die beiden zu mergenden Dateiversionen von unterschiedlichen Grunddateien ausgehen.

Ein weiteres Problem ist die mangelnde Konsistenz bezüglich der von IBM entwickelten Versionen der Aglet Workbench AWB. Im Verlauf meiner Studienarbeit ist es öfter vorgekommen, daß die AWB geändert wurde, wobei die alte Version dann ungültig wurde und nicht mehr benutzt werden konnte. Leider ist dabei nicht immer die Kompatibilität zu den früheren Versionen gewährleistet gewesen, so daß bereits programmierte Agenten nicht mehr funktionierten oder sich nicht mehr so wie zum Zeitpunkt ihrer Programmierung verhielten. Während meiner Studienarbeit ist es zum Beispiel vorgekommen, daß bei einer neueren Version der AWB einige von mir benutzte Methoden nicht mehr existierten, was zur Folge hatte, daß einige Aglets umgeschrieben werden mußten.

Eine Einschränkung dieses Programmes ist die Verwendung der sternförmigen Client/Server-Architektur, wobei der Server-Context das Zentrum des Sterns darstellt und die einzelnen Contexts der Clients nur mit diesem Zentrum verbunden sind. Ist nämlich der Server-Context mit den serverseitigen Agenten nicht vorhanden, so funktioniert die agentenbasierte Versionskontrolle nicht mehr. Aus diesem Grund beseitigen sich die clientseitigen Agenten, die die Abwesenheit der serverseitigen Agenten feststellen. Im Rahmen einer Weiterentwicklung dieses Programmes könnte diese Einschränkung behoben werden, indem beispielsweise ein Client-Context beim Nichtvorhandensein des Server-Contextes diese Aufgabe übernehmen würde.

7. Verwendete Materialien

1. Binas-Holz, Antje: Das Programmierbuch JAVA, SYBEX-Verlag GmbH, Düsseldorf, 1996
2. Campione, Mary und Walrath, Kathy: Das Java™ Tutorial: Objektorientierte Programmierung für das Internet, Addison-Wesley-Longman, Bonn, 1997
3. Doberenz, Walter: JAVA, Carl Hanser Verlag, München, Wien, 1996
4. Cederqvist, Per et al: Version Management with CVS, Signum Support AB, Linköping, Schweden, 1992, 1993
5. IBM's Aglets Workbench und die zugehörigen Dokumente bezüglich der agentenbasierten Computer-Technologie, IBM Research's Aglets Workbench Home Page:
<http://www.trl.ibm.co.jp/aglets/>
6. Sommers, Bret: Agents: Not just for Bond anymore, März 1997
<http://www.javaworld.com/javaworld/jw-04-1997/jw-04-agents.html>
7. Venners, Bill: Under the Hood: The architecture of aglets, März 1997
<http://www.javaworld.com/javaworld/jw-04-1997/jw-04-hood.html>
8. Venners, Bill: Under the Hood: Solve real problems with aglets, a type of mobile agent, April 1997
<http://www.javaworld.com/javaworld/jw-05-1997/jw-05-hood.html>
9. Lange, Danny B. und Oshima, Mitsuru: Auszüge aus: Programming Mobile Agents in Java™ - With the Java Aglet API, Tokio, 1997
<http://www.trl.ibm.co.jp/aglets/aglet-book/index.html>