FACTA UNIVERSITATIS (NIŠ)
Series: Electronics and Energetics vol. 14, No. 1 April 2001, 33-43

KONZEPT ZUR GESICHERTEN NACHRICHTEN-ÜBERMITTLUNG IN EINEM VERTEILTEM SYSTEM

Thomas Droste and Johannes Jußen

Kurzfassung – In einem Rechnersystem sind Warnmeldungen und Nachrichten eine wichtige Möglichkeit, um Verantwortliche auf mögliche Fehler und Systemverletzungen hinzuweisen. Speziell in einem verteilten Sicherheitsmanagement [4] ist es zwingend notwendig, dass Informationen und Sicherheitsmeldungen auch bei Fehlern, Systemabstürzen oder Netzwerkausfall gesichert und ein Vorgang im Rechnerverbund nachträglich rekonstruiert werden kann. Es darf kein Datenverlust auftreten, Redundanz der Nachrichten ist hingegen erlaubt. Ziel dieses Artikels ist es, ein Konzept vorzustellen, welches am Beispiel eines verteilten Sicherheitsmanagements einen realisierten Nachrichtendienst erläutert und angewendete Mechanismen zur Datensicherung aufzeigt. Speziell die Rekonstruktion des Zustands, in welchem der Nachrichtendienst in seiner Funktion unterbrochen worden ist, und die damit verbundenen Anforderungen an den Aufbau des Dienstes werden in diesem Artikel näher betrachtet

Schlagworte: Nachrichtenübermittlung, verteiltes Sicherheitsmanagement, Datensicherung.

1. Einführung

Nachrichten über den Zustand eines Rechnerverbundes sollen von den Clients an einen zentralen Server gesendet werden. Der Nachrichtendienst benötigt zur Kommunikation zwischen Sender und Empfänger zwei Komponenten: Client und Server. Je nach Konfiguration kann der Nachrichtendienst Client- und/oder Servereigenschaften besitzen.

Primär sollen kurze Nachrichten versendet werden, welche Ereignisse (Information, Warnung, Fehler, Sicherheitsverletzung, etc.) beschreiben.

Manuscript eingegangen 5. Dezember 2000.

T. Droste ist Wissenschaftlicher Mitarbeiter am Lehrstuhl für Datenverarbeitung, Ruhr-Universität Bochum, 44780 Bochum (e-mail: droste@etdv.ruhr-uni-bochum.de).

Cand.-Ing. J. Jußen, Studentische Hilfskraft und Diplomand am obigen Institut.

Erst wenn eine Nachricht vom Absender zum Nachrichtenempfänger korrekt übertragen und diese auch lokal auf Empfängerseite gesichert worden ist, kann diese Nachricht vom Absender als zugestellt angesehen werden.

Die Übertragung von Nachrichten über eine Netzwerkverbindung ist durch das TCP/IP- Protokoll gesichert. Für die Überprüfung der sicheren Zustellung müssen starke Synchronisationsmechanismen zusätzlich angewandt werden.

Probleme treten jedoch auf, sobald Nachrichten versendet werden sollen, aber der Empfänger nicht bereit ist diese zu empfangen (Netzwerk- und Systemausfall). Die Rekonstruierbarkeit von Abläufen muss im Nachhinein gewährleistet werden. Es dürfen keine Nachrichten verloren gehen. Dies stellt insbesondere höhere Anforderungen an die Realisierung des Client. Nach Unterbrechung muss der Ablauf der Nachrichtenübermittlung fortgesetzt bzw. neu initiiert werden, zwischenzeitlich angefallene Nachrichten müssen nachversendet werden.

Die Implementierung als *multithreaded* Microsoft Windows NT - Dienst erlaubt die Verarbeitung im Hintergrund und gewährleistet die Verfügbarkeit ab Systemstart. Bei Problemen mit der Netzwerkverbindung wird die Nachrichtenübermittlung fortgesetzt, wenn Sender und Empfänger wieder erreichbar und bereit sind. Fällt ein Dienst aus, so wird spätestens bei Neustart die Nachrichtenübermittlung automatisch fortgeführt.

2. Aufbau des Dienstes

Der Nachrichtendienst beinhaltet die Funktionalität von Client und Server, d.h. er kann sowohl Nachrichten versenden als auch empfangen. Die zentrale Komponente *CLogControl* dient zur lokalen Nachrichtenverarbeitung und zur eigentlichen Sicherung von Mitteilungen auf einem Server.

In Abbildung 1 ist die Interaktion zwischen den Dienstkomponenten dargestellt. Das *CLogControl* nimmt dabei Nachrichten entgegen und sichert diese lokal. In Abhängigkeit der Konfiguration werden Nachrichten vom *CLogControl* über den *CLogClient* mit Hilfe der Methode *WriteEntry()* an einen übergeordneten *CLogServer* weitergeleitet. Dieser Server kann während des Betriebs neu spezifiziert werden. Der *CLogClient* wird kurzzeitig deaktiviert, die IP-Adresse des Servers geändert und im Anschluss der *CLogClient* wieder aktiviert.

Nachrichten können entweder vom lokalen System generiert oder vom Server über eine Netzverbindung entgegengenommen werden. Diese werden, wenn das *CLogControl* sich im aktivierten Zustand befindet, über eine Schnittstelle an das *CLogControl* weitergeleitet.

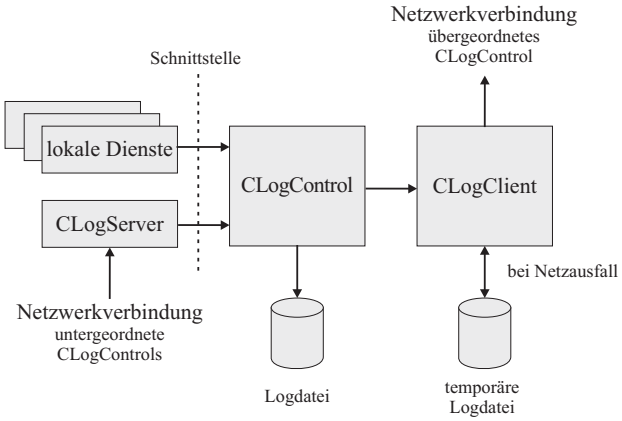

Abbildung 1. Aufbau des Nachrichtendienstes

3. Aufbau einer Nachricht

Eine Nachricht besitzt mehrere Felder, durch die eine eindeutige Identifizierung und Klassifizierung möglich ist. In Tabelle 1 ist der Aufbau einer Nachricht aufgeschlüsselt worden. Dies entspricht einer Instanz der Klasse *CLogEntry*.

Tabelle 1. Nachrichtenaufbau

Level	Priority	EventTime	SaveTime	User	Host	EventSource	EventDescription

Durch das erste Feld wird der Teilbereich der Software charakterisiert, in der die Mitteilung generiert worden ist. Eine Nachricht bekommt dieses Attribut von der erzeugenden Software mitgeteilt. Dadurch ist es möglich das Ereignis einzuordnen in *Application, Service, Threads, Connections* und *Basic Objects*.

Die Priorität gibt die Art der Nachricht an. Dadurch wird eine Klassifizierung in *Information*, *System Error* und *User Error* erreicht. Eine spätere Durchsicht der Nachrichten bzgl. der Relevanz kann somit einfacher stattfinden.

Die nächsten beiden Felder enthalten jeweils einen Zeitstempel: Auslösezeitpunkt und Zeitpunkt der physikalischen Abspeicherung der Nachrichten. Ersteres gibt den tatsächlichen Generierungszeitpunkt der Nachricht auf dem lokalen System an. Die physikalische Abspeicherung und Archivierung der Nachricht wird durch den zweiten Zeitstempel gekennzeichnet. Für eine evtl. lokal gesicherte und für eine auf einen anderen Rechner transferierte Nachricht ergeben sich somit unterschiedliche Zeitstempel für die gleiche

Nachricht, die alleinig durch den Zeitaufwand für die Netzübertragung differieren. Wird eine Nachricht durch einen Fehlerfall (Netzwerkfehler, Rechnerausfall, etc.) erst später oder verzögert übermittelt, so erhöht sich die Differenz zwischen Erzeugung und Sicherung der Nachricht entsprechend der Ausfallzeit. Damit eine annährend gleiche Zeitbasis für die Zeitstempel existiert, wird mit Hilfe eines Zeit-Servers die Zeit aller Rechner mehrmals täglich synchronisiert, um die Nachrichten auch netzübergreifend zeitlich verfolgen zu können.

Der Benutzer- und der Rechnername kennzeichnen eine individuelle Umgebung der die Nachricht zugeordnet werden kann. Der Rechnername wird durch die IP-Adresse gekennzeichnet.

Der eigentliche Ursprung der Nachricht wird in dem darauf folgendem Feld eingetragen. Hier wird die entsprechende Stelle in der auslösenden Software angegeben, an der die Mitteilung generiert worden ist.

Die Art der Mitteilung im letztem Feld enthält eine genaue Spezifikation des Ereignisses.

4. Nachrichtenübermittlung

Die Netzwerkverbindungen werden auf Basis von Win32-Sockets realisiert. Diese werden in Instanzen einer Klasse *CBlockingSocket* gekapselt, welche folgende Merkmale besitzt:

- Ein zu versendendes Datenpaket besteht aus einem Header (50 Byte), in dem die Klassenbezeichnung und die Datenmenge der zu übertragenen serialisierten Instanz spezifiziert sind, und darauf folgend der serialisierten Instanz selbst.
- Innerhalb eines Funktionsaufrufs von *CBlockingSocket* wird grundsätzlich auf Ereignisse gewartet, die Netzwerk-Befehlen unmittelbar vorangestellt werden. Somit warten Threads nicht in blockierenden Aufrufen der Bibliothek *winsock* und können auch von einem nebenläufig ablaufenden Thread signalisiert werden. Die Ereignisse müssen entsprechend bei der Bibliothek *winsock* als Netzwerkereignisse registriert werden, damit diese Ereignisse ausgelöst werden, wenn Daten anliegen. Über eine Fallunterscheidung mit einem Flag *cancel* ist es möglich, z.B. im Rahmen eines *shutdown* Methodenaufrufe der Klasse *CBlockingSocket* von Threads, sofort zu unterbrechen, ohne dass ein Timeout für einen Netzwerkbefehl abgewartet werden muss. Der unterbrochene Thread verlässt die Methode umgehend mit einer Fehlermeldung und kann nach Freigabe aller Ressourcen direkt terminieren.
- Klassen, die über das Netzwerk übertragen werden sollen, müssen *Mar-*

shalling unterstützen und von einer Basisklasse *CLogMessage* erben. Dazu müssen die Funktionen *Serialize()* und *Unserialize()* jeweils individuell zu jeder Klasse implementiert werden. *CBlockingSocket* bietet beim Empfangen und Senden von Daten nur diese Schnittstelle *CLogMessage* an. Dadurch wird jedoch die universelle Verwendung einer Netzwerkverbindung deutlich vereinfacht.

- Alle Instanzen von Klassen, die über die Netzwerkverbindung übertragen werden sollen, müssen zusätzlich auf dem Server bei der Schnittstelle *CLogMessage* registriert sein. Diese erlaubt es *CBlockingSocket* beim Empfang von serialisierten Instanzen, Konstruktoren für die zugeordneten empfangenen Klassen aufzurufen. Dazu ist in jeder Klasse eine statische Funktion *CreateInstance()* zu implementieren, die den Aufruf des Konstruktors realisiert. In einer Tabelle hat die Schnittstelle *CLogMessage* die Zuordnung zwischen Klassenbezeichner aus dem Header und dem Einsprungpunkt der Funktion *CreateInstance()* gespeichert. Durch Polymorphismus wird automatisch das Marshalling in das *CBlockingSocket* eingebunden. Zur Laufzeit wird innerhalb der *Methode Receive()* immer die zur empfangenen Klasse gehörige Funktion *Unserialize()* aufgerufen, die es erlaubt den Bytestrom richtig zu interpretieren und die vorher mit *CreateInstance()* erzeugte Instanz richtig zu initialisieren.
- Ein zusätzliches Funktionenpaar *Confirm()* auf dem Server und *IsConfirmed()* auf dem Client erlaubt die starke Synchronisation von Instanzen der Klasse *ClogMessage*. Im Rahmen des Report-Dienstes ist dies erforderlich, um eine Sicherheit für den korrekten Empfang einer Mitteilung auf dem Server zu gewährleisten.
- Im Rahmen dieses Report-Dienstes werden nur zwei Klassen zur Datenübertragung verwendet. *CLogEntry* repräsentiert eine Mitteilung während eine Instanz der Klasse *CLogMultipleMessage* einen Vektor von Instanzen der Klasse *CLogEntry* überträgt. Diese Differenzierung hat den Vorteil, dass bei der Versendung von mehreren Mitteilungen gleichzeitig diese nicht einzeln synchronisiert werden müssen. Die einmalige Bestätigung des Servers reicht aus. So können auch größere Mengen von Mitteilungen versendet werden, ohne durch die Synchronisation Performance zu verlieren.

5. Nachversenden von Mitteilungen

Um das Nachversenden von Mitteilungen sicher zu gewährleisten, ist der *CLogClient* auf die Fragestellung hin untersucht worden, ob Abläufe

im Rechnerverbund durch Mitteilungen auch im Nachhinein rekonstruiert werden können. Dies ist insbesondere wichtig, wenn der lokale Rechner keine Verbindung zum Server aufnehmen kann oder ein Rechner mutwillig ausgeschaltet wird. Mittels eines Zustandsdiagramms ist versucht worden, sich diesem Idealfall ohne Nachrichtenverlust möglichst gut anzunähern (vgl. Abbildung 2).

Auf dem lokalen Rechner legt der *CLogClient* im Bedarfsfall drei verschiedene Dateien mit zwischengespeicherten Mitteilungen an, welche mit *log*, *cpy* und *old* bezeichnet sind.

Im Zustand *NORMAL* - der Server ist verfügbar - befindet sich nur die Datei *old* auf dem lokalen System, welche gleichzeitig mit dem Attribut *ARCHIVE* versehen ist. Das Attribut kennzeichnet die Daten in der Datei *old* als bereits an den übergeordneten Server versendet und bestätigt. Werden Mitteilungen versendet, so verbleibt *CLogClient* in diesem Zustand. Bei Start des Systems wird die Existenz der Dateien und das Attribut *ARCHIVE* der Datei *old* überprüft und in diesen Zustand *NORMAL* gewechselt, wenn sich das lokale System bei letztem Betrieb ebenfalls im Zustand *NORMAL* befunden hat.

Tritt während des Betriebs ein Netzwerkfehler auf - *ERROR DETECTED* - so wird die Mitteilung, die nicht versandt werden kann, in der Datei *log* gespeichert. Alle weiteren, vom lokalen System generierten, Mitteilungen werden ebenfalls in dieser Datei *log* gespeichert. *CLogClient* wechselt in den Zustand *TEST SERVER*. In diesem Zustand wird auf dem *CLogClient* ein zusätzlicher Thread gestartet, der mit der Operation *Test* in regelmäßigen Zeitabständen versucht, mit der ersten Mitteilung aus der Datei *log*, die in diesem Zustand immer vorhanden ist, den Server zu kontaktieren.

Gelingt dies, benennt der Thread zunächst die Datei *log* in die Datei *cpy* um. Dies ist notwendig, damit bei einem nochmaligen Netzwerkausfall während des Nachversendens von Mitteilungen diese sich in der Datei *cpy* befinden, und nicht in der Datei *log* überschrieben werden. Es können also nebenläufig von *CLogControl* durch den Aufruf einer Methode *WriteEntry()* Mitteilungen bei nochmaligem Ausfall der Netzwerkverbindung in die Datei *log* zwischen gespeichert und Daten aus dem Arbeitsspeicher, die noch nicht versendet worden sind, von *CLogClient* in die Datei *cpy* zurückgeschrieben werden.

Unabhängig davon, ob der Vorgang *send old data* erfolgreich verlaufen ist oder nicht wird, zunächst die Datei *old* entfernt und die Datei *cpy* in *old* umbenannt. Die Datei *old* bekommt dabei das Attribut *NOARCHIVE*, da

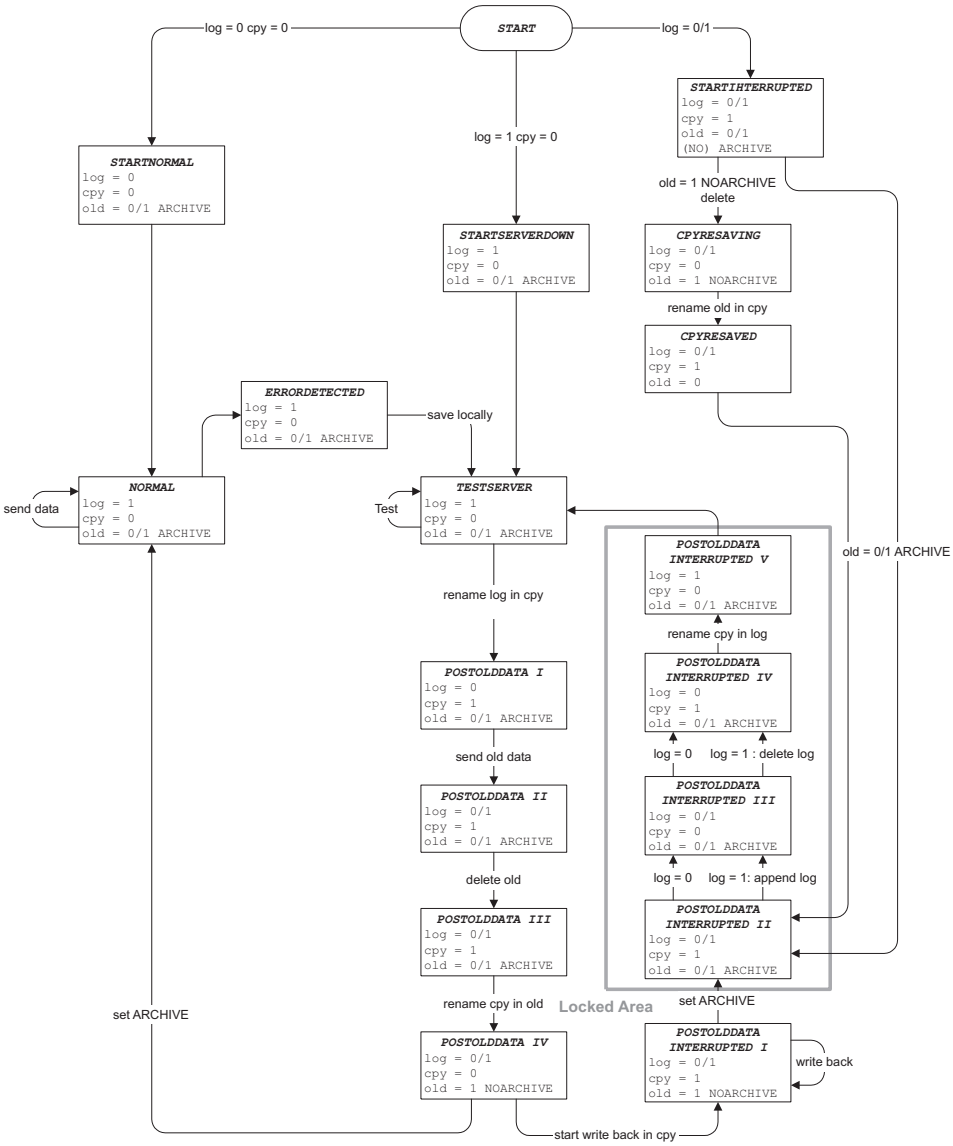

Abbildung 2. Zustandsdiagramm des CLogClient

noch nicht ermittelt worden ist, ob die Mitteilungen erfolgreich vom Server bestätigt worden sind. Dies wird erst im Zustand *POSTOLDDATA IV* er-

mittelt.

Sind die Daten erfolgreich versandt worden, so ist die Datei *log* leer, da der Server erreichbar und *CLogControl* durch den Aufruf der Methode *WriteEntry()* nebenläufig Mitteilungen an den Server versenden konnte. Mit dem Setzen des Attributs *ARCHIVE* der Datei *old* gesetzt wird wieder in den Zustand *NORMAL* gewechselt. Der in *TEST SERVER* zusätzlich eingerichtete Thread terminiert.

Ist beim Nachversenden von Mitteilungen ein Netzwerkfehler aufgetreten, so müssen alle noch nicht versandten und vom Server bestätigten Mitteilungen in die Datei *cpy* zurückgeschrieben werden. Wird der Rechner genau in diesem Zustand *POSTOLDDATAINTERRUPTED I* ausgeschaltet, erkennt das System beim Neustart im Zustand *STARTINTERRUPTED*, dass die Datei *old* zwar vorhanden, aber das Attribut *NOARCHIVE* gesetzt ist. Die Mitteilungen werden dann im Zustand *CPYRESAVING/RESAVED* in die Datei *cpy* zurückgespeichert, da diese noch nicht vom Server bestätigt worden sind. Ist hingegen die Datei *cpy* vorhanden und das Attribut ARCHIVE der Datei *old* gesetzt, so ist das System im Zustand *POSTOLDDATAINTERRUPTED II* und folgende unterbrochen worden.

Der grau markierte Bereich *Locked Area* kennzeichnet die wenigen Zustände, in denen das lokale System (*CLogControl*) weder eine Mitteilung an den Server versenden noch auf die Datei *log* zugreifen kann. Die Anwendung ist darauf optimiert worden, dass sich der *CLogClient* in nur sehr kurzen Zeitbereichen in dieser *Locked Area* befindet und somit die Datensicherheit nur für einen kurzen Augenblick in Gefahr ist. *CLogControl* wird in der Locked Area durch ein Semaphore innerhalb der Methode *WriteEntry()* davon abgehalten, auf die Datei *log* schreibend zuzugreifen. Der Thread innerhalb von *CLogControl* ist folglich kurzfristig blockiert. Innerhalb dieses Bereiches *Locked Area* wird die Datei *log* an die Datei *cpy* angefügt, die Datei *log* gelöscht und darauf die Datei *cpy* in die Datei *log* umbenannt. Durch diese Vorgehensweise geht der Client wieder in den Zustand *TEST SERVER* über, der bereits oben beschrieben worden ist.

6. Anwendung

Eine Instanz der Klasse *CLogControl* ist global im Projekt definiert worden. Mit Hilfe der Methode *WriteLogEntry()* können in jedem Teil des gesamten Projektes Mitteilungen an den übergeordneten Server versendet werden. Zusätzlich erlaubt es *CLogServer*, Mitteilungen über das Netzwerk entgegenzunehmen und über *CLogControl* und *CLogClient* wiederum einem übergeordneten zentralen Server zukommen zu lassen. In einem

außenstehenden Projekt muss nur die Klasse *CLogClient* und die abhängigen Klassen integriert werden, um einen zentralen Server zu kontaktieren.

7. Implementierung als Dienst

Der Report-Dienst ist als *Microsoft Windows NT - Dienst* implementiert worden. Mittels des *Service Control Manager* (SCM) kann der Dienst direkt beim Hochfahren des Systems gestartet werden. Der Dienst ist auch aktiviert, wenn kein Benutzer eingeloggt ist.

In der Systemsteuerung ist eine weitere Systemsteuerungsoption (Dateiendung **.cpl*) integriert worden. Diese besteht aus einer *DLL*, die im Rahmen einer Schnittstellendefinition Einsprungpunkte für das Betriebssystem zur Verfügung stellen muss. Bei einem Doppelklick auf das entsprechende Symbol in der Systemsteuerung wird diese *DLL* aufgerufen und es erscheint eine grafische Benutzungsoberfläche, die es erlaubt, Parameter und Schalter von *CLogServer*, *CLogControl* und *CLogClient* zu konfigurieren (vgl. Abbildung 3).

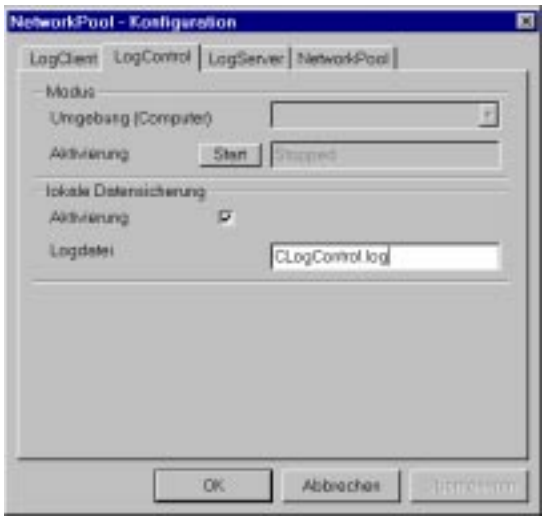

Abbildung 3. Systemsteuerungsoption für CLogControl

Die Systemsteuerungsoption kommuniziert über eine INI-Datei mit dem Dienst. Parameter und Schalter werden nach jeder Modifikation in der Systemsteuerungsoption neu in die INI- Datei geschrieben und dem Dienst über den SCM eine Mitteilung SERVICE_CONTROL_UPDATE geschickt. Diese

veranlasst den Dienst, die INI-Datei zu lesen und Zustandsvariablen zu aktualisieren. Der Wert dieser Zustandsvariablen wird wiederum in der Systemsteuerungsoption angezeigt, die ebenfalls in regelmäßigen Zeitabständen die INI-Datei auf Veränderungen überwacht.

8. Zusammenfassung

Der Report-Dienst erlaubt es, kurze Mitteilungen aus dem lokalen System oder über eine Netzwerkverbindung entgegenzunehmen. Diese werden in einer Logdatei in Tabellenform gespeichert, so dass diese Datei anschließend ausgewertet werden kann. Der *CLogClient* ist mit Hilfe eines Zustanddiagramms so optimiert worden, dass Mitteilungen im Störungsfall auf dem lokalen System zwischen gespeichert und bei reaktiviertem Netz im Nachhinein an den Server versendet werden können. Dabei ist jeder Programmschritt im *CLogClient* daraufhin untersucht worden, welche Folgen bei spontanem Abschalten des lokalen Rechners auftreten können. Die negativen Folgen eines solchen Abschaltens sind durch konsequente Umsetzung einer starken Synchronisation auf dem lokalen persistenten Speichermedium realisiert worden. Durch Definition eines weiteren Datentyps *CLogMultipleMessage* sind durch die starke Synchronisation resultierende Nachteile in der Performance des Report-Dienstes kompensiert worden.

9. Ausblick

Im Rahmen einer Diplomarbeit soll der Kommunikationsmechanismus derartig erweitert, dass auch entfernte Funktionsaufrufe ermöglicht werden. Zielsetzung ist die Verteilung von Analyseprozessen auf den gesamten Rechnerverbund zur gleichmäßigen Auslastung der verfügbaren Ressourcen (Load balancing). Ein lokaler Prozess soll auch entfernte ungenutzte Rechenkapazitäten ausnutzen können, um z.B. umfangreiche Logdateien auf bestimmte Muster hin zu untersuchen. Dies ist notwendig, da das verteilte Sicherheitsmanagement im Hintergrund zu aktuell von Nutzern verwendeten Applikationen ablaufen soll. Der lokale Nutzer eines Rechners soll nicht durch den Start eines Analyseprozesses benachteiligt werden. Analyseprozesse werden somit auf diejenigen Rechner ausgelagert, die zum aktuellen Zeitpunkt am wenigsten ausgelastet sind. Zur Bestimmung des optimalen Rechners soll ein stochastischer Ansatz verfolgt werden.

Dank

Herrn Prof. Dr.-Ing. Wolfgang Weber sei für die bisherige und auch weiterführende Unterstützung gedankt, durch die es möglich gewe-

sen ist, diesen Ansatz zu verfolgen und auch in weiteren Arbeiten diesen fortzuführen.

REFERENCES

1. H. Balzert: *Lehrbuch der Softwaretechnik.* Software-Entwicklung, Spektrum Akademischer Verlag, Heidelberg 1998.
2. H. Balzert: *Lehrbuch der Softwaretechnik.* Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung, Spektrum Akademischer Verlag, Heidelberg 1998.
3. T. Droste: *Rechnerorientierte Datenaufnahme und dessen Verteilung zur Analyse.* Facta Universitatis, Series: Electronic and Energetics vol. 12, No.3 (1999), pp. 47-55. (`http://factaee.elfak.ni.ac.yu/facta9903/facta5.html`).
4. T. Droste: *Aspekte eines verteilten Sicherheitsmanagements.* In: Patrick Horster (Hrsg.): Systemsicherheit, DuD-Fachbeiträge, Vieweg & Sohn, Braunschweig 2000.
5. K. Miller: *Professional NT Services.* Wrox Press, Birmingham 1998.