Universität Dortmund
Lehrstuhl Verteilte Systeme

Dr. Michael Zapf

Michael.Zapf@cs.uni-dortmund.de
Michael.Zapf@uni-kassel.de

vkiv@ls4.cs.uni-dortmund.de

Übung zur Vorlesung „Verteilt-kooperative Informationsverarbeitung“ - SS 2007

Lösungen zu Blatt 3

Aufgabe 3.1 – 15 Punkte

a) Migriert der Agent lediglich von der sicheren Heimatplattform auf eine fremde Plattform und schickt die Ergebnisse zurück oder migriert ohne Umwege nach Hause, bezeichnet man dies als „Single-Hop“-Mobilität. Würde der Agent mit einer weiteren Plattform kommunizieren oder dorthin migrieren, spricht man von „Multiple-Hop“-Mobilität. Welche Auswirkungen hat die Single-Hop-Mobilität auf die Sicherheit?

Bei der Verwendung von Single-Hop-Agenten nähert man sich wieder den traditionellen Ausführungsformen, etwa Remote Evaluation. Das Schicken eines Agenten an eine entfernte Stelle und die Rückkehr desselben oder dessen Ergebnissen gleicht einer Klient-Server-Struktur.

Hier sind Sicherheitsmechanismen einsetzbar, die von der Klient-Server-Welt bekannt sind. Es kann ein Vertrauensniveau zwischen der Heimatstelle und der entfernten Stelle aufgebaut werden, etwa durch Aufbauen einer gesicherten Verbindung mittels Kryptografie und dafür eigens verifizierten Schlüsseln.

Malicious-Agent-Angriffe werden durch die Verwendung längerer Reiserouten gefährlicher: Die Angriffe auf verschiedene Stellen haben bei Single-Hop-Agenten alle denselben Ausgangspunkt, sodass dieser leicht nachzuweisen ist. Die Herkunft eines herumreisenden Agenten muss erst durch Analyse dessen Zustands bestimmt werden.

Multiple-Hop-Agenten sind durch Malicious-Host-Angriffe besonders gefährdet:

Ein Fehlverhalten der Stelle kann bei einem Single-Hop-Agenten unmittelbar nachgewiesen werden, da der Agent keine andere Stelle mehr besucht. Verwendet der Agent mehrere Schritte (Multi-Hop), so kann ein bösartiger Einfluss nicht mehr zweifelsfrei einer Stelle zugeordnet werden.

Ferner ist es möglich, den Zustand eines Single-Hop-Agenten zu signieren, bevor er das erste Mal migriert. Die Zielstelle kann die Signatur überprüfen. Wenn der Agent seine Reise fortsetzen würde, wäre der Zustand verändert, und die Signatur müsste erneuert werden. Dies ist nur möglich, wenn ein privater Schlüssel mitgeführt würde. Dies gefährdet hochgradig die Vertraulichkeit des privaten Schlüssels, da dieser ausgelesen werden könnte.

Aus diesem Grund ist die Vertraulichkeit der Daten des Agenten bei Multi-Hop nicht gewährleistet. Der Agent könnte seinen Zustand verschlüsseln. Jedoch kann er ihn nur mit seinem privaten Schlüssel wieder entschlüsseln. Liegt dieser auf der Heimatstelle, muss er wiederholt zurückkehren, was ihn zu einem Single-Hop-Agenten werden lässt.

Schließlich ist bei Multi-Hop-Agenten die Reiseroute (Itinerary) ein mögliches Angriffsziel. Diese kann von bösartigen Stellen modifiziert werden, um den Agenten zum Besuch bestimmter anderer Stellen zu bringen. Single-Hop-Agenten müssen eine Reiseroute nicht bearbeiten und sind für diesen Angriff somit nicht anfällig.

b) Erklären Sie (in eigenen Worten) anhand des Textes in

http://java.sun.com/javase/6/docs/technotes/guides/concurrency/threadPrimitiveDeprecation.html

warum in Java gewisse Methoden zur Threadbeeinflussung nicht verwendet werden sollen. Führen Sie jeweils ein konkretes Beispiel an.

Als Beispiel einer Datenstruktur, die mit Monitoren geschützt wird, nehme man einen BoundedStack, der die Methoden push und pop besitze und maximal 10 Speicherplätze habe. „null“ darf nicht eingetragen werden.

```
public class BoundedStack {
  Object[] m_aList = new Object[10];
  int m_nPosition = 0;
  public synchronized Object pop() {
    if (m_nPosition==0) return null;
    Object obj = m_aList[m_nPosition-1];
    m_nPosition--;
    return obj;
  }

  public synchronized void push(Object obj) {
    if (obj==null) return;
    if (m_nPosition==10) return;
    m_nPosition++;    (*)
    m_aList[m_nPosition-1] = obj;
  }
}
```

Angenommen, ein Thread A führt gerade push() aus und erhält ein **Thread.stop**, wenn er gerade (*) ausgeführt hat. Der Stack gerät dadurch in einen inkonsistenten Zustand: Der Zeiger wird weitergerückt, aber es wird kein Wert eingetragen.

Ein anderer Thread B wartet am Monitor von BoundedStack darauf, in die Methode pop eingelassen zu werden. Dadurch, dass A ein stop() bekommen hat, verlässt A den Monitor und gibt diesen frei. B darf nun in die pop-Methode eintreten und wird „null“ lesen. Die push-Methode sichert zwar zu, dass keine „null“ eingetragen wird, aber dies wird durch den inkonsistenten Zustand verletzt.

Wenn A hingegen ein **Thread.suspend** erhielt, so bleibt er im push stehen. Dadurch wird er den Monitor nicht zurückgeben. Der Stack ist blockiert, bis B **Thread.resume** aufruft.

Wenn nun aber Thread.resume nur von B aufgerufen wird, aber B noch vorher ein push oder pop aufruft, dann blockieren sich A und B gegenseitig am Monitor: A wurde von B schlafen gelegt, A kann den Monitor nicht verlassen, B kann nicht in den Monitor eintreten, und A wird nie aufgeweckt, da B einem linearen Programmablauf folgen muss. Ein Deadlock entsteht.

Thread.destroy wurde nie implementiert, hätte aber ähnliche Wirkungen wie Thread.stop.

Zusatz (hat nicht direkt mit der Threadbeeinflussung zu tun):

Mit **Runtime.runFinalizersOnExit** wird bestimmt, dass die finalize-Methoden aller Objekte vor dem Beenden der Java-VM ausgeführt werden, sofern sie noch nicht zuvor aufgerifen wurden. Normalerweise würden die finalize-Methoden nur aufgerufen, wenn es zu diesem Objekt keine Referenzen mehr gibt und das Objekt dadurch über die „Garbage collection“ entsorgt werden soll. Werden die finalize-Methoden jedoch zwangsweise bei Beendigung ausgeführt, so kann es passieren, dass ein anderer aktiver Thread das Objekt bearbeitet, während der Systemthread in finalize eine andere Aktion zugleich auf dem Objekt ausführt.

Bespielsweise könne beim Stackbeispiel Folgendes angefügt werden:

```
public void finalize() throws Throwable {
  m_aList = null;
}
```

Normalerweise würde dieser Code nur ausgeführt, wenn niemand mehr eine Referenz auf den Stack hätte – insbesondere kein Thread. Es dürfte also zu keinen unerwünschten Effekten kommen. Wird nun System.exit aufgerufen bei gesetztem **runFinalizersOnExit**, dann würde dieser Code zwangsweise ausgeführt. Wenn nun gerade ein Thread an der Stelle (*) steht, wird es in der nächsten Zeile zu einer Exception kommen, da das Feld entfernt wurde. Es könnte auch passieren, dass eine Aktion in finalize dazu führt, dass der Finalizer-Thread an einem Monitor wartet, sodass das ganze System in einen Deadlock gerät.

c) Legen Sie dar, wie ein Reputationssystem die Sicherheit in Agentensystemen gewährleisten kann. Geben Sie Details an, wie ein solches System realisiert werden kann. Hinweis: Eine gute Reputation (öffentliches Ansehen, Ruf) „öffnet Türen“.

Das „Malicious-Host-Problem“ ist sehr wahrscheinlich nicht allgemein lösbar. (Eine „echte“ Lösung würde zugleich alle Formen von Vertrauensproblemen zwischen Kommunikationspartnern lösen, was ebenfalls nicht erreichbar ist.) Man kann jedoch ein Reputationssystem vorsehen:

Ein bekanntes Beispiel eines Reputationssystems findet sich in Auktionsportalen wie eBay. Hier entspricht der Host dem Verkäufer und der Agent dem Käufer; dies kann aber auch andersherum genutzt werden, wenn sich ein Verkäufer über die Verlässlichkeit eines Käufers Klarheit verschaffen möchte.

Beim Auftreten von negativen Unregelmäßigkeiten sinkt der Ruf des Teilnehmers; entsprechend kann seine Konkurrenz (als Käufer oder Verkäufer) einen Vorteil daraus ziehen, dass Interessenten den Teilnehmer künftig meiden. Mit der Zeit kann der „schlechte“ Ruf wieder aufgebessert werden. Die Hoffnung besteht, dass ein schlechter Ruf in einem elektronischen Marktplatz einen so schwer wiegenden Nachteil darstellt, dass sich durch diesen Druck ein Regulativ ausbildet, das die technische Unzulänglichkeit des Nachweises von bösartigen Stellen aufwiegt.

Die Reputation kann in Form eines Zertifikats ausgestellt werden, das dann in kurzen Abständen erneuert werden muss. Kunden melden ihre Erfahrungen einer Aufsichtsstelle, die dann diese Zertifikate ausstellt. Es könnten auch Testagenten dieser Aufsichtsstelle zu den unterschiedlichen Teilnehmern verschickt werden und deren Verhalten analysiert werden.

Weiterhin kann auch ein „Web-of-trust“, ähnlich jenem im PGP-System, aufgebaut werden. In einem solchen Netzwerk bescheinigen sich die Teilnehmer gegenseitig ihr Vertrauen.

Aufgabe 3.2 – 20 Punkte

Quellcode auf unserer Webseite erhältlich.

Agent:

```
import AMETAS.agentdev.*;
import AMETAS.data.*;
import AMETAS.data.type.*;

public class Uebung3Agent extends AMETASAgent {

    /** Zustand des Agenten. Der Agent ist gerade gestartet worden. */
    private final static int FIRST = 0;

    /** Zustand des Agenten. Der Agent hat den Auftrag erhalten und ist an die entfernte
        Stelle migriert. Dort setzt er die Nachricht an den Dienst ab. */
    private final static int MIGRATED = 1;

    /** Zustand des Agenten. Der Agent wartet auf die Antwort des Dienstes. */
    private final static int WAITING = 2;

    /** Zustand des Agenten. Der Agent hat die Nachricht des Dienstes bekommen und kehrt
        zurück.*/
    private final static int GETBACK = 3;

    /** Zustand des Agenten. Der Agent ist nach Hause gekommen. */
    private final static int BACK = 4;

    /** Zustand des Agenten. Alles erledigt. */
    private final static int DONE = 5;

    /** Der Zustand des Agenten. */
    int m_nState = FIRST;

    /** Der Auftrag. Die erste Komponente. */
    String m_sZahl;

    /** Der Auftrag. Die zweite Komponente. */
    String m_sName;

    /** Das Resultat. */
    String m_sResultat;

    /** Die ID des Dienstes. Der Agent merkt sich diese ID, um die richtige Antwort
        zu identifizieren. */
    AMETASPlaceUserID m_puidService = null;

    /** Die ID der Anfrage des Benutzeradapters. Diese dient als Referenz für die Antwort. */
    AMETASMessageID m_midRequest = null;

    /** Die ID des Benutzeradapters. Damit der Agent weiß, wem er antworten soll. */
```

```
AMETASPlaceUserID m_puidUA = null;

/** Es ist ein Fehler aufgetreten. */
boolean m_bFehler = false;

/** Konstruktor. Der Anfangszustand für auf FIRST gesetzt. */
public Uebung3Agent() {
    super("Uebung3Agent");
    m_nState = FIRST;
    m_bFehler = false;
}

/** Hauptmethode des Agenten. */
public void invoke() {
    while (m_nState!=DONE) {
        switch (m_nState) {
        case FIRST:
            // Wir holen uns die Initialisierungsnachricht - es ist garantiert,
            // dass sie in der Mailbox liegt.
            AMETASMessage[] amesInit = m_Driver.getMessages(true);
            for (int i=0; i < amesInit.length; i++) {
                if (amesInit[i].getSubcategory()==AMETASMessage.INIT) {
                    m_sName = (String)(amesInit[i].getBody())[0];
                    m_sZahl = (String)(amesInit[i].getBody())[1];
                    m_midRequest = amesInit[i].getID();
                    m_puidUA = amesInit[i].getSenderID();
                    break;
                }
            }
            if (m_sZahl==null) {
                m_Driver.output("Fehler, Initnachricht nicht gefunden");
                m_nState = DONE;
                m_bFehler = true;
                break;
            }

            try {
                // Erst den Zustand setzen, denn nach der Migration kommt nichts mehr
                m_nState = MIGRATED;
                m_Driver.go("place1.vkiv.uniks.ametas.");
            }
            catch (MigrationException mx) {
                m_Driver.output("Fehler, konnte nicht migrieren: " + mx.getMessage());
                m_nState = DONE;
                m_bFehler = true;
            }
            break;
        case MIGRATED:
            // Wir suchen den Dienst
            AMETASMediationResult[] amr = m_Driver.request(
                    new AMETASMediationRequest("srv_blatt3", true));
            if (amr==null) {
                // Fehler bei der Mediation
                m_nState = GETBACK;
                m_sResultat = "*** Konnte Dienst nicht finden - Mediationsfehler";
                m_bFehler = true;
            }
            else {
                if (amr.length==0) {
                    // Keinen Dienst gefunden!
                    m_nState = GETBACK;
                    m_sResultat =
                          "*** Konnte Dienst nicht finden - er läuft anscheinend nicht";
                    m_bFehler = true;
                }
                else {
                    // Dienst gefunden; Senden der Nachricht
                    m_puidService = amr[0].getPlaceUserID();
                    String[] asBody = new String[3];
                    asBody[0] = "meineZahl";
                    asBody[1] = m_sZahl;
                    asBody[2] = m_sName;
                    AMETASMessage mes = new AMETASMessage(m_puidService, asBody);
                    try {
                        submitMessage(mes);
                        m_nState = WAITING;
                    }
                    catch (IllegalArgumentException iax) {
```

```
                        m_nState = GETBACK;
                        m_sResultat =
                         "*** Konnte mit Dienst nicht kommunizieren: " + iax.getMessage();
                        m_bFehler = true;
                    }
                }
            }
            break;
        case WAITING:
            // Wir warten auf Antwort. Diese Antwort stammt vom Dienst. Wir wissen
            // noch, wie die ID dieses Dienstes lautet.
            AMETASMessage[] ames = m_Driver.getMessages(true);
            if (ames==null) m_Driver.idle(1000); // einen Moment warten
            else {
                for (int i=0; i < ames.length; i++) {
                    if (ames[i].getSenderID().equals(m_puidService)) {
                        Object[] aBodyReply = ames[i].getBody();
                        if (((String)aBodyReply[0]).equals("Fehler")) {
                            m_bFehler = true;
                            m_sResultat = (String)aBodyReply[1];
                        }
                        else m_sResultat = (String)aBodyReply[0];
                        m_nState = GETBACK;
                        break;
                    }
                    // else gehts uns nichts an
                }
            }
            break;
        case GETBACK:
            // Alles erledigt, heim gehts.
            try {
                m_nState = BACK;
                m_Driver.go(m_sHomePlace);
            }
            catch (MigrationException mx) {
                m_Driver.output("Fehler, konnte nicht migrieren: " + mx.getMessage());
                m_nState = DONE;
                m_bFehler = true;
            }
            break;

        case BACK:
            // Wieder zu Hause; der Benutzeradapter bekommt das Ergebnis
            String[] asBody = new String[2];
            if (m_bFehler) {
                m_Driver.output("*** Das war nicht erfolgreich: " + m_sResultat);
                asBody[0] = "Fehler";
                asBody[1] = m_sResultat;
            }
            else {
                m_Driver.output("ERFOLG! Ergebnis = " + m_sResultat);
                asBody[0] = m_sResultat;
            }

            AMETASMessage mesReply = new AMETASMessage(m_puidUA, m_midRequest, asBody);
            try {
                submitMessage(mesReply);
            }
            catch (IllegalArgumentException iax) {
                m_Driver.output("Fehler beim Senden des Ergebnisses; " +
                                iax.getMessage());
            }
            m_nState = DONE;  // Das war's. Die Schleife wird verlassen.
            break;
        default:
            m_Driver.output("Unbekannter Zustand: " + m_nState);
        }
    }
  }
}
```

Benutzeradapter (Oberfläche)

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
```

```
/** Oberfläche für den einfachen Benutzeradapter auf Blatt 3. */
public class Uebung3Frame extends JFrame implements WindowListener, ActionListener {

    private Uebung3UA m_ua;
    private JLabel m_jlResultValue;
    private JButton m_jbAgent;
    private JTextField m_jtName;
    private JTextField m_jtNumber;
    private Color m_colBackground;

    public Uebung3Frame(String sTitle, Uebung3UA ua) {
        super(sTitle);
        m_ua = ua;
        setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
        JPanel jp = new JPanel();
        JPanel jpName = new JPanel();
        jpName.setLayout(new BoxLayout(jpName, BoxLayout.LINE_AXIS));
        JLabel jlName = new JLabel("Name:");
        m_jtName = new JTextField(20);
        jpName.add(jlName);
        jpName.add(Box.createHorizontalStrut(5));
        jpName.add(m_jtName);

        JPanel jpNumber = new JPanel();
        jpNumber.setLayout(new BoxLayout(jpNumber, BoxLayout.LINE_AXIS));
        JLabel jlNumber = new JLabel("Zahl:");
        m_jtNumber = new JTextField(20);
        m_jtNumber.setText("0");
        jpNumber.add(jlNumber);
        jpNumber.add(Box.createHorizontalStrut(5));
        jpNumber.add(m_jtNumber);

        m_jbAgent = new JButton("Agent senden");

        JPanel jpResult = new JPanel();
        jpResult.setLayout(new BoxLayout(jpResult, BoxLayout.LINE_AXIS));
        JLabel jlResult = new JLabel("Resultat:");
        m_jlResultValue = new JLabel("-");

        jpResult.add(jlResult);
        jpResult.add(Box.createHorizontalStrut(5));
        jpResult.add(m_jlResultValue);

        jp.setLayout(new BoxLayout(jp, BoxLayout.PAGE_AXIS));
        jp.add(Box.createVerticalStrut(5));
        jp.add(jpName);
        jp.add(Box.createVerticalStrut(5));
        jp.add(jpNumber);
        jp.add(Box.createVerticalStrut(5));
        jp.add(m_jbAgent);
        jp.add(Box.createVerticalStrut(5));
        jp.add(jpResult);
        jp.add(Box.createVerticalStrut(5));

        m_colBackground = m_jlResultValue.getBackground();

        add(jp);
        addWindowListener(this);
        m_jbAgent.addActionListener(this);
        pack();
        setVisible(true);
    }


    /** Wird vom Benutzeradapter aufgerufen. Die Methode stellt das Ergebnis
        dar.
        @param aBody Nutzlast der Nachricht.
    */
    void showResult(Object[] aBody) {
        if (((String)aBody[0]).equals("Fehler")) {
            m_jlResultValue.setBackground(Color.red);
            m_jlResultValue.setText((String)aBody[1]);
        }
        else m_jlResultValue.setText((String)aBody[0]);
    }

    /** Wird aufgerufen, wenn der Knopf gedrückt wird. */
```

```
    public void actionPerformed(ActionEvent ae) {
        if (ae.getSource().equals(m_jbAgent)) {
            m_ua.output("Agent verschicken");
            m_jlResultValue.setText("Bitte warten...");
            m_jlResultValue.setBackground(m_colBackground);
            m_ua.sendAgent(m_jtName.getText(), m_jtNumber.getText());
        }
    }

    public void windowActivated(WindowEvent we) {
    }

    public void windowDeactivated(WindowEvent we) {
    }

    public void windowOpened(WindowEvent we) {
    }

    public void windowClosed(WindowEvent we) {
        m_ua.debug("closed");
    }

    public void windowClosing(WindowEvent we) {
        m_ua.debug("closing");
        m_ua.done();
    }

    public void windowIconified(WindowEvent we) {
    }

    public void windowDeiconified(WindowEvent we) {
    }
}
```

Benutzeradapter (Hauptklasse)

```
import AMETAS.agentdev.*;
import AMETAS.data.*;
import javax.swing.*;

/** Einfacher Benutzeradapter für Blatt 3. */
public class Uebung3UA extends AMETASUserAdapter {

    /** Referenz auf den Rahmen. Wir müssen dort noch Ausgaben vornehmen. */
    Uebung3Frame m_ua = null;

    /** Nichts tun, nur Nachrichten abfragen. */
    private final static int LISTEN=0;

    /** Agent verschicken. */
    private final static int SENDAGENT=1;

    /** Den Benutzeradapter beenden lassen. Dies geschieht dadurch, dass die
        invoke-Methode verlassen wird. */
    private final static int DONE=3;

    /** Die aktuelle Aufgabe. */
    int m_nTask;

    /** Die Kennung der Anfrage, die von hier an den Agenten geht. */
    AMETASMessageID m_midRequest = null;

    /** Eingabe im Namensfeld. */
    String m_sMyName;

    /** Eingabe im Zahlenfeld. */
    String m_sMyNumber;

    /** Konstruktor. */
    public Uebung3UA() {
        super("Uebung3UA");
        // Hier bitte keine Aktionen starten. Der SecurityManager erlaubt
        // es nicht, im Konstruktor "gefährliche" Aktionen durchzuführen
        m_nTask = LISTEN;
    }

    /** Sorgt dafür, dass der Benutzeradapter geschlossen wird. */
```

```
void done() {
    m_nTask = DONE;
    m_Driver.wakeup();        // Aufwachen!
}

/** Gibt Meldungen im Stellenlog aus. */
void debug(String sMessage) {
    m_Driver.output(sMessage);
}

/** Hauptmethode des Stellennutzers. */
public void invoke() {
    m_ua = new Uebung3Frame("Beispiel", this);
    m_Driver.output("Benutzeradapter gestartet.");
    while (m_nTask != DONE) {
        // Polling, wir haben es nicht eilig
        m_Driver.idle(500);
        switch (m_nTask) {
        case LISTEN:
            // Auf Antwortnachrichten achten
            if (m_midRequest!=null) {
                AMETASMessage[] ames = m_Driver.getMessages(true);
                if (ames!=null) {
                    for (int i=0; i < ames.length; i++) {
                        if (ames[i].getRepliedMsg().equals(m_midRequest)) {
                            m_ua.showResult(ames[i].getBody());
                            break;
                        }
                    }
                }
            }
            break;
        case SENDAGENT:
            String[] asBody = new String[2];
            asBody[0] = m_sMyName;
            asBody[1] = m_sMyNumber;
            AMETASMessage mesInit = new AMETASMessage(null, asBody);
            m_midRequest = null;
            try {
                // Hiermit wird ein Agent erzeugt und ihm gleich eine
                // Nachricht mitgegeben, so wie bei einem Konstruktor.
                // Deshalb ist die Empfängeradresse null.
                // Diese Nachricht bekommt automatisch die Subkategorie
                // AMETASMessage.INIT; der Agent kann also seine
                // erhaltenen Nachrichten danach durchsuchen.
                m_Driver.requestPUStartup("Uebung3Agent", mesInit);
                // Wir merken uns die ID, damit wir wissen, ob die
                // hier später eintreffende Nachricht zu dieser Anfrage
                // gehört.
                m_midRequest = mesInit.getID();
            }
            catch (CreationFailedException cfx) {
                error("Konnte Agent nicht starten: " + cfx.getMessage());
            }
            catch (CreationDeniedException cdx) {
                error("Durfte Agent nicht starten: " + cdx.getMessage());
            }
            m_nTask = LISTEN;
            break;
        case DONE:
            break;
        }
    }
    m_Driver.output("So, fertig.");
}

/** Bringt den Benutzeradapter dazu, einen Agenten loszuschicken. Dabei
    werden der eingegebene Name und Wert hierher übergeben und dann in
    Instanzvariablen abgelegt.
    @param sName Eingegebener Name.
    @param sNumber Eingegebene Zahl.
*/
void sendAgent(String sName, String sNumber) {
    m_sMyName = sName;
    m_sMyNumber = sNumber;
    m_nTask = SENDAGENT;
    m_Driver.wakeup();
}
```

```
    /** Gibt ein Fehlerfenster aus.
        @param sMessage Fehlernachricht.
    */
    void error(String sMessage) {
       JOptionPane.showMessageDialog(m_ua, sMessage);
    }
}
```