

## Flash und Java: verschiedene Wege für die Kommunikation zwischen Flash und Java

# Helles Licht am Himmel

■ VON THOMAS KIESL

Gerne wird Flash als banales Tool für Banner-Animationen und Browser-Spiele abgetan. Die Möglichkeiten einer gekonnten Anwendung von Flash bei komplexeren Frontend-Projekten werden dabei häufig unterschätzt. So führt spätestens bei der Integration von multimedialen Inhalten kein Weg mehr an Flash vorbei, das hier eine Fülle von unbestrittenen Vorteilen bietet.

Während „seriöse“ Internet-Applikationen meist farblos und uninspiriert wirken, lassen sich mit Flash optisch ansprechende Frontends erstellen. Technologien wie Ajax ermöglichen zwar das Erstellen von komponenten- statt der herkömmlichen seitenorientierten Applikationen und erhöhen somit erheblich die Benutzerfreundlichkeit. Für zielgruppenorientierte multimediale Anwendungen fehlen ihnen jedoch die direkte Unterstützung zur Wiedergabe der Audio- und Videodateien (bzw. Streams) und die Anzeige von skalierbaren Vektorgrafiken. Hierfür sind Plug-ins erforderlich, die von verschiedenen Herstellern in großer Zahl angeboten werden, beim User allerdings nicht vorausgesetzt werden können. Stattdessen müssen den Benutzern verschiedene Formate zur Auswahl angeboten werden, da die meisten Plug-ins eigene Formate nutzen – die Folge sind deutlich erhöhte Entwicklungskosten.

Stark frequentierte Angebote wie beispielsweise YouTube ([www.youtube.com](http://www.youtube.com)) oder MyVideo.de ([www.myvideo.de](http://www.myvideo.de)) bieten ihre Inhalte ausschließlich über den Flash Player an und sorgen so für eine zunehmende Verbreitung des aktuellen Flash-Plug-ins. Selbst wenn das Plug-in beim User nicht vorhanden sein sollte, ist die Installa-

tion für Laien unkompliziert ohne Rechner- oder Browser-Neustart durchführbar.

Neben der direkten Unterstützung multimedialer Inhalte bietet Flash seit der Einführung von ActionScript 2 eine vergleichsweise mächtige, objektorientierte Script-Sprache, mit der sich Webapplikationen gezielt dynamisieren lassen. Somit ist Flash das ideale Werkzeug, um anspruchsvolle, interaktive und zielgruppenorientierte Webapplikationen zu erstellen.

Für dynamische und interaktive Webapplikationen ist jedoch eine gut abgestimmte Kommunikation mit dem Server zwingend notwendig. In diesem

Artikel werden Beispiele für die Kommunikation über XML Sockets, das proprietäre Action Message Format (AMF) von Adobe und die Kommunikation über HTTP vorgestellt. Der Quellcode (sowohl die Java-Sourcen als auch die ActionScript2-Sourcen) und die Konfigurationsdateien sind auf der Heft-CD enthalten.

### Einfache Kommunikation über HTTP

Die einfachste Form der Kommunikation ist ein einzelner HTTP Request und eine anschließende HTTP Response durch den Server. Flash bietet dazu die Klasse *LoadVars* an: Es sollten zwei Instanzen

#### Listing 1

*send\_email fla* (ActionScript 2.0)

```
sendMail=function() {  
    /**  
     * Loadvars-Objekt zum Senden der Anfrage  
     */  
    var sendLv:LoadVars = new LoadVars();  
    /**  
     * Ergebnis : Loadvars-Objekt  
     */  
    var resultLv:LoadVars = new LoadVars();  
  
    sendLv.action = "sendEmail";  
    sendLv.email = escape("thomas.kiesl@bluemars.net");  
    sendLv.subject = escape("Dies ist eine Testemail");  
    sendLv.subject = escape("Hallo Welt");  
  
    sendLv.sendAndLoad("http://localhost:8080/flash  
        comm/sendEmail.jsp", resultLv, "POST");  
  
    /** Definition des EventHandlers für die Antwort des  
     * Servers  
     * resultLv.onLoad = function(success:Boolean):Void {  
     * if (success) {  
     *     for (var i in this) {  
     *         /**  
     *          * Ausgabe aller Rückgabewerte des Servers  
     *          */  
     *         trace(i + "-" + this[i]);  
     *     }  
     * } else {  
     *     trace("loading failed");  
     * }  
     * };  
     * }  
     */  
}
```



Quellcode auf CD



Abb. 1: Flash Client zum Versenden einer E-Mail

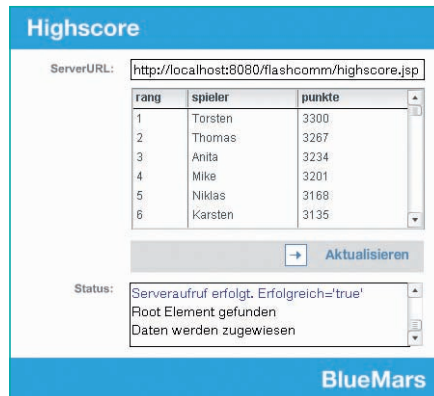


Abb. 2: Anzeige einer Highscore in einem Flash Client

(einmal für den Request und einmal für die Response) instanziiert werden. In diesem Beispiel wird eine E-Mail mittels Flash-Java-Kommunikation verschickt. Dazu ist das ActionScript aus Listing 1 clientseitig zur Übergabe der E-Mail-Daten an den Webserver nötig.

Der Code-Ausschnitt zeigt eine Funktion zum Versenden einer E-Mail. Bei einer

Antwort des Servers wird die Funktion *onload* des Objektes *resultLv* aufgerufen. Innerhalb der Methode kann auf die Antwort des Servers reagiert werden. Die Anfrageparameter werden nach der eingestellten Methode (Mechanismen POST oder GET des HTTP-Protokolls) übertragen. Die zu übertragenden Daten sollten URL-kodiert sein (nach RFC 1738), um Sonderzeichen

sicher ans andere Ende zu bringen. URL-Kodierung und URL-Dekodierung erfolgen mit den Methoden *escape(text:String)* und *unescape(text:String)*, serverseitig erfolgt die Kodierung bzw. die Dekodierung mit *java.net.URLDecoder.decode(text, "UTF-8")*; und *java.net.URLEncoder.encode(text, "UTF-8")*;

Die Antwort des Servers ist im einfachsten Fall ein String, der nach folgendem Schema aufgebaut ist: *&key1=value1 &key2=value2...* Um den Kommunikationsaufwand möglichst gering zu halten und um eine Manipulation (z.B. durch einen modifizierten Client) serverseitig ausschließen zu können, sollten die Parameter der Anfrage zunächst client- und anschließend bei Erfolg serverseitig validiert werden.

Der Server kann dem Client mit Fehlercodes die Gültigkeit der Anfrage zurückgeben, z.B. im Fehlerfall *&errors=2 &error\_1=email &error\_2=subject* oder bei Erfolg *&errors=0 &status=sent*.

## Listing 2

### highscore fla (ActionScript 2.0)

```

/**
 * Loadvars-Objekt zum Senden der Anfrage
 */
var sendLv:LoadVars = new LoadVars();
/**
 * XML-Objekt für Speicherung der Antwort
 */
var xmlReply:XML = new XML();
/**
 * Ignorieren von Leerzeichen/Zeilenumbrüchen
 */
xmlReply.ignoreWhite = true;
/**
 * Definition der URL
 */
var highscore_url:String = "http://localhost:8080/flashcomm/highscore.jsp";
/**
 * Array - Highscore-Ergebnisse
 */
var entries:Array = new Array();
/**
 * Event-Handler für die Server-Antwort
 * @param success Boole'scher Wert, ob der Aufruf des Servers erfolgreich war.
 */
xmlReply.onLoad = function(success:Boolean):Void {
    if (success) {
        printStatus("Serveraufruf erfolgt. Erfolgreich="
            + success + """, "server");
        var node:XMLNode = this.childNodes;
        for (var i in node) {
            /**
             * Die Rückgabe erfolgt in Form eines XML-Dokumentes,
             * nur das XML-Element-Dokument ist relevant
             */
            if (node[i].nodeName == "root") {
                printStatus("Root Element gefunden");
                /**
                 * Alle Kinder-Elemente des Root-Elements durchlaufen
                 */
                for (var j:Number = 0; j < node[i].childNodes.length; j++) {
                    var entryNode:XMLNode = node[i].childNodes[j];
                    var rangInt:Number = Number(entryNode.attributes.rang)+1;
                    var punkteInt:Number = Number(entryNode.attributes.punkte);
                    entries.push({rang:rangInt, spieler:entryNode.attributes.spieler, punkte:punkteInt});
                }
            }
        }
    } else {
        trace("loading failed");
    }
};
/**
 * Starten des Aufrufs an den Server als POST-Anfrage
 */
sendLv.sendAndLoad(highscore_url, xmlReply, "POST");

```

## Rückgabe eines XML-Dokumentes

Wenn die Struktur der Antwort komplexer aufgebaut ist, sollte diese in einem XML-Format zurückgegeben werden. Dabei sollte darauf geachtet werden, dass nicht zu große XML-Dokumente und nur solche mit einer möglichst flachen Struktur an den Flash-Client geschickt werden. Ansonsten können schnell Performanceprobleme entstehen.

In Listing 2 erhält der Flash-Client nach der Anfrage die Highscore-Tabelle eines Browser-spiels. Neben dem rekursiven Durchsuchen des XML-Baumes besteht auch die Möglichkeit, über XPath darauf zuzugreifen. Dazu bieten sich von Adobe XPathAPI und Bibliotheken wie z.B. von xfactor Studio ([www.xfactorstudio.com](http://www.xfactorstudio.com)) an. Allerdings sollten aufgrund der recht langsamen XML-Verarbeitung von Flash diese Zusatzbibliotheken nur gezielt für komplexe Zugriffe eingesetzt werden. Dagegen ist das rekursive Verarbeiten der XML-Knoten mit den Standardmethoden von Flash aus Performancegründen durchaus zu empfehlen.

## Dauerhafte Verbindungen

Benötigt man dauerhafte Verbindungen zum Server, gibt es zwei Möglichkeiten:

XML Sockets und das Action Message Format (AMF). Dauerhafte Verbindungen werden z.B. bei Multiplayer

Games, Chats oder zeitkritischen Anwendungen wie Börsenticker, Auktionen u.Ä. benötigt. Von einem verschwenderischen

Umgang mit dauerhaften Verbindungen ist abzuraten: Sie binden auf dem Server Ressourcen, weshalb auch nur eine

### Listing 3

#### ThreadedServer.java

```
import .... }

public class ThreadedServer { /**
    * Methode zum Starten des Servers
    * @throws IOException
    */
    static final int port = 3000;

    private Vector clientThreads = new Vector();

    // JDOM-Klasse zu Ausgabe von XML-Elementen
    private org.jdom.output.XMLOutputter xmlOut;

    public ThreadedServer() {
        xmlOut = new org.jdom.output.XMLOutputter(Format.
            getCompactFormat());
    } /**
    * Startmethode für den Server
    * @param args
    * @throws Throwable
    */
    public static void main(String args[]) throws Thrown
        able {
        ThreadedServer server = new ThreadedServer();
        server.startServer();
    }

    /**
    * verschicke Nachricht an einen bestimmten Client
    * @param element
    * @param clientThread
    */
    public void sendMessageToClient(Element element,
        ClientThread clientThread) {
        try {
            StringWriter sw = new StringWriter();
            xmlOut.output(element, sw);
            String message = sw.getBuffer().toString();
            clientThread.getPrintOut().print(message);
            clientThread.getPrintOut().write((byte) 0);
            clientThread.getPrintOut().flush();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
    ...
}
```

### Listing 4

#### ClientThread.java

```
import ... Stream()); catch (IOException ioe) {
    org.jdom.input.SAXBuilder builder = new org.jdom.
    input.SAXBuilder(); ...
}

public class ClientThread extends Thread {
    Socket socket = new Socket(); while (running) {
    BufferedReader readerIn; String str = readerIn.readLine();
    PrintStream printOut; if (str == null) {
    ThreadedServer server; running = false;
    }
    boolean running = true; else {
    try { /**
    // Flash schickt nach der ersten Abfrage immer erst
    // eine 0 und dann die Nachricht
    * @param element
    * @throws NickNameUsedException
    */
    private void handle(Element element) {
    Iterator iter = element.getAttributes().iterator();
    if (element.getName().equals("login")) {
    nickName = element.getAttribute("name")
        .getValue();
    }
    ...
    }
    ...
    }
    }

    /**
    * Nickname
    */
    private String nickName = "noname";

    public ClientThread(Socket socket, ThreadedServer
        server) {
        this.socket = socket;
        this.server = server;
    }

    public void run() {
        try {
            readerIn = new BufferedReader(new
                InputStreamReader(socket.getInputStream()));
            printOut = new PrintStream(socket.getOutputStream());
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

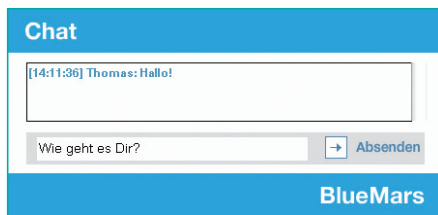


Abb. 3: Flash Chat, der mit XML Sockets kommuniziert

begrenzte Anzahl von parallelen dauerhaften Verbindungen möglich ist.

### XML-Socket-Server

Da Netzwerkprogrammierung mit Java sehr einfach ist, kann man schnell einen eigenen Multithreaded-Server implementieren. Für einen rudimentären Prototyp benötigt man lediglich zwei Klassen:

**Serverklasse:** Diese Klasse lauscht an einem bestimmten Port. Sobald sich ein Client verbindet, wird der ClientTread instanziiert und gestartet (Listing 3).

**ClientThread-Klasse:** Beim Anmelden eines Clients an den Server wird ein Objekt der Klasse *ClientThread* instanziiert und diesem Client zugewiesen. Dieses Objekt ist nun für die Kommunikation zwischen beiden Seiten zuständig, d.h., es empfängt die Daten des Clients und verschickt auch an diesen (Listing 4).

Anfragen und Antworten werden in XML-Elemente/-Attribute übertragen. Der Aufbau der XML-Elemente ist beliebig wählbar, jedoch sollte auf kurze Element- und Attributnamen geachtet werden, um den Overhead der Kommunikation möglichst gering zu halten. Sofern der aufrufende Browser Komprimierung unterstützt, können serverseitig die XML-Dokumente komprimiert gesendet werden, was eine deutliche Reduzierung der Datenmenge zur Folge hat.

### HTTP-Header

Die Es ist möglich, bei HTTP-POST-Anfragen über das Loadvars-Objekt oder das XML-Objekt auch HTTP-Header-Parameter zu setzen. Damit kann man z.B. Cookies oder Basic Authentification (ein schwacher Passwortschutz) setzen:

```
loadvarObject.setRequestHeader(key, value);
```

Die Session lässt sich serverseitig halten, indem man bei jedem Aufruf einen Cookie an den Server überträgt, der die eindeutige ID des Benutzers enthält (JSESSIONID).

Serverseitig wurde in diesem Beispiel die Open-Source-Bibliothek *jdom* ([www.jdom.org](http://www.jdom.org)) zum Parsen und Erstellen der XML-Elemente genutzt. Diese Bibliothek lässt sich sehr einfach nutzen, kann aber durch eine andere bevorzugte Bibliothek ersetzt werden. Auf Clientseite versendet man mit dem Code aus Listing 5 die Anfrage.

Vorteil dieser Methode ist, dass ein sehr schlanker Server implementiert werden kann. Alle Anforderungen, wie beispielsweise Zugriffsrechte oder eine Beschränkung der Anzahl der gleichzeitigen Client-Verbindungen, müssen jedoch selbst programmiert werden.

Aufgrund von Restriktionen des Flash-Plug-ins ist es nicht möglich, eine Verbindung mit einer Portnummer kleiner als 1024 aufzunehmen. Viele Firmen und zunehmend auch private Benutzer verwenden zudem Firewalls, die eine Kommunikation nur über definierte Standard-Ports (für Protokolle wie HTTP, POP3, IMAP, DNS) zulassen.

Bei der Verwendung eines XML-Socket-Servers muss bedacht werden, dass man weiterhin noch einen HTTP-Webserver benötigt, auf dem die Flash-Anwendung und die notwendigen HTML-Dateien abgelegt sind.

### Flash Remoting

Serverseitig betrachtet besteht Flash Remoting aus einem Servlet und Helferklassen. Das Servlet und die Helferklassen werden üblicherweise als Gateway bezeichnet. Ein Gateway kann mehrere Services zur Verfügung stellen, wobei ein Service die eigentliche Methode darstellt, die serverseitig aufgerufen wird. Um diese Methoden zu finden und aufzurufen, nutzt das Gateway die Introspection, die Analyse der Objekte zur Laufzeit. Das Action Message Format (AMF) ist ein proprietäres Binärformat von Adobe. Um über AMF zu kommunizieren, wird Adobe Flash Remoting benötigt. Flash Remoting besteht aus zwei Teilen:

- einer Erweiterung der Flash-Entwicklungsumgebung
- einem Server mit Unterstützung des AMF

Die Erweiterung für die-Flash Entwicklungsumgebung ist kostenlos ([www.adobe.com/de/products/flashremoting/downloads/components/](http://www.adobe.com/de/products/flashremoting/downloads/components/)). Nach Installation ist die Erweiterung in der Entwicklungsumgebung automatisch integriert. Für die Serverseite stehen mehrere kommerzielle und Open-Source-Varianten vorwiegend in den Programmiersprachen Java und PHP zur Verfügung:

Von Adobe gibt es Flex 2 Data Services und den ColdFusion Application Server. Die Open-Source-Varianten sind *amfphp* ([www.amfphp.org](http://www.amfphp.org)) und dessen Umsetzung in Java, *OpenAMF* ([sourceforge.net/projects/OpenAMF/](http://sourceforge.net/projects/OpenAMF/)).

### Lebenszyklus von Flash Remoting

1. Der Flash Client verbindet sich mit dem Gateway.
2. Es wird ein Callback Handler für die Antwort des Servers definiert.
3. Der Flash Client benennt den Service, den er aufrufen möchte.
4. Der Flash Client ruft die Methode auf der Serverseite optional mit Parametern auf.
5. Sind Parameter vorhanden, werden diese serialisiert.
6. Der Flash Player sendet einen HTTP Request an den Server, der die aufzurufende Komponente und Methode

### Adobe Flex 2

Das Framework Adobe Flex 2 umfasst nicht nur die reine Kommunikation, sondern ist vielmehr für die dynamische, serverseitige Generierung von Rich Internet Applications (RIA) ausgelegt. Mit der eigenen XML-basierten Skriptsprache MXML werden die Flash-Module zur Laufzeit erzeugt. Die Komponente Flex Data Services 2 Express ([www.adobe.com/cfusion/tdrc/index.cfm?product=flex](http://www.adobe.com/cfusion/tdrc/index.cfm?product=flex)) ist auch für die kommerzielle Nutzung kostenlos, allerdings darf der Server nur eine CPU haben und die Applikation nicht im Cluster laufen. In der Lieferung sind auch der Server JRun und mehrere Beispiele enthalten. Die Webapplikation läuft aber auch auf gängigen Servern, wie z.B. Tomcat, WebSphere, JBoss u.a., die den Servlet-Standard ab der Version 2.3 unterstützen.

Zu der dynamischen serverseitigen Erstellung von Flash-Applikationen zur Laufzeit gibt es auch eine Open-Source-Alternative: *OpenLaszlo* ([www.openlaszlo.org](http://www.openlaszlo.org)).

zusammen mit den serialisierten Parametern enthält.

7. Das Gateway erhält die Anfrage und deserialisiert die AMF-Objekte in Java-Objekte.
8. Um das entsprechende Objekt zu finden und dessen Methoden auszuführen, nutzt das Gateway entweder die Introspektion oder eine Mapping-Konfigurationsdatei.
9. Die Methode des Objektes wird ausgeführt und die Antwort wird an das Gateway zurück geliefert.
10. Die Antwort wird in das AMF serialisiert.
11. Das Gateway erzeugt und sendet die Antwort an den Flash Client zusammen mit der serialisierten Antwort zurück.
12. Flash empfängt asynchron die Antwort des Servers und deserialisiert den Rückgabewert in ein ActionScript-Objekt.
13. Flash ruft den Callback Handler für die Antwort des Servers auf.

Als Beispiel wird eine kleine Applikation mit OpenAMF als Gateway vorgestellt,

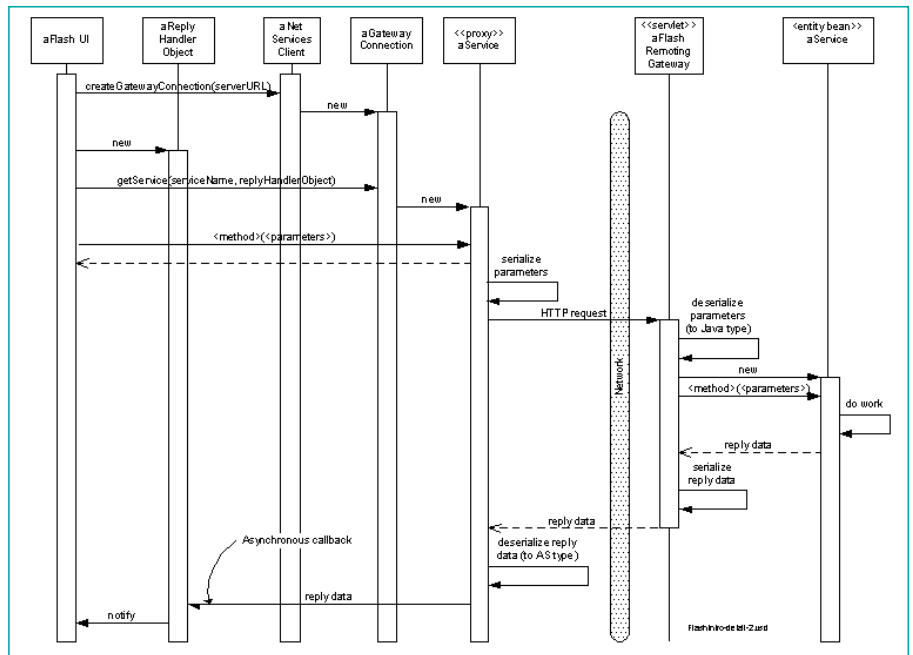


Abb. 4: Flash-Remoting-Infrastructure-Sequence-Diagramm von Adobe

OpenAMF ist für die meisten Anwendungszwecke vollkommen ausreichend. Es gibt zwar von OpenAMF ein umfangreiches Paket mit verschiedenen Beispielen, nur lei-

der ist es in der aktuellen Version 1.0RC12 nicht out of the box lauffähig. Nachfolgend eine Kurzanleitung für eine Beispielapplikation mit OpenAMF auf dem aktuellen

### Listing 5

#### chat fla (ActionScript 2.0)

```

/**
 * Erstellung des XML-Sockets
 */
var xmlsock:XMLSocket = new XMLSocket();
/**
 * Event Handler des XML-Socket-Objektes
 * @param success Boole'scher Wert ob der Aufruf
 * des Servers erfolgreich war.
 */
xmlsock.onConnect = function(success:Boolean) {
    if (success) {
        printStatus("Verbindung zu Server hergestellt.");
        gotoAndPlay(2);
    } else {
        printStatus("Verbindung zu Server fehlgeschlagen!",
            "error");
    }
};
/**
 * Verbindung zum Server erstellen
 */
xmlsock.connect("localhost", 3000);
/**
 * XML-Objekt zum Erzeugen von Nodes
 */
var xmlCreator:XML = new XML();

* Event Handler des XML-Socket-Objektes bei Empfang
* von Daten @param msg XML-Knoten als String
*/
xmlsock.onData = function(msg:String):Void {
    /**
     * Erstellen einer XML Node und Parsen des Strings
     */
    var node:XML = new XML();
    node.parseXML(msg);
    /**
     * Nodename = Befehl, Attribute = Parameter
     */
    var command:String = node.firstChild.nodeName;

    if (command == "message") {
        /**
         * Auslesen der XML-Attribute in Variablen
         */
        var messageText:String = node.firstChild.firstChild.
            nodeValue;
        var messageZeit:String = node.firstChild.
            attributes["zeit"];
        var nickName:String = node.firstChild.
            attributes["nick"];

        /**
         * Chatfenster-Textfeld aktualisieren
         */
        ausgabe_inhalt_str += "["+messageZeit+" "
            +nickName+" "+messageText+"\n";
    }
};
/**
 * Eventhandler fuer den Absenden-Button
 * eingabe_inhalt_str Wert des Eingabefelds für
 * Chatnachrichten abschicken_but Button zum Absenden
 * der Nachricht
 */
abschicken_but.onRelease = function() {
    if ( !(eingabe_inhalt_str == "" || eingabe_inhalt_str ==
        undefined) ) {
        /**
         * Erstellen einer Node und Zuweisen der Attribute
         */
        var node:XMLNode = xmlCreator.createElement
            ("message");
        /**
         * Senden der Anfrage mit folgendem Zeilenumbruch, da
         * die XML-Elemente zeilenweise eingelesen werden
         */
        xmlsock.send(node+"\n");
    }
};

```

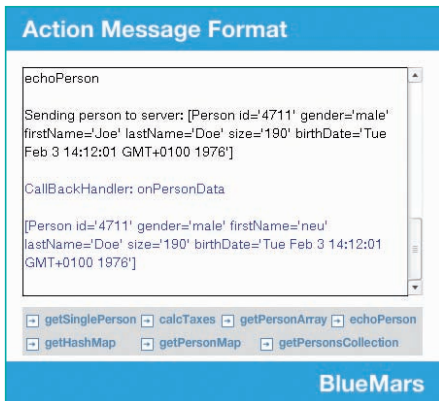


Abb. 5: Flash Client, der Objekte zum Server schickt und empfängt

Tomcat 5.5.20. Das Verzeichnis, in dem der Tomcat-Server installiert ist, wird nachfolgend `$TOMCAT_HOME` genannt, das ausgepackte OpenAMF-Verzeichnis heisst `$OPEN_AMF`.

Zunächst müssen das Tomcat-Archiv entpackt und anschließend folgende Verzeichnisse angelegt werden:

- `$TOMCAT_HOME/webapps/flashcomm`
- `$TOMCAT_HOME/webapps/flashcomm/WEB-INF`

### Listing 6

#### web.xml (Auszug)

```
...
<web-app>
<!-- definition gateway -->
<servlet>
<servlet-name>AdvancedGateway</servlet-name>
<display-name>AdvancedGateway</display-name>
<description>AdvancedGateway</description>
<servlet-class>org.OpenAMF.AdvancedGateway
</servlet-class>

<init-param>
<param-name>OPENAMF_CONFIG</param-name>
<param-value>/WEB-INF/openamf-config.xml
</param-value>
<description>Location of the OpenAMF config file.
</description>
</init-param>
<load-on-startup>1</load-on-startup>
</servlet>
<!-- url mapping -->
<servlet-mapping>
<servlet-name>AdvancedGateway</servlet-name>
<url-pattern>/gateway</url-pattern>
</servlet-mapping>
...
</web-app>
```

- `$TOMCAT_HOME/webapps/flashcomm/WEB-INF/lib`
- `$TOMCAT_HOME/webapps/flashcomm/WEB-INF/classes`

Im Lieferumfang von OpenAMF sind zwei Gateways enthalten, das DefaultGateway und das AdvancedGateway. Das DefaultGateway ist sehr einfach in der Nutzung. Man muss lediglich den Klassennamen, das Package und die Methodensignatur wissen und kann die entsprechenden Methoden aufrufen. Hier liegt jedoch auch eine Sicherheitslücke: Neben den Klassen bzw. Methoden, die aufgerufen werden sollen, können alle sichtbaren *public*-Methoden innerhalb der JVM aufgerufen werden. Deshalb sollte auf Produktionssystemen immer nur das AdvancedGateway genutzt

### Listing 7

#### Person.java (Auszug)

```
public class Person implements Serializable {

private int id;
private String gender;
private String firstName;
private String lastName;
private int age;
private double size;
private Date birthDate;

/**
 * Default-Konstruktor, wird für die Übersetzung eines
 * ActionScript-Objekts in ein Java-Objekt benötigt
 */
public Person() {
super();
}

public Person(int id, String gender, String firstName,
String lastName, int age, double size, Date birthDay) {
super();
this.id = id;
this.gender = gender;
this.firstName = firstName;
this.lastName = lastName;
this.age = age;
this.size = size;
this.birthDate = birthDay;
}

... Getter und Setter ...

public String toString() {
return "[id=" + id + " ... + "];"
}
}
```

werden, da alle Services, die bereitgestellt werden sollen, einzeln in einer Konfigurationsdatei definiert werden müssen. Diese Gateways sind HTTP Servlets und werden im Deployment Descriptor der Applikation definiert werden: `$TOMCAT_HOME/webapps/flashcomm/WEB-INF/web.xml`. Anschließend kopiert man alle Bibliotheken aus der OpenAMF-Webapplikation von `$OPEN_AMF/openamf.war/WEB-INF/lib/* .jar` nach `$TOMCAT_HOME/webapps/flashcomm/WEB-INF/lib`. Für dieses Beispiel wurden zwei Klassen definiert: ein Plain Old Java Object (POJO), das in beiden Richtungen ausgetauscht wird, und eine einfache Logik-Klasse. Die Logik-Klasse soll das Annehmen von Parametern sowie die Rückgabe von Objekten und Arrays übernehmen.

Nun muss die OpenAMF-Konfigurationsdatei für Nutzung des AdvancedGateway angepasst werden: Für jeden von Flash nutzbaren Service und dessen Methode(n) erfolgt ein Eintrag in der Datei `$TOMCAT_HOME/flashcomm/WEB-INF/openamf-config.xml`.

### Listing 8

#### SimpleExample.java

```
...
public class SimpleExample {

public double calcTaxes(double price, double taxes) {
return (price * (taxes+100))/100;
}

public Person getPerson() {
return new Person(1, "male", "joe", "doe", 37, 1.85,
new Date());
}

public Person echoPerson(Person person) {
person.setFirstName("neu");
return person;
}

public Collection getPersons() {
Collection<Person> persons = new HashSet<Person>();
persons.add(new Person(1, "male", "joe", "doe", 37,
1.85, new Date()));
persons.add(new Person(2, "female", "jane", "doe",
35, 1.55, new Date()));

return persons;
}
...
}
```

Die Übersetzung der ActionScript-Objekte in Java-Objekte und umgekehrt erfolgt durch den ASTranslator von Carbonfive, der in der Lieferung von OpenAMF enthalten ist. Dazu muss auf beiden Seiten der Applikation das Mapping des ActionScripts und der Java-Klassen definiert werden. Auf der Serverseite wird dies in der *openamf-config.xml* durch folgenden Eintrag definiert:

```
<custom-class-mapping>
<java-class>net.bluemars.flashcomm.Person</java-class>
```

```
<custom-class>net.bluemars.flashcomm.Person
</custom-class>
</custom-class-mapping>
```

Innerhalb der Flash-Applikation muss dasselbe mit folgendem Eintrag definiert werden:

```
Object.registerClass("net.bluemars.flashcomm.Person",
net.bluemars.flashcomm.Person);
```

Der Code des Flash-Clients sieht nun wie in Listing 10 aus.

Die Nutzung des binären Formats hat große Vorteile: Zum einen sind die Datenmengen deutlich geringer gegenüber der Kommunikation über XML-Elemente/-Dokumente und zum anderen erfolgen die Serialisierung und Deserialisierung der Objekte automatisch, sodass die Daten nicht zwischen Objekten und XML umständlich konvertiert werden müssen. Es kann eine dauerhafte Verbindung zwischen dem Flash Client und dem Server über den Standard-Port 80 hergestellt werden. Wichtig bei der Nutzung von AMF ist, immer das sichere Advanced-Gateway zu nutzen und jeden Service manuell einzutragen, da ansonsten die Anwendung große Sicherheitslücken hat.

### Fazit

Die gezeigten Technologien können Daten zwischen Flash Client und Java-Backend austauschen. Sie sind gut geeignet, um interaktive, zielgruppenorientierte und multimediale Webanwendungen mit Serverkommunikation zu programmieren.

Je nach Komplexität des jeweiligen Projekts kann schon die einfache Kommunikation über HTTP ausreichen. Bei kommunikationsintensiven und zeitkritischen Anwendungen empfehlen sich dauerhafte Verbindungen zum Server über einen eigenen XML-Socket-Server oder über ein AMF-Gateway. Dabei können fertige Bibliotheken wie OpenAMF oder die Flex 2 Data Services verwendet werden.

### Listing 9

#### openamf-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
<!-- true = ActionScript 1.0, false = ActionScript 2.0 -->
<amf-serializer>
<force-lower-case-keys>false</force-lower-case-keys>
</amf-serializer>

<!-- Required for DefaultGateway -->
<invoker>
<name>Java</name>
<class>org.openamf.invoker.JavaServiceInvoker<
/c>class>
</invoker>
<pageable-recordset>
<initial-data-row-count>20</initial-data-row-count>
</pageable-recordset>

<custom-class-mapping>
<java-class>net.bluemars.flashcomm.Person<
/c>java-class>

<custom-class>net.bluemars.flashcomm.Person<
/c>custom-class>
</custom-class-mapping>

<service>
<name>SimpleExample</name>
<service-location>net.bluemars.flashcomm.Simple
Service</service-location>
<invoker-ref>Java</invoker-ref>
<method>
<name>getPerson</name>
<parameter>
<type>*</type>
</parameter>
</method>
... Definition aller Services die bereitgestellt werden
sollen
</service>
</config>
```

### Listing 10

#### amf fla (ActionScript 2.0)

```
import ...
/**
* Registrieren der Klassen
*/
Object.registerClass("net.bluemars.flashcomm.
Person", net.bluemars.flashcomm.Person);
/**
* Erstellen des Services
*/
var myService:Service = new Service
("http://localhost:8080/flashcomm/gateway", null,
"SimpleExample", null, null);
/**
* Handler für Fehlermeldungen
*/
function onEchoFault(rs:ResultEvent) {
this.printStatus(msg.result[i], "error");
}
/**
* Callback Handler für die einzelne Person
*/

function onPersonData(msg:ResultEvent) {
var person:Person = Person(msg.result);
// Ausgabe der Person
trace(person);
}
/**
* Handler für Betätigen der Buttons
*/
single_person_btn.onRelease = function() {
/**
* Objekt, das aus von dem Service zurückgegeben
* wird, um die Ergebnisse und Fehler des Aufrufs zu
* verarbeiten, da dieser Aufruf asynchron erfolgt.
*/
var pc:PendingCall = myService.getPerson();
/**
* Festlegen, welches Objekt und welche Methoden
* beim Eintreffen der asynchronen Antwort des
* Servers bei erfolgreichem oder fehlgeschlagenem
* Aufruf ausgeführt werden.
*/
}
```



Thomas Kiesel ist Dipl. Inf. (FH) und seit 2003 Softwareentwickler bei BlueMars. Seine Tätigkeiten liegen bei der Konzeption und Entwicklung von Java EE-Webapplikationen. Kontakt: [thomas.kiesel@bluemars.net](mailto:thomas.kiesel@bluemars.net).

### Links & Literatur

- [1] [sourceforge.net/projects/openamf/](http://sourceforge.net/projects/openamf/)
- [2] [www.adobe.com/de/products/flex/](http://www.adobe.com/de/products/flex/)
- [3] [carbonfive.sourceforge.net](http://carbonfive.sourceforge.net)
- [4] Flash Remoting for J2EE Developers: [www.onjava.com/pub/a/onjava/2003/02/26/flash\\_remoting.html](http://www.onjava.com/pub/a/onjava/2003/02/26/flash_remoting.html)
- [5] [www.devx.com/webdevArticle/21803/0/page/1](http://www.devx.com/webdevArticle/21803/0/page/1)
- [6] Robert Hülsey: Java & Flash – Rich-Client-Frontends mit Macromedia Flex, in *Java Magazin* 12.2004

Feedback

Die Redaktion freut sich über Ihr Artikel-Feedback: [feedback@javamagazin.de](mailto:feedback@javamagazin.de)

